



Bilkent University

Department of Computer Engineering

CS 353 - Database Systems

Tour Reservation Management System

Design Report

- Group 4 -

Mahmud Sami Aydın

Halil Şahiner

Abdullah Talayhan

Emin Bahadır Tülüçe

Table of Contents

1. Revised E/R Model	4
1.1. Changes made in E/R Diagram	4
1.2. Revised E/R Diagram	5
2. Relation Schemas	7
2.1. Account	7
2.2. StaffAccount	7
2.3. CustomerAccount	7
2.4. Dependent	8
2.5. PromotionCard	9
2.6. CustomerPromotionCards	9
2.7. CustomerTelephones	9
2.8. PaymentMethod	10
2.9. CustomerPaymentMethods	10
2.10. Reservation	11
2.11. ReservationCancel	11
2.12. BackupReservation	12
2.13. IncludedDependents	12
2.14. Tour	13
2.15. TourCancel	13
2.16. Tag	14
2.17. TourTags	14
2.18. TourDay	15
2.19. Accommodation	15
2.20. AccommodationPlace	16
2.21. Hostel	16
2.22. Hotel	16
2.23. TripEvent	17
2.24. City	17
2.25. Country	18
2.26. TravelRoute	18
2.27. TravelCompany	19
2.28. Bus	19
2.29. Train	19
2.30. Plane	20
2.31. CruiseShip	20
2.32. CustomerTravels	20
2.33. CustomerAccommodates	21

2.34. DependentAccommodates	22
3. Functional Dependencies and Normalization	22
4. Functional Components	23
4.1. Use Cases / Scenarios	23
4.1.1. User Account	23
4.1.1.1. Register	23
4.1.1.2. Login	24
4.1.1.3. List the Tours	24
4.1.2 Customer Account	25
4.1.2.1. Make a Reservation to a Tour	26
4.1.2.2. Cancel a Reservation to a Tour	26
4.1.2.3. Manage Account	27
4.1.3. Staff Account	29
4.1.3.1. Create a Tour	29
4.1.3.2 Manage Staff Account	29
4.1.3.2 See Created Tour	30
4.1.3.3 Manage Tour	30
4.2. Algorithms	31
4.2.1. Tour Filtering & Sorting Algorithm	31
4.2.2. Reservation Algorithm	31
4.2.3. Reservation Cancelling Algorithm	31
4.2.4. Earning & Spending Booking Points	31
4.2.5. Using Promotion Cards	32
4.3. Data Structures	32
5. User Interface Design and SQL Statements	33
5.1. General Pages	33
5.1.1. Home Page (without login)	33
5.1.2. Tour Listing Page	34
5.1.3. Login Page	35
5.2. Customer Pages	36
5.2.1. Home Page (as customer)	36
5.2.2. Tour Details Page (as customer)	37
5.2.3. Tour Reservation Page	39
5.2.4. Payment Page	40
5.2.5. My Account Page	41
5.3. Staff Pages	44
5.3.1. Home Page (as staff)	44
5.3.2. Tour Adding Page	45
5.3.3. Tour Details Page (as staff)	47

6. Advanced Database Components	50
6.1. Reports	50
6.1.1. Most visited cities of the month	50
6.1.2. Top revenue-making countries of the year	50
6.2. Views	51
6.2.1. Tour Preview	51
6.2.2. Tour Associations	52
6.3. Triggers	54
6.3.1. Quota Backup Trigger	54
6.3.2. Re-Reservation Trigger	54
6.4. Constraints	55
6.5. Stored Procedures	55
6.5.1. Searching for Tours	55
6.5.2. Filtering Tours	55
7. Implementation Plan	56
8. Website	56

1. Revised E/R Model

1.1. Changes made in E/R Diagram

The following changes are made in our E/R diagram:

- Since our system will operate for only one tour agency, we have deleted AgencyAccount, SystemAdminAccount and TravelAgency entities. There is only CustomerAccount and StaffAccount entities now. A staff account is able to create a new tour on the system where a customer account can see and register to tours. Also, the creator for each tour is stored in the added_by relation. Other than that, we have added telephone_no attribute to the CustomerAccount entity.
- Our tour/sub-tour relation was ambiguous, thus we have revised it to have a Tour and a TourDay entities. Now, all tours are consisting of sub-tours and a sub-tour is defined as a day in the tour. Every day in the tour has a description to make the schedule clearer. Other than that, tours are associated to the travel routes, accommodations and trip events which are included in them.
- Our customer-to-traveler relation and traveler-to-traveler dependency relation were ambiguous, thus we have revised to have a CustomerAccount entity along with its Dependent entities that might be children of them. Those dependents of customers have also have room no's on the tours that they are attending. They share the same PNR number for travel routes, with their dependees.
- We have added the CancelInfo entity to store the cancelled reservations with their cancellation dates. Also, we have added the functionality to cancel a whole tour while storing its reason and date.
- We have added PromoCard entity to represent a defined promotion card that could be associated to customers via a promo-code entering process.
- We have separated the tags from tours as a Tag entity to make the tours filterable with tags.
- We have added City and Country entities, for making it possible to filter tours according to their cities/countries. Also, the nationality of a customer is now not an attribute but a relation with Country entity.

NOTE: Here, tours do not have a specific city or country since they can have different trips in different cities or even different countries. So a tour is

associated to one or more cities indirectly, through its trip cities and accommodation cities.

- The Accommodation entity divided into Accommodation and AccommodationPlace entities for avoiding duplication of information (e.g. for name and address of a hotel) and also for making it possible for different tours to have accommodation in the same place.
- We have revised TripEvent, TravelRoute and Accommodation entities to get rid of the ternary relations with customers. Now, all tour days have some of trip events, travel routes and accommodations defined in them, by the creator of the tour. Now, travel routes have pnr numbers, accommodations have rooms.
 - Moreover, each customer that have made a reservation to a tour has a relation with a Room for all of the accommodations in that tour.
 - Similarly, each customer that have made a reservation to a tour has a relation with a PNR number for all of the travel routes in that tour. Using the PNR number travelers are able to learn their seat numbers and check-in details from the website of the associated travel company.
- For TravelRoutes entity, we have separated the TravelCompany as a different entity to not have duplicated/inconsistent travel company names. This also allows us to have a feature for customers to filter out some tours which have a travel company that they do not desire to travel with.
- We have also added generalization entities as
 - Hotel, Hostel for Accommodation
 - Bus, Train, Plane, CruiseShip for TravelRoute

to make the distinction between those clearer and keep our entities in a more organized way.

1.2. Revised E/R Diagram

The revised version of our E/R diagram is given in Figure 1.

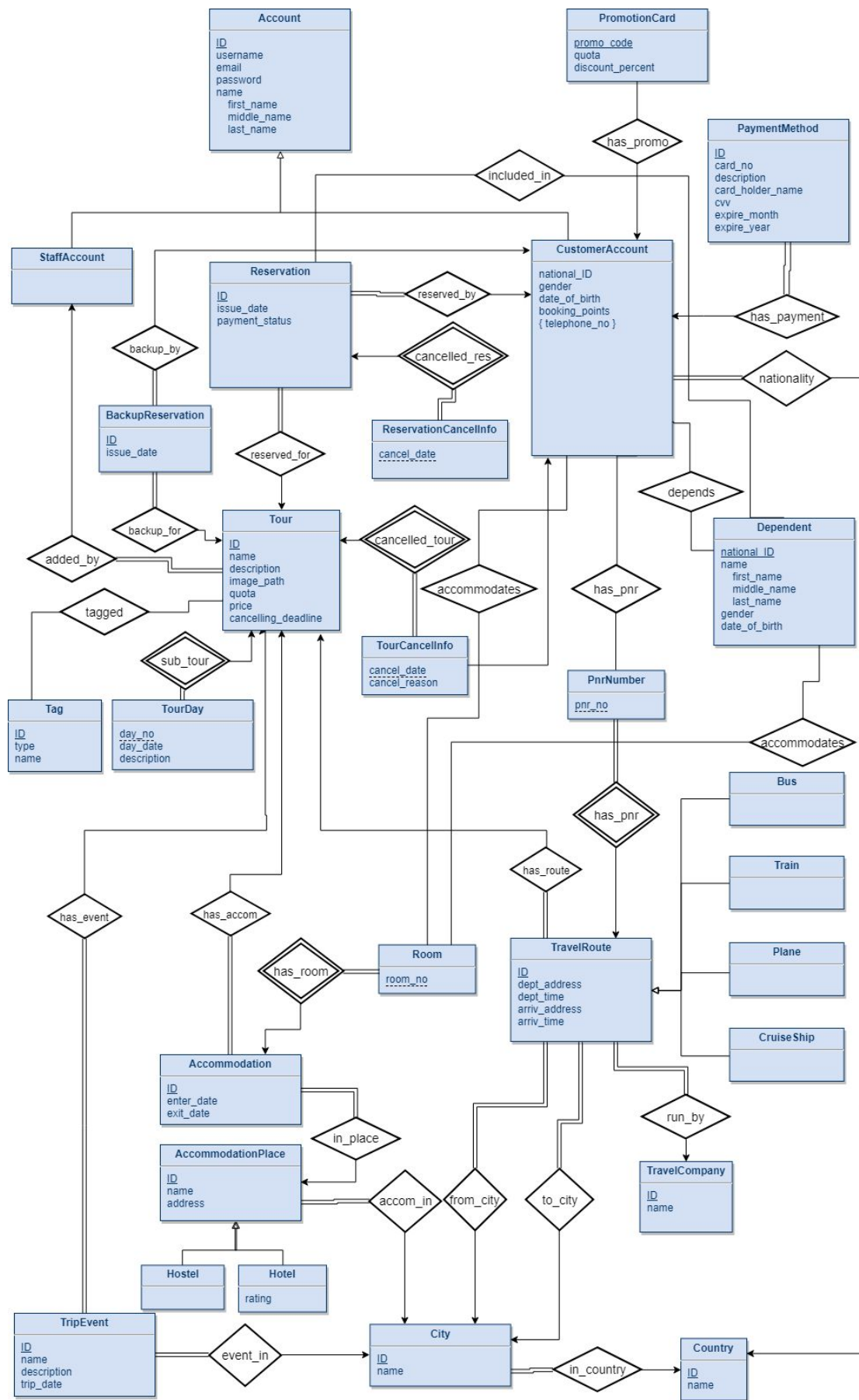


Figure 1: E/R Diagram

2. Relation Schemas

2.1. Account

Relational Model: Account(ID, username, email, passwd, first_name, middle_name, last_name)

Functional Dependencies: { (ID \rightarrow username, email, passwd, first_name, middle_name, last_name), (username \rightarrow ID) }

Candidate Keys: { {ID}, {username} }

Primary Key: {ID}

Foreign Keys: { }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Account (  
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,  
    username VARCHAR(50) NOT NULL,  
    email VARCHAR(50) NOT NULL,  
    passwd VARCHAR(50) NOT NULL,  
    first_name VARCHAR(50),  
    middle_name VARCHAR(50),  
    last_name VARCHAR(50)  
);
```

2.2. StaffAccount

Relational Model: StaffAccount(ID)

Functional Dependencies: { }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { (ID \rightarrow Account.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE StaffAccount (  
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,  
    FOREIGN KEY (ID) REFERENCES Account(ID)  
);
```

2.3. CustomerAccount

Relational Model: CustomerAccount(ID, nationality, national_ID, gender, date_of_birth, booking_point)

Functional Dependencies: { (ID → national_ID, nationality, gender, date_of_birth, booking_points) }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { (ID → Account.ID), (nationality → Country.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE CustomerAccount (  
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,  
    national_ID VARCHAR(15),  
    nationality VARCHAR(10),  
    gender VARCHAR(6) CHECK (gender IN ('Female', 'Male')),  
    date_of_birth DATE,  
    booking_points INTEGER DEFAULT 0,  
    FOREIGN KEY (ID) REFERENCES Account(ID),  
    FOREIGN KEY (nationality) REFERENCES Country(ID)  
);
```

2.4. Dependent

Relational Model: Dependent(customer_ID, national_ID, gender, date_of_birth, first_name, middle_name, last_name)

Functional Dependencies: { (customer_ID, national_ID → gender, date_of_birth, first_name, middle_name, last_name) }

Candidate Keys: { {customer_ID, national_ID} }

Primary Key: {customer_ID, national_ID}

Foreign Keys: { }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Dependent (  
    customer_ID INTEGER,  
    national_ID INTEGER,  
    gender VARCHAR(6) NOT NULL,  
    date_of_birth DATE NOT NULL,  
    first_name VARCHAR(50),  
    middle_name VARCHAR(50),  
    last_name VARCHAR(50),  
    FOREIGN KEY (customer_ID) REFERENCES CustomerAccount(ID),  
    PRIMARY KEY (customer_ID, national_ID),  
    CHECK (gender IN ('Female', 'Male'))  
);
```

2.5. PromotionCard

Relational Model: PromotionCard(promo_code, quota, discount_percent)

Functional Dependencies: { (promo_code → quota, discount_percent) }

Candidate Keys: { {promo_code} }

Primary Key: {promo_code}

Foreign Keys: { }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE PromotionCard (  
    promo_code VARCHAR(10) PRIMARY KEY,  
    quota INTEGER NOT NULL,  
    discount_percent NUMERIC(3,0) NOT NULL,  
    CHECK (0 < discount_percent AND discount_percent <= 100)  
);
```

2.6. CustomerPromotionCards

Relational Model: CustomerPromotionCards(customer_ID, promo_code)

Functional Dependencies: { }

Candidate Keys: { {customer_ID, promo_code} }

Primary Key: {customer_ID , promo_code}

Foreign Keys: { (customer_ID → CustomerAccount.ID, promo_code → PromotionCard.promo_code) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE CustomerPromotionCards (  
    customer_ID INTEGER,  
    promo_code VARCHAR(10),  
    PRIMARY KEY (customer_ID, promo_code ),  
    FOREIGN KEY (customer_ID) REFERENCES CustomerAccount(ID),  
    FOREIGN KEY (promo_code) REFERENCES  
    PromotionCard(promo_code)  
);
```

2.7. CustomerTelephones

Relational Model: CustomerTelephones(customer_ID, telephone_no)

Functional Dependencies: { }

Candidate Keys: { {customer_ID, telephone_no} }

Primary Key: {customer_ID, telephone_no}

Foreign Keys: { (customer_ID → CustomerAccount.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE CustomerTelephones (  
    customer_ID INTEGER,  
    telephone_no NUMERIC(15, 0),  
    PRIMARY KEY (customer_ID , telephone_no),  
    FOREIGN KEY (customer_ID) REFERENCES CustomerAccount(ID),  
    CHECK (telephone_no > 100000000)  
);
```

2.8. PaymentMethod

Relational Model: PaymentMethod(ID, card_no, description, card_holder_name, cvv, expire_month, expire_year)

Functional Dependencies: { (ID → card_no, name, expire_month, expire_year) }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE PaymentMethod (  
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,  
    card_no NUMERIC(16, 0)  
        CHECK (card_no >= 10000000000000000),  
    name VARCHAR(30) NOT NULL,  
    expire_month NUMERIC(2, 0)  
        CHECK (1 <= expire_month AND expire_month <= 12),  
    expire_year NUMERIC(4, 0)  
        CHECK (expire_year >= 2018)  
);
```

2.9. CustomerPaymentMethods

Relational Model: CustomerPaymentMethods(customer_ID, payment_method_ID)

Functional Dependencies: { }

Candidate Keys: { {customer_ID, payment_method_ID} }

Primary Key: {customer_ID, payment_method_ID}

Foreign Keys: { (customer_ID → CustomerAccount.ID), (payment_method_ID → PaymentMethod.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE CustomerPaymentMethods (  
    customer_ID INTEGER,  
    payment_method_ID INTEGER,  
    FOREIGN KEY (customer_ID) REFERENCES CustomerAccount(ID),  
    FOREIGN KEY (payment_method_ID) REFERENCES  
PaymentMethods(ID),  
    PRIMARY KEY (customer_ID, payment_method_ID)  
);
```

2.10. Reservation

Relational Model: Reservation(ID, customer_ID, tour_ID, issue_date, payment_status)

Functional Dependencies: { (ID → customer_ID, tour_ID, issue_date, payment_status), (customer_ID, tour_ID → ID) }

Candidate Keys: { {ID}, {customer_ID, tour_ID} }

Primary Key: {ID}

Foreign Keys: { (customer_ID → CustomerAccount.ID), (tour_ID → Tour.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Reservation (  
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,  
    customer_ID INTEGER,  
    tour_ID INTEGER,  
    issue_date DATETIME NOT NULL,  
    payment_status VARCHAR(6) NOT NULL CHECK (payment_status IN  
( 'Paid' , 'Unpaid' )),  
    FOREIGN KEY (customer_ID) REFERENCES CustomerAccount(ID),  
    FOREIGN KEY (tour_ID) REFERENCES Tour(ID)  
);
```

2.11. ReservationCancel

Relational Model: ReservationCancel(reservation_ID, cancel_date)

Functional Dependencies: { }

Candidate Keys: { {reservation_ID , cancel_date } }

Primary Key: {reservation_ID, cancel_date}

Foreign Keys: { (reservation_ID → Reservation.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE ReservationCancel (  
    reservation_ID INTEGER,  
    cancel_date DATETIME,  
    PRIMARY KEY (reservation_ID, cancel_date),  
    FOREIGN KEY (reservation_ID) REFERENCES Reservation(ID)  
);
```

2.12. BackupReservation

Relational Model: BackupReservation(ID, customer_ID, tour_ID, issue_date)

Functional Dependencies: { (ID → customer_ID, tour_ID, issue_date),
(customer_ID, tour_ID → ID) }

Candidate Keys: { {ID}, {customer_ID, tour_ID} }

Primary Key: {ID}

Foreign Keys: { (customer_ID → CustomerAccount.ID), (tour_ID → Tour.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE BackupReservation (  
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,  
    customer_ID INTEGER,  
    tour_ID INTEGER,  
    issue_date DATETIME NOT NULL,  
    FOREIGN KEY (customer_ID) REFERENCES CustomerAccount(ID),  
    FOREIGN KEY (tour_ID) REFERENCES Tour(ID)  
);
```

2.13. IncludedDependents

Relational Model: IncludedDependents(reservation_ID, customer_ID, dependent_ID)

Functional Dependencies: { }

Candidate Keys: { {reservation_ID, customer_ID, dependent_ID} }

Primary Key: {reservation_ID, customer_ID, dependent_ID}

Foreign Keys: { (reservation_ID → Reservation.ID), (customer_ID → CustomerAccount.ID), (dependent_ID → Dependent.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE IncludedDependents(  
    reservation_ID INTEGER,  
    customer_ID INTEGER,  
    dependent_ID INTEGER,  
    PRIMARY KEY (reservation_ID, customer_ID, dependent_ID),  
    FOREIGN KEY (reservation_ID) REFERENCES Reservation(ID),  
    FOREIGN KEY (customer_ID ) REFERENCES CustomerAccount(ID),  
    FOREIGN KEY (dependent_ID ) REFERENCES Dependent(ID)  
);
```

2.14. Tour

Relational Model: Tour(ID, name, description, image_path, quota, price, creator_ID, cancelling_deadline)

Functional Dependencies: { (ID → name, description, image_path, quota, price, creator_ID, cancelling_deadline) }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { (creator_ID → StaffAccount.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Tour (  
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(20) NOT NULL,  
    description VARCHAR(1000),  
    image_path VARCHAR(500),  
    quota INTEGER DEFAULT 0,  
    price NUMERIC (7,2) DEFAULT 0,  
    creator_ID INTEGER,  
    cancelling_deadline DATETIME NOT NULL,  
    FOREIGN KEY (creator_ID) REFERENCES StaffAccount(ID)  
);
```

2.15. TourCancel

Relational Model: TourCancel(tour_ID, cancel_date, cancel_reason)

Functional Dependencies: { (tour_ID, cancel_date → cancel_reason) }

Candidate Keys: { {tour_ID, cancel_date} }

Primary Key: {tour_ID, cancel_date}

Foreign Keys: { tour_ID → tour.ID }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE TourCancel (  
    tour_ID INTEGER,  
    cancel_date DATETIME NOT NULL,  
    cancel_reason VARCHAR(300),  
    PRIMARY KEY (tour_ID, cancel_date),  
    FOREIGN KEY (tour_ID) REFERENCES Tour(ID)  
);
```

2.16. Tag

Relational Model: Tag(ID, name, type)

Functional Dependencies: { (ID → name, type) }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Tag (  
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR( 15) NOT NULL,  
    type VARCHAR(15) NOT NULL  
);
```

2.17. TourTags

Relational Model: TourTags(tour_ID, tag_ID)

Functional Dependencies: { }

Candidate Keys: { {tour_ID, tag_ID} }

Primary Key: {tour_ID, tag_ID}

Foreign Keys: { (tour_ID → tour.ID), (tag_ID → tag.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE TourTags (  
    tour_ID INTEGER,  
    tag_ID INTEGER,  
    PRIMARY KEY (tour_ID, tag_ID),  
    FOREIGN KEY (tour_ID) REFERENCES Tour(ID),
```

```
FOREIGN KEY (tag_ID) REFERENCES Tag(ID)
);
```

2.18. TourDay

Relational Model: TourDay(tour_ID, day_no, day_date, description)

Functional Dependencies: { (tour_ID, day_no → day_date, description) }

Candidate Keys: { {tour_ID, day_no} }

Primary Key: {tour_ID, day_no}

Foreign Keys: { (tour_ID → Tour.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE TourDay (
    tour_ID INTEGER,
    day_no INTEGER,
    day_date DATE,
    description VARCHAR(1000),
    FOREIGN KEY (tour_ID) REFERENCES Tour(ID),
    PRIMARY KEY (tour_ID, day_no)
);
```

2.19. Accommodation

Relational Model: Accommodation(ID, tour_ID, place_ID, enter_date, exit_date)

Functional Dependencies: { (ID → tour_ID, place_ID, enter_date, exit_date) }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: {(tour_ID → Tour.ID), (place_ID → AccommodationPlace.ID)}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Accommodation (
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,
    tour_ID INTEGER,
    place_ID INTEGER,
    enter_date DATETIME,
    exit_date DATETIME,
    FOREIGN KEY (tour_ID) REFERENCES Tour(ID),
    FOREIGN KEY (place_ID) REFERENCES AccommodationPlace(ID)
);
```


2.20. AccommodationPlace

Relational Model: AccommodationPlace(ID, city_ID, name, address)

Functional Dependencies: { (ID \rightarrow city_ID, name, address) }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { city_ID \rightarrow city.ID }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE AccommodationPlace (  
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,  
    city_ID INTEGER,  
    name VARCHAR(45) NOT NULL,  
    name VARCHAR(1000),  
    FOREIGN KEY (city_ID) REFERENCES City(ID)  
);
```

2.21. Hostel

Relational Model: Hostel(ID)

Functional Dependencies: { }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { (ID \rightarrow AccommodationPlace.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Hostel (  
    ID INTEGER PRIMARY KEY,  
    FOREIGN KEY (ID) REFERENCES AccommodationPlace(ID)  
);
```

2.22. Hotel

Relational Model: Hotel(ID, rating)

Functional Dependencies: { (ID \rightarrow rating) }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { (ID \rightarrow AccommodationPlace.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Hotel (  
    ID INTEGER PRIMARY KEY,  
    rating NUMERIC(1,0) DEFAULT 0,  
    FOREIGN KEY (ID) REFERENCES AccommodationPlace(ID),  
    CHECK ( rating <= 5 )  
);
```

2.23. TripEvent

Relational Model: TripEvent(ID, tour_ID, city_ID, name, description, trip_date)

Functional Dependencies: { (ID → tour_ID, city_ID, name, description, trip_date) }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { (tour_ID → Tour.ID), (city_ID → City.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE TripEvent (  
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,  
    tour_ID INTEGER,  
    city_ID INTEGER,  
    name VARCHAR(20),  
    description VARCHAR(300),  
    trip_date DATETIME,  
    FOREIGN KEY (city_ID) REFERENCES City(ID),  
    FOREIGN KEY (tour_ID) REFERENCES Tour(ID)  
);
```

2.24. City

Relational Model: City(ID, country_ID, name)

Functional Dependencies: { (ID → country_ID , name) }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: {ID}

Normal Form: BCNF

Table Definition:

```
CREATE TABLE City (  
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(10) NOT NULL,  
    country_ID INTEGER NOT NULL,
```

```
FOREIGN KEY (country_ID) REFERENCES country(ID)
);
```

2.25. Country

Relational Model: Country(ID, name)

Functional Dependencies: { (ID → name) }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Country (
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(10) NOT NULL
);
```

2.26. TravelRoute

Relational Model: TravelRoute(ID, tour_ID, from_city_ID, to_city_ID, company_ID, dept_address, dept_time, arriv_address, arriv_time)

Functional Dependencies: { (ID → tour_ID, from_city_ID, to_city_ID, company_ID, dept_address, dept_time, arriv_address, arriv_time) }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { (tour_ID → Tour.ID), (from_city_ID → City.ID), (to_city_ID → City.ID), (company_ID → TravelCompany.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE TravelRoute (
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,
    tour_ID INTEGER,
    from_city_ID INTEGER,
    to_city_ID INTEGER,
    company_ID INTEGER,
    dept_address VARCHAR(60),
    dept_time DATETIME,
    arriv_address VARCHAR(60),
    arriv_time DATETIME,
    FOREIGN KEY (tour_ID) REFERENCES Tour(ID),
    FOREIGN KEY (from_city_ID) REFERENCES City(ID),
    FOREIGN KEY (to_city_ID) REFERENCES City(ID),
    FOREIGN KEY (company_ID) REFERENCES TravelCompany(ID)
);
```

```

FOREIGN KEY (to_city_ID) REFERENCES City(ID),
FOREIGN KEY (company_ID) REFERENCES TravelCompany(ID)
);

```

2.27. TravelCompany

Relational Model: TravelCompany(ID, name)

Functional Dependencies: { (ID → name) }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { }

Normal Form: BCNF

Table Definition:

```

CREATE TABLE TravelCompany (
    ID INTEGER PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(20)
);

```

2.28. Bus

Relational Model: Bus(ID)

Functional Dependencies: { }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { (ID → TravelRoute.ID) }

Normal Form: BCNF

Table Definition:

```

CREATE TABLE Bus (
    ID INTEGER PRIMARY KEY,
    FOREIGN KEY (ID) REFERENCES TravelRoute( ID )
);

```

2.29. Train

Relational Model: Train(ID)

Functional Dependencies: { }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { (ID → TravelRoute.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Train (  
    ID INTEGER PRIMARY KEY,  
    FOREIGN KEY (ID) REFERENCES TravelRoute(ID)  
);
```

2.30. Plane

Relational Model: Plane(ID)

Functional Dependencies: { }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { (ID → TravelRoute.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE Plane (  
    ID INTEGER PRIMARY KEY,  
    FOREIGN KEY (ID) REFERENCES TravelRoute( ID )  
);
```

2.31. CruiseShip

Relational Model: CruiseShip(ID)

Functional Dependencies: { }

Candidate Keys: { {ID} }

Primary Key: {ID}

Foreign Keys: { (ID → TravelRoute.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE CruiseShip (  
    ID INTEGER PRIMARY KEY,  
    FOREIGN KEY (ID) REFERENCES TravelRoute(ID)  
);
```

2.32. CustomerTravels

Relational Model: CustomerTravels(customer_ID, travel_route_ID, pnr_no)

Functional Dependencies: { (customer_ID, travel_route_ID → pnr_no) }

Candidate Keys: { {customer_ID, travel_route_ID} }

Primary Key: {customer_ID, travel_route_ID}

Foreign Keys: { (customer_ID → CustomerAccount.ID), (travel_route_ID → TravelRoute.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE CustomerTravels (  
    travel_route_ID INTEGER,  
    customer_ID INTEGER,  
    pnr_no VARCHAR(10),  
    FOREIGN KEY (travel_route_ID) REFERENCES TravelRoute(ID),  
    FOREIGN KEY (customer_ID) REFERENCES Customer(ID),  
    PRIMARY KEY (travel_route_ID, customer_ID)  
);
```

2.33. CustomerAccommodates

Relational Model: CustomerAccommodates(customer_ID, accommodation_ID, room_no)

Functional Dependencies: { customer_ID, accommodation_ID → room_no }

Candidate Keys: { {customer_ID, accommodation_ID} }

Primary Key: {customer_ID, accommodation_ID}

Foreign Keys: { (customer_ID → CustomerAccount.ID), (accommodation_ID → Accommodation.ID) }

Normal Form: BCNF

Table Definition:

```
CREATE TABLE CustomerAccommodates (  
    accommodation_ID INTEGER,  
    customer_ID INTEGER,  
    room_no VARCHAR(6),  
    FOREIGN KEY (accommodation_ID) REFERENCES  
Accommodation(ID),  
    FOREIGN KEY (customer_ID) REFERENCES Customer(ID),  
    PRIMARY KEY (accommodation_ID, customer_ID)  
);
```

2.34. DependentAccommodates

Relational Model: DependentAccommodates(customer_ID, national_ID, accommodation_ID, room_no)

Functional Dependencies: { (customer_ID, national_ID, accommodation_ID → room_no) }

Candidate Keys: { {customer_ID, national_ID, accommodation_ID} }

Primary Key: {customer_ID, national_ID, accommodation_ID}

Foreign Keys: { (customer_ID → Customer.ID), (national_ID → Dependent.national_ID), (accommodation_ID → Accommodation.ID) }

Normal Form: BCNF

Table Definition:

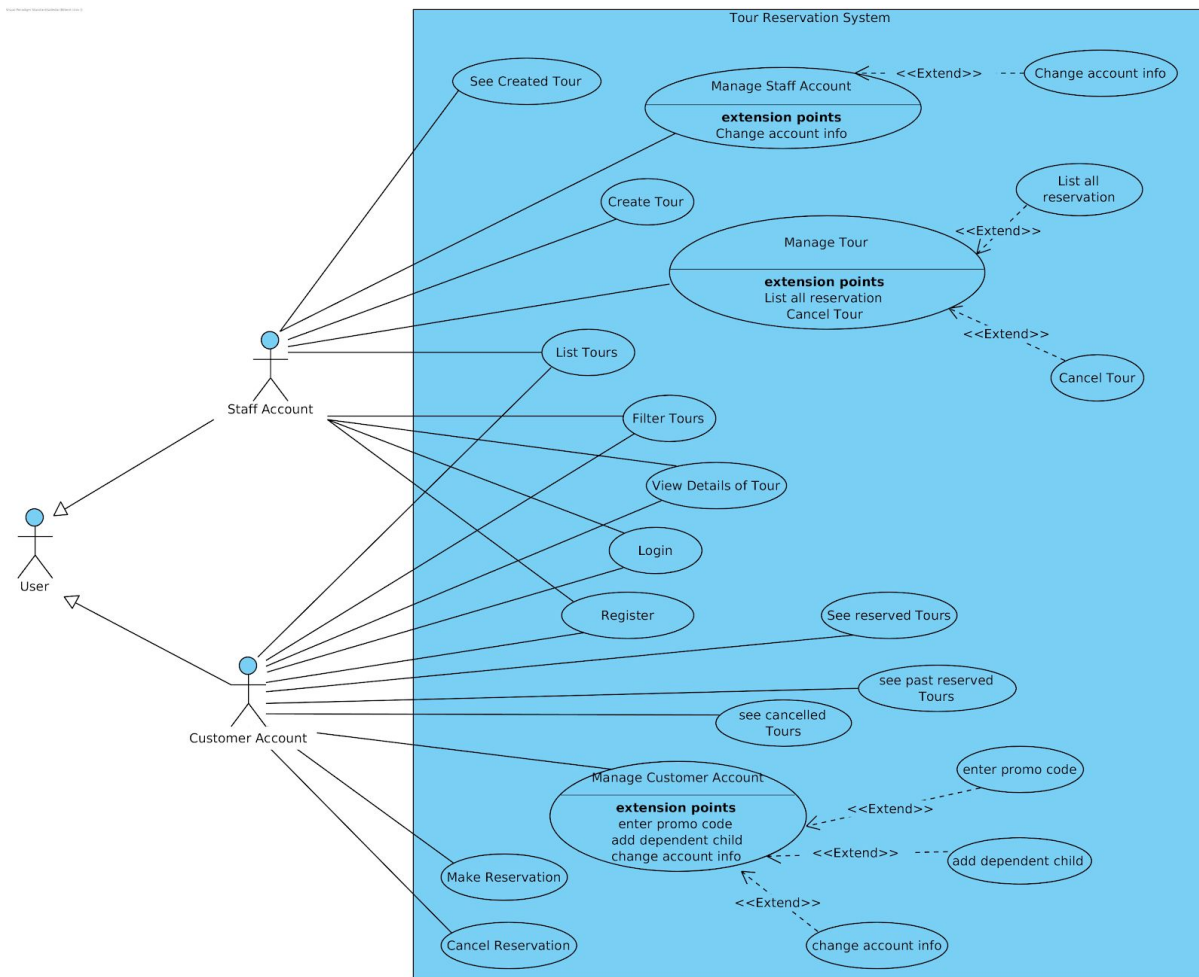
```
CREATE TABLE DependentAccommodates (  
    customer_ID INTEGER,  
    national_ID INTEGER,  
    accommodation_ID INTEGER,  
    room_no VARCHAR(6),  
    FOREIGN KEY (accommodation_ID) REFERENCES  
Accommodation(ID),  
    FOREIGN KEY (customer_ID) REFERENCES Customer(ID),  
    FOREIGN KEY (national_ID) REFERENCES Dependent(national_ID),  
    PRIMARY KEY (accommodation_ID, national_ID ,customer_ID)  
);
```

3. Functional Dependencies and Normalization

In the previous section, we have given functional dependencies of the tables. All of the tables are designed to be in BCNF. Thus, we did not perform any decomposition or normalization on the tables.

4. Functional Components

4.1. Use Cases / Scenarios



4.1.1. User Account

The main use cases for a user are given in this section.

4.1.1.1. Register

Use Case: Register to Tour Reservation System

Primary Actor: User

Stakeholders and interests:

- User who wants to create an account.

Pre-conditions:

- User must log out if any User account logged in on that system

- User must be in Registration screen.

Successful Scenario Event Flow for Register:

1. User enters username, email, and password to register the system.
2. User submit registration information to the system.
3. User receives an acceptance screen.

Unsuccessful Scenario Event Flow for Register:

1. User enters username, email, and password to register the system.
2. User submit registration information to the system.
3. User receives an error message that shows the User couldn't be registered to system.

4.1.1.2. Login**Use Case: Login to Tour Reservation System**

Primary Actor: User

Stakeholders and interests:

- User who wants to login to an account

Pre-conditions:

- User must log out if any account is logged in on the system
- User must be in login screen.

Successful Scenario Event Flow for Login:

1. User enters username and password to register the system.
2. User submit login information to the system.
3. User receives an acceptance screen.

Unsuccessful Scenario Event Flow for Login:

1. User enters username and password to login the system.
2. User submit registration information to the system.
3. User receives an error message that shows the user couldn't be logged in to system.

4.1.1.3. List the Tours**Use Case: List tours in Tour Reservation System**

Primary Actor: User

Stakeholders and interests:

- User who wants to list tours in system.

Successful Scenario Event Flow for List Tours by Name:

1. User enters the name of the tours to list.
2. User clicks to list tours button.
3. User can choose an order of list.

Successful Scenario Event Flow for List All Tours:

1. User clicks to list tours button without entering anything to search bar.
2. User can choose an order of list.

Unsuccessful Scenario Event Flow for List Tours by Name:

1. User clicks to list tours button without entering anything to search bar.
2. User sees an empty list with a message that indicate there is no such tour with that name

4.1.1.4. Filter the Tours

Use Case: Filter tours by tags in Tour Reservation System

Primary Actor: User

Stakeholders and interests:

- User who wants to filter tours in system.

Pre-conditions:

- User must be list tours before filtering it.

Successful Scenario Event Flow for List Tours by Name:

1. User choose filter options.
2. User clicks filter button to see filtered list.

4.1.1.5. View Details of Tour

Use Case: View Details of Tour in Tour Reservation System

Primary Actor: User

Stakeholders and interests:

- User who wants to view details of tours.

Pre-conditions:

- User must list the tours.

Successful Scenario Event Flow for View Details of Tour:

- User clicks the View Details button for a tour.

4.1.2 Customer Account

The main use cases for a customer account are given in this section.

4.1.2.1. Make a Reservation to a Tour

Use Case: Make Reservation in Tour Reservation System

Primary Actor: Customer

Stakeholders and interests:

- Customer who wants to make reservation to a tour in system.

Pre-conditions:

- Customer must be logged in system.
- Customer must be in a tour details screen.

Successful Scenario Event Flow for Make Reservation:

1. Customer selects make reservation for the tour.
2. Customer enters the information to make reservation if the information hasn't already saved for customer account.
3. Customer selects the dependents which will be in tour reservation with customer.
4. Customer makes reservation.
5. Customer receives an confirmation message for reservation.
6. Customer receives booking points for that reservation according to what customer pays.

Unsuccessful Scenario Event Flow for Make Reservation:

1. Customer selects make reservation for the tour.
2. Customer enters the information to make reservation if the information hasn't already saved for customer account.
3. Customer selects the dependents which will be in tour reservation with customer.
4. Customer makes reservation.
5. Customer receives an error message for reservation since the payment method is not valid.

4.1.2.2. Cancel a Reservation to a Tour

Use Case: Cancel Reservation in Tour Reservation System

Primary Actor: Customer

Stakeholders and interests:

- Customer who wants to cancel reservation of a tour in system.

Pre-conditions:

- Customer must be logged in system.
- Customer must be in tour detail of My Tours screen in customer account.

Successful Scenario Event Flow for Cancel Reservation:

1. Customer selects cancel reservation for the tour.
2. Customer confirms the reservation cancellation.
3. Customer gets money refund as booking points.

Unsuccessful Scenario Event Flow for Cancel Reservation:

1. Customer selects cancel reservation for the tour.
2. Customer cannot cancel the reservation since the cancellation deadline is passed.

4.1.2.3. Manage Account**Use Case: Manage Customer Account in Tour Reservation System**

Primary Actor: Customer

Stakeholders and interests:

- Customer who wants to manage customer account information.

Pre-conditions:

- Customer must be logged in system.

Successful Scenario Event Flow for Change Account Info:

1. Customer change account information in manage account screen.
2. Customer click change account information to update account info.
3. Customer receive a confirmation message which indicates the changed information

Unsuccessful Scenario Event Flow for Change Account Info:

1. Customer change account information in manage account screen.
2. Customer click change account information to update account info.
3. Customer receive an error message which indicates the information hadn't been changed

Successful Scenario Event Flow for Add Dependent to Account:

1. Customer clicks add dependent for customer account.
2. Customer enter the information for dependent.
3. Customer confirm the entered information and add it.

Successful Scenario Event Flow for Enter Promo Code:

1. Customer clicks enter the promo code.
2. Customer enters the promo code.
3. Customer sees the promo code in promotion table.

Unsuccessful Scenario Event Flow for Enter Promo Code:

1. Customer clicks enter the promo code.

2. Customer enters the promo code.
3. Customer receives an error message which indicates the promo code is not valid.

4.1.2.4. See Reserved Tours

Use Case: See Reserved Tours in Tour Reservation System

Primary Actor: Customer

Stakeholders and interests:

- Customer who wants to see reserved tours.

Pre-conditions:

- Customer must be logged in system.

Successful Scenario Event Flow for See Reserved Tours:

- Customer enters the My Account screen.
- Customer clicks Current Reserved Tours to see reserved tours.

4.1.2.5. See Past Reserved Tours

Use Case: See Past Reserved Tours in Tour Reservation System

Primary Actor: Customer

Stakeholders and interests:

- Customer who wants to see past reserved tours.

Pre-conditions:

- Customer must be logged in system.

Successful Scenario Event Flow for See Reserved Tours:

- Customer enters the My Account screen.
- Customer clicks Past Reserved Tours to see reserved tours.

4.1.2.6. See Canceled Tours

Use Case: See Canceled Tours in Tour Reservation System

Primary Actor: Customer

Stakeholders and interests:

- Customer who wants to see canceled tours which is canceled by customer.

Pre-conditions:

- Customer must be logged in system.

Successful Scenario Event Flow for See Reserved Tours:

- Customer enters the My Account screen.
- Customer clicks Canceled Tours to see reserved tours.

4.1.3. Staff Account

The main use cases for a staff member are given in this section.

4.1.3.1. Create a Tour**Use Case: Create a Tour in Tour Reservation System**

Primary Actor: Staff

Stakeholders and interests:

- Staff who wants to create a tour in system.

Pre-conditions:

- Staff must be logged in system.
- Staff must be in a Home screen.

Successful Scenario Event Flow for Create a Tour:

1. Staff clicks the Create a Tour button to create a tour.
2. Staff enters information about the tour such as accommodation informations, travel routes, trip events, descriptions of tour days, canceling deadline, quota, tour description, image for tour, tour name and price.
3. Staff clicks the confirm button to complete the creation of a tour.

4.1.3.2 Manage Staff Account**Use Case: Manage Staff Account in Tour Reservation System**

Primary Actor: Staff

Stakeholders and interests:

- Staff who wants to manage staff account information.

Pre-conditions:

- Staff must be logged in system.

Successful Scenario Event Flow for Change Account Info:

1. Staff change account information in manage account screen.
2. Staff click change account information to update account info.
3. Staff receive a confirmation message which indicates the changed information

Unsuccessful Scenario Event Flow for Change Account Info:

1. Staff change account information in manage account screen.
2. Staff click change account information to update account info.
3. Staff receive an error message which indicates the information hadn't been changed

4.1.3.2 See Created Tour**Use Case: See Created Tour in Tour Reservation System**

Primary Actor: Staff

Stakeholders and interests:

- Staff who wants to see created tour.

Pre-conditions:

- Staff must be logged in system.

Successful Scenario Event Flow for See Reserved Tours:

1. Staff enters the My Account screen.
2. Staff clicks Created Tours to see created tour.

4.1.3.3 Manage Tour**Use Case: List All Reservation in Tour Reservation System**

Primary Actor: Staff

Stakeholders and interests:

- Staff who wants to list all reservation in a tour.

Pre-conditions:

- Staff must be logged in system.
- Staff must be in View Details screen of a tour.

Successful Scenario Event Flow for List All Reservation:

1. Staff click List All Reservation button to see all reservations in View Details scene of a tour.

Successful Scenario Event Flow for Cancel Reservation:

1. Staff selects cancel reservation for the tour.
2. Staff confirms the reservation cancellation.

4.2. Algorithms

4.2.1. Tour Filtering & Sorting Algorithm

The users of the system should be able to view the tours that fits their interest. Thus, we need a tour filtering algorithm based on the user input. First we extract the tours that contains the source and destination locations which falls in the selected date and time interval. Then we apply the auxiliary filters such as food type, hotel rating etc. After the filters are applied, the resulting tours are listed to the user. Also, the user can choose the sorting parameter of the tours (date, price etc.).

4.2.2. Reservation Algorithm

The reservation algorithm first checks whether the selected tour has enough quota for the current reservation instance. If there are enough quotas for reservation to be made, the number of quotas in the tour will be decreased by the number of participants in the reservation. Then the reservation is registered to the user and will stay registered to that user until the reservation is canceled or the tour has been passed. If the cancel deadline is passed for the particular tour, the payment must be made during the reservation. If the cancel deadline has not been reached yet, the payment can be made after the reservation.

4.2.3. Reservation Cancelling Algorithm

Users are able to cancel their reservations until the specified cancel deadline for each tour. If the purchase has not been made, the algorithm automatically cancels the reservation after the cancellation date. One day before this automatic cancellation, user gets a notification about his/her payment deadline. If the purchase has been made and user wants to cancel the reservation, the algorithm checks whether the cancellation date has been passed or not. If it has not been passed yet, the system starts the refund process.

If a cancelled reservation is reserved again, it should not appear as a cancelled reservation in the customer's dashboard. For this reason, the old reservation should be deleted from the cancelled reservations. (See Section 6.3.2. for the implementation with a trigger.)

4.2.4. Earning & Spending Booking Points

Users earn booking points for each purchase, there is a constant ratio between the purchase amount and the number of booking points received. To calculate the booking points, 1% of the purchase amount is feed in to the floor function and the resulting number is the booking point received for that purchase. For example if the purchase amount is 266.84 TRY the user will gain 2 booking points ($\text{floor}(2.66) = 2$). Each booking point corresponds to 1 TRY.

4.2.5. Using Promotion Cards

The customer can enter a promotion code to add a promotion card to their account. Each promotion card applies a discount percentage on particular kinds of tours and can be used in at most one tour payment. Each promotion card can be entered by a specific number of customers, that is called quota of the promotion card. After the quota is reached, this promotion code will no longer be valid.

The discount percentage is applied to the reduced price by the booking points. For example, if the user has 5 booking points and a promotion card of 10% discount (and chooses to use them both), a price of 105 TRY will be reduced to 90 TRY.

4.3. Data Structures

In our relation schemas we have made use of the types Integer, Numeric, Varchar and Date.

- "Integer" types are used for numbers that does not have part after decimal point like booking points or tour quota.
- "Numeric" types are used for storing values with decimal points like prices and discount percent. It is also used for numeric values with fixed length like telephone no.
- "Varchar" types are used for storing strings like name of people or tours.
- "Date" types are used for representing specific dates like arrival, departure or cancelling deadline.

Other than these, in the front-end side of the application, the tours in the page will be represented as a list. They will be retrieved from the database as list and they will be rendered as HTML.

5. User Interface Design and SQL Statements

5.1. General Pages

5.1.1. Home Page (without login)

A Web Page

https://ibitur.firebaseio.com

Home | [Tours](#) [Login](#)

Q Where do you want to travel?

IBITUR - Tour Reservation System

Username

e-mail

password

Contact Information: ibitur@gmail.com All Rights Reserved IBITUR 2018.




  

Figure 2: Home Page

SQL queries for registering an account:

(after the e-mail verification process)

```
INSERT INTO Account(username, email, passwd) VALUES('johndoe',  
'john@doe.com', SHA2('johndoe123', 256));  
INSERT INTO CustomerAccount() VALUES ();  
--- Inserted to CustomerAccount with same auto incremented account ID in  
Account schema, and all profile attributes are set to NULL.
```

5.1.2. Tour Listing Page

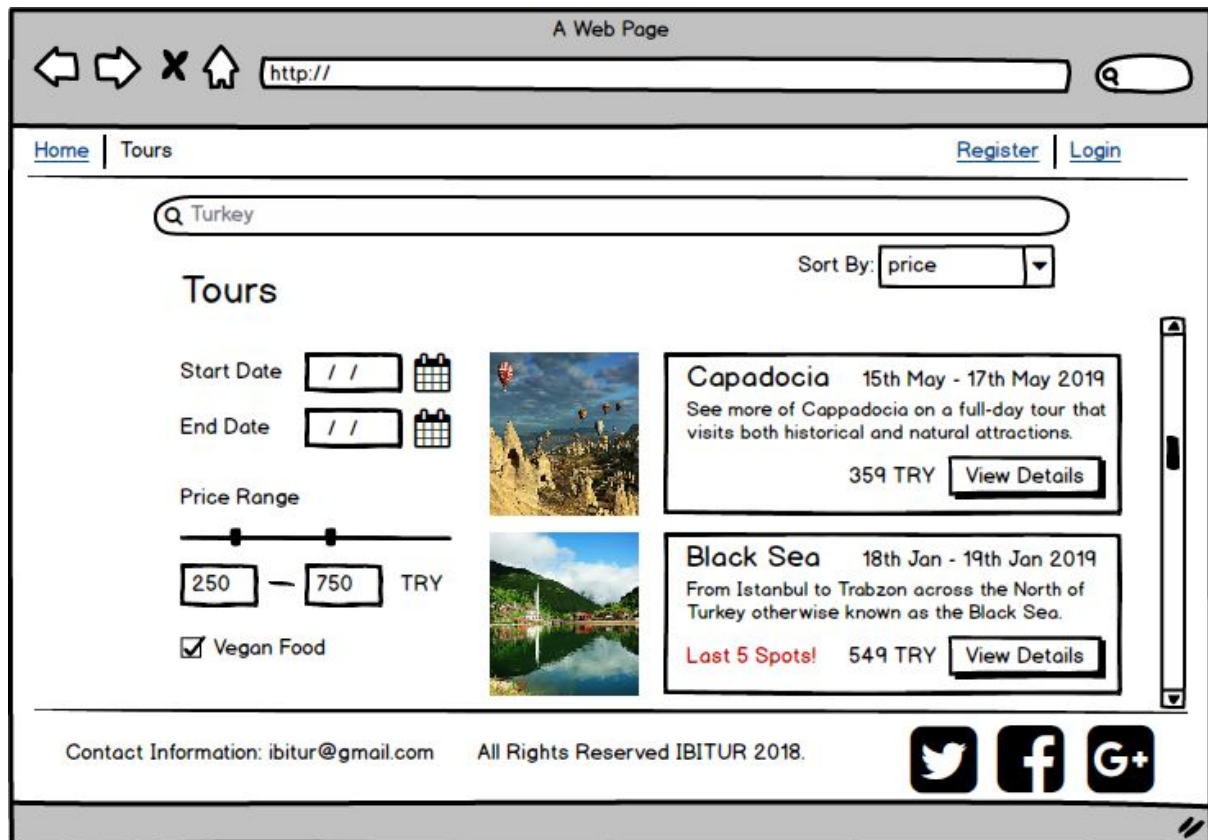


Figure 3: Tour Listing

SQL query to get the list without filters:

```
SELECT * FROM TourPreview;
```

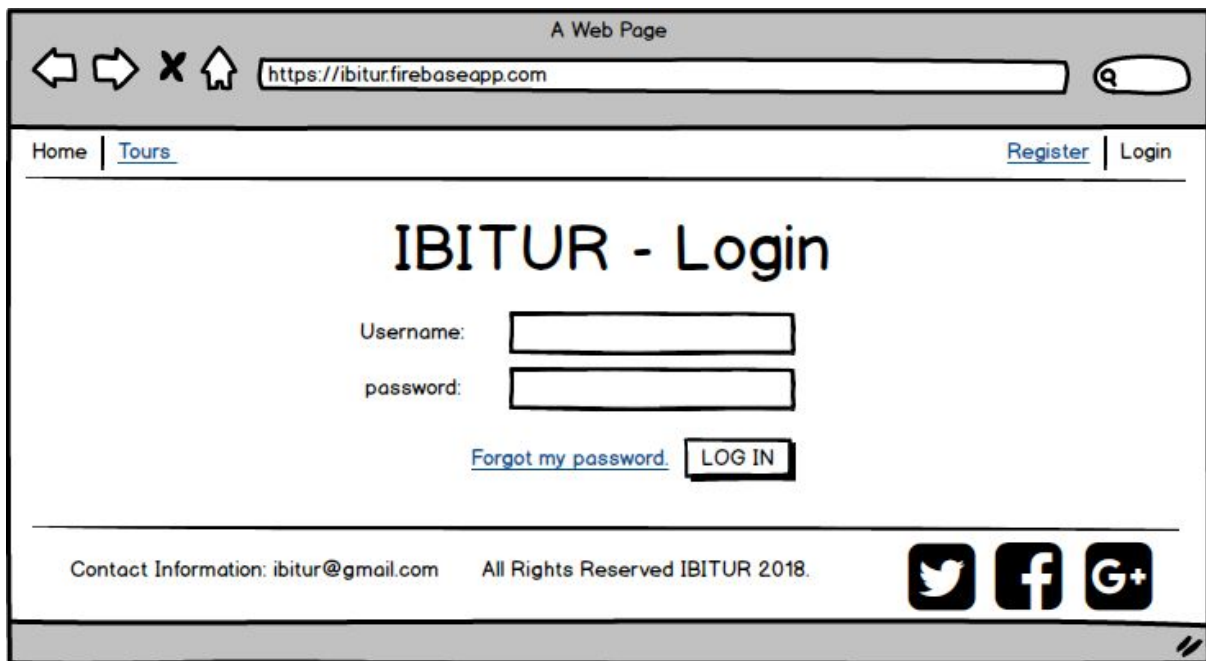
SQL query to get the list with sample filters:

```
SELECT * FROM TourPreview
WHERE price < 700 AND tour_ID IN (
  SELECT tour_ID FROM TourTags NATURAL JOIN Tag
  WHERE name = 'Vegan Food')
ORDER BY price;
```

SQL query to get the list with a sample search:

```
SELECT * FROM (TourPreview
  NATURAL JOIN (EXEC SearchForQuery 'Turkey'));
```

5.1.3. Login Page



A screenshot of a web browser displaying the IBITUR Login page. The browser's address bar shows the URL `https://ibitur.firebaseio.com`. The page has a navigation bar with links for [Home](#), [Tours](#), [Register](#), and [Login](#). The main heading is "IBITUR - Login". Below this, there are input fields for "Username:" and "password:". A link for [Forgot my password.](#) and a "LOG IN" button are positioned below the password field. The footer contains contact information: "Contact Information: ibitur@gmail.com", a copyright notice "All Rights Reserved IBITUR 2018.", and social media icons for Twitter, Facebook, and Google+. The browser window title is "A Web Page".

Figure 4: Login Page

SQL query to check the username and password match:

```
SELECT username, passwd FROM Account
```

```
WHERE username = 'johndoe' AND passwd = SHA2('johndoe123', 256);
```

5.2. Customer Pages

5.2.1. Home Page (as customer)

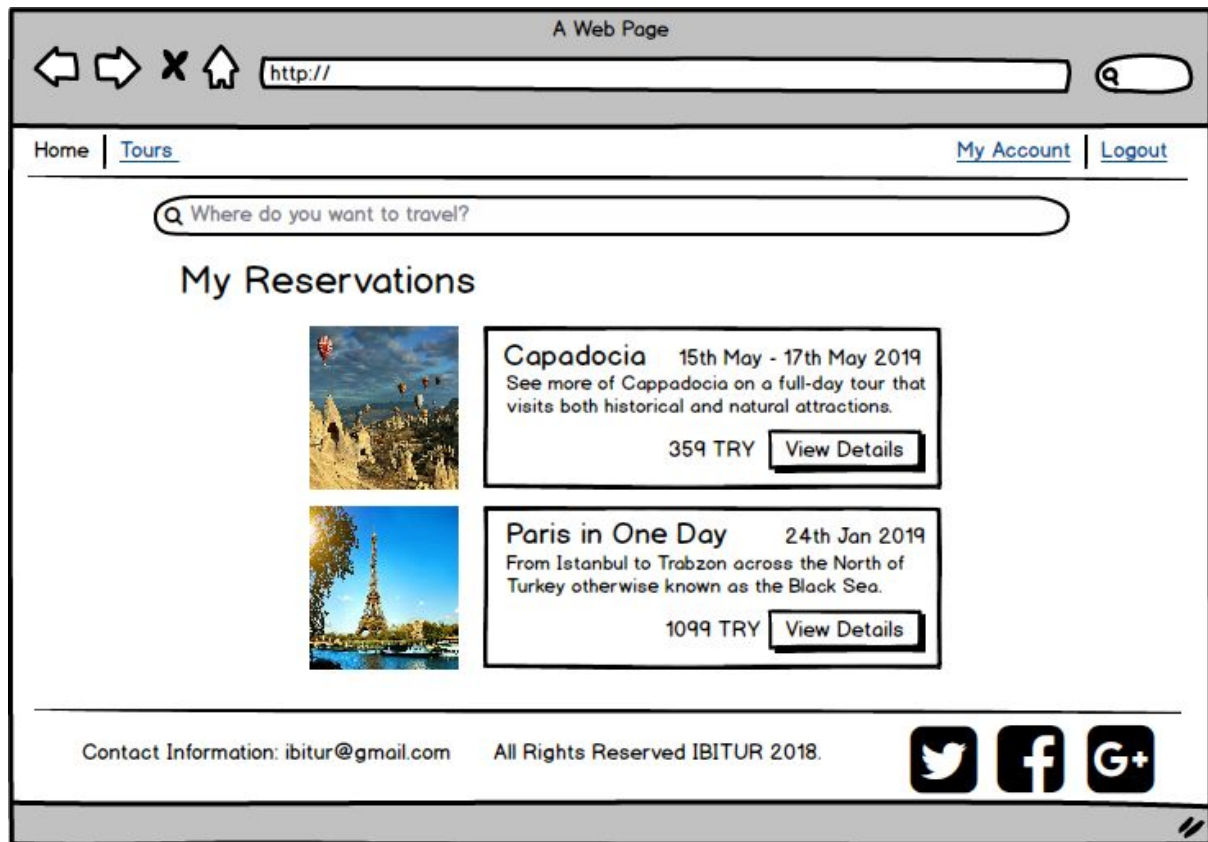


Figure 5: Customer Home Page

SQL query to get the list of reservations of the customer:

```
SELECT name, description, image_path, price, start_date, end_date
FROM (Reservation, TourPreview)
WHERE Reservation.customer_ID = '1234567890'
AND Reservation.tour_ID = TourPreview.tour_ID;
```

NOTE: TourPreview is a view that is defined in Section 6.2.1.

5.2.2. Tour Details Page (as customer)

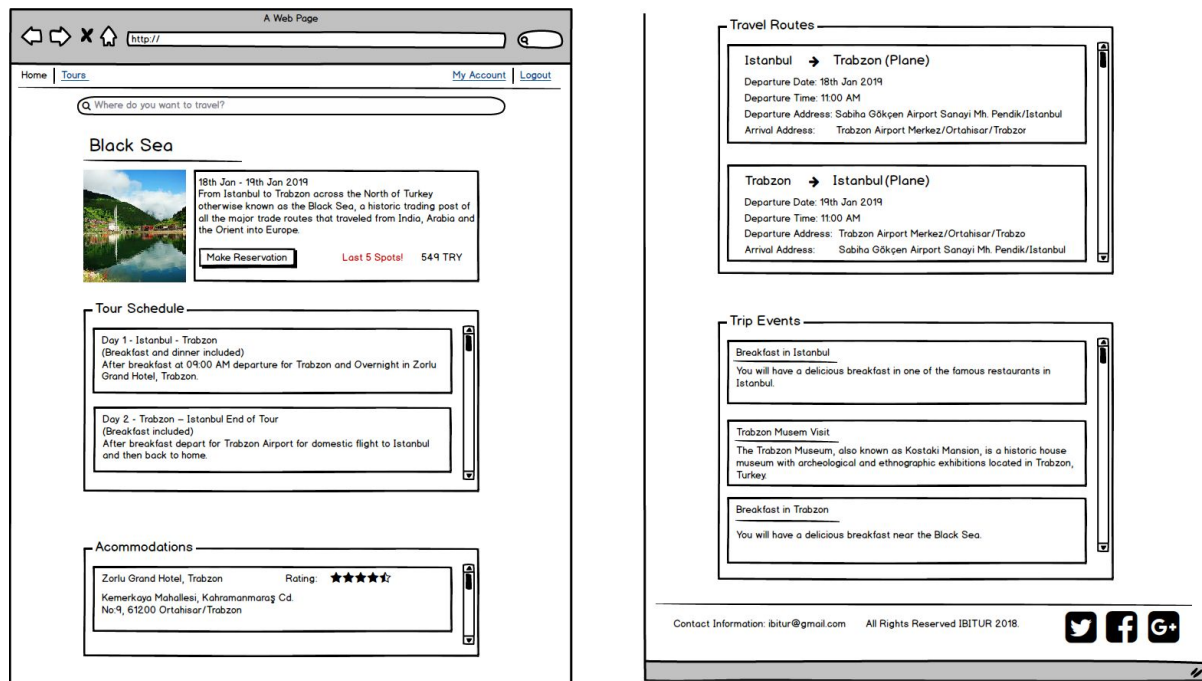


Figure 6: Tour Details Page

NOTE: If the customer has a reservation for the tour, then the “Make Reservation” button will be replaced by “Cancel Reservation”. If the payment is not yet been made, a “Make Payment” button will appear next to it.

SQL query to get the summary of the tour:

```
SELECT * FROM TourPreview WHERE tour_ID = '1234567890';
```

SQL query to get the daily schedule of the tour:

```
SELECT day_no, day_date, description
FROM Tour T, TourDay D
WHERE T.ID = D.tour_ID
ORDER BY day_date;
```

SQL query to get the accommodations in the tour:

```
SELECT ap.name, ap.address, a.enter_date, a.exit_date, ca.room_no
FROM Accommodation a,
AccommodationPlace ap,
CustomerAccommodates ca
WHERE a.tour_ID = '1234567890'
AND a.place_ID = ap.ID
AND a.ID = ca.accommodation_ID
AND ca.customer_ID = '7890456123';
```

SQL query to get the travel routes in the tour:

```
SELECT from_c.name AS from_city_name, to_c.name AS to_city_name,  
       dept_time, dept_address, arriv_address, arriv_time, pnr_no  
FROM TravelRoute tr, City from_c, City to_c, CustomerTravels ct  
WHERE tr.tour_ID = '1234567890'  
      AND from_c.ID = tr.from_city_ID  
      AND to_c.ID = tr.to_city_ID  
      AND tr.ID = ct.travel_route_ID;
```

SQL query to get the trip events in the tour:

```
SELECT name, description, City.name AS city_name, trip_date  
FROM TripEvent, City  
WHERE TripEvent.tour_ID = '1234567890' AND City.ID =  
TripEvent.city_ID;
```

SQL query to cancel the reservation:

```
INSERT INTO ReservationCancel VALUES('0123456789', GETDATE());
```

5.2.3. Tour Reservation Page


A Web Page

http://

Home | Tours


My Account | Logout


Tour Reservation




Black Sea 18th Jan - 19th Jan 2019
From Istanbul to Trabzon across the North of Turkey otherwise known as the Black Sea, a historic trading post of all the major trade routes that traveled from India, Arabia and the Orient into Europe.
Last 5 Spots! 549 TRY

First Name: Last Name:

National ID: Date of Birth: 

Phone Number: Gender: 

Dependents: 




Contact Information: ibitur@gmail.com All Rights Reserved IBITUR 2018.   

Figure 7: Tour Reservation Page

SQL queries to make reservation to a tour:

```
INSERT INTO Reservation VALUES('1234567890', '1234567890',  
    '1234567890', GETDATE(), 'UNPAID');  
INSERT INTO IncludedDependents VALUES('123456789', '123456789',  
    '123456789');  
INSERT INTO IncludedDependents VALUES('987654321', '987654321',  
    '987654321');
```



5.2.4. Payment Page

A Web Page

http://

Home | Tours | My Account | Logout

Payment



Black Sea 18th Jan - 19th Jan 2019
From Istanbul to Trabzon across the North of Turkey otherwise known as the Black Sea, a historic trading post of all the major trade routes that traveled from India, Arabia and the Orient into Europe.
Last 5 Spots! 549 TRY

☒ Use booking points: 6 Points
Booking Points to be received: 5 Points

Payment Method: Saved Credit Cards: Promotion Card:

New Credit Card: Final Amount: 515.85 TRY

Card Holder Name:

Credit Card Number:

Expiry Date:

CVV:

☐ Save this card

Contact Information: ibitur@gmail.com All Rights Reserved IBITUR 2018.




  

Figure 8: Reservation Payment Page

SQL query to get saved payment method for customer:

```
SELECT P.description
```

```
FROM PaymentMethod p, CustomerPaymentMethods c
```

```
WHERE p.ID = c.ID AND c.customer_ID ='1234567890';
```

SQL queries to save the new payment method:

```
INSERT INTO PaymentMethod(ID, card_no, description, card_holder_name,  
cvv, expire_month, expire_year)
```

```
VALUES( '1324567890', '1234567812345678', 'INGBANK', 'John Kennedy',  
'436', '11', '2022');
```

```
INSERT INTO CustomerPaymentMethods(customer_ID,
    payment_method_ID)
VALUES('1234567890', '1324567890');
```

SQL query to get promotion cards of customer:

```
SELECT discount_percent
FROM CustomerPromotionCards NATURAL JOIN PromotionCard
WHERE CustomerPromotionCards.customer_ID = '1234567890';
```

SQL query to get booking points of customer:

```
SELECT booking_point
FROM CustomerAccount
WHERE CustomerAccount.ID = '1234567890';
```

SQL query to get the summary of the tour:

Same with “Get the summary” query in Section 5.2.2.

5.2.5. My Account Page

The screenshot displays the 'My Account Page' for a user named John Doe. The page is divided into several sections:

- My Account - John Doe**: Includes links for Home, Tours, My Account, and Logout.
- Reservations**:
 - Current Reservations**: A dropdown menu.
 - Cancelled Reservations**: A dropdown menu.
 - Black Sea**: A reservation for 18th Jan - 19th Jan 2019, from Istanbul to Trabzon across the North of Turkey, priced at 549 TRY. A 'View Details' button is present.
 - Past Reservations**: A dropdown menu.
- Profile Settings**:
 - First Name: John, Last Name: Doe
 - Nationality: (empty), Date of Birth: / /
 - National ID: (empty), Gender: Male
 - Phone Number: (empty), Save Changes button
- Dependent Travelers**:
 - Dependents: Jane Doe, Joe Doe (selected), Remove Dependent button
 - Add New Dependant**:
 - First Name: (empty), Last Name: (empty)
 - National ID: (empty), Date of Birth: / /
 - Gender: Female, Add Dependant button
- Payment Details**:
 - Saved Credit Cards**: Yapi Kredi World (Default), Garanti Bonus, Set as Default button
 - Add New Credit Card**:
 - Card Holder Name: John Doe
 - Credit Card Number: 1234-567
 - Expiry Date: 1234-567
 - CVV: 1234-567, Add Credit Card button
 - Booking Points**: Currently you have 6 booking points.
- Promotion Cards**:
 - Current Promotion Cards**:
 - 5% Discount on all tours in Turkey
 - 10% Discount on all tours
 - 25% Discount on all tours that costs more than 500 TRY
 - OWCDF2019, Redeem Promotion Card button
- Account Details**:
 - Username: jdoe
 - e-mail: jdoe@email.com, Change e-mail button, Change password button
- Footer**: Contact Information: ibitur@gmail.com, All Rights Reserved IBITUR 2018, social media icons (Twitter, Facebook, Google+).

Figure 9: My Account Page

SQL query to display the current reservations:

```
SELECT name, description, image_path, price,
```

```
    start_date, end_date, remaining_quota
FROM (TourPreview NATURAL JOIN Reservation)
WHERE Reservation.customer_ID = '1234567890'
    AND Reservation.ID NOT IN (SELECT reservation_ID FROM
ReservationCancel)
    AND start_date > GETDATE();
```

SQL query to display the past reservations:

```
SELECT name, description, image_path, price,
    start_date, end_date, remaining_quota
FROM (TourPreview NATURAL JOIN Reservation)
WHERE Reservation.customer_ID = '1234567890'
    AND Reservation.ID NOT IN (SELECT reservation_ID FROM
ReservationCancel)
    AND start_date <= GETDATE();
```

SQL query to display the cancelled reservations:

```
SELECT name, description, image_path, price,
    start_date, end_date, remaining_quota
FROM (TourPreview NATURAL JOIN Reservation), ReservationCancel
WHERE Reservation.customer_ID = '1234567890'
    AND Reservation.ID = ReservationCancel.reservation_ID;
```

SQL to update profile settings:

```
UPDATE CustomerAccount
SET first_name = 'John', middle_name = 'Frederick', last_name = 'Kennedy',
    nationality = 'American', national_ID = '1111111115',
    gender= 'Male', date_of_birth = '19670618'
WHERE CustomerAccount.ID = '1234567890';
```

SQL query to display the dependent travelers:

```
SELECT first_name, middle_name, last_name
FROM Dependent
WHERE Dependent.customer_ID = '1234567890';
```

SQL query to remove the dependent travelers:

```
DELETE
FROM Dependent
WHERE Dependent.customer_ID = '1234567890'
    AND national_ID = '1111111110';
```

SQL query to add new dependent:

```
INSERT INTO Dependent(customer_ID, national_ID, gender, date_of_birth,  
first_name, last_name)  
VALUES( '1234567890' , '1111111110', 'Male', '20120618', 'John', 'Doe' );
```

SQL query to display the credit cards:

```
SELECT P.description  
FROM PaymentMethod P, CustomerPaymentMethods C  
WHERE P.ID = C.ID AND C.customer_ID = '1234567890';
```

SQL query to add new credit card:

Same with “Add new credit card” query in Section 5.2.4.

SQL query to display the booking points:

Same with “Add new credit card” query in Section 5.2.4.

SQL query to display the Promotion Cards:

Same with “Add new credit card” query in Section 5.2.4.

SQL query to add new Promotion Card:

```
INSERT INTO CustomerPromotionCards(customer_ID, promo_code)  
VALUES ( '1234567890', 'A1B2C3D4E5');
```

SQL query to change e-mail:

```
UPDATE Account  
SET Account.email = 'joe@dhon.com'  
WHERE Account.ID = '1234567890';
```

SQL query to change password:

```
UPDATE Account  
SET Account.passwd = SHA('joedhon123')  
WHERE Account.ID = '1234567890';
```

5.3. Staff Pages

5.3.1. Home Page (as staff)

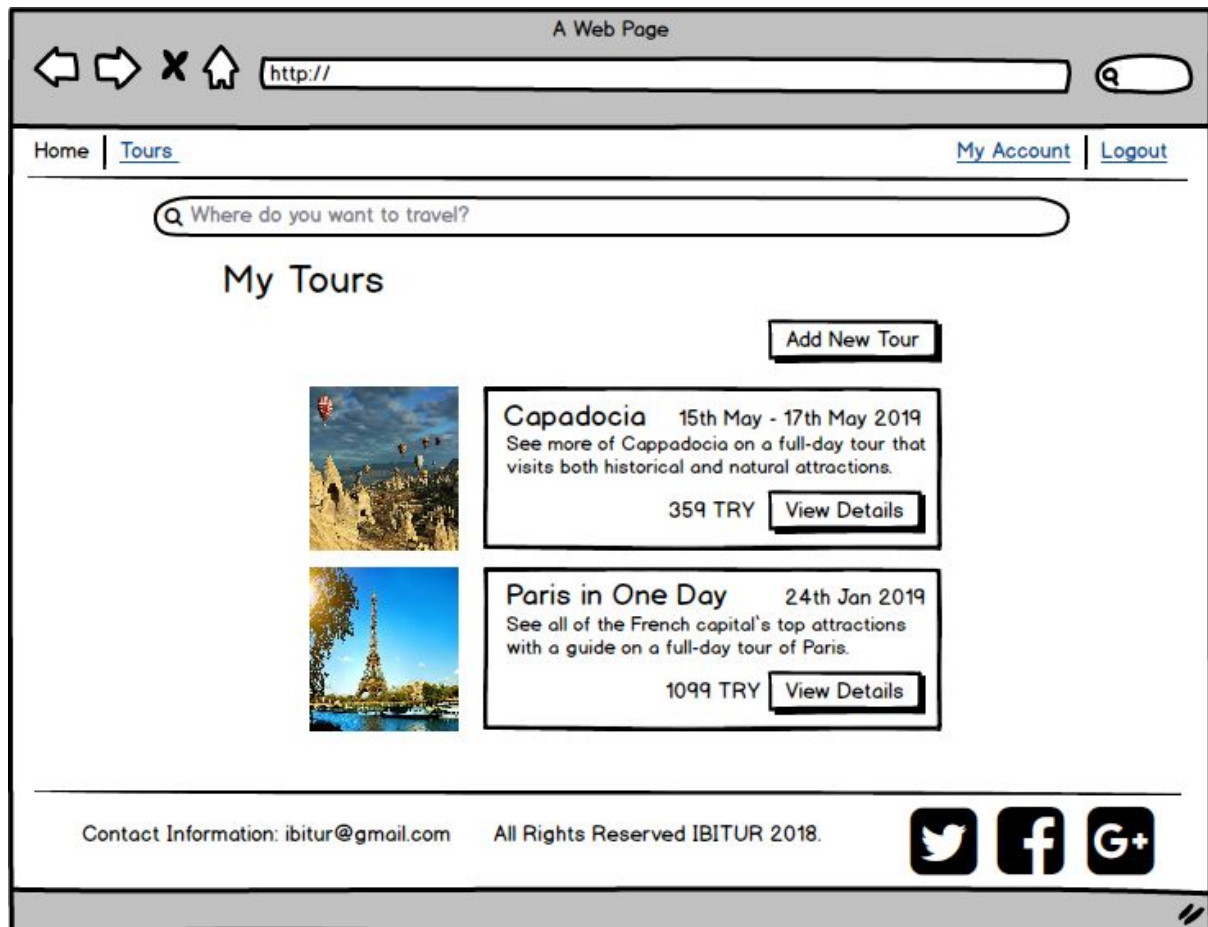


Figure 10: Home Page for Staff

SQL query to get the list of tours added by the staff:

```
SELECT name, description, image_path, price,  
       start_date, end_date, remaining_quota  
FROM TourPreview NATURAL JOIN Tour  
WHERE Tour.creator_ID = '1234512345';
```

5.3.2. Tour Adding Page

The figure displays two side-by-side screenshots of a web application interface for adding tours. The left screenshot shows the 'Add New Tour' form, which includes fields for Title, Description, Tour Image, Price, Number of Spots, and Cancel Deadline. It also features a 'Tour Schedule' section with a table for adding days and an 'Accommodations' section for adding hotels. The right screenshot shows the 'Travel Routes' and 'Trip Events' sections. 'Travel Routes' includes fields for Source City, Destination City, Departure Date, Departure Time, Vehicle Type, and Travel Company. 'Trip Events' includes fields for Event Title, Description, and Date. Both sections have 'Add New' buttons. The bottom of the page shows contact information and social media links.

Figure 11: Tour Adding Page

SQL query to add general tour details:

```
INSERT INTO TOUR(ID, name, description, image_path, quota, price,
creator_ID, cancelling_deadline)
VALUES ( '1234567890', 'Trabzon Tour', 'Beauty of Trabzon city and amazing
trip events in one day Trabzon tour.', '/home/blacksea.png', '60', '549',
'1234512345','20190617');
```

SQL query to add a day to tour schedule:

```
INSERT INTO TourDay( tour_ID, day_no, day_date, description)
VALUES ( '1234567890', '1', '20190618', 'After breakfast at 09:00 AM
departure for Trabzon and Overnight in Zorlu Grand Hotel, Trabzon.');
```

SQL query to add an accommodation:

```
INSERT INTO Accommodation(ID, tour_ID, place_ID, enter_date, exit_date)
VALUES( '123456987', '1234567890', '789456123',
'20160618 10:30:00 AM',
'20160620 10:30:00 PM');
```

SQL query to add a travel route:

```
INSERT INTO TravelRoute(ID, tour_ID, from_city_ID, to_city_ID,  
company_ID, dept_address, dept_time, arriv_address, arriv_time)  
VALUES ('123456987', '1234567890', 'Istanbul', 'Trabzon', '123456987',  
      'Istanbul Sabiha Gökçen Airport Istanbul', '20160620 10:30:00 PM',  
      'Trabzon Airport Merkez/ Ortahisar/ Trabzon', '20160620 12:30:00 PM');
```

SQL query to add a trip event:

```
INSERT INTO TripEvent(ID, tour_ID, city_ID, name, description, trip_date)  
VALUES('123455768', '1456782390', '1234568790',  
      'Trabzon Museum Visit',  
      'The Trabzon Museum, also known as Kostaki Mansion, is a historic house  
museum with archeological and ethnographic exhibitions located in Trabzon,  
Turkey.', '20160620 12:30:00 PM');
```

5.3.3. Tour Details Page (as staff)

A Web Page

X

http://

Q

Home


Tours

My Account

Logout

Tour Details

Black Sea



18th Jan - 19th Jan 2019
From Istanbul to Trabzon across the North of Turkey otherwise known as the Black Sea, a historic trading post of all the major trade routes that traveled from India, Arabia and the Orient into Europe.

549 TRY

Reservations:

Name	Paid	No. of Travelers	Details
John Doe	<input checked="" type="checkbox"/>	3	View
Halil Şahiner	<input checked="" type="checkbox"/>	1	View
Mahmud Sami Aydın	<input type="checkbox"/>	1	View
Emin Bahadır Tülüçe	<input checked="" type="checkbox"/>	1	View
Abdullah Talayhan	<input type="checkbox"/>	1	View

Cancel Tour

Reason for Cancellation:

Bad Weather Conditions.

Cancel Tour

Contact Information: ibitur@gmail.com

All Rights Reserved IBITUR 2018.






Figure 12: Tour Details Page

SQL query to get the tour details:

Listing the tour summary use the same query that is given in Section 5.2.2.

SQL queries to list travelers who have reserved this tour:

```
CREATE VIEW DependetCounts AS (  
    SELECT COUNT(dependent_ID) AS dependent_count,  
           reservation_ID, customer_ID  
    FROM IncludedDependents  
    GROUP BY (reservation_ID, customer_ID)  
);  
SELECT first_name, middle_name, last_name,  
       payment_status, dependent_count  
FROM Reservation, CustomerAccount, DependentCounts  
WHERE Reservation.tour_ID = '1234567890'  
       AND Reservation.customer_ID = CustomerAccount.ID  
       AND DependentCounts.customer_ID = CustomerAccount.ID  
       AND DependentCounts.tour_ID = Reservation.tour_ID;
```

SQL query to cancel the tour:

```
INSERT INTO TourCancel(tour_ID, cancel_date, cancel_reason)  
VALUES( '1234567890', GETDATE(),  
       'Because of the weather conditions in Trabzon, the tour is cancelled.');
```

5.3.4. Customer Profile View (as staff)

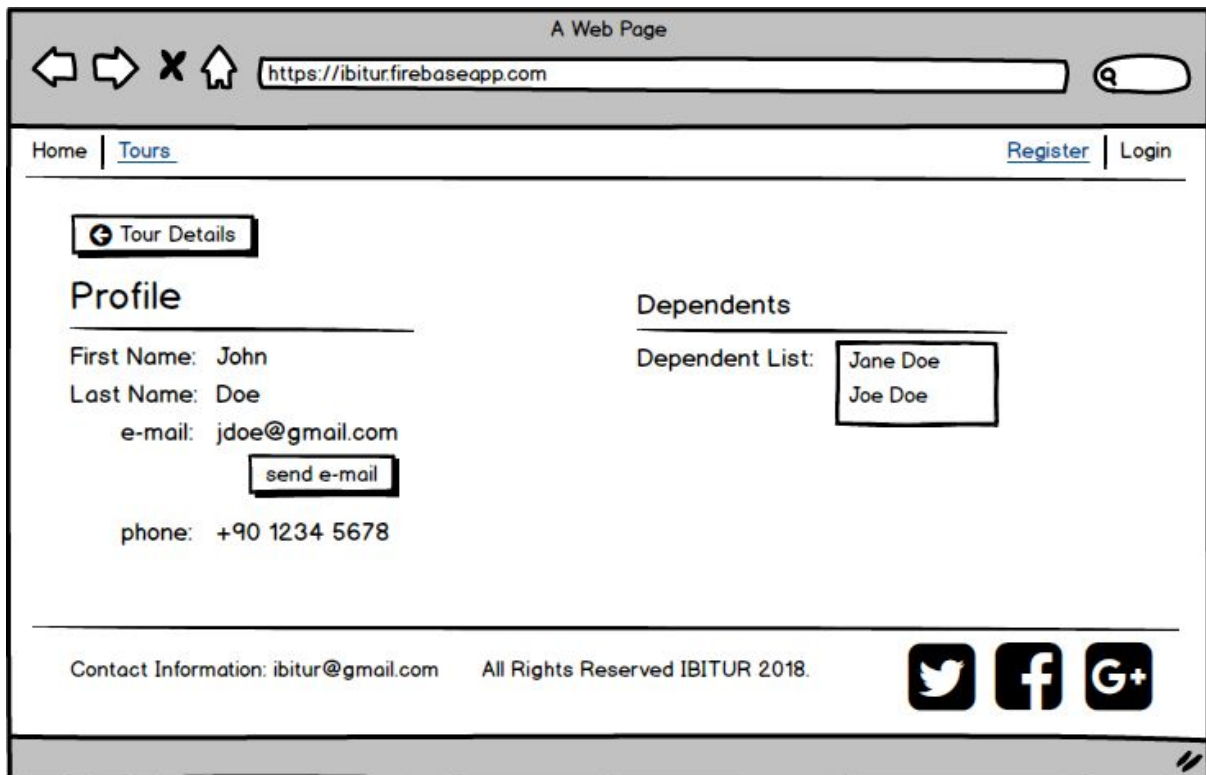


Figure 13: Customer View Profile Page for Staff

SQL query to display necessary customer information:

```
SELECT first_name, middle_name, last_name, email, telephone_no
FROM Tour NATURAL JOIN CustomerAccount NATURAL JOIN
CustomerTelephones
WHERE Tour.ID = '1234567890';
```

SQL query to display the dependents of the customer:

```
SELECT first_name, middle_name, last_name, email, telephone_no
FROM Tour NATURAL JOIN CustomerAccount NATURAL JOIN
CustomerTelephones
WHERE Tour.ID = '1234567890';
```

6. Advanced Database Components

6.1. Reports

6.1.1. Most visited cities of the month

This report will generate a table of cities and their total reservation number, made in a month. If a tour has more than one city in it, then both of the cities will be counted as a reservation made to them.

```
CREATE VIEW CityPopularity AS (  
    SELECT city_name, SUM(resv_no) AS popularity  
    FROM TourAssociations NATURAL JOIN ReservationCount  
    GROUP BY city_name  
    HAVING (tour_ID IN  
        (SELECT tour_ID FROM TourPreview  
         WHERE '2018-07-01' <= start_date AND start_date <= '2018-08-01'))  
    ORDER BY popularity DESC  
);
```

NOTE: TourPreview is a view that is defined in Section 6.2.1.

NOTE: ReservationCount is a view that is defined in Section 6.2.1.

NOTE: TourAssociations is a view that is defined in Section 6.2.2.

6.1.2. Top revenue-making countries of the year

This report will generate a table of countries and the total amount of spending that is done on that tours that include that country.

```
CREATE VIEW CountryRevenues AS (  
    SELECT country_name, SUM(price) AS revenue  
    FROM TourAssociations NATURAL JOIN TourPreview  
    GROUP BY country_name  
    HAVING (tour_ID IN  
        (SELECT tour_ID FROM TourPreview  
         WHERE '2018-01-01' <= start_date AND start_date <= '2018-12-31'))  
    ORDER BY revenue DESC  
);
```

NOTE: TourPreview is a view that is defined in Section 6.2.1.

NOTE: TourAssociations is a view that is defined in Section 6.2.2.

6.2. Views

6.2.1. Tour Preview

A tour has many components in it (accommodations, travel routes, trip events, day descriptions). It's not practical to list all of these components in a short preview of the tour, thus we decided to create a summary view of the tour, including its basic attributes like name, description, image, price and the following extra attributes: We introduced a remaining_quota of the tour, which will be calculated with the count of the reservation. We also introduced start_date and end_date of the tour to be calculated from the components in the tour. The earliest date among the components of the tour will be start_date where the latest date will be end_date of this view.

NOTE: The TourPreview view is created at the last SQL statement, where the first ones are auxillary views.

```
--- *** Tables Used ***
```

```
--- Tour(ID, name, description, image_path, quota, price, creator_ID,  
cancelling_deadline)  
--- Reservation(ID, customer_ID, tour_ID, issue_date, payment_status)  
--- Accommodation(ID, tour_ID, place_ID, enter_date, exit_date)  
--- TravelRoute(ID, tour_ID, from_city_ID, to_city_ID, company_ID, dept_address,  
dept_time, arriv_address, arriv_time)  
--- TripEvent(ID, tour_ID, city_ID, name, description, trip_date)
```

```
CREATE VIEW NonCancelledReservation AS  
(SELECT * FROM Reservation WHERE  
cutomer_ID NOT IN (SELECT customer_ID FROM ReservationCancels));
```

```
CREATE VIEW ReservationCount AS  
(SELECT tour_ID, COUNT(customer_ID) AS resv_no  
FROM NonCancelledReservation GROUP BY tour_ID);
```

```
CREATE VIEW TourSubDates1 AS  
(SELECT tour_ID, arriv_date AS the_date FROM Tour, Accommodation  
WHERE Tour.ID = Accommodation.tour_ID);
```

```
CREATE VIEW TourSubDates2 AS  
(SELECT tour_ID, enter_date AS the_date FROM Tour, Accommodation  
WHERE Tour.ID = Accommodation.tour_ID);
```

```
CREATE VIEW TourSubDates3 AS
(SELECT tour_ID, arriv_date AS the_date FROM Tour, TravelRoute
WHERE Tour.ID = TravelRoute.tour_ID);
```

```
CREATE VIEW TourSubDates4 AS
(SELECT tour_ID, dept_date AS the_date FROM Tour, TravelRoute
WHERE Tour.ID = TravelRoute.tour_ID);
```

```
CREATE VIEW TourSubDates5 AS
(SELECT tour_ID, trip_date AS the_date FROM Tour, TripEvent
WHERE Tour.ID = TripEvent.tour_ID);
```

```
CREATE VIEW TourDates AS
(SELECT DISTINCT * FROM TourSubDates1)
UNION (SELECT DISTINCT * FROM TourSubDates2)
UNION (SELECT DISTINCT * FROM TourSubDates3)
UNION (SELECT DISTINCT * FROM TourSubDates4)
UNION (SELECT DISTINCT * FROM TourSubDates5);
```

```
CREATE VIEW TourInterval AS
(SELECT MIN(date) AS start_date, MAX(date) AS end_date FROM TourDates
GROUP BY tour_ID);
```

```
CREATE VIEW TourPreview AS
(SELECT Tour.ID AS tour_ID, name, description, image_path, price, start_date,
end_date,
(quota - resv_no) AS remaining_quota
FROM Tour, ReservationCount, TourInterval WHERE Tour.ID =
ReservationCount.tour_ID AND Tour.ID = TourInterval.tour_ID);
```

6.2.2. Tour Associations

The tours are associated with cities in an indirect way (their accommodation places, trip events or travel routes have cities but not themselves.). Thus, we decided to create a view that lists all the cities and countries associated with them.

NOTE: The TourAssociations view is created at the last SQL statement, where the first ones are auxillary views.

--- *** Tables Used ***

--- Tour(ID, name, description, image_path, quota, price, creator_ID, cancelling_deadline)

```

--- City(ID, country_ID, name)
--- Country(ID, name)
--- TripEvent(ID, tour_ID, city_ID, name, description, trip_date)
--- Accommodation(ID, tour_ID, place_ID, enter_date, arriv_date)
--- AccommodationPlace(ID, city_ID, name, address)
--- TravelRoute(ID, tour_ID, from_city_ID, to_city_ID, company_ID, dept_address,
dept_time, arriv_address, arriv_time)

```

```

CREATE VIEW TripAssociations AS
(SELECT
    Tour.ID AS tour_ID,
    Tour.name AS tour_name,
    City.name AS city_name,
    Country.name AS country_name
FROM (Tour, TripEvent, City, Country)
WHERE (Tour.ID = TripEvent.tour_ID
    AND TripEvent.city_ID = City.ID
    AND City.country_ID = Country.ID));

```

```

CREATE VIEW AccommodationAssociations AS
(SELECT
    Tour.ID AS tour_ID,
    Tour.name AS tour_name,
    City.name AS city_name,
    Country.name AS country_name
FROM (Tour, Accommodation, AccommodationPlace, City, Country)
WHERE (Tour.ID = Accommodation.tour_ID
    AND Accommodation.place_ID = AccommodationPlace.ID
    AND AccommodationPlace.city_ID = City.ID
    AND City.country_ID = Country.ID));

```

```

CREATE VIEW TravelAssociations AS
(SELECT
    Tour.ID AS tour_ID,
    Tour.name AS tour_name,
    City.name AS city_name,
    Country.name AS country_name
FROM (Tour, TravelRoute, City, Country)
WHERE (Tour.ID = TravelRoute.tour_ID
    AND (TravelRoute.from_city_ID = City.ID OR TravelRoute.to_city_ID = City.ID)
    AND City.country_ID = Country.ID));

```

```
CREATE VIEW TourAssociatons AS
(SELECT DISTINCT * FROM TripAssociations)
UNION (SELECT DISTINCT * FROM AccommodationAssociations)
UNION (SELECT DISTINCT * FROM TravelAssociations);
```

6.3. Triggers

6.3.1. Quota Backup Trigger

When a customer cancels their reservation that is done on a full-quote tour, a quota becomes free. In this case, the first backup-reservation should become a real reservation.

```
--- *** Tables Used ***
```

```
--- ReservationCancel(reservation_ID, cancel_date)
--- Reservation(ID, customer_ID, tour_ID, issue_date, payment_status)
--- BackupReservation(ID, customer_ID, tour_ID, issue_date)
```

```
CREATE TRIGGER QuotaBackupTrigger AFTER INSERT ON ReservationCancel
REFERENCING NEW ROW AS nrow
BEGIN
    SELECT tour_ID INTO @released_tour FROM Reservation
    WHERE Reservation.ID = nrow.reservation_ID;
    SELECT customer_ID INTO @backup_customer FROM BackupReservation
    WHERE BackupReservation.tour_ID = released_tour
    AND BackupReservation.issue_date <
    ALL(SELECT issue_date FROM BackupReservation);
    INSERT INTO Reservation(customer_ID, tour_ID, issue_date, payment_status)
    VALUES(@backup_customer, @released_tour, GETDATE(), 'Unpaid');
    DELETE FROM BackupReservation WHERE tour_id = released_tour;
END;
```

6.3.2. Re-Reservation Trigger

When a customer makes a reservation which he/she have cancelled before, that tour should no longer be appearing as cancelled by the user. Thus, if the previous cancelled reservation record should be deleted.

```
--- *** Tables Used ***
```

```
--- ReservationCancel(reservation_ID, cancel_date)
--- Reservation(ID, customer_ID, tour_ID, issue_date, payment_status)
```

```
CREATE TRIGGER ReReservationTrigger BEFORE INSERT ON Reservation
```

```

REFERENCING NEW ROW AS nrow
BEGIN
    SELECT ID INTO @old_res_ID FROM Reservation
        WHERE customer_ID = nrow. customer_ID AND tour_ID = nrow.tour_ID;
    DELETE FROM Reservation WHERE ID = @old_res_ID;
    DELETE FROM ReservationCancel WHERE ID = @old_res_ID;
END;

```

6.4. Constraints

- A customer can make at most 1 reservation to a tour.
- A customer have to pay the reservation price until the cancellation date. If its not paid until the last day, a notification is sent to the user. When the deadline is past, the reservation is cancelled.
- A customer cannot make reservation to a tour with full quota. (However, they can be added to the backup list.)
- A tour has to have at least 1 element (trip event, accommodation or travel route) to exist.

6.5. Stored Procedures

6.5.1. Searching for Tours

We have decided to design a procedure for obtaining a tour list with a given query. This procedure searches for the query in the tour name, tour city and tour country. It makes use of the TourAssociations view, that has been defined in Section 6.2.

```

CREATE PROCEDURE SearchForQuery @query VARCHAR(50)
AS
    SELECT Tour_ID
    FROM TourAssociations
    WHERE city_name LIKE CONCAT( ' %' , @query , '%')
        OR tour_name LIKE CONCAT( ' %' , @query , '%')
        OR country_name LIKE CONCAT( ' %' , @query , '%')
GO;

```


6.5.2. Filtering Tours

We have decided to design a procedure for filtering tours according to the given TagName and TagType.

```
CREATE PROCEDURE FilterByTagWithType @TagName VARCHAR(15),
@TagType VARCHAR(15)
AS
    SELECT *
    FROM Tour
    WHERE Tour.ID IN ( SELECT tour_ID
                        FROM TourTags NATURAL JOIN Tag
                        WHERE name = @TagName AND type = @TagType )
GO;
```

7. Implementation Plan

In this project we will be using PHP for our back-end web service, along with the MySQL database. In front-end, we will create a website with HTML and make use of Bootstrap CSS library to obtain a modern website view.

8. Website

In this project, our reports are being hosted on the following website:

<https://cs353-01.firebaseio.com>