Bilkent University

Department of Computer Engineering

CS353 Database Systems

Social Network for Check-In - CheckMe

Project URL: http://bit.ly/CS353DB

Project Design Report

Ahmet Çandıroğlu, Albjon Gjuzi, Aurel Hoxha, Eniselda Tusku

Supervisor: Fuat Basık

Feb 26, 2018

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Database Systems, course CS353.

Table of Contents

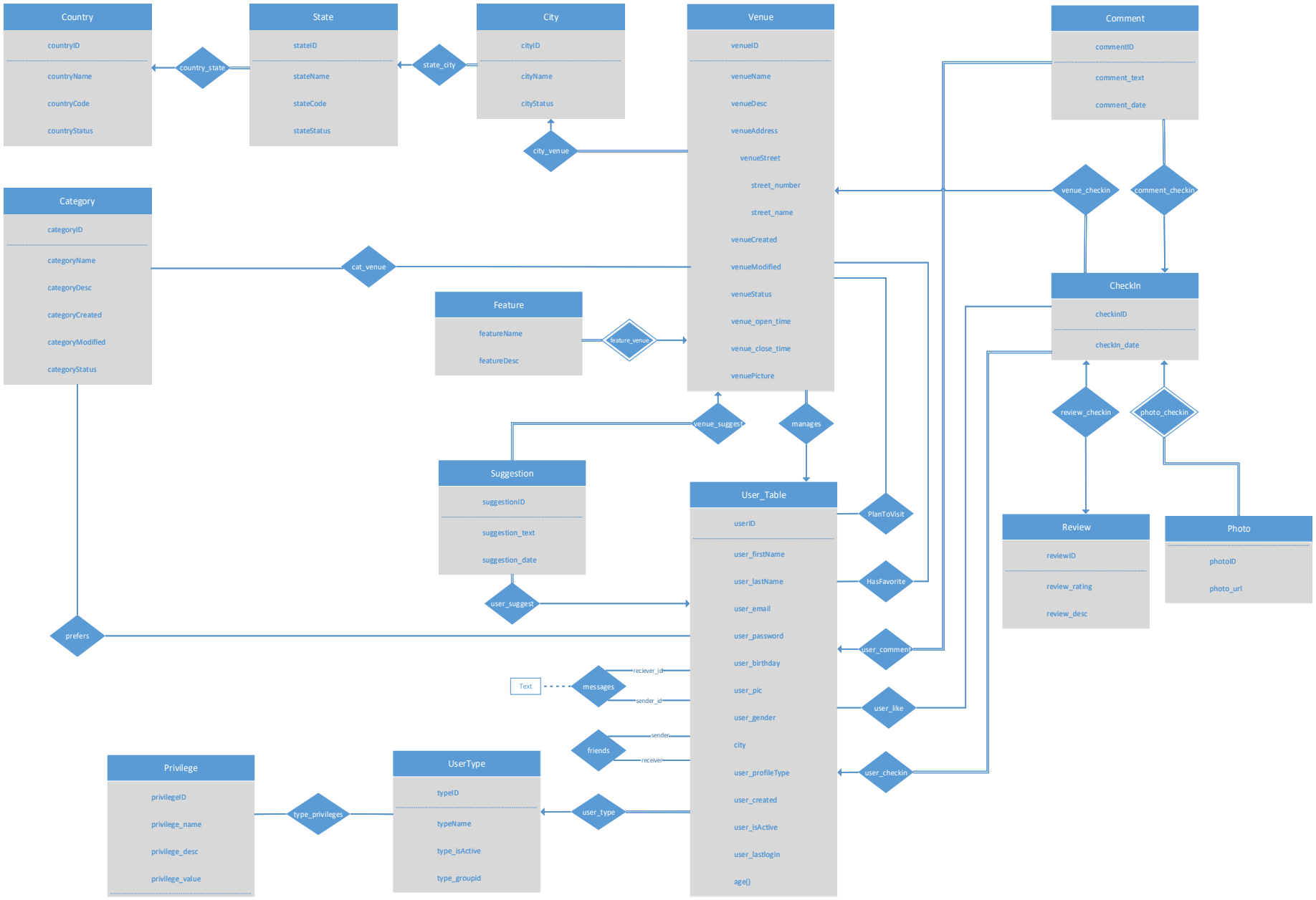
1. Revised E / R Model

According to assistant feedback, we revised the E/R Model as follows:

- The names of the relationships were changed in order to have no relationships with the same name.
- All missing total participation as in case of (country_state, state_city, city_venue) were fixed.
- The entity of the feature was changed from a normal one to a weak one. We realized that a feature does not exist if a venue does not.
- The is-a-relationship between user and manager was removed and updated using a roleId that indicate the type of the user such as (user, admin, manager).
- The relationship between venue and user was changed from 1-to-1 to 1-to-many considering the fact that a user that is manager can manages more than 1 venue.
- CheckIn was converted from a relationship to an entity and the required relationships were added.
- Comment was converted from a relationship to an entity and the required relationships were added.
- The lists inside the entities were removed and substituted with other relationships.

The E/R diagram was completed adding the following modification:

- Review entity was added and it has a 1-to-1 relationship with CheckIn considering the fact that a review corresponds only to one checkIn.
- Suggestion entity was added to provide a way for the user to send suggestions to different venues.
- UserType and Privilege entities were added such that provide a way to maintain different privileges for different types of user.
- Photo entity was added as e weak entity to CheckIn. The reason for this is because it is in user preferences whether or not to merge a photo with the CheckIn.
- Comment entity was added and a relationship that connect the CheckIn and the user was added.
- A relationship that contains the likes of different user to different checkIn-s was added.
- Friends relationship between two different users was added to contain the list of friends for different user.
- Message relationship between two different user was added to save the messages


Country
countryID
countryName
countryCode
countryStatus
country_state
State
stateID
stateName
stateCode
stateStatus
state_city
City
cityID
cityName
cityStatus
city_venue
Venue
venueID
venueName
venueDesc
venueAddress
venueStreet
street_number
street_name
venueCreated
venueModified
venueStatus
venue_open_time
venue_close_time
venuePicture
Comment
commentID
comment_text
comment_date
venue_checkin
comment_checkin
Category
categoryID
categoryName
categoryDesc
categoryCreated
categoryModified
categoryStatus
cat_venue
Feature
featureName
featureDesc
feature_venue
CheckIn
checkinID
checkIn_date
review_checkin
photo_checkin
venue_suggest
manages
Suggestion
suggestionID
suggestion_text
suggestion_date
user_suggest
User_Table
userID
user_firstName
user_lastName
user_email
user_password
user_birthday
user_pic
user_gender
city
user_profileType
user_created
user_isActive
user_lastlogin
age()
PlanToVisit
HasFavorite
user_comment
user_like
user_checkin
Review
reviewID
review_rating
review_desc
Photo
photoID
photo_url
prefers
Text
messages
reciever_id
sender_id
friends
sender
receiver
Privilege
privilegeID
privilege_name
privilege_desc
privilege_value
type_privileges
UserType
typeID
typeName
type_isActive
type_groupid
user_type

2. Relation Schemas

2.1. Country

Relational Mode:

Country(<u>countryID</u>, countryName, countryCode, countryStatus)

Functional Dependencies:

countryID -> countryName countryCode countryStatus

Candidate Keys:

{(countryID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE country (
        countryID int PRIMARY KEY AUTO_INCREMENT,
        countryName varchar(40) NOT NULL,
        countryCode varchar(10) NOT NULL,
        countryStatus  int NOT NULL )
        ENGINE=InnoDB;
```

2.2. State

Relational Mode:

State(stateID, stateName, stateCode, stateStatus, countryID)

Functional Dependencies:

stateID -> stateName stateCode stateStatus countryID

Candidate Keys:

{(stateID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE state (
        stateID int PRIMARY KEY AUTO_INCREMENT,
        stateName varchar(40) NOT NULL,
        stateCode varchar(10) NOT NULL,
        stateStatus  int NOT NULL,
        countryID int NOT NULL,
        FOREIGN KEY(countryID) references country  )
        ENGINE=InnoDB;
```

2.3. City

Relational Mode:

City(cityID, cityName, cityStatus, stateID)

Functional Dependencies:

cityID -> cityName cityStatus stateID

Candidate Keys:

{(cityID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE city (
        cityID int PRIMARY KEY AUTO_INCREMENT,
        cityName varchar(40) NOT NULL,
        cityStatus  int NOT NULL,
        stateID int NOT NULL,
        FOREIGN KEY(stateID) references state  )
        ENGINE=InnoDB;
```

2.4. Category

Relational Mode:

Category(categoryID, categoryName, categoryDesc, categoryCreated, categoryModified, categoryStatus)

Functional Dependencies:

categoryID -> categoryName categoryDesc categoryCreated categoryModified categoryStatus

Candidate Keys:

{(categoryID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE category (

        categoryID int PRIMARY KEY AUTO_INCREMENT,

        categoryName varchar(40) NOT NULL,

        categoryDesc varchar(150),

        categoryCreated date NOT NULL,

        categoryModified date NOT NULL,

        categoryStatus int NOT NULL)

        ENGINE=InnoDB;
```

2.5. Venue

Relational Mode:

Venue(venueID,venueName,venueDesc,street_number, street_name, venueCreated, venueModified, venueStatus, venue_open_time, venue_close_time, venue_picture, cityID, managerID)

Functional Dependencies:

venueID -> venueName venueDesc street_number street_name venueCreated venueModified venueStatus venue_open_time venue_close_time venue_picture cityID managerID

Candidate Keys:

{(venueID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE venue (
        venueID int PRIMARY KEY AUTO_INCREMENT,
        venueName varchar(40) NOT NULL,
        venueDesc varchar(150),
        street_number int,
        street_name varchar(40),
        venueCreated date NOT NULL,
        venueModified date NOT NULL,
        venue_open_time time,
        venue_close_time time,
        venue_picture blob,
        venueStatus  int  NOT NULL,
        cityID int NOT NULL,
        managerID int,
        FOREIGN KEY(cityID) references city,
        FOREIGN KEY(managerID) references user_table(UserID))
        ENGINE=InnoDB;
```

2.6. Feature

Relational Mode:

Feature(VenueID, featureName, featureDesc)

Functional Dependencies:

VenueID featureName -> featureDesc

Candidate Keys:

{(VenueID, featureName)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE feature (
        venueID int,
        featureName varchar(40),
        featureDesc varchar(50) NOT NULL,
        PRIMARY KEY(venueID, featureName),
        FOREIGN KEY(venueID) references venue )
        ENGINE=InnoDB;
```

2.7. Cat_Venue

Relational Mode:

Cat_Vanue(categoryID, venueID)

Functional Dependencies:

Candidate Keys:

{(categoryID, venueID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE cat_venue (
        categoryID int,
        venueID int,
        PRIMARY KEY(categoryID, venueID),
        FOREIGN KEY(categoryID) references category,
        FOREIGN KEY(venueID) references venue )
        ENGINE=InnoDB;
```

2.8. Privilege

Relational Mode:

Privilege(privilegeID, privilege_name, privilege_desc, privilege_value)

Functional Dependencies:

privilegeID -> privilege_name privilege_desc privilege_value

Candidate Keys:

{(privilegeID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE privilege (
        privilegeID int PRIMARY KEY AUTO_INCREMENT,
        privilege_name varchar(40) NOT NULL,
        privilege_desc varchar(150) NOT NULL,
        privilege_value varchar(50) NOT NULL)
        ENGINE=InnoDB;
```

2.9. UserType

Relational Mode:

UserType(typeID, typeName, type_isActive, type_groupid)

Functional Dependencies:

typeID-> typeName type_isActive type_groupid

Candidate Keys:

{(typeID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE usertype (
        typeID int PRIMARY KEY AUTO_INCREMENT,
        typeName varchar(40) NOT NULL,
        type_isActive int NOT NULL,
        type_groupid int NOT NULL )
        ENGINE=InnoDB;
```

2.10. Type_Privilege

Relational Mode:

Type_Privilege (typeID, privilegeID)

Functional Dependencies:

Candidate Keys:

{(typeID, privilegeID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE type_privilege (
        typeID int,
        privilegeID int,
        PRIMARY KEY(typeID, privilegeID),
        FOREIGN KEY(typeID ) references usertype,
        FOREIGN KEY(privilegeID) references privilege )
        ENGINE=InnoDB;
```

2.11. User_Table

Relational Mode:

User_Table(<u>userID</u>, user_firstName, user_lastName, user_email, user_password, user_birthdate, user_pic, user_gender, city, user_profileType, user_created, user_isActive, user_lastlogin, typeID)

Functional Dependencies:

<u>userID</u> -> user_firstName user_lastName user_email user_password user_birthdate user_pic user_gender city user_profileType user_created user_isActive user_lastlogin typeID

Candidate Keys:

{(userID,user_email)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE user_table (
        userID int PRIMARY,
        user_firstName varchar(50) NOT NULL,
        user_lastName varchar(50) NOT NULL,
        user_email varchar(100),
        user_password varchar(30) NOT NULL,
        user_birthdate date NOT NULL,
        user_pic blob,
        user_gender character(1) NOT NULL,
        city varchar(50) NOT NULL,
        user_profileType int NOT NULL,
        user_created date NOT NULL,
        user_isActive int NOT NULL,
        user_lastlogin time NOT NULL,
        typeID int NOT NULL,
        FOREIGN KEY(typeID) references usertype)
        ENGINE=InnoDB;
```

2.12. Prefers

Relational Mode:

Prefers(userID, categoryID)

Functional Dependencies:

Candidate Keys:

{(userID, categoryID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE prefers (
        userID int,
        categoryID int,
        PRIMARY KEY(userID, categoryID),
        FOREIGN KEY(userID) references user_table,
        FOREIGN KEY(categoryID) references category )
        ENGINE=InnoDB;
```

2.13. Messages

Relational Mode:

Messages(userID1, userID2, message)

Functional Dependencies:

userID1 userID2 -> message

Candidate Keys:

{(userID1, userID2)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE messages (
        userID1 int,
        userID2 int,
        message varchar(500),
        sent_date date,
        PRIMARY KEY(userID1, userID2),
        FOREIGN KEY(userID1) references user_table (userID),
        FOREIGN KEY(userID2) references user_table (userID))
        ENGINE=InnoDB;
```

2.14. Friends

Relational Mode:

Friends(<u>userID1, userID2</u>)

Functional Dependencies:

Candidate Keys:

{(userID1, userID2)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE friends (
        userID1 int,
        userID2 int,
        PRIMARY KEY(userID1, userID2),
        FOREIGN KEY(userID1) references user_table (userID),
        FOREIGN KEY(userID2) references user_table (userID))
        ENGINE=InnoDB;
```

2.15. Suggestion

Relational Mode:

Suggestion(<u>suggestionID</u>, suggestion_text, suggestion_date, venueID, userID)

Functional Dependencies:

suggestionID -> suggestion_text suggestion_date venueID userID

Candidate Keys:

{(suggestionID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE suggestion (
        suggestionID int PRIMARY KEY AUTO_INCREMENT,
        suggestion_text varchar(250) NOT NULL,
        suggestion_date date NOT NULL,
        venueID int NOT NULL,
        userID int NOT NULL,
        FOREIGN KEY(venueID) references venue,
        FOREIGN KEY(userID) references user_table)
        ENGINE=InnoDB;
```

2.16. PlanToVisit

Relational Mode:

PlanToVisit(<u>userID, venueID</u>)

Functional Dependencies:

Candidate Keys:

{(userID, venueID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE plan_to_visit (
        userID int,
        venueID int,
        PRIMARY KEY(userID, venueID),
        FOREIGN KEY(userID) references user_table,
        FOREIGN KEY(venueID) references venue)
        ENGINE=InnoDB;
```

2.17. HasFavorite

Relational Mode:

HasFavorite(<u>userID, venueID</u>)

Functional Dependencies:

Candidate Keys:

{(userID, venueID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE has_favorite (
        userID int,
        venueID int,
        PRIMARY KEY(userID, venueID),
        FOREIGN KEY(userID) references user_table,
        FOREIGN KEY(venueID) references venue)
        ENGINE=InnoDB;
```

2.18. CheckIn

Relational Mode:

CheckIn(checkinID, checkin_date, userID, venueID, reviewID)

Functional Dependencies:

checkinID -> checkin_date userID venueID reviewID

Candidate Keys:

{(checkinID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE checkin (
        checkinID int PRIMARY KEY AUTO_INCREMENT,
        checkin_date date NOT NULL,
        userID int NOT NULL,
        venueID int NOT NULL,
        reviewID int NOT NULL,
        FOREIGN KEY(userID) references user_table,
        FOREIGN KEY(venueID) references venue,
        FOREIGN KEY(reviewID) references review )
        ENGINE=InnoDB;
```

2.19. Photo

Relational Mode:

Photo(checkinID, photoID, photoFile)

Functional Dependencies:

checkinID photoID -> photoFile

Candidate Keys:

{(checkinID, photoID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE photo (
        checkinID int,
        photoID int,
        photoFile blob NOT NULL,
        PRIMARY KEY(checkinID,photoID),
        FOREIGN KEY(checkinID) references checkin )
        ENGINE=InnoDB;
```

2.20. Review

Relational Mode:

Review(reviewID, review_rating, review_desc)

Functional Dependencies:

reviewID -> review_rating review_desc

Candidate Keys:

{(reviewID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE review (
        reviewID int PRIMARY KEY AUTO_INCREMENT,
        review_rating int NOT NULL,
        review_desc varchar(150) )
        ENGINE=InnoDB;
```

2.21. Comment

Relational Mode:

Comment(commentID, comment_text, comment_date, userID, checkinID)

Functional Dependencies:

commentID -> comment_text comment_date userID, checkinID

Candidate Keys:

{(commentID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE comment (
        commentID int PRIMARY KEY AUTO_INCREMENT,
        comment_text varchar(250) NOT NULL,
        comment_date date NOT NULL,
        userID int NOT NULL,
        checkinID int NOT NULL,
        FOREIGN KEY(userID) references user_table,
        FOREIGN KEY(checkinID) references checkin)
        ENGINE=InnoDB;
```

2.22. User_Like

Relational Mode:

User_Like(userID, checkinID)

Functional Dependencies:

Candidate Keys:

{(userID, checkinID)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE user_like (
        userID int NOT NULL,
        checkinID int NOT NULL,
        PRIMARY KEY(userID, checkinID),
        FOREIGN KEY(userID) references user_table,
        FOREIGN KEY(checkinID) references checkin)
        ENGINE=InnoDB;
```

3. Functional Dependencies and Normalization of Tables

All functional dependencies and normal forms are indicated in Relation Schemas in Section 2 of this Project Design Report. First we check whether all our relations are in Boyce-Codd normal form. We concluded that no decomposition is required.

4. Functional Components

4.1. Use Cases / Scenarios

In CheckMe Social Network, there will be two types of user, normal user and manager. Since the admin will have total privileges it will not be included in the use cases. Manager and Users have both similarities and differences and in order to use the system all user have to log in into the system selecting the type of the account they want to create.

User

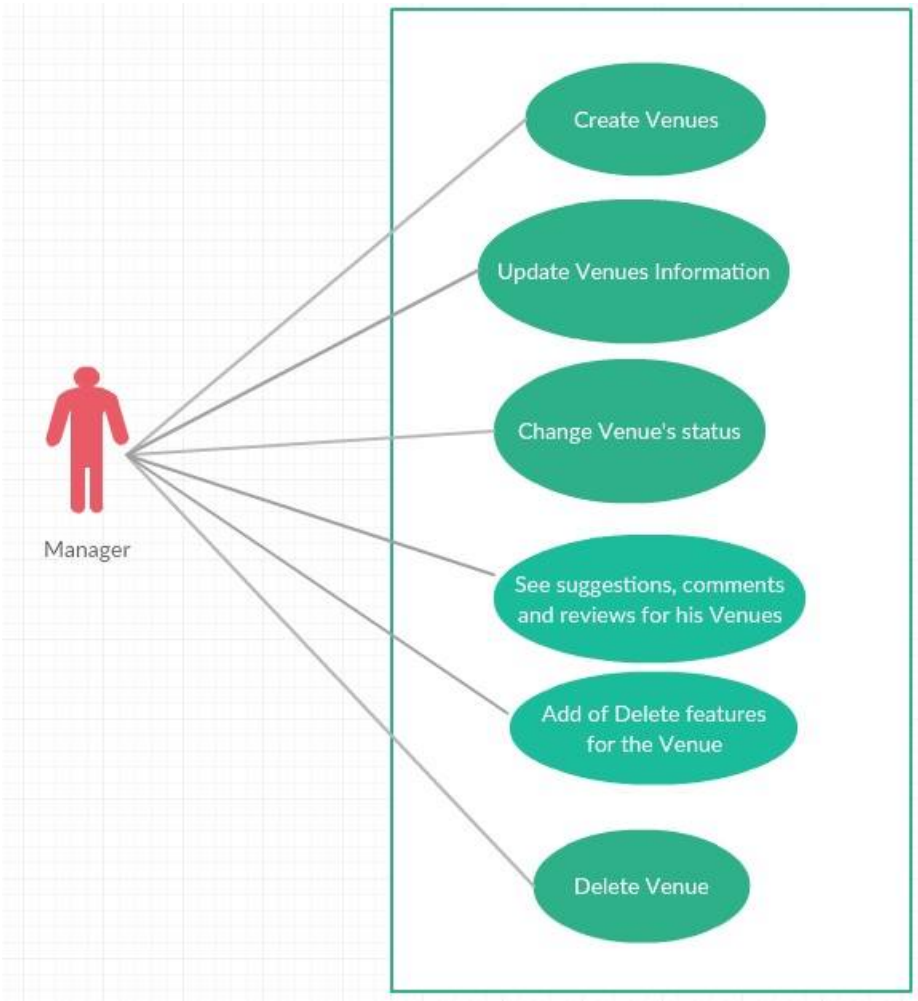
- A user can sign up and register to our system using a unique email, password, and selecting he is a simple user.
- User can login to the system with the user ID and password.
- User can access his profile page: which includes his first and last name, email address, birthday, profile picture, gender and address.
- User can change password from profile page.
- User can make his profile private. (Public by default)
- User can update the information of his profile.
- User can make friends from other users.
- User can search for venues based on country, state, or city.
- User can search for venues based on venue category.
- User can select features which the venues must have.
- User can view the venue's details which include venue name, description, address, online-offline status, open time, close time and the rating determined by other users.
- User can view previous check-ins at different venues together with the review.
- User can check-in at venues and the check-in includes optional photos, a description about user's experience at the venue and a review of the venue which is composed of a rating from 1-10 and a short description.
- User can send suggestion to the venues.
- Users can add friends.
- User can like or comment check-ins of their friends or people whose profile is public.
- User can have a list of check-ins of his friends and category preferences in chronological order in his news feed.
- User can send messages to his friends, or to people whose profile is public.
- User can create and save a list of venues he plans to visit in the future.
- User can recommend a venue to his friends.
- User can deactivate his account.

Every use-case requires Registration and later Login to access the system. They are omitted for simplicity of the diagram.

Manager

- A manager can sign up and register to our system using a unique email, password, and selecting he is a manger.
- Manager can manage venues.
- Manager can create venues.
- Managers can update the information of the venues he manages which consist of venue name, description, address, status, open time and close time.
- Manager can delete venues that he is managing from the system.
- Manager can also act as an ordinary user, which means check-in at different venues, like and comment, add friends, sends messages to them and all other functionalities that users have.

Every use-case requires Registration and later Login to access the system. They are omitted for simplicity of the diagram.



4.2. Algorithms

Search Related Algorithm
All users interacting with the application can search through the website using keywords. The user will firstly have to select what he is looking for. The options to be selected will be Country, State, City, Venue and People.

The search algorithm will be different for all of these options. For example if user searches for the country, all the states and cities in those states will be automatically selected. Therefore, the venues that will be displayed will be those of the country he selected. The same will be for the state, all the cities of the state will be automatically selected and the venues displayed will all belong to the state searched from the user. To make searching process easier for the user, after the first character for every character that the user enters in the search box the options containing that keyword will be displayed below the text box.

News Feed Algorithm
The data management system will allow user to keep track of check-ins from their favorite venues or from their favorite categories plus their friends.

In the news feed page there will be two checkboxes called categories and venues which will be combined with the user's friend's check-ins. Besides the checkboxes there will be two radio buttons called most recent and top. If the user selects most recent the algorithm will be bases on chronological order, meanwhile if the user selects top we will display them the posts with more likes and comments combined from the last 24 hours. Likes will affect 70% of the algorithm and comments the rest 30%.

Rating Algorithm
Users in our system will have different impact in the review rating of the venues. There will be a simple algorithm to make sure that the rating of the venue is closer to the value it should really have.

The first issue with the rating algorithm is that the managers will be prevented from rating the venues they manage, because this is not fair as they would definitely give a big 10 to their venues. Besides the manager, all users' will affect the overall rating in different ways. Users with less than 5 review rating in total will have only 15% weight in the total rating, users with more than 5 and less than 50 review ratings will have 35% weight in the total rating and users with more than 50 review ratings will have the remaining 50% of the weight.

Logical Requirements

In order to prevent logical errors inside our system, there will be a number of spots that the system ought to approach in a sensitive manner. For example the open hours of a specific venue should be followed carefully.

Two attributes in the *venue* table are venue_open_time and venue_close_time. Considering the values of those attributes, the difference between them should be less than 24. For example a venue cannot open at 16:00pm and close at 15:00pm. However, the open time can be bigger than the close time. This will mostly occur in night clubs which can open at 20:00pm and close at 06:00am. Such restrictions are made in order to prevent mistakes while entering the open and close hours of the venue.

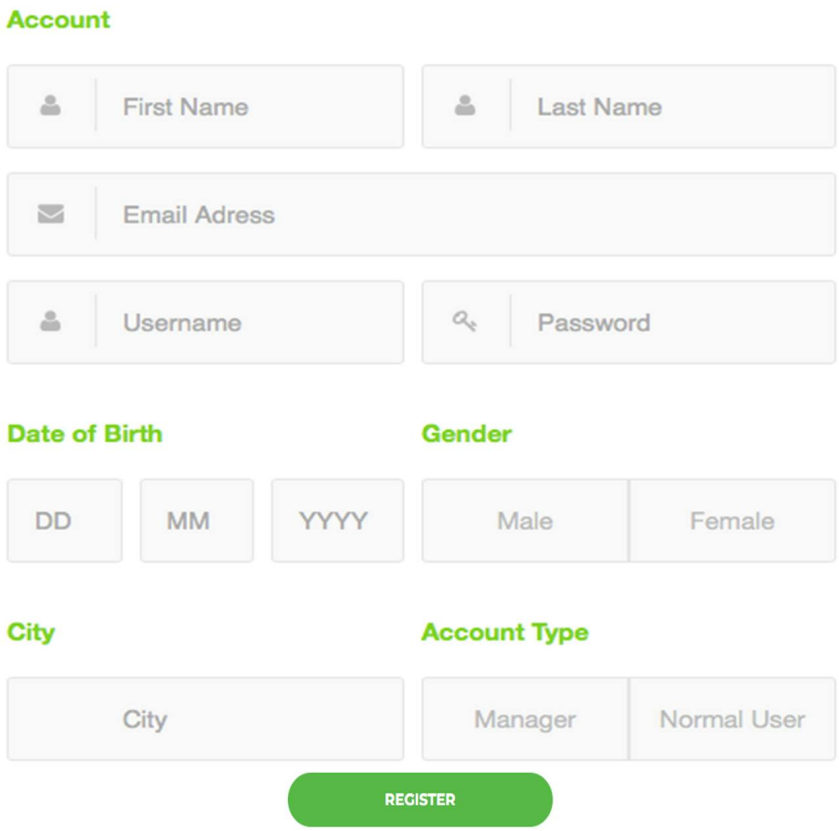
Checking in at a venue should be atomic. This implies that the check-in will not be uploaded to the database if the photos are not already uploaded. We will provide an algorithm in order to make sure that the photos are uploaded together with the check-in description.

4.3. Data Structures

For the attribute domains we will use Numeric type, Date type, Time type, Blob type and other types that are supported from MySQL.

5. User Interface Design And Corresponding SQL Statements

5.1. Signup



Inputs: @first_name, @last_name, @email_address, @username, @password, @birthday, @gender, @city, @profile_type

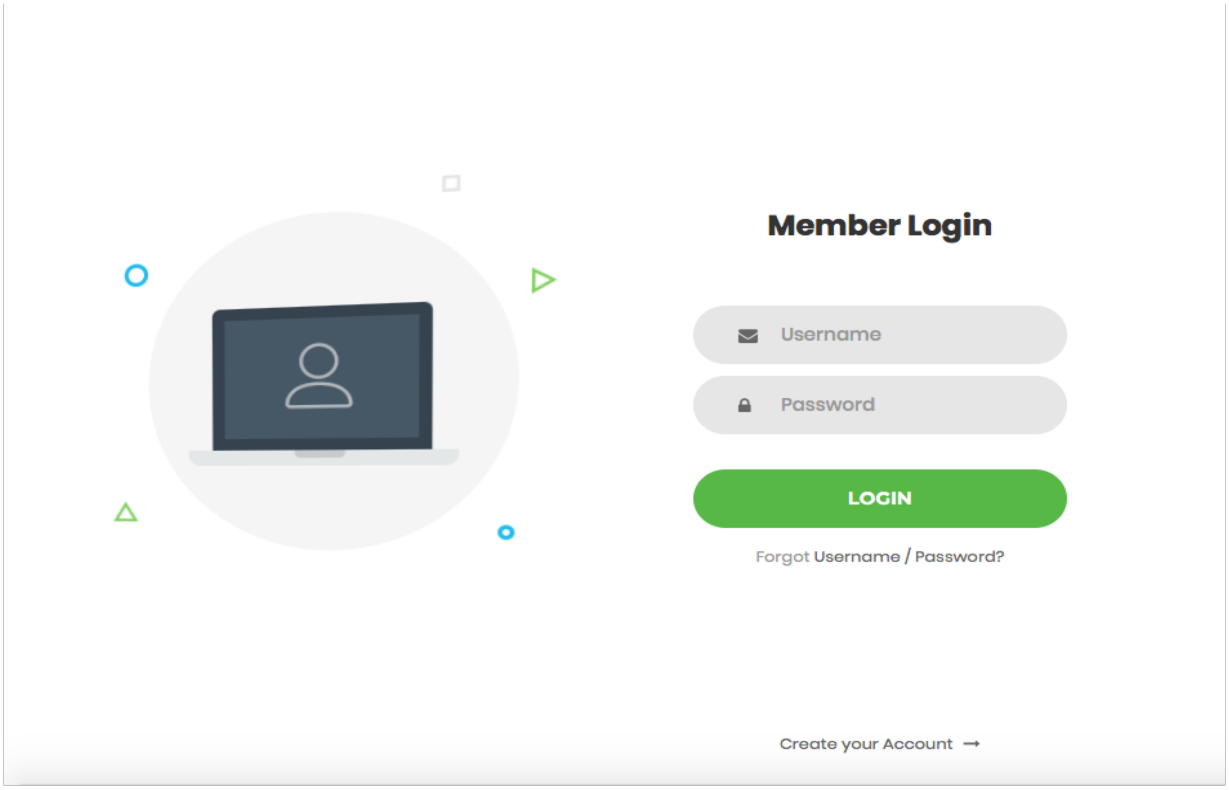
Process: The user enter his information to register to CheckMe System. The information include user's first name, last name, email, username, password, date of birth, gender, city and the account type.

SQL Statement:

Registering

INSERT INTO user_table
VALUES(@username, @first_name, @last_name, @email_address, @password, @birthday, NULL, @gender, @city, @profile_type, NOW(), 1, NOW()

5.2. Login



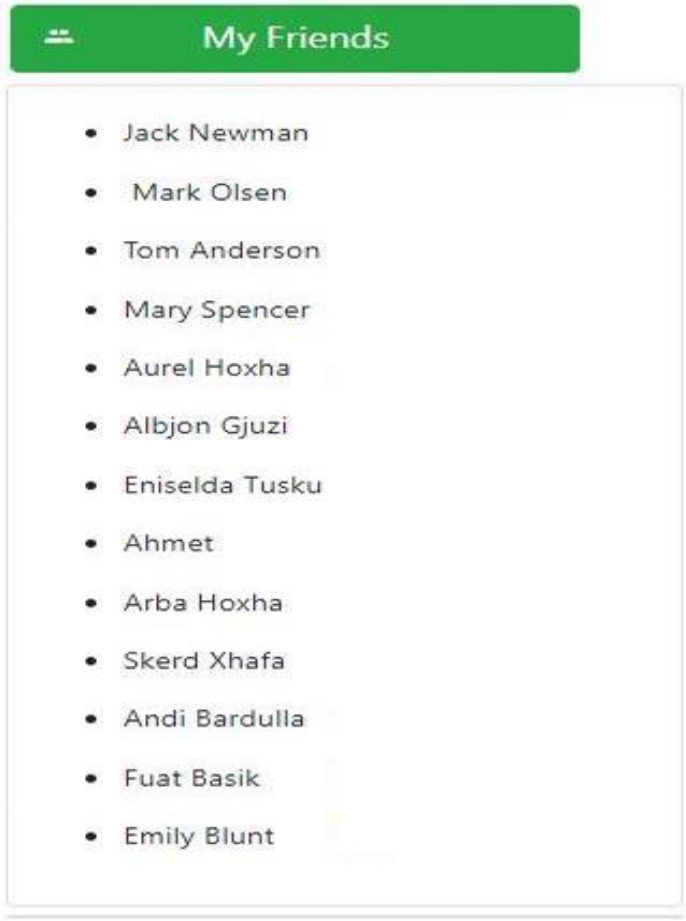
Inputs: @username, @password

Process: The user enter his username and his password to enter to the *CheckMe* System.

SQL Statement:

Logging in
SELECT userID, password
FROM user_table
WHERE userID = @username AND password = @password

5.3. Friends



Inputs: @username

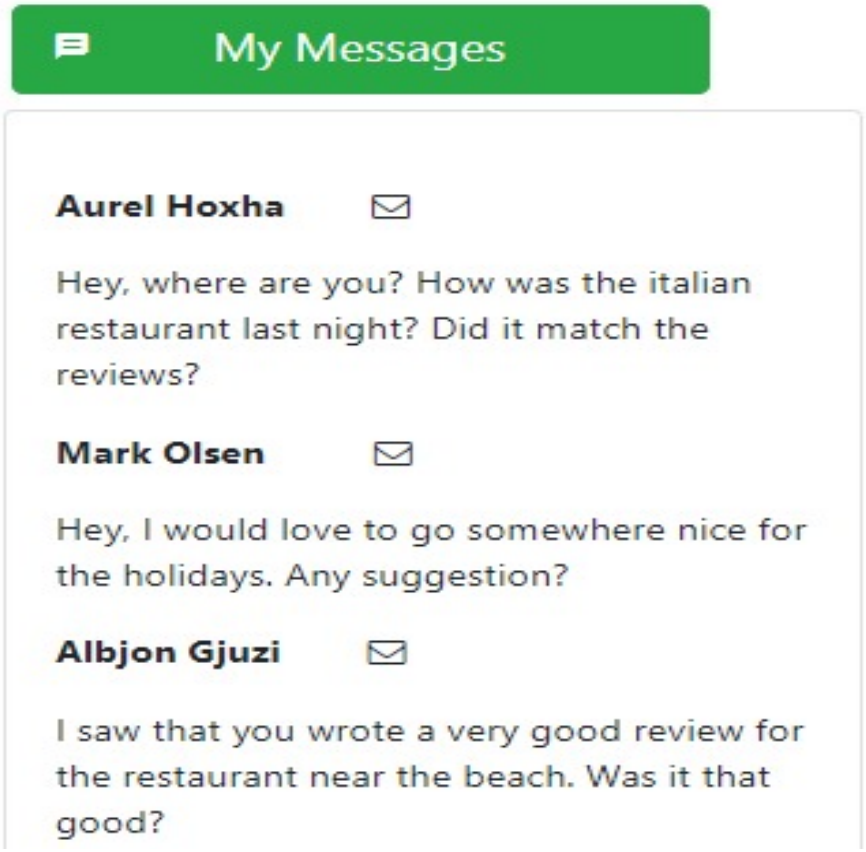
Process : When a user open the friends screen, all his friends will be listed without any filter.

@username: Is the username of the user who is currently signed in.

SQL Statement:

```
SELECT friends.userID2
FROM friends
WHERE friends.userID1 = @username
```

5.4. Messages



Inputs: @username

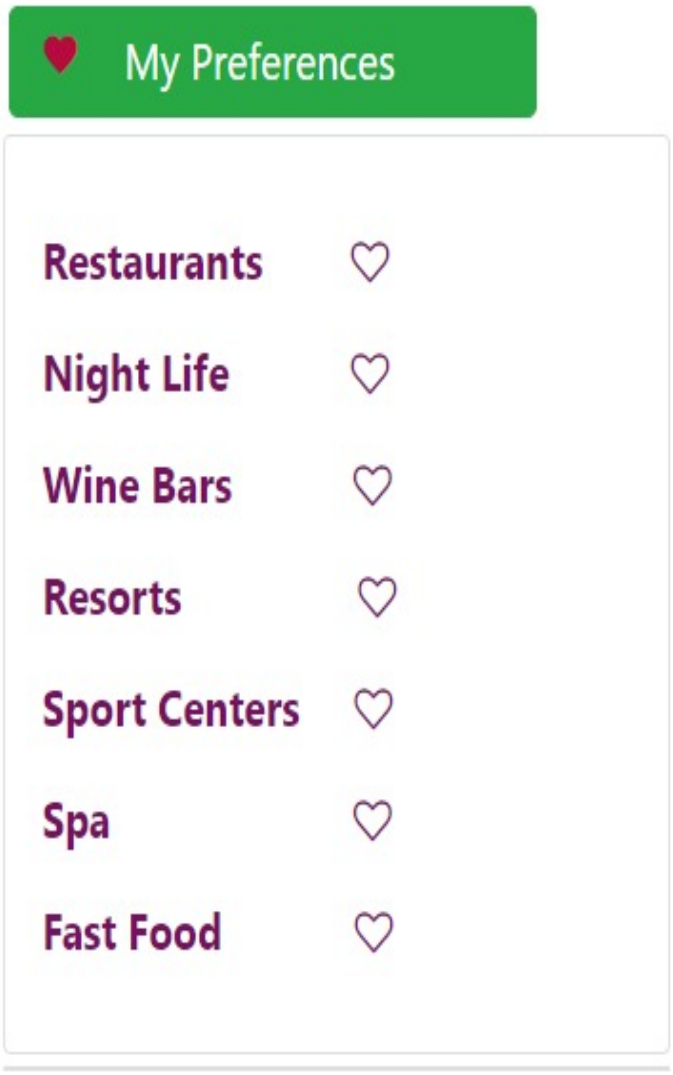
Process : When a user open the messages screen, all his messages will be displayed

@username: Is the username of the user who is currently signed in.

SQL Statement:

```
SELECT messages.userID2, messages.text, message.sent_date
FROM messages
WHERE messages.userID1 = @username
GROUP BY messages.userID2
```

5.5. User Preference



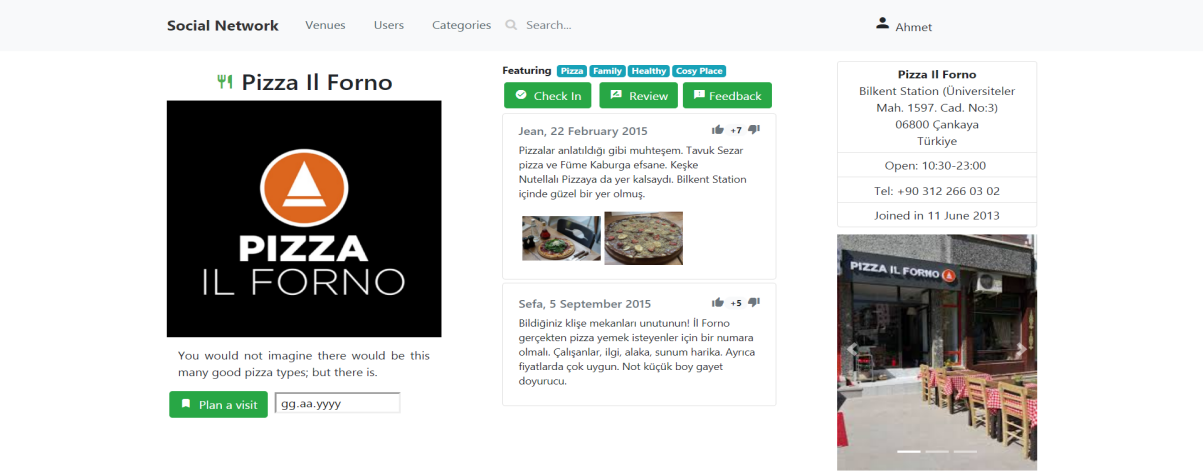
Inputs: @username

Process : When a user open the preference screen, all his preferences will be displayed.

@username: Is the username of the user who is currently signed in.

SQL Statement:
SELECT prefers.categoryID
FROM prefers
WHERE friends.userID = @username

5.6. Venue



Inputs: @userID, @venueID

Process: Venue page displays all the information about that Venue and Review's done about that Venue. Users can Check In, Review and give Feedback (Suggestion) to the Venue.

@userID: ID of the user signed in.
@venueID: ID of the venue which is being displayed.

SQL Statements:

Loading Venue information

```
SELECT    V.venueName, V.venueDesc, V.venue_picture, V.street_number, V.street_name,
          V.venueCreated, V.venue_open_time, V. venue_close_time,
          Category.categoryName, City.cityName, S.stateName, Country.countryName
FROM      Venue V, Country CO, State S, City CI, Category CA, Cat_Venue
WHERE     V.venueID = venueID and V.cityID = CI.cityID and CI.stateID = S.stateID
          and S.countryID =  CO.countryID and CA.categoryID = Cat_Venue.categoryID
          and venueID = Cat_Venue.venueID
```

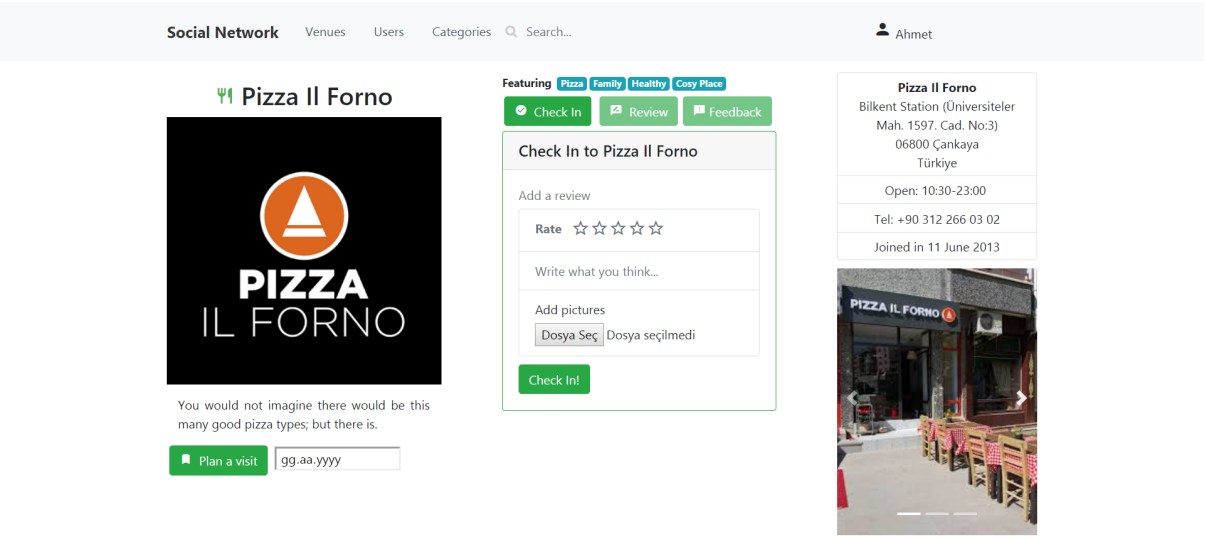
Loading Feature information

```
SELECT    F.featureName
FROM      Venue V, Feature F
WHERE     venueID = V.venueID and F.venueID = venueID
```

Loading Check In and Review Information

```
SELECT    U.user_first_name, C.checkin_date, C.checkin_like_num, R.review_desc,
          P.photoFile
FROM      Venue V, CheckIn C, Review R, User U, Photo P
WHERE     venueID = V.venueID and C.venueID = venueID and C.reviewID = R.reviewID
          and P.checkinID = C.checkinID
```

5.7. Check In



Inputs: @userID, @venueID, @reviewDesc, @rating, @photoFile, @date

Process: User can check in and add review if he/she wants. If the user does not want to add review to the check in only check in will be created. Note that this display is not a new page, it is Venue page when Check In button is pressed.

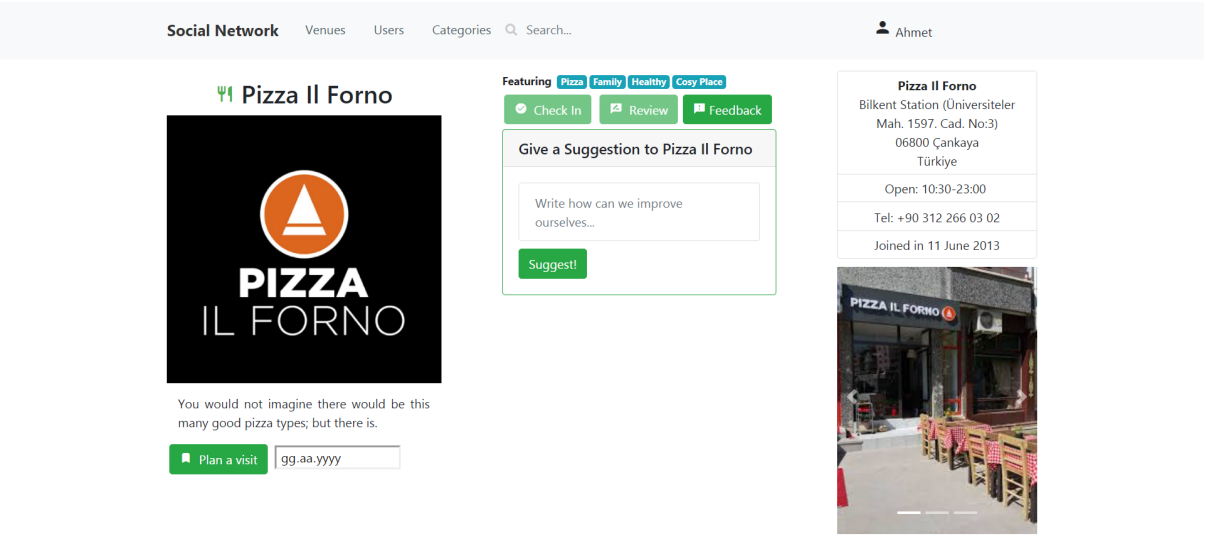
@userID: ID of the signed in user
@venueID: Venue which is currently being displayed
@reviewDesc, @rating and @photoFile: Input of user to add review
@date: Date of check in which is taken from the server

SQL Statements:

Adding Review (If the User wants to add)
INSERT INTO Review (reviewID, review_rating, review_desc)
VALUES (@reviewID, @rating, @reviewDesc)

Adding Check In
INSERT INTO CheckIn(checkinID, checkin_date, userID, venueID, reviewID)
VALUES (@checkInID, @date, @userID, @venueID, @reviewID)

5.8. Suggestion



Inputs: @userID, @venueID, @text, @date

Process: User can give suggestions to the Venue which will be only shown to Venue managers, it won't be public. Note that this display is not a new page, it is Venue page when Feedback button is pressed.

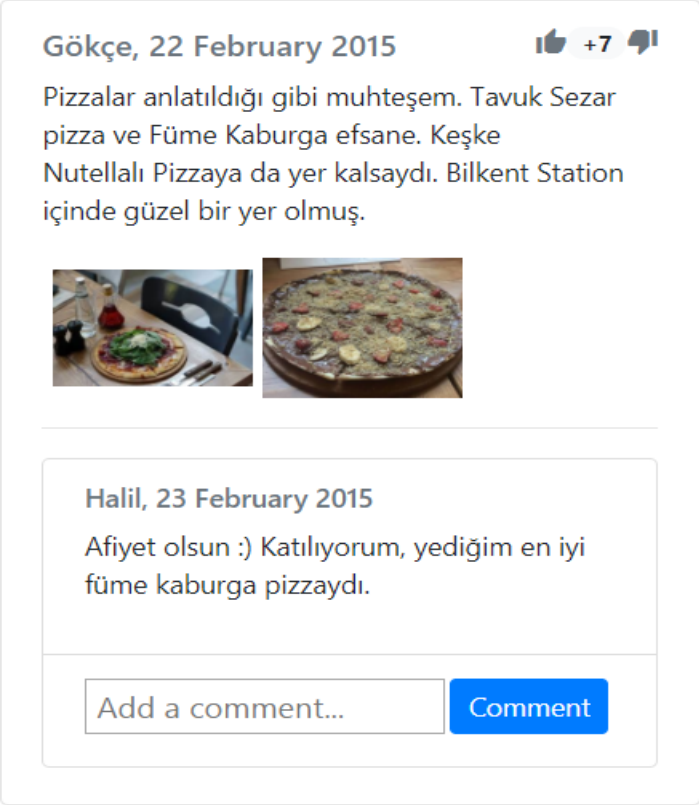
@userID: ID of the signed in user
@venueID: Venue which is currently being displayed
@text: Input of user to add suggestion
@date: Date of suggestion which is taken from the server

SQL Statements:

Adding Suggestion

```
INSERT INTO Suggestion (suggestionID, suggestion_text, suggestion_date, venueID,
                        userID)
VALUES        (@suggestionID, @text, @date, @venueID, @userID)
```

5.9. Comment



Inputs: @userID, @checkInID, @text, @date

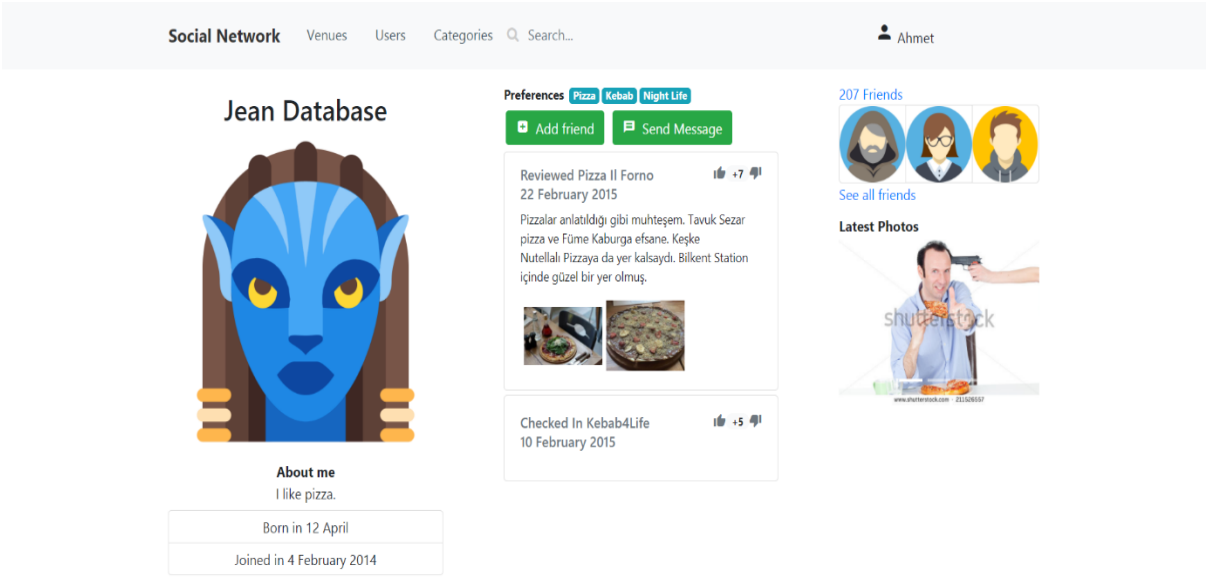
Process: User can comment on check-ins of people which includes only a text. Note that this display can be shown in user profile page, home screen and venue profile; this is not a separate page.

@userID: ID of the signed in user
@checkInID: Check in to comment on
@text: Input of user to add comment
@date: Date of comment which is taken from the server

SQL Statements:

Adding Comment
INSERT INTO Comment (commentID, comment_text, comment_date, userID, checkInID)
VALUES (@commentID, @text, @date, @userID, @checkInID)

5.10. User



Inputs: @userProfileID, @userID, @messageText

Process: User profile page displays all the information about that User and Review's he/she has done. Currently signed in User can add viewed person as a friend, send a message and like/comment on his/her check-ins.

@userID: ID of the user signed in.
@userProfileID: ID of the venue which is being displayed.

SQL Statements:

Loading User information

```
SELECT      U.user_first_name, U.user_last_name, U.user_picture, U.user_created
FROM        User U
WHERE       userProfileID = U.userID
```

Loading Check In and Review Information

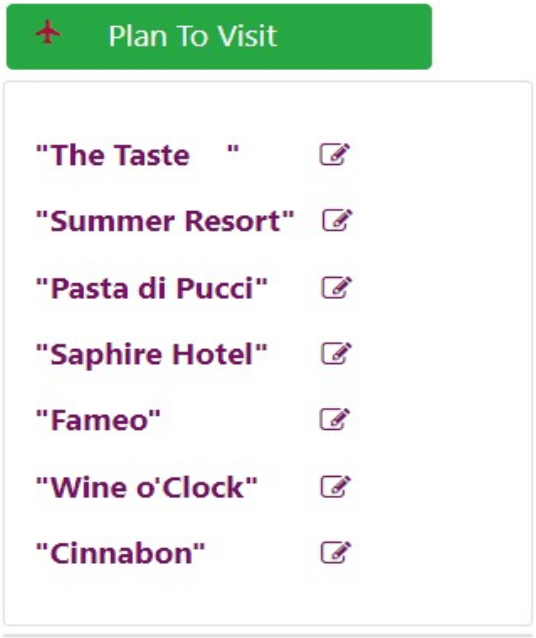
```
SELECT      U. user_first_name, C.checkin_date, C.checkin_like_numbers, R.review_desc,
            P.photoFile
FROM        CheckIn C, Review R, User U, Photo P
WHERE       userProfileID = U.userID and userProfileID = C.userID and C.reviewID =
            R.reviewID and P.checkinID = C.checkinID
```

Loading Friends Information

```
SELECT    U1.user_picture
FROM      User U1, (    SELECT F.userID2 as friendID
                        FROM User U, Friends F
                        WHERE userProfileID = U.userID and userProfileID =
                        F.userID1)
                        as friendList
WHERE     U1.userID = friendList.friendID

SELECT    COUNT(*)
FROM      User U, Friends F
WHERE     userProfileID = U.userID and userProfileID = F.userID1
```

5.11. PlanToVisit



Inputs: @username

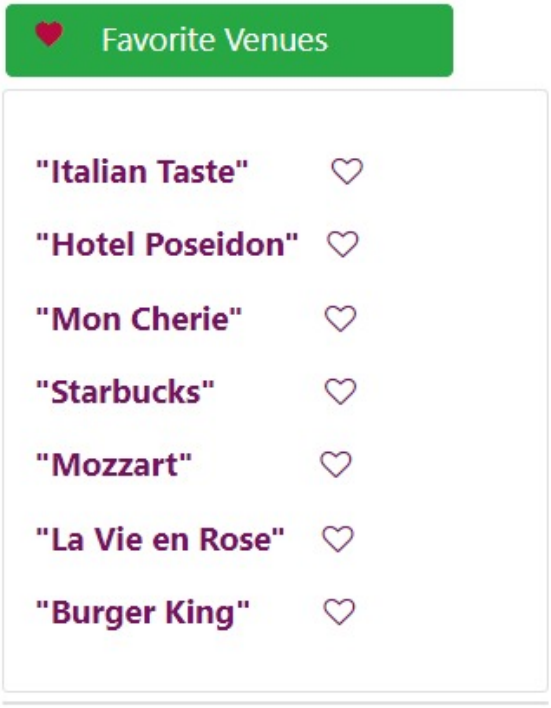
Process : When a user open the PlanToVisit screen, all the venues that the user has planned to visit in a near future will be displayed.

@username: Is the username of the user who is currently signed in.

SQL Statement:

```
SELECT PlanToVisit.venueID
FROM PlanToVisit
WHERE PlanToVisit.userID = @username
```

5.12. HasFavorite



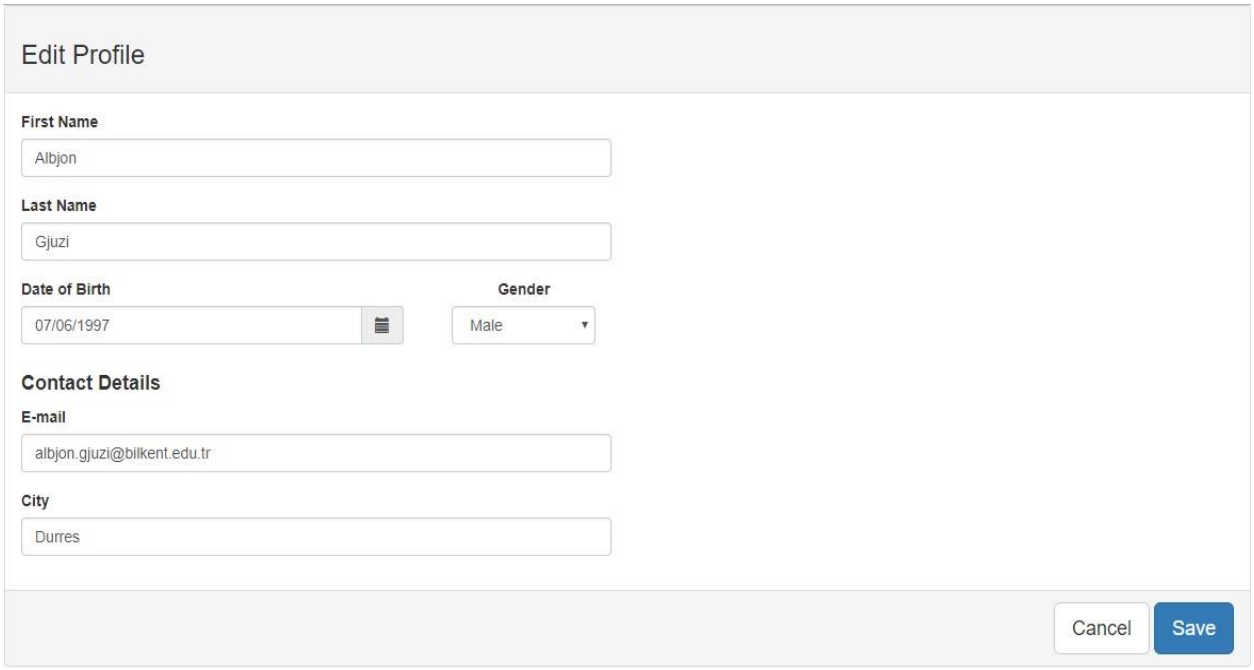
Inputs: @username

Process : When a user open the hasFavorite screen, all the favorite venues of the user will be displayed on the screen. User than can remove any of the them if he/she wants.

@username: Is the username of the user who is currently signed in.

SQL Statement:
SELECT HasFavorite.venueID
FROM HasFavorite
WHERE HasFavorite.userID = @username

5.13. Edit User Profile



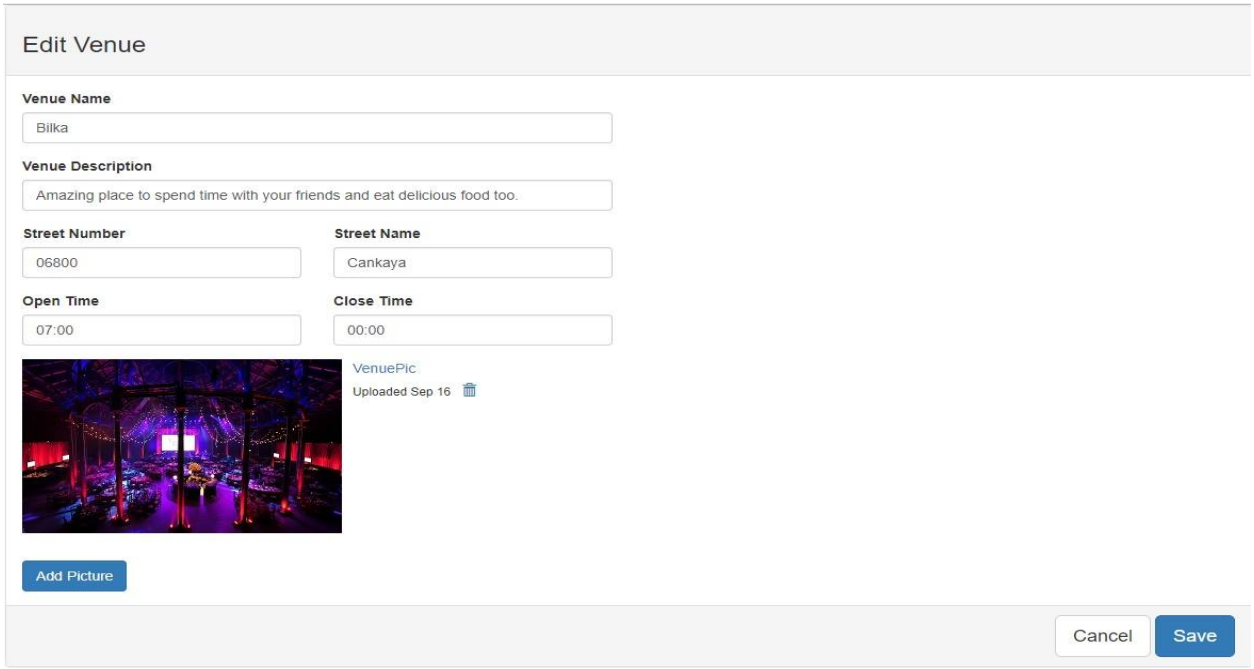
Inputs: @username, @firstname, @lastname, @birthday, @gender, @email, @city

Process : When a user open the Edit User Profile screen, the user will be available to change his basic information. He cannot edit the important information such as username, or others.

@username: Is the username of the user who is currently signed in.
@firstname: The new value of the first name
@lastname: The new value of the last name
@birthday: The new value corresponding to the birthday of the user.
@gender: Saves the new gender of the user.
@email: Save the new email of the user.
@city: Save the new city of the user.

SQL Statement:
UPDATE user_table
SET user_firstName = @firstname, user_lastname = @lastname,
user_birthday = @birthday, user_gender = @gender, user_email = @email,
city = @city
WHERE user_table.userID = @username

5.14. Edit Venue Profile



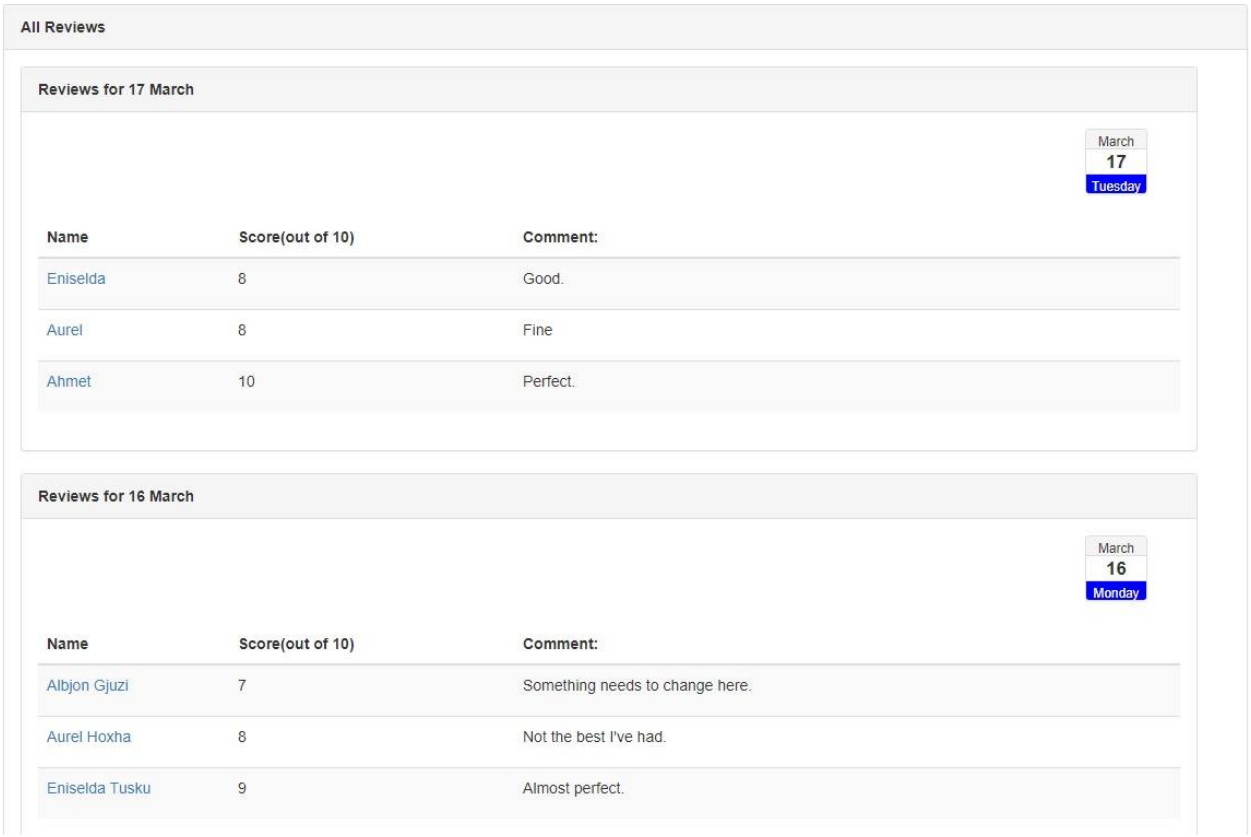
Inputs: @username, @venueID, @venueName, @venueDesc, @streetNumber, @streetName, @open_time, @close_time, @picture

Process : When a manager open Edit Profile of the Venue he will have the opportunity to change the information of the venue.
@username: Is the username of the user who is currently signed in.
@venueID: Is the ID if the venue that the manager is editing.
@venueName: The new value of the venue name
@venueDesc: The new value of the venue description
@streetNumber: The new value corresponding to the venue street number
@streetName: The new value corresponding to the venue street name
@open_time: Save the new open time of the venue.
@close_time: Save the new close time of the venue.
@picture: Save the new picture for the venue profile.

SQL Statement:

```
UPDATE venue
SET venueName = @venueName, venueDesc = @venueDesc,
street_number = @streetNumber, street_name = @streetName,
venue_open_time = @open_time, venue_close_time = @close_time,
venuePicutre = @ picture
WHERE venueID = @venueID
```

5.15. Users Rating To Venue



Inputs: @username, @venueID

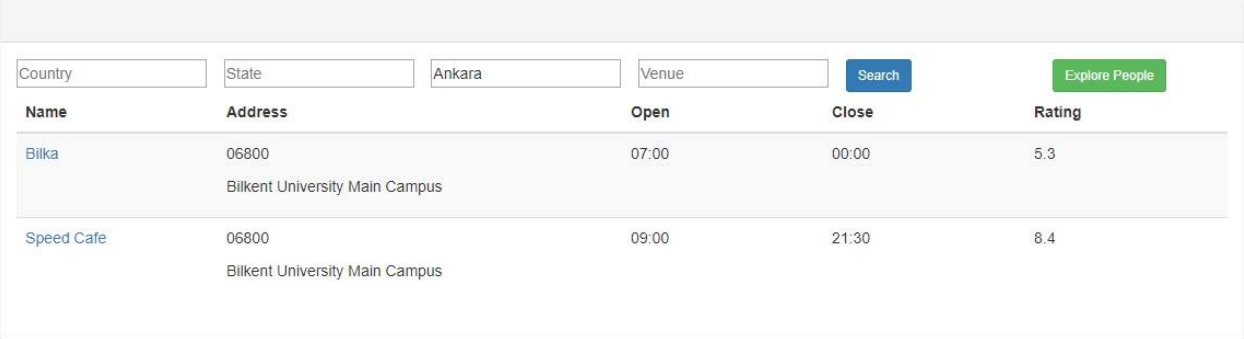
Process : When a manager checks the reviews that the users have written for that venue he has opened.

@username: Is the username of the user who is currently signed in.
@venueID: Is the ID if the venue that the manager has opened

SQL Statement:

```
SELECT userID, review_rating, review_desc
FROM checkin natural join review
WHERE venueID = @venueID
```

5.16. Search for a Venue



Inputs: @username, @countryName, @stateName, @cityName, @venueName

Process : The process of searching for a venue based on the keywords given in the boxes provided in the system.

@username: Is the username of the user who is currently signed in.
@countryName: The box to query entering the name of the country
@stateName: The box to query entering the name of the state
@cityName: The box to query entering the name of the city
@venueName: The box to query entering the name of the venue

SQL Statement:

```
SELECT venueName, venueDesc, venueAddress, venue_open_time,
        venue_close_time
FROM venue natural join city natural join state natural join country
WHERE (
            countryName likes @countryName or @countryName IS NULL
        )
AND     (
            stateName likes @stateName or @stateName IS NULL
        )
AND     (
            cityName likes @cityName or @cityName IS NULL
        )
AND     (
            venueName likes @venueName or @venueName IS NULL
        )
```

6. Advance Database Components

6.1. Views

Manager Suggestion View

This view restricts the manager to access user names that sent suggestions.

```
create view manager_suggestion as
select suggestionID, suggestion_text, suggestion_date, venueID
from suggestion
```

Manager Review View

This view restricts the manager to access user names that wrote the reviews.

```
create view manager_review as
select reviewRating, reviewDescription, checkin_date, venueID
from checkin natural join review
```

6.2. Stored Procedures

The most important operations on our system will be adding venues and check-ins at venues. Therefore, we can use some stored procedures to avoid using long queries all the time.

This procedure will be used to add check-ins to the database.

```
Create Procedure addCheckin
        (@checkinID int, @checkin_date date, @userID int, @venueID int, @reviewID
        int)
As
Begin
        Insert Into checkin
        Values (@checkinID, @checkin_date, @userID, @venueID, @reviewID)
End
```

This procedure will be used to add messages to the database.

```
Create Procedure addVenue
        (@userID1 int, @userID2 int, @message varchar(500), @sent_date date)
As
Begin
        Insert Into messages
        Values (@userID1, @userID2, @message, @sent_date)
End
```

This procedure will be used to add venues to the database.

```
Create Procedure addUser
        (@userID int, @user_firstName varchar(50), @user_lastName varchar(50), @user_email
        varchar(100), @user_password varchar(30), @user_birthdate date, @user_pic blob,
        @user_gender character(1), @city varchar(50), @user_profileType int, @user_created date,
        @user_isActive int, @user_lastlogin time, @typeID int)
As
Begin
        Insert Into user_table
        Values (@venueID, @venueName, @venueDesc, @street_number, @street_name,
                @venueCreated, @venueModified, @venue_open_time, @venue_close_time,
                @venueStatus int, @cityID, @managerID, @cityID, @managerID)
End
```

This procedure will be used to display the number of friends in user's profile

```
Create Procedure countFriends as
Begin
        (SELECT U.userID, count(*)
        FROM user_table U, friends F
        WHERE U.userID = F.userID1
        GROUP BY U.userID)
End
```

6.3. Profile Reports

Total number of check-ins uploaded by each user:

```
SELECT C.userID, count(*)
FROM checkin C
GROUP BY C.userID;
```

Total number of suggestions sent from each user:

```
SELECT S.userID, count(*)
FROM suggestion S
GROUP BY S.userID;
```

Total number of venues for each category

```
SELECT CV.categoryID, CV.categoryName, count(*)
FROM cat_venue CV
GROUP BY CV.categoryID, CV.categoryName;
```

Total number of venues for each city

```
SELECT CV.cityID, count(*)
FROM city_venue CV
GROUP BY CV.cityID;
```

Total number of planToVisit venues for each user
SELECT C.userID, count(*)
FROM PlanToVisit P
GROUP BY P.userID;

Total number of planToVisit venues from each user
SELECT P.userID, count(P.venueID)
FROM PlanToVisit P
GROUP BY P.userID;

Total number of planned visits for all venues
SELECT P.venueID, count(P.userID)
FROM PlanToVisit P
GROUP BY P.venueID

6.4. Triggers

- When a check-in is deleted from system, review, photos and comments related with this check-in are also deleted.
- When a new category is inserted the total number will be increased and when a category is deleted the total number will be decreased.
- When a user likes or dislikes a check-in the corresponding number of total likes will be updated and shown in the check-in.
- When a user becomes friend with someone else the number of the friends in his profile will increase and when he unfriends someone this number will decrease.
- When a user gives a rating together with the check-in it will affect the overall rating of the venue according to our algorithm.
- When a user adds a check-in, it is shown in the user's profile, venue's profile and his friend's news feed.
- When a user adds to his plan to visit list a particular venue it will automatically increment the total number of plannedToVisit venues in the profile of the user and when the user checks in at one of the venues in his plannedToVisit list it will automatically be removed from there.
- When a user deactivates the account all the information such as suggestions, comments, likes, check-ins, messages will be removed from the system.
- When a manager deletes a venue all the information regarding that venue such as venue profile, check-ins, suggestions etc. will be removed from the system.
- When a manger changes the status of the venue to offline, no other user besides the manager can view the venue's profile and the check-ins done there.
- When a user sends a message, the message count in his friends profile and it will remain like that until the friend responds.
- When a user makes a review about a venue, it will increment the number of reviews this user has made which will be later used in the algorithm about the venue rating.

6.5. Constraints

- To enter the system a registration is required.
- The system cannot be accessed without logging in.
- All IDs of the system cannot be null.
- Users can only log in using their username and password.
- A user can check-in multiple times in a venue, but each check-in has only review.
- There cannot be two same name categories.
- A venue cannot have two same name features.
- The suggestions are seen only from manager of the venue.
- Users can only see profiles of their friends or of people whose profile is public.

- Users can filter the data based on the available resources such as country, state, city and category.
- Users can edit only their profiles.
- Users cannot remove comments of their friends from their check-ins.
- A venue cannot be managed by more than one manager.
- Deactivating the account and registering again with the same username and email will not preserve the old information.

7. Implementation Plan

For the implementation plan we are planning to use MySQL Server at data layer in our project as database management system. Furthermore, for the logic and user interface of the project we are planning to code it in PHP and a use a small amount of JavaScript or Node.js. The core of our system will be using MySQL and phpMyAdmin.