



CS 353 - Database Systems

## **Hotel Database Management System Design Report**

Group 38

Maryam Shahid	21801344
Bora Çün	21802838
Çağrı Eren	21801831
Hammad Khan Musakhel	21801175

# Contents

<b>1. Revised E/R Diagram</b>	<b>4</b>
<b>2. Table Schemas</b>	<b>7</b>
2.1. Users	7
2.2. Employee	8
2.3. Manager	9
2.4. Security-Staff	10
2.5. Housekeeper	101
2.6. Recruiter	112
2.7. Receptionist	123
2.8. Candidate	134
2.9. Guests	145
2.10. Room	156
2.11. Room-type	167
2.12. Reservation	178
2.13. Comment	189
2.14. QnA	20
2.15. Building	201
2.16. Security-walk	212
2.17. Cleaning-Duty	223
2.18. Location	234
2.19. Event	245
2.20. Guest-Activity	26
2.21. Activity	267
2.22. Guest-Tours	278
2.23. Training-Program	289
2.24. Ticket	30
2.25. Tourist-Attraction	301
2.26. Places-to-visit	312
2.27. Sec-staff-applies-to	323
2.28. Evaluates-sec-staff-application	334
2.29. HK-applies-to	345
2.30. Evaluates-hk-application	356
2.31. Food-Drink	367
2.32. Order-contains	378
2.33. Food-Order	389
2.34. Restaurant	40
2.35. Job-Application	401
2.36. Approves	412

2.37. Security-Report	423
2.38. Leave-Request-Form	434
<b>3. User Interface Design and Corresponding SQL Statements</b>	<b>45</b>
3.1. Homepage	45
3.2. Guest - Login	46
3.3. Guest - Signup	47
3.4. Guest - Homepage/ Reservation	48
3.5. Guest - Booking	49
3.6. Guest - Order Food	50
3.7. Guest - Order History	51
3.8. Employee - Login	52
3.9. Manager - Homepage	53
3.10. Manager - Assign Orders	54
3.11. Manager - Evaluate Training Application	55
3.12. Housekeeper - Orders	56
3.13. Housekeeper - Training Programs	57
3.14. Housekeeper - Current Applications	58
3.15. Candidate Login	59
3.16. Candidate Application	60
<b>5. Implementation Plan</b>	<b>61</b>
<b>6. Web Page</b>	<b>61</b>

# 1. Revised E/R Diagram

The E/R diagram from the Proposal Document, shown in Figure 1.1, has been improved according to feedback. The revised diagram is shown in Figure 1.2 below. The changes that leads to the revised diagram are:

- All the derived attributes are removed.
- Unnecessary discriminators in the weak entity sets are changed to regular attributes.
- Ternary relationship “Applies-to”, which allowed the employees to apply for training programs and managers to approve the applications, has been rearranged such that there are two separate aggregations: one for security staff, one for housekeepers. This way, the manager id’s do not become temporarily null and only the security staff and housekeepers can apply to training programs.
- Housekeeper cardinality is now 1 in “assigned-delivery” relationship.
- Building participation is now total in “Security-walk” relationship.
- Room participation is now total in “Cleaning-Duty” relationship.
- "totals" in Employee, Guest Activity, and Users are removed.
- Job-Application is now an entity.
- “Ordered-from” relationship is removed.
- Reservation is now an entity.
- Comment is now an entity.
- Ticket is now an entity.

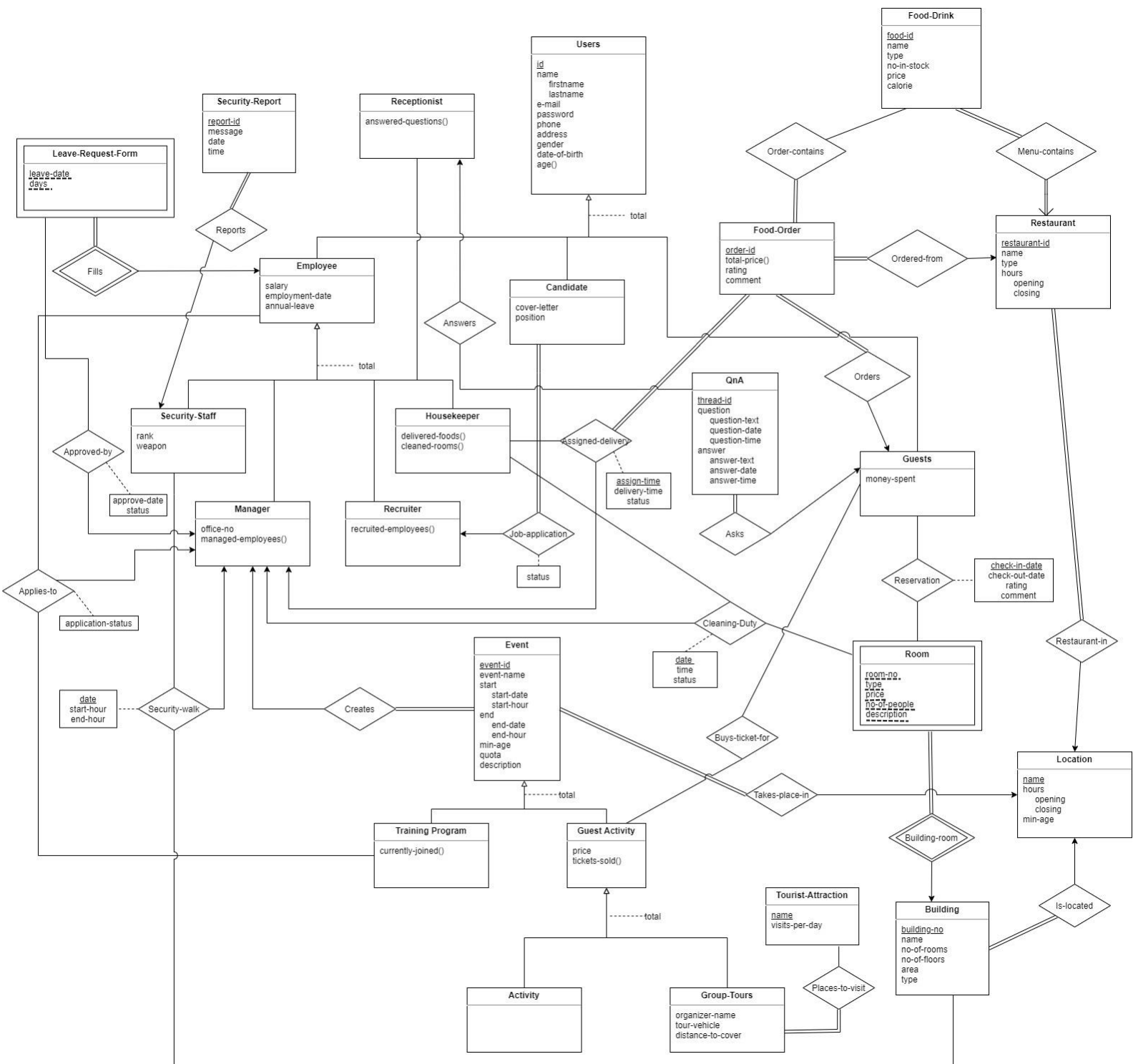


Figure 1.1: E/R Diagram from the Proposal Document



## 2. Table Schemas

### 2.1. Users

Users(id, firstname, lastname, email, password, phone, address, gender, date-of-birth)

Functional Dependencies:

id -> firstname, lastname, email, password, phone, address, gender, date-of-birth

Candidate Keys:

{id}

{email}

Primary Key:

{id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Users(  
    id INT,  
    firstname VARCHAR(50) NOT NULL,  
    lastname VARCHAR(50) NOT NULL,  
    email VARCHAR(75) NOT NULL UNIQUE,  
    password VARCHAR(20) NOT NULL,  
    phone VARCHAR(20),  
    address VARCHAR(100),  
    gender VARCHAR(20),  
    date-of-birth DATE  
    PRIMARY KEY (id)  
    CHECK (date-of-birth < CAST(GETDATE() AS Date) AND id > 0 ) );
```

## 2.2. Employee

Employee(id, salary, employment-date, annual-leave )

Functional Dependencies:

id -> salary, employment-date, annual-leave

Candidate Keys:

{id}

Primary Key:

{id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Employee(  
    id INT,  
    salary NUMERIC(18, 2) NOT NULL,  
    employment-date DATE NOT NULL,  
    annual-leave INT NOT NULL,  
    PRIMARY KEY (id)  
    CHECK (salary >= 0 AND id > 0 AND annual-leave > 0) );
```



### 2.3. Manager

Manager(id, office-no )

Functional Dependencies:

id -> office-no

Candidate Keys:

{id}

Primary Key:

{id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Manager(  
    id INT,  
    office-no VARCHAR(10) NOT NULL,  
    PRIMARY KEY (id),  
    FOREIGN KEY (id) REFERENCES Employee(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (id > 0) );
```

## 2.4. Security-Staff

Security-staff(id, rank, weapon )

Functional Dependencies:

id -> rank, weapon

Candidate Keys:

{id}

Primary Key:

{id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Security-Staff(  
    id INT,  
    rank VARCHAR(20) NOT NULL,  
    weapon VARCHAR(20),  
    PRIMARY KEY (id)  
    FOREIGN KEY (id) REFERENCES Employee(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (id > 0) );
```

## 2.5. Housekeeper

Housekeeper(id)

Candidate Keys:

{ id }

Primary Key:

{ id }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Housekeeper(  
    id INT,  
    PRIMARY KEY (id),  
    FOREIGN KEY (id) REFERENCES Employee(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (id > 0));
```

## 2.6. Recruiter

Recruiter(id)

Candidate Keys:

{ id }

Primary Key:

{ id }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Recruiter(  
    id INT,  
    PRIMARY KEY (id),  
    FOREIGN KEY (id) REFERENCES Employee(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (id > 0));
```

## 2.7. Receptionist

Receptionist(id)

Candidate Keys:

{ id }

Primary Key:

{ id }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Receptionist(  
    id INT,  
    PRIMARY KEY (id),  
    FOREIGN KEY (id) REFERENCES Employee(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (id > 0));
```

## 2.8. Candidate

Candidate( id, cover-letter )

Functional Dependencies:

id -> cover-letter

Candidate Keys:

{id}

Primary Key:

{id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Candidate(  
    id INT,  
    cover-letter VARCHAR(5000) NOT NULL,  
    PRIMARY KEY (id),  
    FOREIGN KEY (id) REFERENCES Users(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (id > 0));
```

## 2.9. Guests

Guests(id, money-spent )

Functional Dependencies:

id -> money-spent

Candidate Keys:

{id}

Primary Key:

{id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Guests(  
    id INT,  
    money-spent NUMERIC(18, 2) NOT NULL,  
    PRIMARY KEY (id),  
    FOREIGN KEY (id) REFERENCES Users(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (id > 0 AND money-spent > 0) );
```

## 2.10. Room

Room(room-no, building-no, type, description)

Functional Dependencies:

room-no, building-no -> type, description

Candidate Keys:

{room-no, building-no}

Primary Key:

{room-no, building-no}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Room (  
    room-no INT,  
    building-no VARCHAR(50),  
    type VARCHAR(50) NOT NULL,  
    description VARCHAR(5000)  
    PRIMARY KEY (room-no, building-no),  
    FOREIGN KEY (type) REFERENCES Room-type(type)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (building-no) REFERENCES Building(building-no)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (room-no > 0 AND LEN(building-no) > 0 AND LEN(type) > 0) );
```



## 2.11. Room-type

Room-type(type, price, no-of-people)

Functional Dependencies:

type->price, no-of-people

Candidate Keys:

{type}

Primary Key:

{type}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Room-type (  
    type VARCHAR(50),  
    price NUMERIC(18, 2) NOT NULL,  
    no-of-people INT NOT NULL,  
    PRIMARY KEY (type)  
    CHECK ( LEN(type) > 0) );
```

## 2.12. Reservation

Reservation(reservation-id, guest-id, room-no, building-no, check-in-date, check-out-date )

Functional Dependencies:

reservation-id -> guest-id, check-in-date, check-out-date

Candidate Keys:

{reservation-id}

Primary Key:

{reservation-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Reservation(  
    reservation-id INT,  
    guest-id INT NOT NULL,  
    room-no INT,  
    building-no VARCHAR(50),  
    check-in-date DATE NOT NULL,  
    check-out-date DATE NOT NULL,  
    PRIMARY KEY (reservation-id),  
    FOREIGN KEY (guest-id) REFERENCES Guests(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (room-no,building-no) REFERENCES Room(room-no, building-no)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (reservation-id > 0 AND guest-id > 0 AND check-in-date < check-out-date  
AND room-no > 0 AND LEN(building-no) > 0) );
```

## 2.13. Comment

Comment(comment-id, reservation-id, text, date, topic )

Functional Dependencies:

comment-id -> text, reservation-id, date, topic

Candidate Keys:

{comment-id}

Primary Key:

{comment-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Comment(  
    comment-id INT,  
    reservation-id INT NOT NULL,  
    text VARCHAR(5000) NOT NULL,  
    date DATE NOT NULL,  
    topic VARCHAR(50),  
    PRIMARY KEY (comment-id),  
    FOREIGN KEY (reservation-id) REFERENCES Reservation(reservation-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (comment-id > 0 AND reservation-id > 0) );
```

## 2.14. QnA

QnA(thread-id, guest-id, question-text, question-date, question-time, answer-text, answer-date, answer-time )

Functional Dependencies:

thread-id -> guest-id, receptionist-id, question-text, question-date, question-time, answer-text, answer-date, answer-time

Candidate Keys:

{thread-id}

Primary Key:

{thread-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE QnA(  
    thread-id INT,  
    guest-id INT NOT NULL,  
    receptionist-id INT,  
    question-text VARCHAR(5000) NOT NULL,  
    question-date DATE NOT NULL,  
    question-time TIME NOT NULL,  
    answer-text VARCHAR(5000),  
    answer-date DATE,  
    answer-time TIME,  
    PRIMARY KEY (thread-id),  
    FOREIGN KEY (guest-id) REFERENCES Guests(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (receptionist-id) REFERENCES Receptionist(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (thread-id > 0 AND guest-id > 0 AND receptionist-id > 0) );
```

## 2.15. Building

Building(building-no, location-name, name, no-of-rooms, no-of-floors, area,type )

Functional Dependencies:

building-no -> location-name, name, no-of-rooms, no-of-floors, area,type

Candidate Keys:

{building-no}

Primary Key:

{building-no}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Building(  
    building-no VARCHAR(50),  
    location-name VARCHAR(50) NOT NULL,  
    name VARCHAR(50) NOT NULL,  
    no-of-rooms INT NOT NULL,  
    no-of-floors INT NOT NULL,  
    area INT NOT NULL,  
    type VARCHAR(50) NOT NULL,  
    PRIMARY KEY (building-no),  
    FOREIGN KEY (location-name) REFERENCES Location(name)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (no-of-rooms > 0 AND no-of-floors > 0 AND area > 0 AND LEN(building-  
no) > 0 AND LEN(location-name) > 0) );
```

## 2.16. Security-walk

Security-walk(manager-id,security-staff-id, building-no, date, start-hour, end-hour)

Candidate Keys:

{ manager-id,security-staff-id, building-no, date, start-hour, end-hour }

Primary Key:

{ manager-id,security-staff-id, building-no, date, start-hour, end-hour }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Security-walk(  
    manager-id INT,  
    security-staff-id INT,  
    building-no VARCHAR(50),  
    date DATE,  
    start-hour TIME,  
    end-hour TIME,  
    PRIMARY KEY (manager-id,security-staff-id, building-no, date, start-hour, end-  
hour),  
    FOREIGN KEY (manager-id) REFERENCES Manager(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (security-staff-id) REFERENCES Security-staff(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (building-no) REFERENCES Building(building-no)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (manager-id > 0 AND security-staff-id > 0 AND LEN(building-no) > 0));
```

## 2.17. Cleaning-Duty

Cleaning-Duty(housekeeper-id, room-no, building-no, date, time, status, manager-id)

Candidate Keys:

{housekeeper-id, room-no, building-no, date }

Primary Key:

{housekeeper-id, room-no, building-no, date }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Cleaning-Duty(  
    housekeeper-id INT,  
    room-no INT,  
    building-no VARCHAR(50),  
    date DATE,  
    time TIME,  
    status VARCHAR(20) NOT NULL,  
    manager-id INT NOT NULL,  
    PRIMARY KEY (housekeeper-id, room-no, building-no, date),  
    FOREIGN KEY (housekeeper-id) REFERENCES Housekeeper(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (room-no) REFERENCES Room(room-no)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (building-no) REFERENCES Building(building-no)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (manager-id) REFERENCES Manager(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (housekeeper-id > 0 AND room-no > 0 AND LEN(building-no) > 0 AND  
        manager-id > 0 AND CAST(GETDATE() AS Date ) <= date AND  
        CAST(GETDATE() AS Time ) <= time ));
```

## 2.18. Location

Location(name, opening, closing, min-age )

Functional Dependencies:

name -> opening, closing, min-age

Candidate Keys:

{name}

Primary Key:

{name}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Location(  
    name VARCHAR(50),  
    opening TIME,  
    closing TIME,  
    min-age INT,  
    PRIMARY KEY (name),  
    CHECK ( LEN(name) > 0 AND opening < closing AND min-age > 0 ) );
```



## 2.19. Event

Event(event-id, event-name, location-name, start-date, start-hour, end-date, end-hour, min-age, quota, description, manager-id)

Functional Dependencies:

event-id -> event-name, location-name, start-date, start-hour, end-date, end-hour, min-age, quota, description, manager-id

Candidate Keys:

{event-id}

Primary Key:

{event-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Event(  
    event-id INT,  
    event-name VARCHAR(50) NOT NULL,  
    location-name VARCHAR(50) NOT NULL,  
    start-date DATE NOT NULL,  
    start-hour TIME NOT NULL,  
    end-date DATE NOT NULL,  
    end-hour TIME NOT NULL,  
    min-age INT,  
    quota INT,  
    description VARCHAR(3000),  
    manager-id INT NOT NULL,  
    PRIMARY KEY (event-id),  
    FOREIGN KEY (location-name) REFERENCES Location(name)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (manager-id) REFERENCES Manager(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK((event-id > 0) AND ((start-date < end-date) OR (start-date = end-date AND  
        start-hour < end-hour))));
```

## 2.20. Guest-Activity

Guest-Activity(event-id, price)

Functional Dependencies:

event-id -> price

Candidate Keys:

{event-id}

Primary Key:

{event-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Guest-Activity(  
    event-id INT,  
    price NUMERIC (18, 2) NOT NULL,  
    PRIMARY KEY (event-id),  
    FOREIGN KEY (event-id) REFERENCES Event(event-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (event-id > 0 AND price >= 0.0));
```

## 2.21. Activity

Activity(event-id)

Candidate Keys:

{event-id}

Primary Key:

{event-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Activity(  
    event-id INT,  
    PRIMARY KEY (event-id),  
    FOREIGN KEY (event-id) REFERENCES Event(event-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (event-id > 0));
```

## 2.22. Guest-Tours

Group-Tours(event-id, organizer-name, tour-vehicle, distance-to-cover)

Functional-Dependencies:

event-id -> organizer-name, tour-vehicle, distance-to-cover

Candidate Keys:

{event-id}

Primary Key:

{event-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Group-Tours(  
    event-id INT,  
    organizer-name VARCHAR(50) NOT NULL,  
    tour-vehicle VARCHAR(20),  
    distance-to-cover INT NOT NULL,  
    PRIMARY KEY (event-id),  
    FOREIGN KEY (event-id) REFERENCES Guest-Activity(event-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (event-id > 0 AND distance-to-cover > 0));
```

## 2.23. Training-Program

Training-Program(event-id)

Candidate Keys:

{event-id}

Primary Key:

{event-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Training-Program(  
    event-id INT,  
    PRIMARY KEY (event-id),  
    FOREIGN KEY (event-id) REFERENCES Event(event-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (event-id > 0));
```

## 2.24. Ticket

Ticket(ticket-id, event-id, guest-id)

Functional Dependencies:

ticket-id -> event-id, guest-id

Candidate Keys:

{ticket-id}

Primary Key:

{ticket-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Ticket(  
    ticket-id INT,  
    event-id INT NOT NULL,  
    guest-id INT NOT NULL,  
    PRIMARY KEY(ticket-id),  
    FOREIGN KEY event-id REFERENCES Guest-Activity(event-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY guest-id REFERENCES Guests(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (ticket-id > 0 AND event-id > 0 AND guest-id > 0));
```

## 2.25. Tourist-Attraction

Tourist-Attraction(name, visits-per-day)

Functional Dependencies:

name -> visits-per-day

Candidate Keys:

{name}

Primary Key:

{name}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Tourist-Attraction(  
    name VARCHAR(50),  
    visits-per-day INT NOT NULL,  
    PRIMARY KEY (name),  
    CHECK (visits-per-day >= 0));
```

## 2.26. Places-to-visit

Places-to-visit(event-id, attraction-name)

Candidate Keys:

{event-id, attraction-name}

Primary Key:

{event-id, attraction-name}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Places-to-visit(  
    event-id INT,  
    attraction-name VARCHAR(50),  
    PRIMARY KEY (event-id, attraction-name),  
    FOREIGN KEY (event-id) REFERENCES Group-Tours(event-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (attraction-name) REFERENCES Tourist-Attraction(name)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (event-id > 0));
```



## 2.27. Sec-staff-applies-to

Sec-staff-applies-to(sec-staff-id, training-program-id)

Candidate Keys:

{sec-staff-id, training-program-id}

Primary Key:

{sec-staff-id, training-program-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Sec-staff-applies-to(  
    sec-staff-id INT,  
    training-program-id INT,  
    PRIMARY KEY (sec-staff-id, training-program-id),  
    FOREIGN KEY (training-program-id) REFERENCES Training Program(event-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (sec-staff-id) REFERENCES Security-Staff(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (training-program-id > 0 AND sec-staff-id > 0));
```

## 2.28. Evaluates-sec-staff-application

Evaluates-sec-staff-application(sec-staff-id, training-program-id, manager-id, application-status)

Candidate Keys:

{sec-staff-id, training-program-id}

Primary Key:

{sec-staff-id, training-program-id}

Functional Dependencies:

sec-staff-id, training-program-id -> manager-id, application-status

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Evaluates-sec-staff-application(  
    sec-staff-id INT,  
    training-program-id INT,  
    manager-id INT,  
    application-status VARCHAR(50),  
    PRIMARY KEY (sec-staff-id, training-program-id),  
    FOREIGN KEY (training-program-id) REFERENCES Training Program(event-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (sec-staff-id) REFERENCES Security-Staff(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (manager-id) REFERENCES Manager(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (training-program-id > 0 AND sec-staff-id > 0 AND manager-id > 0));
```

## 2.29. HK-applies-to

HK-applies-to(housekeeper-id, training-program-id)

Candidate Keys:

{housekeeper-id, training-program-id}

Primary Key:

{housekeeper-id, training-program-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE HK-applies-to(  
    housekeeper-id INT,  
    training-program-id INT,  
    manager-id INT,  
    PRIMARY KEY (sec-staff-id, training-program-id, manager-id),  
    FOREIGN KEY (training-program-id) REFERENCES Training Program(event-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (housekeeper-id) REFERENCES Housekeeper(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (training-program-id > 0 AND housekeeper-id > 0));
```

## 2.30. Evaluates-hk-application

Evaluates-hk-application(housekeeper-id, training-program-id, manager-id, application-status)

Functional Dependencies:

housekeeper-id, training-program-id -> manager-id, application-status

Candidate Keys:

{housekeeper-id, training-program-id}

Primary Key:

{housekeeper-id, training-program-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Evaluates-HK-application(  
    housekeeper-id INT,  
    training-program-id INT,  
    manager-id INT,  
    application-status VARCHAR(50),  
    PRIMARY KEY (housekeeper-id, training-program-id),  
    FOREIGN KEY (training-program-id) REFERENCES Training Program(event-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (housekeeper-id) REFERENCES Housekeeper(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (manager-id) REFERENCES Manager(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (housekeeper-id > 0 AND training-program-id > 0 AND manager-id > 0));
```

## 2.31. Food-Drink

Food-Drink(food-id, name, type, no-in-stock, price, calorie, restaurant-id)

Functional Dependencies:

food-id -> name, type, no-in-stock, price, calorie, restaurant-id

Candidate Keys:

{ food-id }

Primary Key:

{ food-id }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Food-Drink(  
    food-id INT,  
    name VARCHAR(50) NOT NULL,  
    type VARCHAR(20),  
    no-in-stock INT NOT NULL,  
    price NUMERIC(18, 2) NOT NULL,  
    calorie INT,  
    restaurant-id INT NOT NULL,  
    PRIMARY KEY (food-id),  
    FOREIGN KEY (restaurant-id) REFERENCES Restaurant(restaurant-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (food-id > 0 AND no-in-stock >= 0 AND price >= 0 AND restaurant-  
        id > 0));
```

## 2.32. Order-contains

Order-contains(order-id, food-id)

Candidate Keys:

{order-id, food-id}

Primary Key:

{order-id, food-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Order-contains(  
    order-id INT,  
    food-id INT,  
    PRIMARY KEY (order-id, food-id),  
    FOREIGN KEY (order-id) REFERENCES Food-Order(order-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (food-id) REFERENCES Food-Drink(food-id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (order-id > 0 AND food-id > 0));
```

### 2.33. Food-Order

Food-Order(order-id, rating, comment, guest-id, manager-id, housekeeper-id, assign-time, delivery-time, status)

Functional Dependencies:

order-id -> rating, comment, guest-id, manager-id, housekeeper-id, assign-time, delivery-time, status

Candidate Keys:

{order-id}

Primary Key:

{order-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Food-Order(  
    order-id INT,  
    rating INT,  
    comment VARCHAR(1000),  
    guest-id INT NOT NULL,  
    manager-id INT,  
    housekeeper-id INT,  
    assign-time TIMESTAMP NOT NULL,  
    delivery-time TIMESTAMP,  
    status VARCHAR(20),  
    PRIMARY KEY (order-id),  
    FOREIGN KEY (guest-id) REFERENCES Guests(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (manager-id) REFERENCES Manager(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (housekeeper-id) REFERENCES Housekeeper(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (order-id > 0 AND guest-id > 0 AND assign-time <=  
        CURRENT_TIMESTAMP()));
```

## 2.34. Restaurant

Restaurant(restaurant-id, location-name, name, type, hours-opening, hours-closing)

Functional Dependencies:

restaurant-id -> name, location-name, type, hours-opening, hours-closing

Candidate Keys:

{restaurant-id}

Primary Key:

{restaurant-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Restaurant(  
    restaurant-id INT,  
    location-name VARCHAR(50) NOT NULL,  
    name VARCHAR(50) NOT NULL,  
    type VARCHAR(20),  
    hours-opening TIME,  
    hours-closing TIME,  
    PRIMARY KEY (restaurant-id),  
    FOREIGN KEY (location-name) REFERENCES Location(name)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (restaurant-id > 0 AND LEN(location-name) > 0));
```



### 2.35. Job-Application

Job-Application(id, position, status)

Functional Dependencies:

id, position -> status

Candidate Keys:

{id, position}

Primary Key:

{id, position}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Job-Application (  
    id INT,  
    position VARCHAR(50),  
    status VARCHAR(20) NOT NULL,  
    PRIMARY KEY (id, position),  
    FOREIGN KEY (id) REFERENCES Candidate(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (id > 0));
```

## 2.36. Approves

Approves(candidate-id, position, recruiter-id)

Candidate Keys:

{candidate-id, position, recruiter-id}

Primary Key:

{candidate-id, position, recruiter-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Approves(  
    candidate-id INT,  
    position VARCHAR(50),  
    recruiter-id INT,  
    PRIMARY KEY (candidate-id, position, recruiter-id),  
    FOREIGN KEY (candidate-id) REFERENCES Job-Application(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (position) REFERENCES Job-Application(position)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (recruiter-id) REFERENCES Recruiter(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (candidate-id > 0 AND recruiter-id > 0));
```

## 2.37. Security-Report

Security-Report(report-id, message, date, time, security-staff-id)

Functional Dependencies:

report-id -> message, date, time, security-staff-id

Candidate Keys:

{report-id}

Primary Key:

{report-id}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Security-Report(  
    report-id INT,  
    message VARCHAR(1000) NOT NULL,  
    date DATE NOT NULL,  
    time TIME NOT NULL,  
    security-staff-id INT NOT NULL,  
    PRIMARY KEY (report-id),  
    FOREIGN KEY (security-staff-id) REFERENCES Security-Staff(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (report-id > 0 AND security-staff-id > 0));
```

## 2.38. Leave-Request-Form

Leave-Request-Form(id, leave-date, days, manager-id, approve-date, status)

Functional Dependencies:

id, leave-date -> days, manager-id, approve-date, status

Candidate Keys:

{id, leave-date}

Primary Key:

{id, leave-date}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE Leave-Request-Form(  
    id INT,  
    leave-date DATE,  
    days INT NOT NULL,  
    manager-id INT,  
    approve-date DATE,  
    status VARCHAR(20) NOT NULL,  
    PRIMARY KEY (id, leave-date),  
    FOREIGN KEY (id) REFERENCES Employee(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    FOREIGN KEY (manager-id) REFERENCES Manager(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE,  
    CHECK (leave-date >= CAST(GETDATE() AS Date) AND id > 0 AND days > 0));
```

### 3. User Interface Design and Corresponding SQL Statements

#### 3.1. Homepage

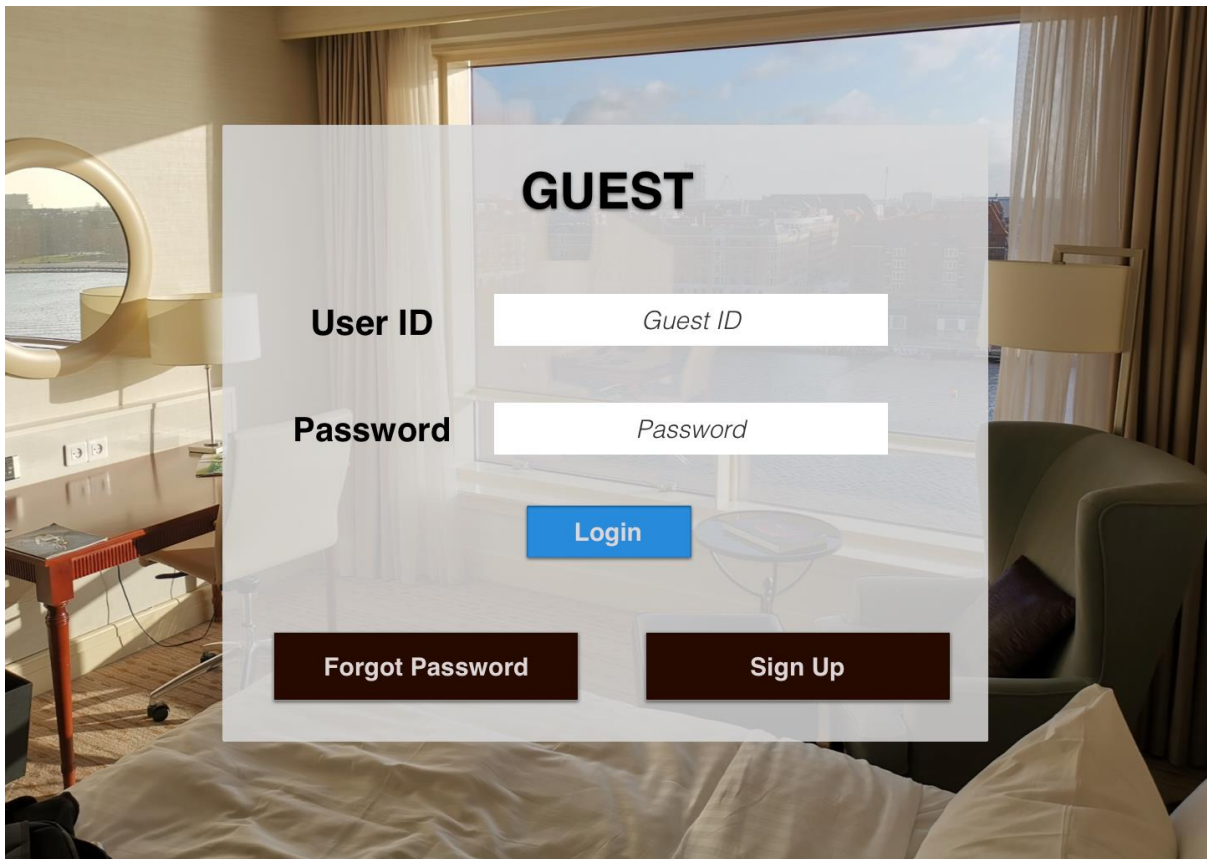


Input: N/A

Process: This is the main menu from where the different users choose their specified category to go with. Each category is going to take the user to different screens.

SQL: N/A

### 3.2. Guest - Login



Inputs: @user\_ID, @password

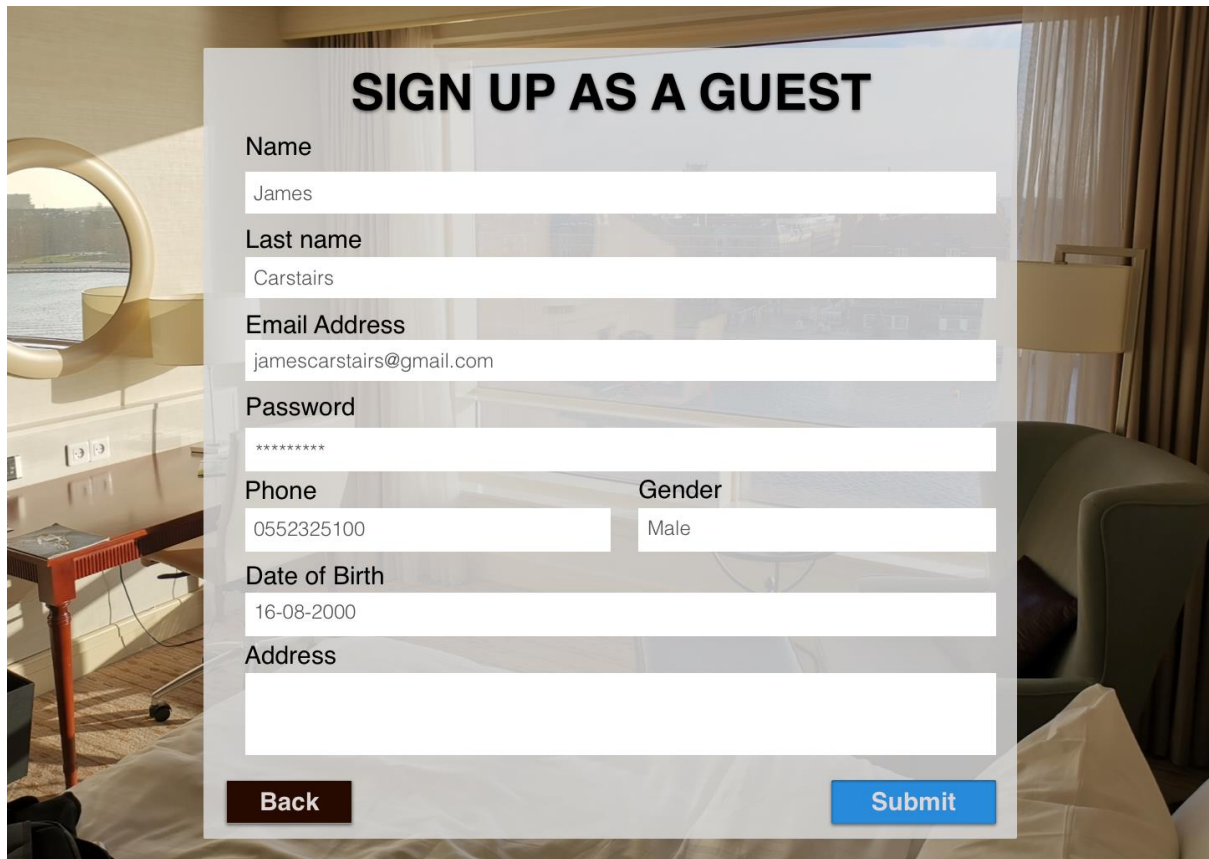
Process: After selecting the GUEST option from the main menu, the guest has to enter his/her guest ID and password in order to proceed. One can also go for sign up that will create a new user entry into the guest table. The user can also use forget password to reset his/her password.

SQL Statements:

#### Logging in

```
SELECT *  
FROM guest  
WHERE @user_ID = id AND @password = password;
```

### 3.3. Guest - Signup



**SIGN UP AS A GUEST**

Name  
James

Last name  
Carstairs

Email Address  
jamescarstairs@gmail.com

Password  
\*\*\*\*\*

Phone  
0552325100

Gender  
Male

Date of Birth  
16-08-2000

Address

**Back** **Submit**

Input: @name, @lastname, @email\_address, @password, @phone, @gender, @date\_of\_birth, @address

Process: The user will enter the specified information required to meet the criteria of the guest entity.

SQL:

**Sign Up:**

```
INSERT INTO guest
VALUES(@name, @lastname, @email_address, @password, @phone, @gender,
@data_of_birth, @address);
```



### 3.4. Guest - Homepage/ Reservation

The screenshot shows a web application interface for a hotel database management system. The top navigation bar includes links for 'Reservation' (highlighted in red), 'Order Food', 'Order History', 'Comment', 'QnA', and 'Logout'. The main content area is divided into two panels. The left panel, titled 'Contact Information', displays guest details: Guest ID: 21801344, Name: James Carstairs, Phone: 05523285100, Address: 100 W College St, Granville, OH 43023, Date of birth: 16/08/2000, and Membership: Gold. An 'Update Information' button is at the bottom. The right panel, titled 'Welcome, James Carstairs', shows booking details: Reservation Status: Booked, Check-in date: 29/03/2021, Room: 104, Double, Check-out date: 05/04/2021, and Room Location: East Wing, B2. It includes buttons for 'Cancel', 'Extend Stay', and 'New Reservation'. Below this is a 'Registered Activities' section with a table listing activities and their prices, and buttons for 'Search Activities', 'Buy Tickets', and 'Payment History'.

Name	Time and Date	Price	Cancel
Pool Party	20:00 - 30/03/2021	350 TL	X
Group Tour - Museum	13:00 - 01/04/2021	265 TL	X

Input: N/A

Process: The reservation part of the guest screen leads the guest user to a screen where it is customised according to the user's selections of previous reservations. The guest user can further book a room only after cancelling the existing one though. The user can register for new activities.

SQL:

#### Bookings

```
SELECT reservation status, room, room location, check_in, check_out
FROM reservation JOIN guest
WHERE user_ID = id;
```

#### Registered activities

```
SELECT *
FROM guest NATURAL JOIN buys-ticket-for NATURAL JOIN Event;
```



### 3.5. Guest - Booking

Hotel Database Management System

Reservation Order Food Order History Comment QnA Logout

**Book a Room**

Room type: Suite

Check-in date:

Check-out date:

**Suite**

Two king sized beds

One multipurpose kitchen

City View + Sea View

Premium Location

Choose Room

Input: N/A

Process: The guest will specify the reservation details in this page by entering the necessary information.

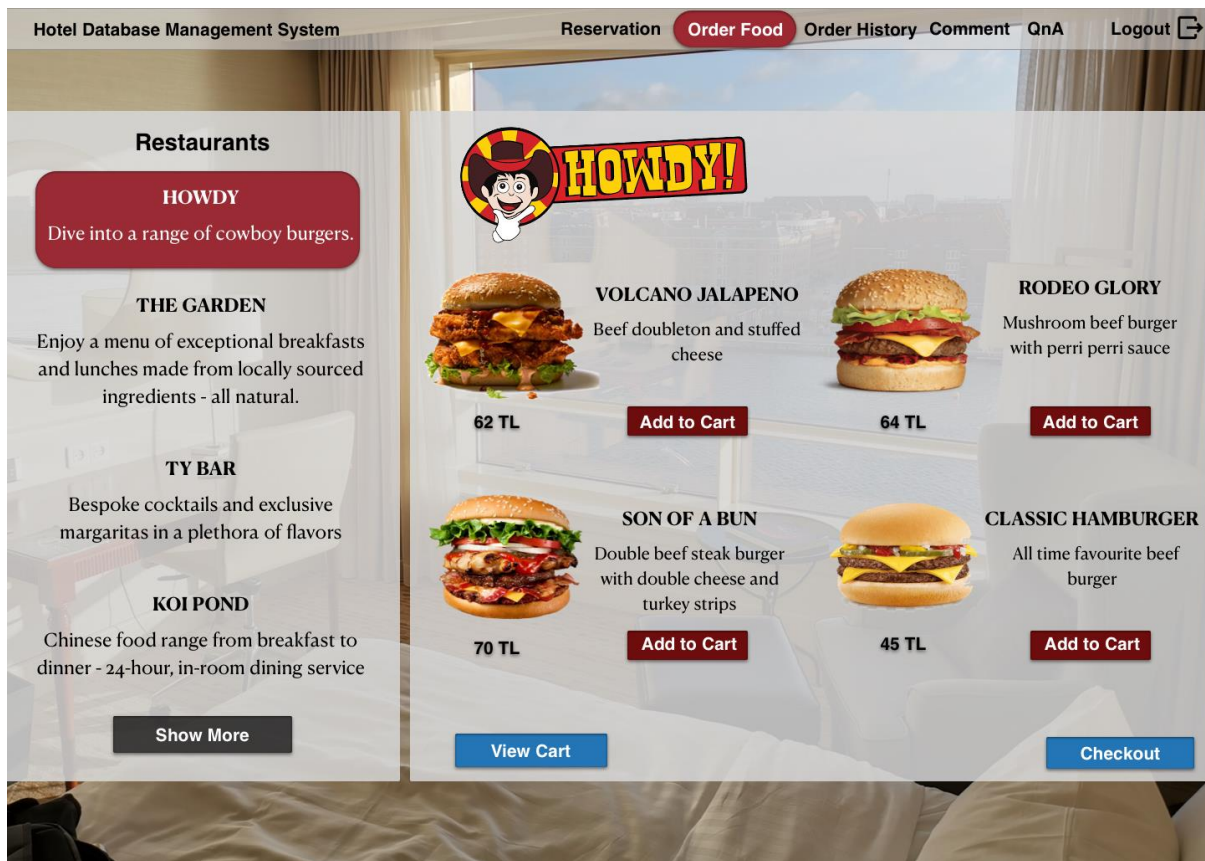
SQL:

**Room Reservation:**

```
SELECT * FROM Room WHERE Room.type = @type;
```

```
INSERT INTO (Reservation) VALUES ( @reservationId, @guestId, @room-no, @building-no, CURDATE(), @check-out-date);
```

### 3.6. Guest - Order Food



Input: N/A

Process: Here the user can select the food of his/her choice from different restaurants. Restaurants are listed in the left part of the screen and the food items the restaurant has to offer are on the right side with details of food items also portrayed.

SQL:

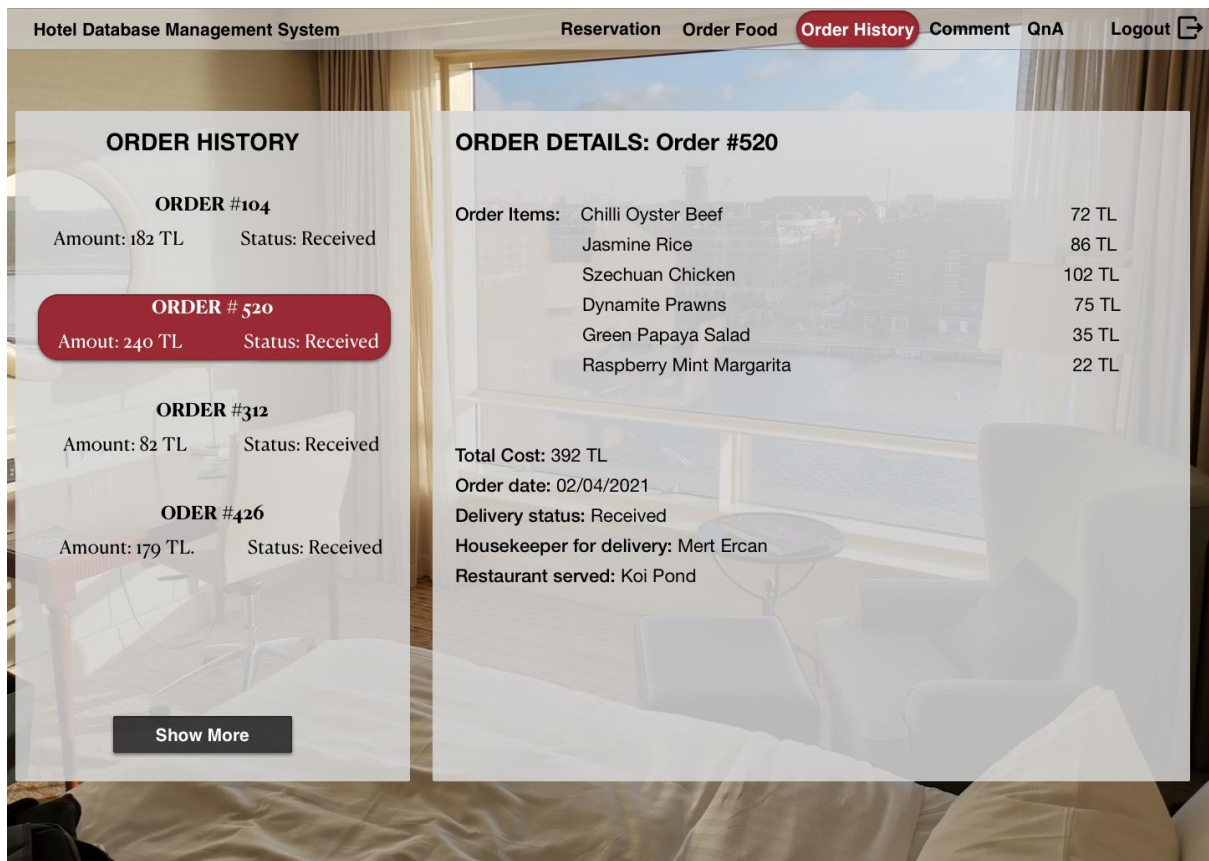
#### Restaurants

```
SELECT name, type, hours
FROM restaurant
GROUP BY restaurant_id;
```

#### Menu items

```
SELECT t.name, t.type, t.price, t.calories,
FROM food-drink t NATURAL JOIN restaurant
ORDER BY t.price;
```

### 3.7. Guest - Order History



Input: N/A

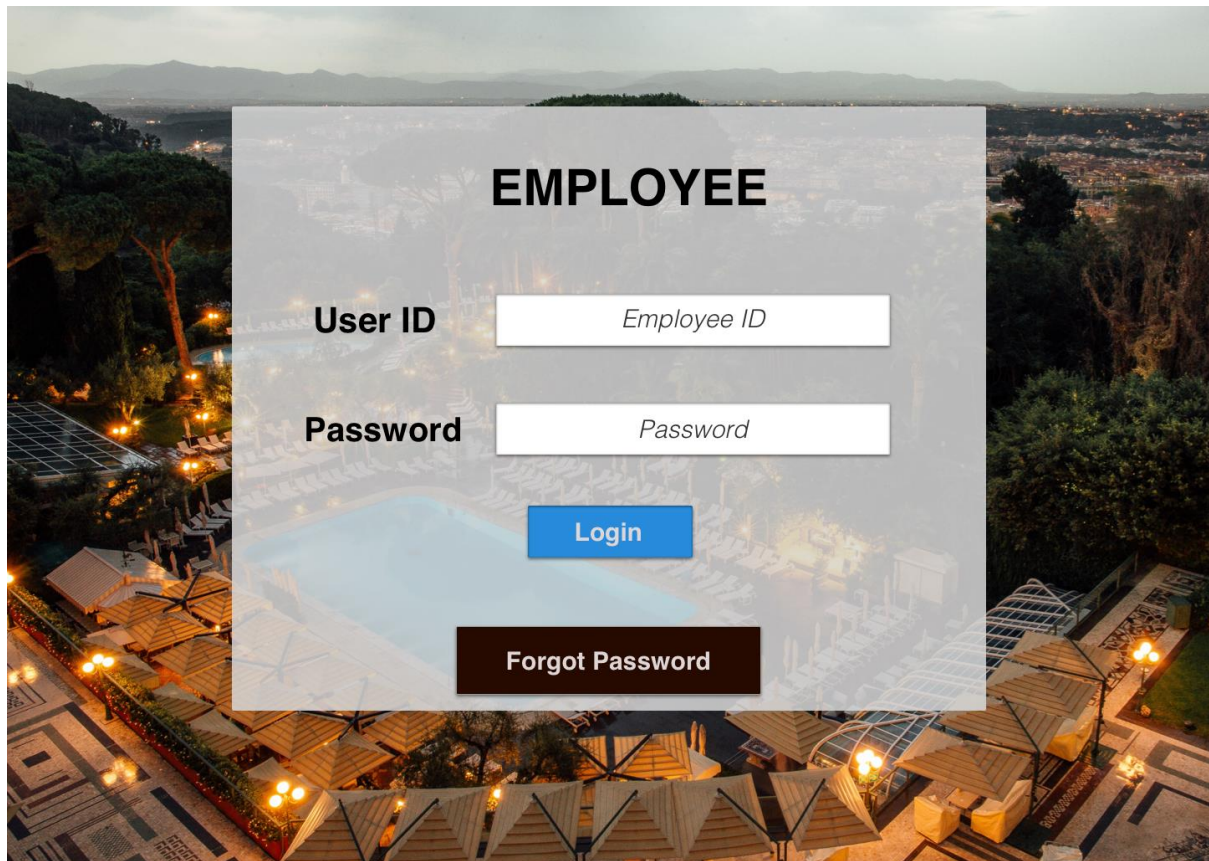
Process: The left half of the window contains the order history of the guest where the list of all orders made by that guest are listed. Secondly, the right half of the screen will have the order details, such as ordered items, cost, order date, etc.

SQL:

#### Order History

```
SELECT status, order-id, SUM(x.price)
FROM order-contains NATURAL JOIN food-drink x NATURAL JOIN food-order t
WHERE user_ID = t.guest-id
GROUP BY order-id;
```

### 3.8. Employee - Login

The image shows a login form titled "EMPLOYEE" overlaid on a background image of a resort at night. The form has two input fields: "User ID" with a placeholder "Employee ID" and "Password" with a placeholder "Password". Below these fields is a blue "Login" button and a dark brown "Forgot Password" button.

**EMPLOYEE**

**User ID**

**Password**

[Login](#)

[Forgot Password](#)

Input: @user-id, @password

Process: Here the user employee is expected to enter the user ID and password which will be tested against the ID column in the employee entity. The screen will have forgot password option as well where one can renew their password as the database contains emails of each employee.

SQL:

#### Logging in

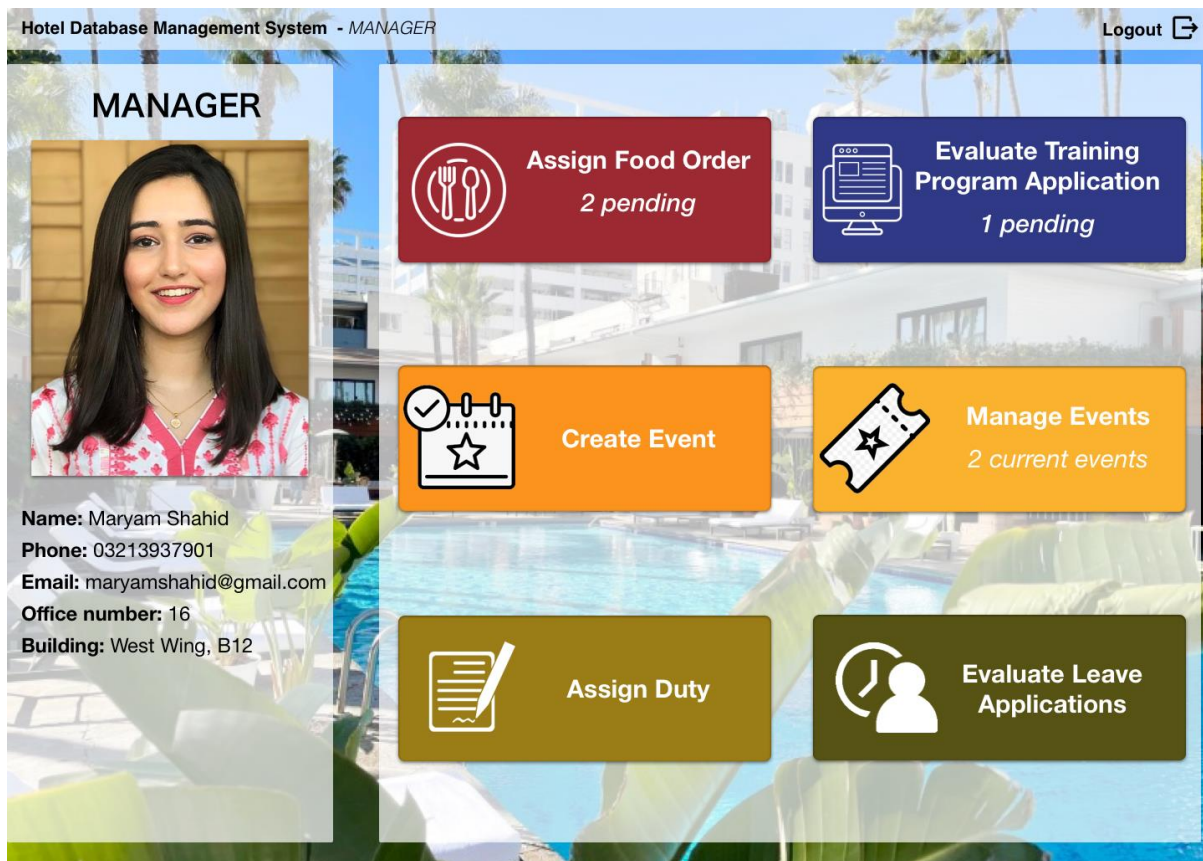
SELECT \*

FROM employee

WHERE @user-id = id AND @password = password;



### 3.9. Manager - Homepage



Input: N/A

Process: After logging in, the manager employee will have this format as output on the screen. The left quarter of the window will have name, phone, email, office number, and location of the manager's office portrayed. The right three-quarters of the window will have several scenarios that the manager can go for, such as assign duty, evaluate leave application, manage events, create events, assign food order, evaluate training program application.

SQL:

#### Manager information

```
SELECT name, phone, email, office-no, address  
FROM manager NATURAL JOIN user;
```

### 3.10. Manager - Assign Orders

The screenshot shows the 'Hotel Database Management System - MANAGER' interface. On the left, there's a sidebar with 'Food Orders' and two sections: 'PENDING:' with 'Order# 07' (highlighted in red) and 'Order# 42', and 'COMPLETE:' with 'Order# 18', 'Order# 29', 'Order# 102', and 'Order #480'. The main area is titled 'Assign Order - Order# 07'. It displays order details: 'Order number: 07', 'Guest name: James Carstairs', 'Room details: Room number 104, West Wing, Main Building', and 'Status: Pending'. Below this, there's a section 'Choose housekeeper:' with three options: 'Mert Ercan', 'Ali Mehmet', and 'Selen Dinc', each with an unchecked checkbox. At the bottom right of the main area is a blue 'Assign' button. The background of the interface features a tropical pool scene.

Input: N/A

Process: This window would show the manager the complete and pending orders. The complete orders would have assigned housekeepers with status completed. Whereas, pending will allow the manager to assign the task to the housekeeper for completion.

#### SQL:

##### Specific Order:

```
SELECT o.order-id, g.firstname, g.lastname, r2.room-no, b.name, b.location-name FROM
Food-order o, Guest g, ( Reservation r NATURAL JOIN Room r2 NATURAL JOIN
Building b NATURAL JOIN User u )
WHERE o.guest-id = g.id AND g.id = r.guest-id AND o.order-id = @order-id;
```

##### Housekeepers:

```
SELECT * FROM Housekeeper h;
```

##### Left list:

```
SELECT o.order-id, o.status FROM Food-order;
```

##### Assign button:

```
UPDATE Food-order SET manager-id = @manager-id, housekeeper-id = @housekeeper-id,
Assign-time = CURRENT_TIMESTAMP;
```

### 3.11. Manager - Evaluate Training Application

The screenshot shows a web application interface for a hotel database management system, specifically the 'MANAGER' role. The background is a blurred image of a hotel pool area with palm trees. The interface has a top navigation bar with the text 'Hotel Database Management System - MANAGER' and a 'Logout' button with an external link icon. On the left, there is a sidebar titled 'Training Programs' with three items: 'Room Delivery Training' (with a '+' icon), 'Event Planner Training' (with a '+' icon), and 'Tour Guide Training' (with a '+' icon). Under 'Room Delivery Training', there is a list of names: '- Mert Ercan', '- Shamin S.', and '- Amna Ahmad' (highlighted with a red background). The main content area is titled 'Application - Amna Ahmad'. It displays the following information: 'Application number: 180', 'Employee ID: 21755', 'Employee rank: Senior', and 'Date Applied: 20/03/2020'. Below this, there is a 'Status' section with two options: 'Accepted' (with an unchecked checkbox) and 'Rejected' (with an unchecked checkbox). At the bottom of the main content area, there is a blue 'Submit' button.

Input: N/A

Process: When clicked on training programs, this window would open up. It will allow the manager to view different training programs made by the manager and the employees that have applied for each training program. The manager would be allowed to either accept or reject the applicant with relevant information regarding the applicant depicted.

SQL:

#### List of training events

```
SELECT event-name
FROM manager m NATURAL JOIN event t NATURAL JOIN training-program;
```

#### List of employees applied to training event

```
SELECT u.name
FROM event p NATURAL JOIN training-program t NATURAL JOIN evaluate-sec-staff-
application q NATURAL JOIN employee s NATURAL JOIN user u
WHERE p.name = (SELECT event-name
FROM manager m NATURAL JOIN event t NATURAL JOIN training-program)
ORDER BY q.training-program-id;
```

#### Application if accepted

```
INSERT INTO Evaluates-sec-staff-application VALUES (@sec-staff-id, @training-program-
id, @manager-id, 'Accepted');
```



### 3.12. Housekeeper - Orders

The screenshot shows the 'Hotel Database Management System - Housekeeper' interface. The top navigation bar includes 'Orders', 'Training', 'Leave', and 'Logout'. The left sidebar displays the user's profile: **HOUSEKEEPER**, Name: Hammad Khan Musakhel, Phone: 03038468251, Email: hkmusakhel@gmail.com, Rank: Senior, Building: East Wing, C-22. The main content area shows a 'PENDING:' order (Order# 07) with details: Date: 29/03/2021, Delivery status: Pending, Delivered food: Son of a bun, wrangler, medium coke, Restaurant served: Howdy, Guest ordered: Mary Jane. There are checkboxes for 'Delivered: Yes' and 'No', and a 'Submit' button. Below this, a list of 'COMPLETE:' orders is shown: Order# 42, Order# 18, Order# 29, Order# 102, and Order# 480.

Input: N/A

Process: When the housekeeper user opens the web application, the first option on the top bar has Orders. Orders opens up a window that enlist all the completed and pending orders. The housekeepers can update the status of a pending order by clicking on YES and submitting it.

SQL:

#### Left list

```
SELECT name, phone, email, address
FROM housekeeper h NATURAL JOIN user;
```

#### Pending

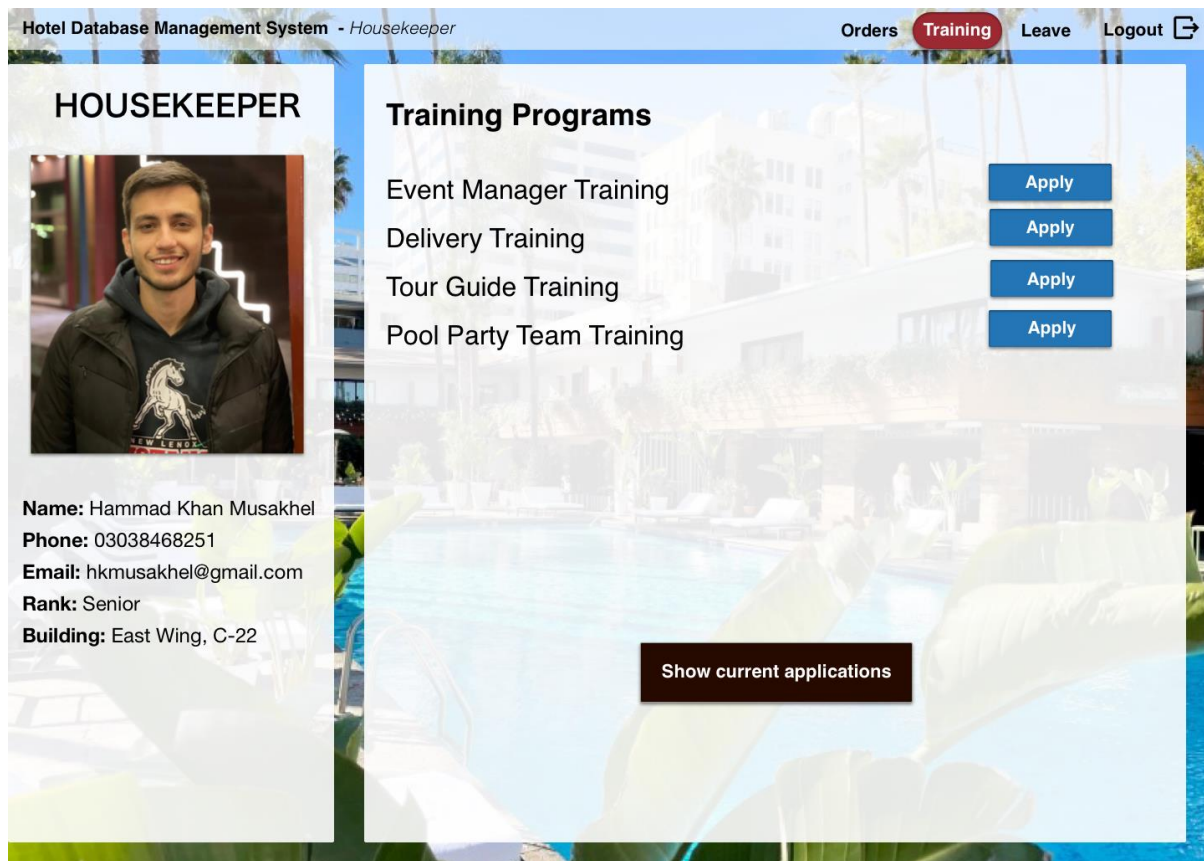
```
SELECT fo-status, fo.assign-time, fd.name, r.name, u.name
FROM food-order fo NATURAL JOIN order-contains oc NATURAL JOIN food-drink fd
NATURAL JOIN restaurant r NATURAL JOIN housekeeper h NATURAL JOIN user u;
```

#### Delivered 'Yes'

```
UPDATE food-order
SET fo.status = delivered
FROM food-order fo NATURAL JOIN order-contains oc NATURAL JOIN housekeeper h
WHERE fo.status = not delivered AND oc.order-id = @order-id;
```



### 3.13. Housekeeper - Training Programs



Input: N/A

Process: This window shows the training programs the housekeeper can apply. The housekeeper can apply to a program by clicking the apply button next to the program name. Using the “Show current applications” button, they can see the programs they apply, as shown in section 3.13.

SQL:

**Training Programs:**

```
SELECT event-name
FROM Event NATURAL JOIN Training-Program
WHERE quota > 0 AND (@housekeeper-id NOT IN (SELECT housekeeper-id
                                              FROM HK-applies-to
                                              WHERE training-program-id = event-
                                              id));
```

### 3.14. Housekeeper - Current Applications

**HOUSEKEEPER**

**HOUSEKEEPER PROFILE:**

- Name: Hammad Khan Musakhel
- Phone: 03038468251
- Email: hkmusakhel@gmail.com
- Rank: Senior
- Building: East Wing, C-22

**Current Applications**

Application	Acceptance status	Date of program	Manager	Action
Event Manager Training	PENDING	08/05/2021	Maryam Shahid	Cancel
Tour Guide Training	PENDING	08/05/2021	Ozcan Ozturk	

Input: N/A

Process: Here, the housekeeper can view his/her application status for different training applications submitted. The left side will have the general information about the housekeeper and the right side will have each application, its status, date of the program, and manager name. The cancel button will allow the housekeeper to cancel the application.

SQL:

#### Left list

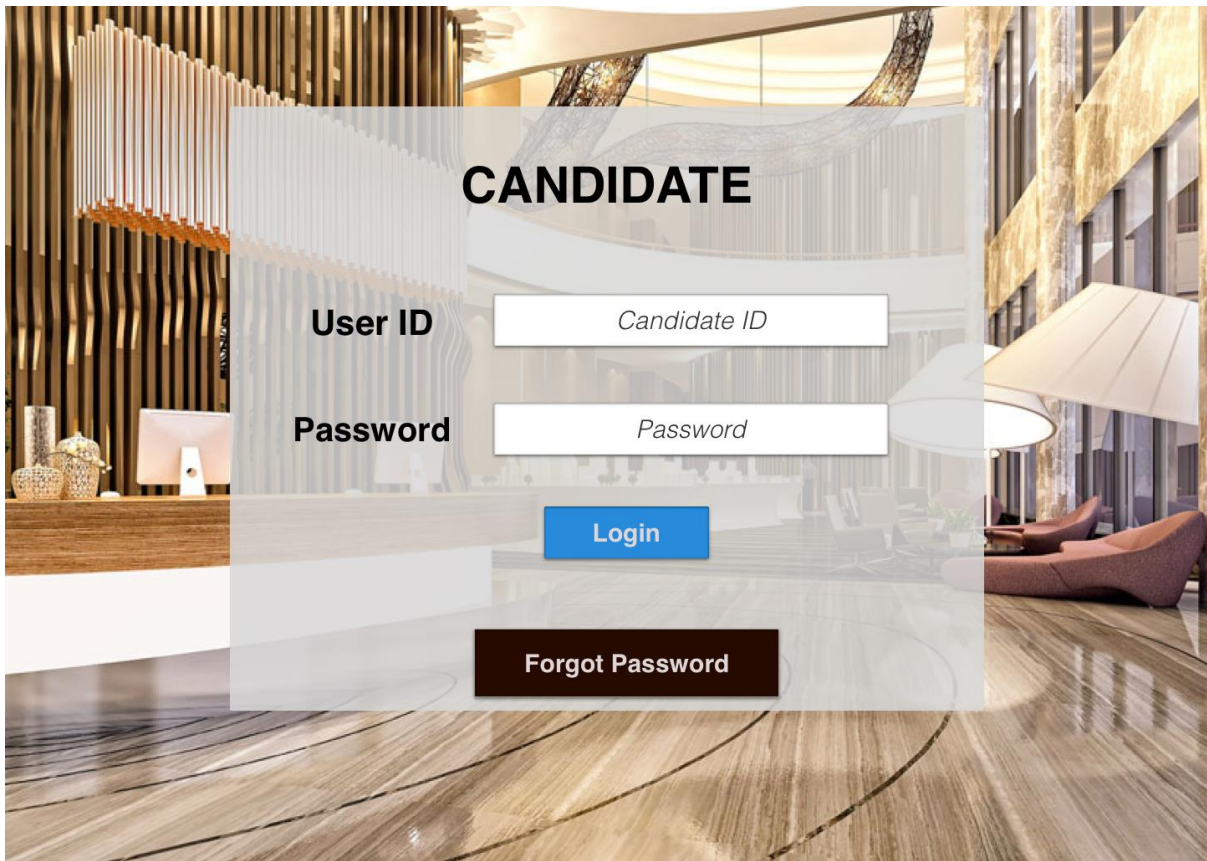
```
SELECT name, phone, email, address  
FROM housekeeper NATURAL JOIN user;
```

#### Application Status

```
SELECT *  
FROM ( Housekeeper h NATURAL JOIN User u ) housekeepers, ( Manager m2 NATURAL  
JOIN User u3 ) creatingManagers ,( Event NATURAL JOIN Training Program) et, ( HK-  
Applies-to at NATURAL JOIN Evaluates-hk-application ) hke
```

```
WHERE creatingManagers.id = et.manager-id AND hke.training-program-id = et.event-id  
AND housekeepers.id = hke.housekeeper-id AND housekeeper.id = @housekeeper-id;
```

### 3.15. Candidate Login



Input: @user-id, @password

Process: Here the user candidate is expected to enter the user ID and password which will be tested against the ID column in the candidate entity. The screen will have a 'forgot password' option as well, where the user can renew their password as the database contains emails of each candidate.

SQL:

#### Logging in

```
SELECT *
```

```
FROM candidate
```

```
WHERE @user-id = id AND @password = password;
```

### 3.16. Candidate Application

The screenshot shows a web application interface for a hotel database management system. At the top, there is a header bar with the text 'Hotel Database Management System' on the left and a 'Logout' button with an external link icon on the right. The main content area is a semi-transparent white box titled 'APPLICATION' in bold black letters. Inside this box, there are several input fields arranged in two columns. The left column contains 'First name', 'Email', 'Address', 'Attach CV', and 'Attach Cover Letter'. The right column contains 'Last name', 'Gender', 'Date of birth', 'Position', and 'Add Info'. Each text input field is accompanied by a small file upload icon (a square with a plus sign and a document icon). At the bottom right of the form box is a blue button with the text 'Apply' in white. The background of the page is a blurred image of a modern hotel lobby with a large circular chandelier and wooden floors.

Input @position

Process: Here the numerous inputs are dummy inputs for the time being. The candidate user will add the required information, especially Position. The candidate will have the opportunity to make numerous applications and apply. Candidate is an additional feature of our web application. A candidate will be assigned a new Employee ID once recruited. A candidate can be assumed as a portal to apply for jobs at the hotel.

SQL:

**Add into job application list; submit button**

```
INSERT INTO Job-Application VALUES( @user-id, @position, 'Pending');
```

## **5. Implementation Plan**

- MySQL will be used as the DBM system in the implementation.
- Spring framework will be used in development.

## **6. Web Page**

The reports of this project will be published on the following web page:

<https://cs353-group38.github.io/>