DATABASE MANAGEMENT SYSTEM

University Student Housing
Project Report

Team 7
Saurav Shekhar (sshekha3)
Akond Rahman (aarahman)
Mithilesh Jaywant Gawande (mgawand)
Anuja Chandrashekhar Yawalkar (ayawalk)

Table of Content

Table of Figures

Problem Statement

University consists of a large amount of population. Earlier the records of students and related information were stored in files at the university office. At this time for doing menial tasks like updating the address, changing the parking spot required the housing staff to remove the records, scan the records to remove the file associated with respect to that particular student. Make the necessary and keep back the records. For generation of requests for parking, leasing etc. the student will have to walk to the management office and make a request. He would also have to go to the office to check if the request is approved or not. This would make his life very tedious. The housing staff also has to keep on tallying every-time information like the total number of students in a particular residential hall, the number of vacancies in the apartments, availability of parking spot in the nearby lot. All this tasks if done manually are very tedious and time consuming. Hence in order to maintain the records of these huge amount of people we require a database that can efficiently and effectively function to provide the users (students/guests) and housing staff the data that they require. This allows the housing staff to manage the information like lease, maintenance and parking with respect to a student effectively. Whenever a student is admitted into an institution his basic records like date of birth, sex, course taken etc. are needed to be stored to uniquely identify that person and store information regarding him. At the same time he is given a ID which is associated with any transaction he makes as a student in the university. Every student in a university requires facilities like housing, parking, etc. For this he needs to request the housing, parking etc. If the student is living far away from the university then it would take time for him to communicate his requirements to the housing staff. Using database systems and making this process online will make it easier for the student to ask for these facilities remotely. In order to make the process smooth, robust and accurate we need to design a robust database system which will keep track of all the student information. Similarly we also need the information of guests to be stored so that whenever a guest arrives we can provide him with facilities like housing/ parking. Also it becomes conducive for the administrative staff to have all the records stored in a system so that he/she can view previous records without having to search much. The students can also monitor their past spending on parking, maintenance and housing by viewing his past invoices, etc. The student will also get invoices to automatically reminding him of his monthly dues. Thus we require a database system which is accessible to students/guests and housing staff so that they can view/retrieve/update/use the information effectively.

Entities and Relationships

Entities

1. People

This entity maintains the records of all the students in the university system. It also maintains information about the university scholars that may visit the university or students that may come to university for competitions etc. It stores the basic profile information of the students in the table like their name, date of birth, sex, courses, category etc. A new row is created in this table whenever a given student is granted an admission in the university and whenever a guest arrives at the university.

2. Kin

University may need the contact information of close friend or relative of a student to contact to in case of an emergency. For this reason each student is associated with information of a kin in this entity (name, address, contact number etc.). As soon as a student entry is entered in the people table after the admission the information of the kin is also added to the kin table. As the attributes of kin entity are not used anywhere in the database and as its primary key is the ID of the student taken from peopleID hence kin is a weak entity.

3. Family

If a student is admitted in the university who has a family to support and the family is going to live with him in the university campus the student entry is associated with the family(Yes) in the people entity and correspondingly the information of the family the student supports is added to the Family table with peopleID as the foreign key. For the students who do not have any dependents there is no entry in this entity.

4. Housing Staff

For administration of the university housing and other departments some staff is assigned. The information regarding this staff like their name, residence hall, address, contact, etc is stored in the housing staff entity. The housing staff has roles like hall manager, etc. with certain privileges. Like a hall manager can approve a leasing request, can process the maintenance tickets, etc.

5. Lease

This entity is designed to store information about the current housing location of a student. Duration of his stay at that particular address, etc. whenever a student is admitted in the university he has a choice to either opt for private accommodation or university housing. IF he opts for university housing he has to request a lease which will be processed by the administrators. If the desired apartment or residence hall is available then the person is granted with the lease and a row is added in the lease entity which stores information like start of the lease / move in date and end of the lease move out date, etc. Once the lease in completed the entry will get removed from the lease table.

6. Lease Request

Whenever a student enters a university or is given an admission he has to decide whether he is opting for living on campus or in private accommodation. If he decides to opt for a private accommodation he has to send a lease request to the administrative department, this request includes information like the students preference for housing, period for which the student is wishing for live in that place and the payment method he/she is opting for. If a student's preference is available at a particular time then his request is processed and he is granted the lease. Or he is put into waitlist students list. This list has higher priority over new lease requests.

7. **Lease Termination(Terminate Request**)

If a student wants to terminate the lease the he can raise a terminate lease requests. The termination lease request includes information like the reason for leaving and the date at which the person wants to leave. Then an entry will be then added to this entity set. The administrative staffs will then reviews the request add the additional charges to the account of the persons whose name is on the lease update the inspection date.

8. Parking Permit

Whenever a person’s request for a parking spot is approved an entry is added in the parking permit entity storing information about the person and the parking slot that is allocated to him.

9. Parking Request

A person may raise a parking request only after he has a lease associated with him. His request is then processed and a parking spot is granted to him depending on his needs like(handicapped/not)(bike/car, etc.) and his address (parking spot that is need the persons housing is preferred while assigning depending on the availability of spot in nearby parking lots.

10. Parking Spot

Parking spot is entity consisting of the unique identifier for the parking spots on the campus. It also consists of information like the type of parking spot and its availability. If a certain category parking spot is available in a nearby table for a particular person then it is assigned to that person and its availability is changed to not available. Simultaneously the spotID is updated in the persons profile and he is given a parking permit.

11. Parking Lot

The parking lot entity keeps a list of all the parking lots that are available in the university premises. Whenever a student/guest raise a request for a parking spot a lot nearest to his housing address is selected and a parking spot that matches his requirement is granted to him. Some parking lots are reserved for only students and guests are not allowed to request a spot for these lots.

12. Housing

The university housing entity consists of variety of housing options like residence halls and apartments for family as well as shared apartments for students who are single. Each of these housing options are uniquely identified by their name and housing ID and an address is associated with them. This information is stored in this entity. Whenever a student is requesting lease he has to enter 3 preferences. The preferences are among the entries in the housing table.

13. Residence Hall

The residence hall is a type of housing option that is available for people without family. The resident hall table consists of information like available rooms, rent for a room, nearby parking lots etc. Some residence halls are reserved for only graduate students. Other students are not allowed to stay in these halls. Also each hall has a hall manager associated with it.

14. Apartment

Apartment is a type of university housing. Apartment can be classified as either family or shared apartment. There is an attribute in the Apartment entity that distinguishes it as shared and family. A shared apartment has a place ID associated with each bedroom and each bedroom has a distinct rent and other facilities associated with it. Apartment table also has a manager associated with it.

15. Private Accommodation

The people who do not wish to live in university housing and prefer living in the private houses around campus have entries in this entity. However a university freshman student is not allowed to live in this type of housing facility.

16. Invoice

Invoice message is a reminder message that maybe auto-generated or generated by the administrator. This message contains information like parking rent, housing rent, additional charges, etc. and other billing information. The invoice is generated depending on the type of payment method a person is using. If a person is using a monthly payment method a monthly invoice is generated whereas if he/she opts for a semester payment only 2 invoices are generated 1 at the start and one at the end of the semester.

17. Maintenance Ticket

Maintenance Ticket is raised by the resident (student/guest) if there is some problem in the housing. The problem can be related to water, electricity, etc. According to the severity of the problem the administrative staff processes the request. The options are provided as high, medium and low severity while making the request for the ticket.

18. Address

This entity stores the addresses associated with the housing ID , the parking lot. Whenever a student/guest is granted a lease the address value associated with the resident hall/apartment where the student/guest is residing is updated in the people profile. Also the

address of the parking lot is stored here and these addresses are used to populate the nearby table which is used to decide the nearby parking lot for the given housing option.

19. Nearby

This entity set stored the information related to the nearby parking spot that is associated with the housing option. This table is used to decide in which parking lot the parking request will be sent by a student/guest belonging to a particular housing option.

20. Login

This entity is an independent entity that stores the information required for authentication and authorization for a student, guest or an administrator to gain access to the database and perform the following tasks required to manage the accounts.

21. Parking Fees

This table consists of the parking fees associated with the respected parking spot. Each parking spot category and each category has a fixed value of parking rent. This table is created to map the parking fees with the category of parking spot.

22. Linkeness

This table is used to check the similarity in people. Students who are searching for potential roommates use this facility; this table matches students on basis of their sex, course, dateOfBirth, comments, etc. information retrieved from the people table.

Relationships

1. Inovice_People

This is a binary relationship between Invoice and People entities. Each invoice has a people ID associated with it. This is a many to one relationship. Here one person can have multiple invoices but each Invoice is generated for only one student/guest at a time.

2. Invoice_Lease

This is a binary relationship between Invoice and Lease entities. Each invoice has a Lease ID associated with it. This is a many to one relationship. Here one lease can have multiple invoices but each Invoice is generated for only one lease at a time.

3. Maintenance Ticket_Housing

This is a binary relationship between Maintenance Ticket and Housing entities. Each Maintenance ticket has a unique Housing option associated with it. This is a many to one relationship. Here one housing option can have multiple maintenance tickets but each maintenance tickets are generated for only one housing option at a time.

4. Maintenance Ticket_People

This is a binary relationship between Maintenance Ticket and People entities. Each Maintenance ticket has a unique People ID associated with it. This is a many to one relationship. Here one Person can have multiple maintenance tickets but each maintenance tickets is generated by only one person at a time.

5. MaintenanceTicket_Housing Staff

This is a binary relationship between Maintenance Ticket and People entities. Each Maintenance ticket has a unique People ID associated with it. This is a one to many. Here one Housing staff can be responsible for managing and processing multiple maintenance tickets but each maintenance ticket is managed by only one housing staff that is associated with the given housing ID .

6. People_Kin (weak entitity)

This is a binary relationship between People and Kin. Each kin is associated with once people ID. Each person has to have a kin associated with him. Here the kin entity is a weak entity.

7. People_Family(weak entitity)

This is a binary relationship between People and Family. Each Family is associated with once people ID. There is no neccisity that each person has to have a family associated with him. Here the Family entity is a weak entity.

8. Kin_Address

This is a binary relationship between Kin and Address. Each Kin is associated with once Address. This is a one to one relationship. That is each kin has once unique address and each address has only one kin.

9. Lease Request_People

This is a binary relationship between Lease Request and People entities. Each Lease request has a unique People ID associated with it. This is a one to one relationship. Here one Person can have only one lease request and one lease request must have only one person associated with it.

10. Lease_People

This is a binary relationship between Lease and People entities. Each Lease request has a unique People ID associated with it. This is a one to one relationship. Here one person can have only one lease and one lease must have only one person associated with it.

11. Lease_Housing

This is a binary relationship between Lease and Housing entities. Each lease has a unique housingID associated with it. This is a one to many relationships. Here one housingID can have only many leases but one lease request must have only one Housing ID associated with it. For example a residence hall has a unique housing ID and it has many students living in it having distinct lease ID

12. Lease_Places

Whenever a student/guest has been giving housing he is given a unique place having a unique place ID information associated with it. This is a one to one relationship. Each lease has one people ID associated with it and vice versa.

13. Lease_Terminate_Request

This is a binary relationship between Lease and Terminate Request entities. Each Lease can have only one terminate request associated with it and each terminate request has a lease ID associated with it. Hence it is a one to one relationship. Though a student/guest may or may not always generate a terminate lease request.

14. People_Terminate_Request

This is a binary relationship between People and Terminate Request entities. Each Person can have only one terminate request associated with one lease. But each terminate request has a lease ID associated with it. Hence it is a one to one relationship. Though a student/guest may or may not always generate a terminate lease request.

15. Parking Lot_Parking Spot

This is a binary relationship between Parking Lot and Parking Spot entities. Each Parking Lot has many spots. But each parking spot has only 1 lot associated with it. Thus it is a many to one relationship between Parking Spots and Parking Lot.

16. Parking Lot_Housing

This is a binary relationship between housing option and parking Lot. Where each housing has a parking id associated with it. We can thus find the nearby parking lot for a given housing ID.

17. Parking Spot_People

This is a binary relationship between People and Parking Spot entities. Each Person can have only one Parking Spot associated with one lease. But each Parking spot may or may not have a people ID. That is it may be occupied or it may be available.

18. Parking Spot_Parking Rent

This is a binary relationship between Parking Rent and Parking Spot entities. Each Parking spot has a classification and according to that classification the rent is decided. Here the Parking Rent is a weak entity.

19. Parking Request_Parking Lot

This is a binary relationship between Parking Request and Parking Lot entities. Each housing option has a parking lot asa nearby parking lot. In the parking request if a vacancy is available in the desired parking lot then the request is granted. This is a one to one relationship where each request has exactly one parking lot entry.

20. Parking Request_Parking Spot

This is a binary relationship between Parking Request and Parking Spot entities. Each Parking spot has an explanation of the type of spot required and the classification and according to that classification the rent is decided. If the desired type of spot is available in the lot then the request is granted.

21. Parking Request_People

This is a binary relationship between Parking Request and People entities. Each Parking request can be raised only by a person that has a valid lease ID associated with him. He can raise only one request at a time. This is a one to one relationship but there each parking request is associated with one people ID but each person does not necessarily raise a request.

22. Housing_Address

This is a binary relationship between Housing and Address entities. Each housing has a unique address. This is a one to one relationship.

23. Housing_Resident Hall

This is a hierarchical relationship between Housing and Residence Hall entities. Each housing is either a Housing Apartment or a residence hall.

24. Housing_Places

This is a binary relationship between Housing and Places entities. Each Place is located in one of the housing options. That is each place ID has exactly 1 housing ID associated with it. But each housing option has many places.

25. Housing_Apartment

This is a hierarchical relationship between Housing and Address entities. Each Housing has a address. This is a one to one relationship.

26. People_Address

Once the lease of a person is approved his address is stored. It is a one to many relationships. As many people can have the same address (can be in the same residence hall for example) but each person as a unique place ID to exactly locate him/her.

27. People_Places

This is a binary relationship with each person living in the housing university has a unique place ID associated with him/her. Each place ID similarly has only one person associated with it.

28. Housing Staff_Housing

This is a binary one to many relationships between Housing Staff and Housing Entities. The housing staff is associated with exactly one housing option. Here as housing can have various people working for it like. However each housing option has exactly one manager according to the data given hence it becomes a one to one relationship in this case.

USERS

There are three types of users namely:-
Student, Guest, and Housing staff/ administrative staff

1. Student

A student can log into the application from where he can login into his online account which consists the information about his profile, housing, parking, maintenance etc.
He can select on either of the options to get further details:-

a. Housing option:- Selecting this option allows the user to view his current/former invoices, current/former lease. It also allows him to generate or cancel his request and view vacancies.
b. Parking Option:- Selecting this option provides the user the options further to request, view, renew, return a parking spot and view the parking lot information like the address, etc.
c. Maintenance:- Through this option a user can view the current ticket status or raise a new ticket providing the required information.
d. Profile:- Through this option the person can view all his profile detail that are stored in the people entity and edit the details if required.

2. Guest

Every guest has an approval id that is assigned to him uniquely. Using this approval ID a person can login from the student's side and follow all the options that are available to the student like housing, parking, maintenance and profile.

3. Supervisor/Admin

The supervisor or admin has access to all the information in the database relating to the students and the guests. The requests made by the students or requests are approved, waitlisted, processed, etc. by the system administrators. The supervisor has the following options available when he logs into: -

a. Lease Requests: - From here the administrator can either approve the lease request or put the request on wait-list by checking the availability of the housing option preferred by the student. If the housing option is of available the request is put to waitlist.
b. Terminate lease request: - The admin can approve the terminate lease request and then assign the inspection date and the enter damage fees if any to the terminate request. After the inspection date the terminate lease request status is changed to complete. If there are any damaged fees than an invoice is generated to inform the student/guest about it.

c. Maintenance Ticket: - Here the admin can view the tickets raised by the students/guests according to the severity of the problem as indicated in the ticket itself, once he views the ticket its status is changed to processing and after half an hour its status is changed to completed.
d. Parking Request: - The admin sequentially checks the incoming parking requests and if the student/guest matches the criterion for the requested parking slot assigns him the required slot or else rejects the request.

E-R DIAGRAM

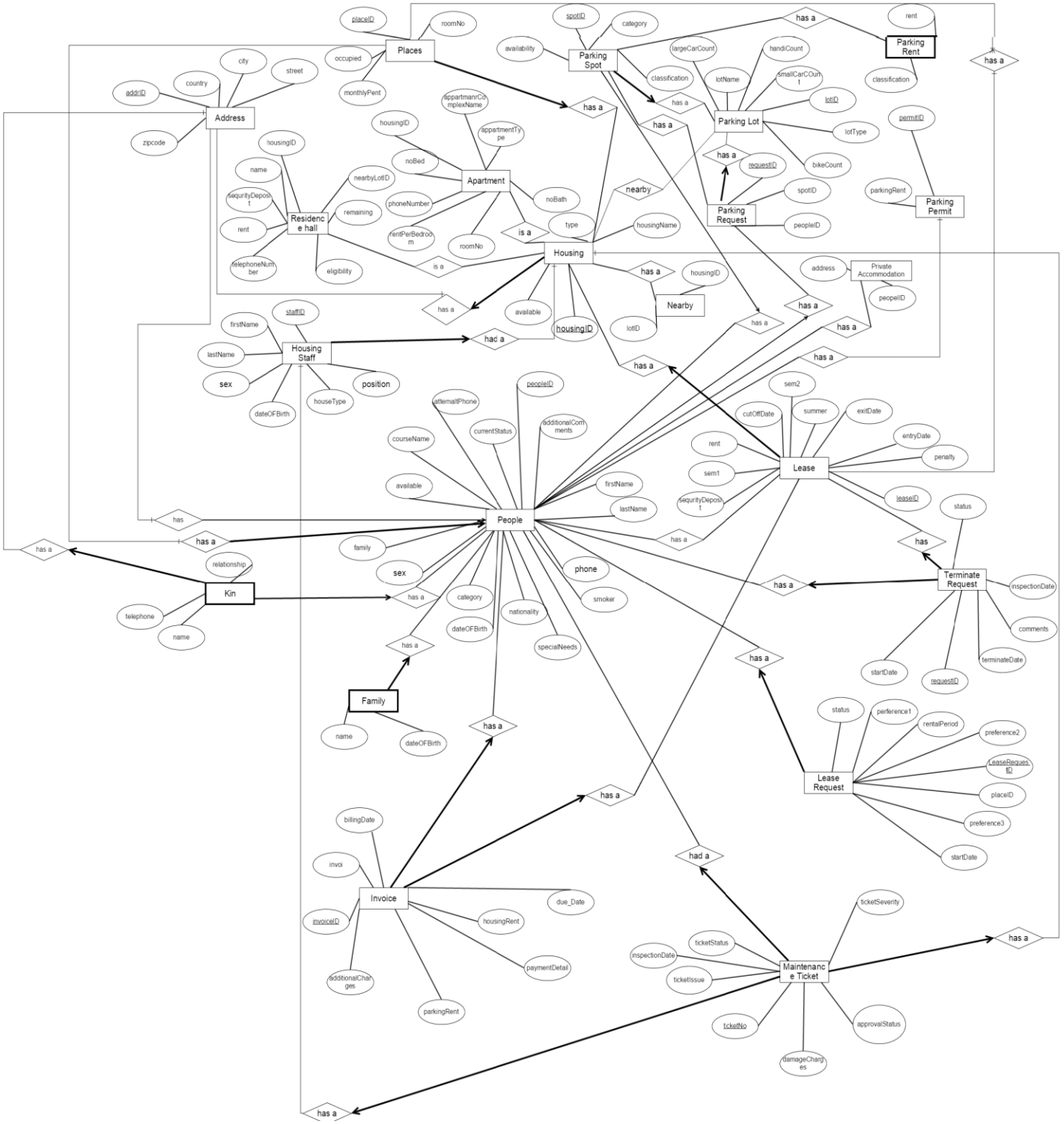

Figure 1 : ER Model for University Housing

Relational Schemas

1. Address (addrID:int, street:varchar(50), city:varchar(20), country:varchar(20), zipcode:int)

2. Parking_Lot (lotID:int, lotName:varchar(20),handiCount:int, bikeCount:int, smallCarCount:int, largeCarCount:int, lotType:varchar(20))

3. Parking_Spot (spotID:int, lotID:int, classification:varchar(20), availability:int, category:int)

4. people (peopleID:int, firstName:varchar(20), lastName:varchar(20), phone:varchar(20), alternatePhone:varchar(20), addrID:int, dateOfBirth:date, sex:char(10), category:varchar(20), nationality:varchar(20), smoker:varchar(20), specialNeeds: varchar(100), additionalComments:varchar(20), currentStatus:varchar(20), courseName:varchar(50), family:varchar(20), housingID:int, placeID:int(20), spotID:int(20), leaseID:int(20), available:int)

5. Kin (peopleID:int, name:varchar(50), relationship:varchar(20), telephone:varchar(20), addrID:int)

6. Family (name:varchar(50), peopleID:int, dateOfBirth:date)
7. Housing (housingID:int, staffID:int, housingname:varchar(20), addrID:int, available:int(1), type:varchar(20))

8. Residence_Hall (housingID:int, name:varchar(20), addrID:int, telephoneNo:varchar(20), staffID:int(20), noRooms:int, rent:int, sequrityDeposit:int, nearbyLotID:int(20), eligibility:int, remaining:int)

9. Apartment (housingID:int(20), addrID:int(20), noBed:int(11),noBath:int(11), apartmentComplexName:varchar(20), phoneNumber:varchar(20), rentPerBedroom:int(11), secuDeposit:int(11), managerID:int, roomNumber:int(11), apartmentType:int)

10. Housing_Staff (staffID:int(20), firstName:varchar(50), lastName:varchar(50), addrID:int, dateOfBirth:date, sex:char(10), position:varchar(30), housingID:int, houseType:varchar(20))

11. Places (placeID:int(20), housingID:int(20), monthlyRent:int, roomNo:int, peopleID:int(20), occupied:varchar(10))

12. Lease (leaseID:int(20), peopleID:int(20), placeID:int(20), housingID:int(20), sem1:int, sem2:int, summer:int, entrydate:date, exitDate:date, securityDeposit:int, penalty:int, cutOffDate:date, rent:int, paymentOption:varchar(20))

13. Parking_Requests (requestID:int, lotID:int(20), spotID:int(20), peopleID:int(20), status:varchar(20), housingID:int, classification:varchar(20), nearby:int)

14. Nearby (lotID:int, housingID:int)

15. Maintenance_Ticket (ticketNo:int, peopleID:int(20), peopleName:varchar(20), ticketStatus:varchar(20), ticketDate:Date, ticketIssue:varchar(20), ticketSeverity:int, comments:varchar(40), housingID:int(20), staffID:int(20))

16. Lease_Request (LeaseRequestID:int, peopleID:int, PlaceID:int, status:varchar(10), startDate:date, endDate:date rentalPeriod:int, preference1:int, preference2:int, preference3:int, paymentMode:varchar(10), parkingNeeded:int)

17. Terminate_Request (RequestID:int(11), peopleID:int(20), placeID:int(20), terminateDate:date, status:varchar(10), inspectionDate:date, leaseID:int(20), comments:varchar(100))

18. parking_fees (classification:varchar(20), rent:int)

19. invoice (invoiceNo:int, peopleID:int(20), rent:int, parkingRent:int, lateFees:int, otherCharges:int, totalDue:int, leaseID:int, billDate:date, dueDate:date, placeID:int, roomNo:int, housingID:int, housingname:varchar(20), paymentDate:date, methodPayment:varchar(20), finalChargeDeducted:int, refund:int, status:varchar(20))

20. login(ID:int,password:varchar(32),firstName:varchar(20),lastName:varchar(20),author:varchar(20))

21. Private_Accommodation (peopleID:int(20), address:varchar(20))

Functional Dependencies and Normal Forms-

1. In Table **Address** every attribute can be uniquely identified using the primary key - addrID. It has only one functional dependency. The candidate key here is - {addrID}. As the left side is the candidate key for the relation the relation is in **BCNF.**

addrID -> street, city, country, zipcode

2. In Table **Parking_Lot** every attribute can be uniquely identified using the primary key - lotID. It has only one functional dependency. The candidate key here is - {lotID}. The lotID here displays the details of the parking lot in general and displays the information related to a particular parking lot. As the left side is the candidate key for the relation the relation is in **BCNF.**

lotID -> lotName, handiCount, bikeCount, smallCarCount, largeCarCount, lotType

3. In Table **Parking_Spot** every attribute can be uniquely identified using the primary key - spotID. It has only one functional dependency. The candidate key here is - {spotID}. The spotID is completely global so it doesn't matter what the lotID is since we can uniquely find the details using just the spotID. As the left side is the candidate key for the relation the relation is in **BCNF.**

spotID -> lotID, classification, availability, category

4. In Table **People** every attribute can be uniquely identified using the primary key - peopleID. It has only one functional dependency. The candidate key here is - {peopleID}. The peopleID will be unique to all the users present in the table. As the left side is the candidate key for the relation the relation is in **BCNF.**

peopleID -> firstName, lastName, phone, alternatePhone, addrID, dateOfBirth, sex, category, nationality, smoker, specialNeeds, additionalComments, currentStatus, courseName, family, housingID, placeID, spotID, leaseID, available

5. In Table **Family** every attribute can be uniquely identified using the primary keys - name and peopleID. It has only one functional dependency. The candidate key here is - {peopleID,name}. As the left side is the candidate key for the relation the relation is in **BCNF.**

name,peopleID -> dateOfBirth

6. In Table **Kin** every attribute can be uniquely identified using the primary keys - peopleID and name. It has only one functional dependency. The candidate key here is - {peopleID,name}. As the left side is the candidate key for the relation the relation is in **BCNF.**

peopleID,name -> relationship, telephone, addrID

7. In Table **Housing** every attribute can be uniquely identified using the primary key - housingID. It has only one functional dependency. The candidate key here is - {housingID}. The housingID can be used to identify whether it is an Apartment or a REsidence_Hall. As the left side is the candidate key for the relation the relation is in **BCNF.**
The housingID attribute in **Residence_Hall** and **Apartment** can also be used to identify each and every attribute of the respective table. Thus both the tables are in **BCNF.**

housingID -> housingname, addrID, available, type

8. In Table **Invoice** every attribute can be uniquely identified using the primary key - invoiceID. It has only one functional dependency. The candidate key here is - {invocieID}. . As the left side is the candidate key for the relation the relation is in **BCNF.**

invoiceID -> housingRent,parkingRent,housingRent,due_Date_paymentDetail

Assumptions

1. Invoice Generation, Lease Request, Parking Request, Maintenance Ticket and Lease Termination Request are implemented using stored procedures. So our requirement is to call the required stored procedures.

2. Lease cannot be renewed midway during the semester.

3. Referring to the sample data we have implemented that each person (peopleID) can be associated with only one kin.

Reporting Queries

1. For each hall or apartment, display the total number of nearby parking spots (both available and not available)

```
SELECT DISTINCT h.housingname,COUNT(*)
FROM Nearby n, Housing h
WHERE n.housingID = h.housingID
GROUP BY n.housingID;
```

2.For each hall, display total number of graduate students on the request list that do not yet have approved leases.
For Each Preference

```
select count(*), preference1 from lease_Request where status <> 'Approved' and peopleID IN ( select peopleID from people where category = 'Graduate' ) and preference2 IN ( select housingID from housing where type='Residence Hall') group by preference2;

select count(*), preference2 from lease_Request where status <> 'Approved' and peopleID IN ( select peopleID from people where category = 'Graduate' ) and preference2 IN ( select housingID from housing where type='Residence Hall') group by preference2;

select count(*), preference3 from lease_Request where status <> 'Approved' and peopleID IN ( select peopleID from people where category = 'Graduate' ) and preference3 IN ( select housingID from housing where type='Residence Hall') group by preference3;
```

3. For each student renting, print their next invoice.

First we run the stored procedure to generate the next invoice. By default, invoice generation is set to whatever preference was selected (Monthly / Semester). This can be tweaked for the purpose of debugging by remove CURDATE() functions to check for valid dates.

Once the invoice is generated, it can be viewed by:

```
SELECT billDate, dueDate FROM invoice WHERE leaseID = _leaseID ORDER BY dueDate DESC LIMIT 1;

DELIMITER //
```

```
DROP PROCEDURE IF EXISTS AddNewInvoice;
CREATE PROCEDURE AddNewInvoice(_leaseID int)
BEGIN
DECLARE _paymentOption varchar(20);
DECLARE _rent int;
DECLARE _peopleID int;
DECLARE _placeID int;
DECLARE _housingID int;
DECLARE _entryDate DATE;
DECLARE _firstDueDate DATE;
DECLARE _exitDate DATE;
DECLARE _finalInvoice DATE;
DECLARE _roomNo int;
DECLARE _housingname varchar(20);
DECLARE _checkInitInvoice int;
DECLARE _securityDeposit int;
DECLARE _aggregateDue int;
DECLARE _spotRent int;
DECLARE _classification varchar(20);
DECLARE _totalDue int;
DECLARE _otherCharges int;
DECLARE _nextBillDate DATE;
DECLARE _nextDueDate DATE;
DECLARE _currentBill int;
DECLARE _refund int;
DECLARE _prevPeriod int;
DECLARE _period int;
DECLARE _nextPeriod int;
DECLARE _currDate DATE;
DECLARE _spotID int;
DECLARE _curDate DATE;
SELECT paymentOption, rent, peopleID, placeID, housingID  INTO _paymentOption, _rent,
_peopleID, _placeID, _housingID FROM lease WHERE leaseID = _leaseID;
SELECT spotID INTO _spotID FROM people WHERE peopleID = _peopleID;
SELECT entryDate, DATE_ADD(entryDate, INTERVAL 10 DAY) INTO _entryDate, _firstDueDate
 FROM lease WHERE leaseID = _leaseID;
SELECT exitDate, DATE_SUB(exitDate, INTERVAL 1 MONTH) INTO _exitDate, _finalInvoice
 FROM lease WHERE leaseID = _leaseID;
SELECT roomNo INTO _roomNo FROM places WHERE placeID = _placeID AND housingID =
_housingID;
SELECT housingname INTO _housingname FROM housing WHERE housingID = _housingID;
SELECT count(*) INTO _checkInitInvoice FROM invoice WHERE leaseID = _leaseID;
SELECT securityDeposit INTO _securityDeposit FROM lease WHERE leaseID = _leaseID;
```

```
SELECT SUM(totalDue) INTO _aggregateDue FROM invoice WHERE leaseID = _leaseID GROUP
BY status = 'billed';
IF _spotID IS NULL THEN
SET _spotRent = 0;
ELSE
SELECT classification INTO _classification FROM parking_spot WHERE spotID = _spotID;
SELECT rent INTO _spotRent FROM parking_fees WHERE classification = _classification;
END IF;
SET _totalDue = _rent + _spotRent;
SET _otherCharges = 0;
IF _paymentOption = 'Monthly' THEN

IF _checkInitInvoice = 0 THEN
INSERT INTO invoice VALUES(NULL, _peopleID, _rent, _spotRent, 0, 0, _totalDue, _leaseID,
_entryDate, _firstDueDate, _placeID, _roomNo, _housingID, _housingname, NULL, 'NONE',0, 0,
'billed' );
ELSE
SELECT CURDATE() INTO _curDate;
SELECT DATE_ADD(billDate, INTERVAL 1 MONTH), DATE_ADD(dueDate, INTERVAL 1 MONTH)
INTO _nextBillDate, _nextDueDate FROM invoice WHERE leaseID = _leaseID ORDER BY dueDate
DESC LIMIT 1;
IF _curDate = _nextBillDate THEN
IF _finalInvoice = _nextBillDate THEN
SET _currentBill = _totalDue ;
IF _securityDeposit < _aggregateDue THEN
SET _otherCharges = _aggregateDue - _securityDeposit;
SET _refund = 0;
ELSE
SET _securityDeposit = _securityDeposit - _aggregateDue;
IF _currentBill > _securityDeposit THEN
SET _refund = 0;
SET _currentBill = _currentBill - _securityDeposit;
ELSE
SET _refund = _securityDeposit - _currentBill;
SET _currentBill = 0;
END IF;
END IF;
SET _totalDue = _currentBill + _otherCharges;
INSERT INTO invoice VALUES(NULL, _peopleID, _rent, _spotRent, 0, _otherCharges, _totalDue,
_leaseID, _finalInvoice, _nextDueDate, _placeID, _roomNo, _housingID, _housingname, NULL,
'NONE', 0, _refund, 'billed' );
ELSE
```

```
SELECT DATE_ADD(billDate, INTERVAL 1 MONTH), DATE_ADD(dueDate, INTERVAL 1 MONTH)
INTO _nextBillDate, _nextDueDate FROM invoice WHERE leaseID = _leaseID ORDER BY dueDate
DESC LIMIT 1;
SET _totalDue = _totalDue + _otherCharges;
INSERT INTO invoice VALUES(NULL, _peopleID, _rent, _spotRent, 0, _otherCharges, _totalDue,
_leaseID, _nextBillDate, _nextDueDate, _placeID, _roomNo, _housingID, _housingname, NULL,
'NONE', 0, 0, 'billed' );
END IF;
END IF;
END IF;
ELSE
IF _paymentOption = 'Semester' THEN
IF _checkInitInvoice = 0 THEN
SET _currDate = _entryDate;
IF MONTH(_currDate) <= 7 AND MONTH(_currDate) >= 6 THEN
SET _prevPeriod = 5;
SET _period = 2;
SET _nextPeriod = 5;
ELSE
IF MONTH(_currDate) <= 5 AND MONTH(_currDate) >= 1 THEN
SET _prevPeriod = 5;
SET _period = 5;
SET _nextPeriod = 2;
ELSE
SET _prevPeriod = 2;
SET _period = 5;
SET _nextPeriod = 5;
END IF;
END IF;

ELSE
SELECT billDate INTO _currDate FROM invoice WHERE leaseID = _leaseID ORDER BY dueDate
DESC LIMIT 1;
IF MONTH(_currDate) <= 7 AND MONTH(_currDate) >= 6 THEN
SET _prevPeriod = 2;
SET _period = 5;
SET _nextPeriod = 5;
ELSE
IF MONTH(_currDate) <= 5 AND MONTH(_currDate) >= 1 THEN
SET _prevPeriod = 5;
SET _period = 2;
SET _nextPeriod = 5;
ELSE
SET _prevPeriod = 5;
```

```
SET _period = 5;
SET _nextPeriod = 2;
END IF;
END IF;
END IF;
SET _rent = _rent * _period;
SET _spotRent = _spotRent * _period;
SET _totalDue = _rent + _spotRent + _otherCharges;
IF _checkInitInvoice = 0 THEN
IF _entryDate = DATE_SUB( DATE_ADD(_exitDate, INTERVAL 1 DAY), INTERVAL _Period
MONTH) THEN
SET _currentBill = _totalDue ;
IF _currentBill > _securityDeposit THEN
SET _refund = 0;
SET _currentBill = _currentBill - _securityDeposit;
ELSE
SET _refund = _securityDeposit - _currentBill;
SET _currentBill = 0;
END IF;

SET _totalDue = _currentBill + _otherCharges;
INSERT INTO invoice VALUES(NULL, _peopleID, _rent, _spotRent, 0, 0, _totalDue, _leaseID,
_entryDate, _firstDueDate, _placeID, _roomNo, _housingID, _housingname, NULL, 'NONE',0,
_refund, 'billed' );
ELSE
INSERT INTO invoice VALUES(NULL, _peopleID, _rent, _spotRent, 0, 0, _totalDue, _leaseID,
_entryDate, _firstDueDate, _placeID, _roomNo, _housingID, _housingname, NULL, 'NONE',0, 0,
'billed' );
END IF;
ELSE
SELECT DATE_ADD(billDate, INTERVAL _prevPeriod MONTH), DATE_ADD(dueDate, INTERVAL
_prevPeriod MONTH) INTO _nextBillDate, _nextDueDate FROM invoice WHERE leaseID =
_leaseID ORDER BY dueDate DESC LIMIT 1;
IF _curDate = _nextBillDate THEN
IF _nextBillDate = DATE_SUB( DATE_ADD(_exitDate, INTERVAL 1 DAY), INTERVAL _period
MONTH) THEN
SET _currentBill = _totalDue ;
IF _securityDeposit < _aggregateDue THEN
SET _otherCharges = _aggregateDue - _securityDeposit;
SET _refund = 0;
ELSE
SET _securityDeposit = _securityDeposit - _aggregateDue;
IF _currentBill > _securityDeposit THEN
SET _refund = 0;
```

```
SET _currentBill = _currentBill - _securityDeposit;
ELSE
SET _refund = _securityDeposit - _currentBill;
SET _currentBill = 0;
END IF;
END IF;
SET _totalDue = _currentBill + _otherCharges;
INSERT INTO invoice VALUES(NULL, _peopleID, _rent, _spotRent, 0, _otherCharges, _totalDue,
_leaseID, _nextBillDate, _nextDueDate, _placeID, _roomNo, _housingID, _housingname, NULL,
'NONE', 0, _refund, 'billed' );
ELSE
INSERT INTO invoice VALUES(NULL, _peopleID, _rent, _spotRent, 0, _otherCharges, _totalDue,
_leaseID, _nextBillDate, _nextDueDate, _placeID, _roomNo, _housingID, _housingname, NULL,
'NONE', 0, 0, 'billed' );
END IF;
END IF;
END IF;
END IF;
END IF;
END IF;
END //
DELIMITER ;
CALL AddNewInvoice(1);
```

4. For each available parking spot, print its information (id, lot, location and type)

```
SELECT A.street,A.city,A.country,A.zipcode, S.lotID, S.spotID, S.classification
FROM Parking_Spot S, Address A, Housing H, Nearby N
WHERE S.availability=1 AND A.addrID=H.addrID AND H.housingID=N.housingID AND
N.lotID=S.lotID;
```

Retrieval Queries

1. Find the most popular hall or apartment (hall or apartment with the most number of requests approved, pending, or denied)

```
SELECT h1.housingID, h1.housingName FROM housing h1 WHERE h1.housingID=(SELECT
h.housingID
FROM Housing h, Lease_Request l WHERE h.housingID = l.preference1 OR h.housingID =
l.preference2 OR h.housingID = l.preference3 GROUP BY h.housingID ORDER BY COUNT(*) DESC
LIMIT 1);
```

2. Find all students who have paid their rent after the due date for any of the past three months.

```
SELECT p.peopleID, p.firstName, p.lastName
FROM invoice i, people p
WHERE p.peopleID=i.peopleID AND i.billDate > i.paymentDate AND DATEDIFF(i.billDate,
i.paymentDate) <= 90 ;
```

3. Print information about pending lease requests where student have not requested a parking spot.

```
SELECT * FROM Lease_Request l WHERE l.status LIKE '%Pending%' AND l.parkingNeeded=0;
```

Use Cases

1. Login - Student/Guest/Admin

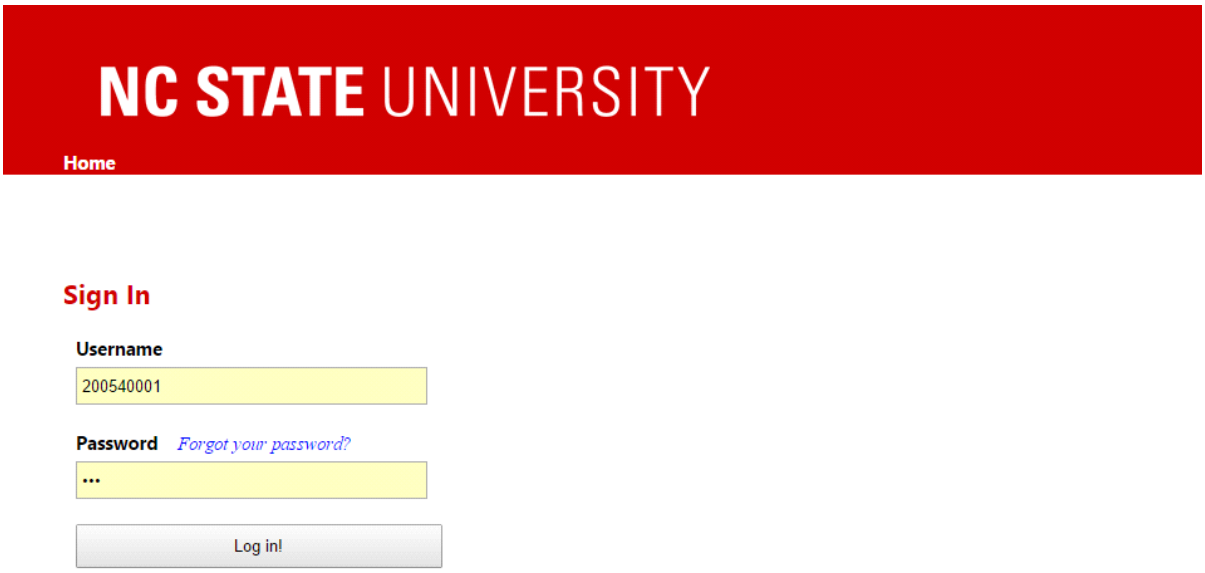

Figure 2: Student Login

Use Case Name	Login
Use Case Goal	To be able to log into the application
Preconditions	The user has username and password in the system
Result	The user is able to successfully log into the system if valid credentials are provided as input. The user is unable to log into the system if the credentials are not correct.
Primary Users	Admin, Student, Guest

2. View Current Invoice:- Student

Guest Section

Housing
Parking
Maintenance
Profile

Back

Figure 3: Guest Options

View Current Invoices

Invoice No	Student ID	Total Rent	Total Parking Rent	Late Fees	Other Charges	Total Due	Lease ID	Billing Date	Due Date	Place ID	room No	House Name	Payment Made On	Method Payment	Final Charge Deducted	Refund	Status
8	200540001	350	35	0	0	385	8	2014-03-01	2014-03-11	4	1	Ravenclaw		NONE	0	0	billed

View Former Invoices
Pay Invoice

Figure 4: View current Invoice

Use case Name	View current invoice
Goal In Context	To be able to view the information present in current invoice, like Billing Due Date, Housing Rent, Parking Rent, etc.

Pre Condition	The student / guest should be logged into their user account and should have selected the housing option and in that view current invoice option.
Post Condition	Student is able to view the most recent invoice and its details like Parking Rent, Housing Rent, Billing Date, etc.
Normal Flow	1. User clicks on the housing options button. 2. In housing option he clicks on the view invoice button. 3. In view invoice he clicks on the view current invoice button.
Assumptions	The user is authenticated and logged into his account.

Similar use case scenario is applicable for other options in the housing options menu.

3. View current Parking Spot- Student/Guest

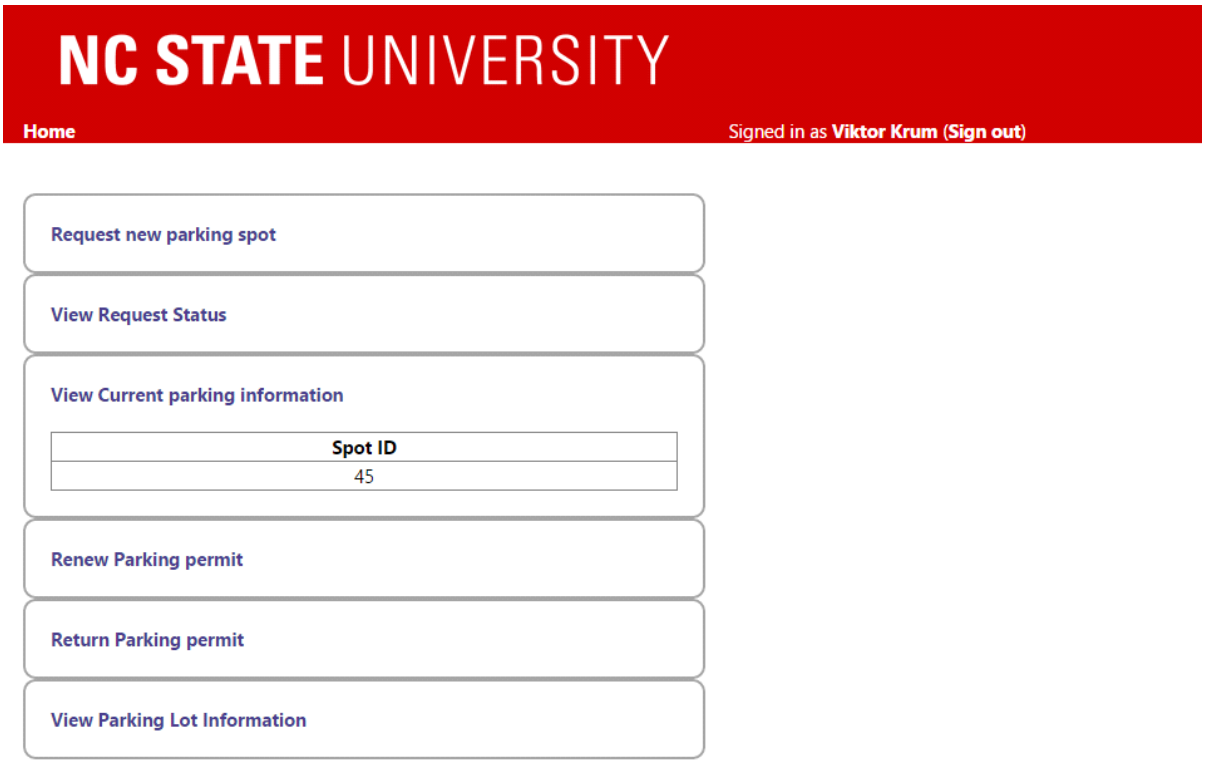

Figure 5: View current parking Information

Use case name	View current Parking Spot

Goal in context	To view the spot ID of the current parking spot allotted to the user
Pre condition	1. The Student/Guest must be logged in. 2. He must have selected the parking option button from the menu. 3. He must have selected the view current parking spot option.
Post condition	User is able to view the parking spot.
Normal Flow	1.User logs into his account 2. He selects the parking option 3. From the parking options menu he selects the view parking spot
Assumptions	

4. Update Profile :- Student/Guest

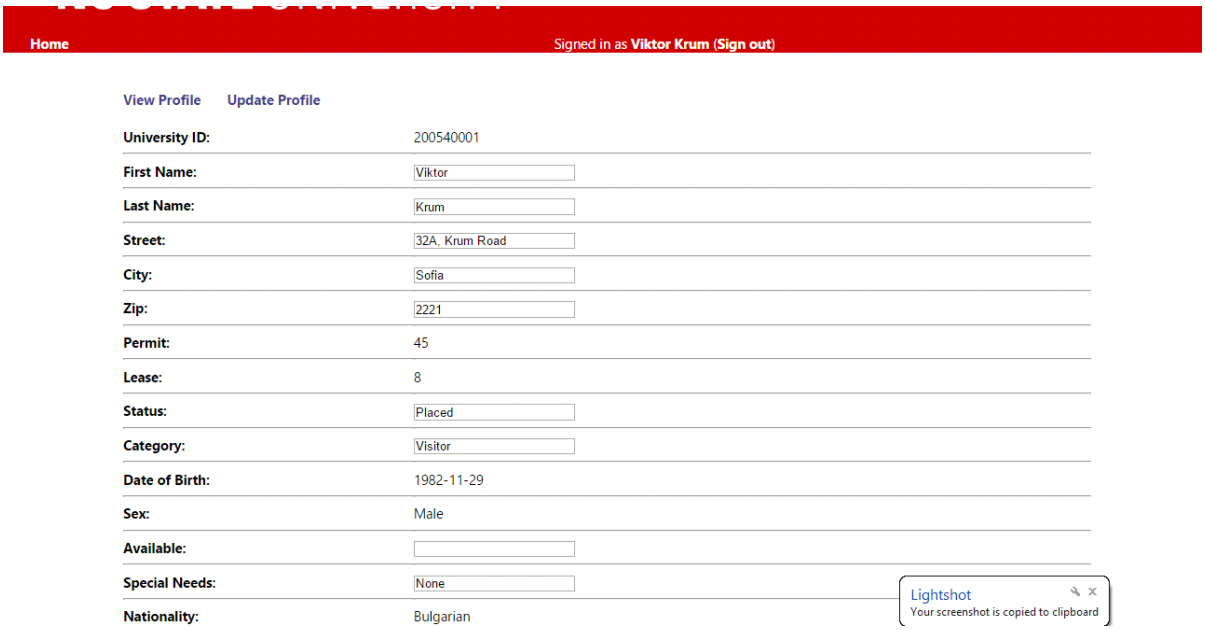

Figure 6: Student/Guest Update profile

Use case Name	Updating Profile
Goal in context	To update the fields in the student profile

Pre Condition	1.The student/guest must be logged into his account.
Post Condition	The student is able to change the values of different attributes associated with his ID.
Normal flow	1.After Logging into his account the user selects the option Profile 2.In the profile menu the user selects the option of update profile.
Assumptions	

5. View Maintenance Ticket :- Administrator

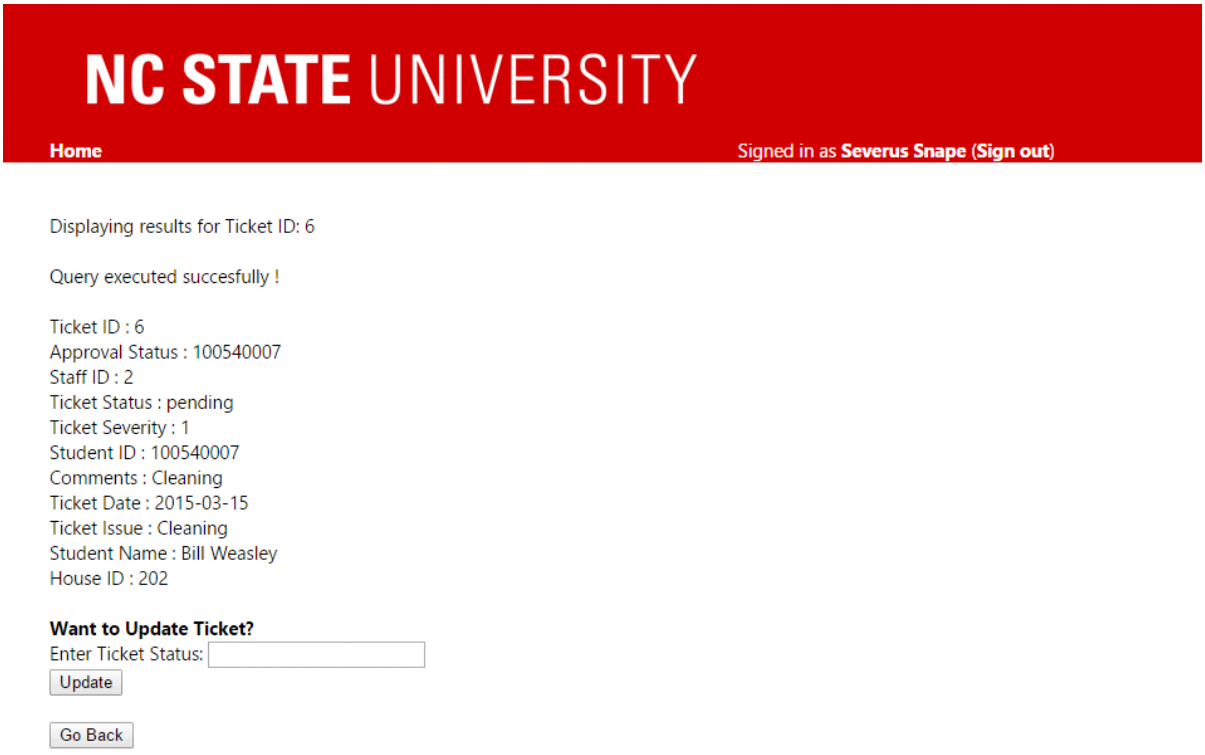

Figure 7: Maintenance Ticket Display

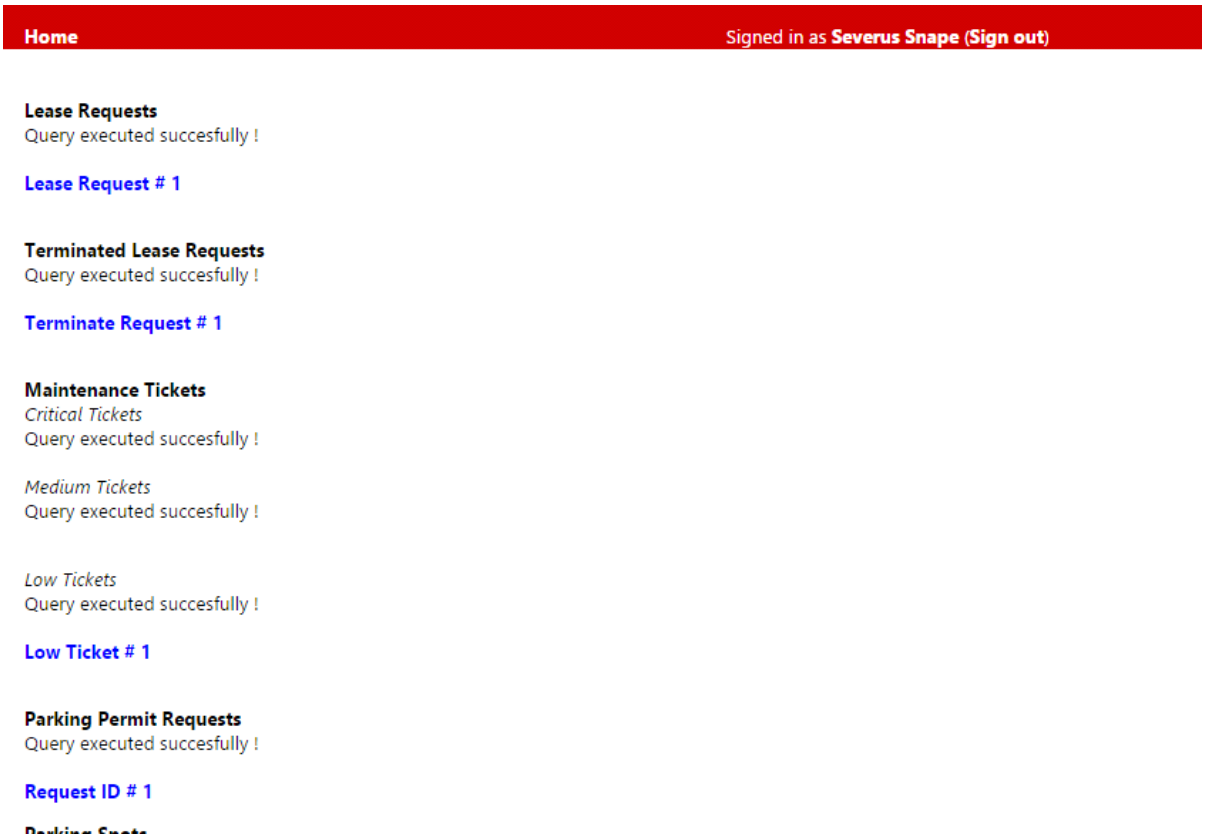

Figure 8:Maintenance ticket window

Use case Name	View Maintenance Ticket
Goal in context	To view the Maintenance tickets raised by users and to process the tickets
Pre Condition	The administrator must be logged into his account.
Post Condition	The administrator is able to view the ticket requests and process them
Normal flow	1. After Logging into his account the administrator selects the option view maintenance tickets. 2. He then is able to see the tickets according to their priority.
Assumptions	There are no such assumptions

Extra Credit Executed

New Lease Request

Terminate Lease Request

Find Roommate with best matches !

Name: Draco Malfoy
ID: 100540002
Housing ID: 101
Housing Name: Ravenclaw
Likeness Rating: 3

Figure 9: Roommate Search Result 1

New Lease Request

Terminate Lease Request

Find Roommate with best matches !

Name: Fred Weasley
ID: 100540005
Housing ID: 103
Housing Name: Hufflepuff
Likeness Rating: 5

Figure 10:Roommate Search Result 2

Stored Procedure to Find Roommates

```
DELIMITER //

CREATE PROCEDURE roommatematching(peopleIDinput int)
BEGIN

        DECLARE _sex varchar(20);
        DECLARE _dateOfBirth DATE;
        DECLARE _nationality varchar(20);
        DECLARE _category varchar(20);
        DECLARE _smoker varchar(20);
        DECLARE _family varchar(20);
        DECLARE _peopleID int;
        DECLARE _roommateName varchar(20);
        DECLARE _kin varchar(20);
        DECLARE finished int;
        DECLARE xkin varchar(20);
        DECLARE xfamily varchar(20);
        DECLARE xsex varchar(20);
        DECLARE xdateOfBirth DATE;
        DECLARE xnationality varchar(20);
        DECLARE xcategory varchar(20);
        DECLARE xsmoker varchar(20);
        DECLARE lotCursor CURSOR FOR SELECT peopleID, family, sex, dateOfBirth, nationality, category,
        smoker from people WHERE peopleID <> peopleIDinput;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

        SET finished = 0;
        SELECT name INTO xkin FROM kin WHERE peopleID = peopleIDinput;
        SELECT family, sex, dateOfBirth, nationality, category, smoker INTO xfamily, xsex, xdateOfBirth,
        xnationality, xcategory, xsmoker from people WHERE peopleID = peopleIDinput;
        OPEN lotCursor;
        loop_label:     LOOP
                FETCH lotCursor INTO _peopleID, _family, _sex, _dateOfBirth, _nationality, _category,
        _smoker;

                IF finished = 1 THEN
                LEAVE loop_labele
                END IF;
```

```
SELECT name INTO _kin FROM kin WHERE peopleID = _peopleID;
        IF xkin = _kin THEN
        UPDATE likeness SET likeness = likeness + 1 WHERE peopleID = _peopleID;
                END IF;
                IF _family = xfamily THEN

                                        UPDATE likeness SET likeness = likeness + 1 WHERE
peopleID = _peopleID;

                                END IF;

                                IF _sex = xsex THEN

                                        UPDATE likeness SET likeness = likeness + 1 WHERE
peopleID = _peopleID;

                                END IF;

                                IF _dateOfBirth = xdateOfBirth THEN

                                        UPDATE likeness SET likeness = likeness + 1 WHERE
peopleID = _peopleID;

                                END IF;

                                IF _nationality = xnationality THEN

                                        UPDATE likeness SET likeness = likeness + 1 WHERE
peopleID = _peopleID;

                                END IF;

                                IF _category = xcategory THEN

                                        UPDATE likeness SET likeness = likeness + 1 WHERE
peopleID = _peopleID;

                                END IF;

                                IF _smoker = xsmoker THEN

                                        UPDATE likeness SET likeness = likeness + 1 WHERE
peopleID = _peopleID;

                                END IF;
```

```
                END LOOP loop_label;

        CLOSE lotCursor;

    SELECT * FROM likeness WHERE likeness = (SELECT MAX(likeness) FROM likeness) LIMIT 1;

END //
```