Bellevue College Residence Hall Database Management System

Design Development Report

November 28, 2018

Report prepared by:

Derek Avery, Evgeniya Kokareva, and Stuart Larsen

DES

01000100 01000101 01010011

SOFTWARE CONSULTANTS

Table of Contents

1 - Introduction 2

2 - EER Model 2

2.1 - Overview 2

2.2 - EER Model Entities 3

2.2.1 - person 3

2.2.2 – application 4

2.2.3 – lease 5

2.2.4 – bed 5

2.2.5 – bed_type 6

2.2.6 – maintenanceRequest 6

2.3 - Relationships 7

2.3.1 – student –submits – application 7

2.3.2 – application –becomes – Lease 7

2.3.3 – lease –has- bed 7

2.3.4 – lease –makes- maintenanceRequest 8

2.3.5 – bed –associated with- bed_type 8

2.3.6 – application –has preference- bed_type 8

3 – Relational Model Mapping 9

4 – SQL 9

5 - References 10

1 - Introduction

Due to the recent opening of the Bellevue College Residence Hall (BCRH), it has become necessary to design a database management system (DBMS) to manage information pertaining to leasing and occupancy of units in the BCRH. The purpose of this report is to elaborate on the schematic design of the DBMS schema by DES Engineering Consultants (DES) up to the date on which this report was submitted.

Design to this point has included development of an enhanced entity-relationship (EER) model for the database. The EER model was designed to include entities which store all data associated with the BCRH reservation system. The model was carefully designed to allow data to be entered, retrieved, and filtered in such a way that the final database application can be used to provide the functionality specified in the design requirements (Appendix A). Additional information about design choices during development of the EER model may be found in the "EER Model" section of this report.

After designing the EER model, an additional step was taken to map the model to a valid relational database schema. During this step, care was taken to confirm that mapping did not result in loss of information, violation of EER model constraints, or other deviations from the design of the EER model. Additional information about this mapping process may be found in the "Relational Model Mapping" section of this report.

Finally, this schema was used to develop a script in Structured Query Language (SQL) which may be used to create a relational database using a DBMS; in this case, the SQL commands were selected to be compatible with MySQL software.

2 - EER Model

2.1 - Overview

The EER model for this project has gone through multiple iterations, with modifications being made at each stage to refine the model to meet design constraints. Some of the preliminary EER sketches may be seen in Appendix B.

Ultimately, the final design of the EER model has been refined to accomplish several things, including:

- Accurate and complete storage of information needed for the BCRH reservation system
- Including meaningful relationships between different entities to allow data retrieval as required by the design specifications
- Minimize the duplication of data, or inclusion of non-necessary information which would unecessarily increase storage requirements

The final EER model established after these multiple design iterations is located in Appendix C. The primary design decisions for this EER model are outlined in greater detail in the following subsections. Additionally, you can refer to Appendix D to find a full list of entity types, their attributes, and a description of each attribute.

2.2 - EER Model Entities

One of the primary concerns during the design of the EER model was selection of non-redundant entities to store in the database. As mentioned above, selection of these entities was performed to minimize duplicate data, separate unrelated information, and logically organize data of interest to database administrators. The following entities were selected for the BCRH database system:

2.2.1 - person

As the name implies, the 'person' entity type represents a person in the BCRH system. The person entity type is further inherited by two child entity types (student and administrator) to separate these different user types. The child entities are mutually exclusive (i.e. an administrator can NOT be a student) which is why this inheritance is shown as disjoint.

It is assumed in this design that this system will be integrated with the primary Bellevue College student/staff database system for authentication, hence the lack of username and password attributes. Additionally, it is the understanding

of DES that the identification numbers for students and staff are unique and managed identically (that is, student ID's and staff ID's are not distinguished by, or managed separately in, the primary Bellevue College database). For this reason, this attribute was chosen as the primary key for the 'Person' entity type.

Information specified in the design specifications (Appendix A) such as name, social security number, etc. are included as attributes for the 'person' entity type except for those which are specific to an admin or student; in such cases, the attributes were included in the child entities. Also note that students are stored in a single table, rather than dividing them further into current students and alumni; this is accomplished via the inclusion of a graduation date field, which allows the status of a student as current/alum to be determined.

2.2.2 – application

The application entity type represents an individual application, from one student, to the BCRH system. Each applicant may submit multiple applications to live in the residence hall. It has also been decided that for this design, applications' 'application_date' attribute will be *modified* when applicants revise their applications, in order to prevent applicants from securing higher priority during a single application period if their application preferences change. Additionally, the application includes a 'start_quarter' and 'end_quarter' to allow applicants to choose the period of their stay.

It is also worth noting that during the initial design of the EER model, consideration was given to the idea of managing 'Leases' only and discarding 'Application' records after leases are accepted. However in order to store and allow access to data in accordance with the design specifications, it was determined that for the purposes of this system it makes more sense manage and store individual applications in addition to accepted leases. This decision was made to allow the system to retrieve statistics about all applicants regardless of whether or not they accepted or received a lease. This was done to maintain conformance with design specifications, which note that applicant data (rather than resident data) must be retrievable by system administrators.

The application entity type was designed to store attributes which do not pertain to individual leases, such as bed type preference and in the case of students with a preferred roommate, the preferred roommate BC_ID.

Therefore, the attributes stored in each instance of the 'application' entity contain information which only pertains to the application; the concrete details of each lease will instead be stored in 'Lease' entity types, which are discussed in greater depth in the next subsection. Each application entity also has a unique application ID, which functions as the primary key in the resulting entity table.

2.2.3 – lease

After initial consideration of managing leases for individual units, it was determined that a better approach for this database system would be to manage leases for individual student. This decision was made because it is the understanding of DES that housing contracts at Bellevue College are managed on a per-resident basis; that is, there are individual contracts made between BCRH administration and each individual student. Additionally, each lease a student begins is stored individually, and the system may retrieve non-current leases a student has made in the past in addition to each current lease. This allows the system to retrieve statistics about current and non-current leases, which is a design feature specified in the design specifications in Appendix A.

Leases have attributes which characterize each lease such as lease period (start/end date), cost, and a unique lease id which functions as the primary key.

2.2.4 – bed

Rather than managing information about each apartment/suite unit in the new BCRH, it was determined that a more optimal solution would be to store information about each individual bed. This accomplishes several things. For one, it facilitates a simpler way of tracking whether or not a space is available or not. For example, if a unit were to have four beds, the system would need to track the availability status of each individual bed, which could be difficult to implement. Also, if units were tracked as an aggregate, additional information about the different bed types (e.g. if they are in a shared or individual room, whether they can house married couples, etc.) would need to be stored with each unit instance, AND for each bed in the unit.

This type of information must be tracked in order to provide the required functionality per the design specifications in Appendix A, searching for a unit which satisfies an applicant's preferences. However, storing this much

information in a single entity would lead to a very complex system. Ultimately, it was decided that tracking individual beds in addition to unit configuration information would offer a less complex solution. In this design, it is possible to track each bed's status as available/unavailable with one of multiple, specific values. It also allows the system to store attributes about each bed which factor into lease assignment such as whether or not the bed is in a shared unit or if it can permit married couples. As each bed may exist in either a shared or single unit, each bed in a unit is given a unique number, and this number in combination with the unit number (elaborated further in section 2.2.5) serves as a unique primary key for each bed instance.

2.2.5 – bed_type

As discussed in section 2.2.4, several attributes about the residence bed_type configurations must be stored in order to offer students accommodations consistent with their preferences. However, it was also noted that it did not make sense to store this information in the same entity as beds. Therefore, for this database it was determined that the information about each bed_type configuration such as number of rooms and whether or not the residence is in an apartment/suite, could be stored in a unique entity to order data in a more logical way. The bed_type entities (which may be thought of as configurations) are each assigned a unique unit ID, which serves as the primary key for each bed_type entity. This offers greater flexibility to define new and unique variations to each unit type (e.g. to specify whether or not a unit is ADA accessible) which could be used to refine this reservation system in the future.

2.2.6 – maintenanceRequest

In accordance with the design specifications, one additional entity type is required in this EER model to track maintenance requests made by occupants in the BCRH. These maintenance request entities each contain attributes which characterize the nature of the request, such as the open/close date, the names of technicians who resolved each request, and a unique maintenance request ID to serve as a primary key. Additionally, note that these maintenance requests will be stored even after resolution in the same way that each of the other five entity types will be stored indefinitely (that is, until a database administrator decides they are no longer applicable).

2.3 - Relationships

In the EER model developed for this database system, as with any EER model, there are several relationships between the different entities. These relationships have been designed to create a model with logical links between related entity types. The relationships described below are consistent with the relationships shown visually in Appendix C.

2.3.1 – student –submits – application

The first relationship of note is that between the student entity and the application entity. Each student may submit applications using their registered account, and as this database system must provide means of retrieving information about specific students' leases and applications, there must be a relationship between each student entity and their applications. This relationship between students and applications is **one to many**, as one student may make several applications during different time periods. Additionally, the application entity type has mandatory participation in this relationship as an application cannot exist without a student.

2.3.2 – application –becomes – Lease

Similarly to how students must be related to their applications, so too must students be related to any leases they sign onto in the BCRH. However, as applications correspond directly to any given application, it was determined that a relationship between each application and the resulting lease (if any) should be maintained rather than a direct relationship with each student. In any case, due to the relationship between students and applications, students and leases may still be related indirectly in the database. As any one application can result in (at most) one lease, this relationship is **one to one**. Additionally, the lease has mandatory participation in this relationship, as a lease cannot exist without an application being submitted.

2.3.3 – lease –has- bed

Due to the fact that individual students' make leases rather than leasing a unit as a whole, each lease must relate to an individual bed in the BCRH. This must be the case because any student's lease is for one (and only one) specific bed.

Therefore, there is a relationship between each lease and one bed. Again, this relationship is **many to one** as any one bed can be associated with many leases since a bed can be leased at different times. In this relationship, the lease has mandatory participation because each lease must have a bed associated with it.

2.3.4 – lease –makes- maintenanceRequest

Maintenance requests are to be stored by this database, and it is desirable for this system to be capable of retrieving historical information pertaining to the maintenance of a unit or the person(s) making request. For this reason, it is deemed necessary to have a relationship between either a student or a lease and each maintenance request. It was decided for this design that relating the lease entities to maintenance requests would provide the desired functionality. Due to the other relationships of the lease entity type, it is possible to query this system for any type of statistical data that might be desired (e.g. maintenance requests made by one student, for a specific unit, during a given term, etc.). This relationship is **one to many** because a unit might receive maintenance on multiple occasions during a lease period. The maintenance request entities have mandatory participation in this relationship, as a unit must be leased in order for maintenance requests to be made for it.

2.3.5 – bed –associated with- bed_type

Due to the choice to store the information about bed_type configuration separate from each available bed, it was necessary to make a relationship between each bed and which bed_type it corresponds to. This relationship allows each beds to be searched by type in order to match application preferences with the beds meeting this specification. This relationship is **many to one** as many different beds can exist with the same configuration. The bed entity has mandatory participation in this relationship because a bet must be of a specific type.

2.3.6 – application –has preference- bed_type

In order to log the preferred bed type of each application, a relationship was created between the application and bed type entities. This allows specific applications to be paired with the preferred bed types of each applicant, which

will be useful for database administrators while filtering available beds (e.g. they may wish to find all available beds of an applicant's first preference, and in order to know what an applicant's preference is, their application must be related to a bed type). This relationship is **many to many,** as applicants can have up to three preferences and many applications can specify the same bed type as a preference. The application has mandatory participation in this relationship because applicants must specify at least one preference in their applications.

3 – Relational Model Mapping

After establishing a complete EER model for this database, the next step taken was to map entities and relationships to a functional database schema. This step was necessary to convert the high-level database model to a format which is usable with traditional database management software (in this case, MySQL).

Mapping of relational the EER Model to a relational model was performed following the guidelines of Chapter 9 of the Fundamentals of Database Systems[1] text. The resulting relational database schema may be found in Appendix E.

The design intent while mapping was to create a simple and readable relational schema while minimizing null values, preserving data, and keeping data redundancy minimal. With those design decisions in mind, mapping the hierarchical entities (Person, Student, Admin) into unique relations maintains the design decisions listed above when analyzing the schema for data stored within those relations. The mapping of the binary relation types followed rules specified in the Fundamentals of Database such that relations each store primary and foreign keys to conform with the EER model design. Primary and foreign keys are indicated in Appendix C with underlined attribute names and connecting arrows, respectively.

4 – SQL

After establishing all of the higher-level design for the database (i.e. the EER model and relational model schema) the final step in this part of the design process was to create a series of SQL statements to create a database which

conforms with the abstract design. Primary concerns during this process were to include primary and foreign keys in a manner consistent with the relational schema, and to enforce domain constraints (e.g. only permitting 'A' or 'S' in the unit_type field of the bed_type relation). This portion of the design was performed following the rules specified in the database text[1].

The final SQL script which may be used to create a database consistent with this design is located in Appendix F.

5 - References

[1] Elmasri, Ramez, and Shamkant B. Navathe. *Fundamentals of Database Systems, 7th Ed*. Pearson Education, 2017.

Appendix A –
Design Specifications

CS331- Project Assignment
Problem Description

Bellevue College Student Housing
Reservation System

In September 2018, Bellevue college open the doors to on-campus housing! This building will be home to 400 students in a mix of units and includes a variety of public areas for studying, meeting and socializing. The building is designed to meet requirements for LEED Gold certification and will have many environmentally-friendly features. These include a rain garden, smart water systems, utility monitoring, permeable paving, and abundant natural lighting.

If you're a student, you have the option of living in a furnished Suite (1 or 2 bedroom) or Apartment (2 or 4 bedroom) — and can enjoy private kitchen and bathroom within your living space. **Take a Tour** to view room layouts and more!

- 24/7 staffing, including Front Desk operations, mail service, information and equipment rental
- Community spaces on each floor
- Laundry facility
- Wi-fi access
- Food services
- Free cultural, academic and community programs and events
- Student volunteer and job opportunities

This project is designed with 350+ beds in 137 units. The units consist a mix of Studios, 2 Bedroom, 2 Bedroom/4 bed (double occupancy) and 4-bedroom apartments (see table below). All units are designed with kitchens and private bathrooms inside the unit. The project also includes a main level lounge with a café' open to all students on campus and second level multi-purpose meeting space. The project features private study areas and several floor lounges.

The current system to apply for resident housing at Bellevue College, students have to complete online application then wait for acceptance and housing offer email confirmation. Instead we like to design a complete automated reservation system for Bellevue College on Campus resident housing

You have been asked to design a solution for needed automatic reservation system for Bellevue college housing. The system should allow complete automatic reservation as well as administrator features for creating reports. The admin needs to have a username and password.

The aim of the proposed system is to achieve:

- User can enjoy the cool and comfort User Interface (optional for this project and not part of the requirements)
- The applicant does not have to come to the administrator's office to make a reservation, so that the effort for the applicant can be reduced.
- There will be an account for each and every user which validates them based on their Bellevue college ID. As a result, the applicant does not have to give their information as many times as they visit.
- The information reusability can be easily achieved.
- The user data can be stored in a database.

- The administrator will have an account through which he can login and know the business statistics like revenue and profit

Here is a summary for the project requirements please make sure to read carefully and ask if you have any question to clarify.

The college Housing Office receives many applications from **Alumni**, **Current student** requesting housing on campus. The housing is available in two categories, Suite and apartment as listed below.

Category	Options	Open for Marriage couple (Yes/No)
Apartment	• Two Bedroom Apartment, Four Persons • Four Bedroom Apartment, Four Persons	• No • No
Suits	• One Bedroom Suite, One Person • One Bedroom Suite, Two Persons • Two Bedroom Suite, Two Persons • Two Bedroom Suite, Three Persons	• No • Yes • Yes • No

The assigned cost of each option is different see for the cost details.

All living units are furnished so as to impose less impact on the environment and to provide a move-in-ready space for students. Furnishing in suites includes a bed, desk, task chair and storage for each resident; in apartments, living room furniture is provided. The students can bring their own matters sheets and different matters sizes dimensions will be provided based on the room.

To be eligible to apply, you'll need to have been admitted to Bellevue College, or a recent Alumni of Bellevue college. As an option if there is someone whom a student would like to have as your roommate, both of the students must specify each other in their housing applications. The system will try to accommodate their request but no guarantee that they will be housed together. For two current students who are married and both are applied for housing they need to specify that in their application and apply for the suitable option.

Priorities in assigning based on application date. Current student has highest priority, then Alumni.

Applicants can mark three preferences for the suit and apartment type desired. Based on availability, they will be offered an apartment/Suit. Applicants can reject the offer. A non-refundable $25 housing application fee t will be placed on Student Account is required for each application. Only one application is permitted for each student. Married couples, however, can have two separate applications and both have to be enrolled students. Notification for vacating premises is required one month prior to the move-out date. After graduation, students are allowed to stay in the apartment at most one more month.

Current student has to have a valid Student ID that they use for login to complete the booking. The system is connected with the student information in the college to verify the id and place on student account any required fees

Alumni can use their student id to login and verify with the college alumni details.

For eligible married students' option, they have to request one-bedroom suite, two persons or two-bedroom suite, two persons to be placed together.

Once student has booked, the system allows two other features

- Cancel reservation
- Submit common maintenance request for any problem. Such as, elevators, heat, electricity, replacing light bulbs, central air-condition or Laundry machines

Below some information on possible ways on how the student and other users can use the system:

Students:

1. The student will login into the Online housing reservation System by entering his/her student number and password on the login screen.
2. The system authenticates the student
3. The system will show the students details and application form
4. The student checks his details
5. The student choose room and send the application form
6. The system checks the room availability and correspondence between the room chosen and the student's year of study.
7. The system provides the Student Application Reference Number.
8. Student prints the reference number
9. Student edit / cancel the application
10. The system update information
11. The system shows the updated information
12. The student logout

Administrators:
Scenario 1

1. The administrator login
2. The system authenticates the user
3. The system displays the payment page

4. The administrator enters the reference number of the student
5. The system retrieves the student information from the database
6. The system displays the student information
7. The administrator enters the amount for payment
8. The administrator delete / edit / search information about payment
9. The system updates the information
10. The system shows the updated information
11. Logout.

Scenario 2

1. The administrator login
2. The system authenticates the user
3. The system displays the main page
4. The administrator selects the student's year of study
5. The system displays the list of applicants in that year
6. The system shows the list of unoccupied rooms for that year
7. The administrator allocated rooms to students
8. The administrator search / edit / delete information
9. The system updates information
10. The system displays the information
11. The administrator sends the confirmation to the student.
12. Logout

The following are sample of the main entity types of the Housing department database and some of the main data you need to keep

Housing office keeps the following information about the

1. **Resident**: The main attributes are identification number, name, gender, marital status, address, telephone number, the SSN of the family head, name, address, telephone number, the name of the major college and department in which they are enrolled in.
2. **Applicant** (a student who has applied for on-campus housing but has not yet been assigned an apartment): The main attributes are identification number, name, gender, address, telephone number, marital status, college, department, preference about the village (the name of the village they want to stay in), preference about the housing (Suite, apartment, single bedroom, double bedroom) SSN of the applicant, name, address, telephone number, marital status, the name of the major college and department in which the family head is enrolled in, the preference about the apartment category.

3. **Maintenance Request**: The main attributes are address (this includes building and apartment numbers), description of the maintenance problem, name of the resident, submission date, date the maintenance problem was fixed, name of the employee responsible, etc.
4. **Apartment:** The main attributes are address, building number, and apartment number), number of bedrooms, Room status.
5. **Suit**: The main attributes are address, building number, and apartment number), number of beds, bed status.
6. **Administrator**: name, staff ID, password, SSN, department, phone number, gender, job title

Please note these are only sample of the needed information, you need to include in your design any more entities with clear explanation of your assumption to meet the design needs.

Here is some of the options of how the final application should be presented to help you understand your database design requirements and achieve the CRUD operation.

- If the option "Resident" is chosen, it will open a list of activities that involve residents, such as add a new resident, update the information about a current resident, delete the information about a resident from the database, check a resident's financial status, maintenance requests submitted, etc.
- If the option "Applicant" is chosen, it will open a list of activities that involve applicants, such as add a new applicant, update the information about a current applicant, delete the information about an applicant from the database, check the application status, etc.
- If the option "Maintenance" is chosen, it will open a list of activities that involve maintenance services, such as add a new maintenance request, update the status of a request, list the requests submitted or due on a particular date, etc.
- If the option "Demographical Studies" is chosen, it is an option for admins to create some statistics study on resident Demographics.
- If the option" Administrators" is chosen, the system will have database administrative functions: residents, applicants, maintenance orders, etc. These will allow the user to add, update, and delete information about residents, applicants, maintenance orders, etc.

The following are some of the queries, forms, and reports that the system can create to increase the functionality of the database.

Applicant

- The system should allow the user to check the availability of apartments in a particular category. It shows a list with all the available apartments for users to check availability of apartments in a particular category.
- Admins can request a list all the name and current address of the applicants that need an apartment next semester.
- Admins can request a list all the applicant who are in the waiting list
- In assigning an apartment, the Housing department prioritizes based on application date. Current student has highest priority, then graduate students. The system should sort the information about the applicants based on their priority level.

Maintenance

- Most of the apartments are vacated at the end of the quarter. On the checkout date, an employee should visit the apartment and check its status and inventory. The Housing department has a limited number of employees, and, therefore, for scheduling purposes it is necessary to know in advance which apartments will be vacant and when the checkout date is. List the addresses of the apartments that will be vacated at the end of the quarter together with the checkout dates.
- In order to help the department with scheduling for apartment maintenance service, prepare a list of the apartments that have submitted a maintenance order for a particular date.
- For maintenance department: every day, a report with the maintenance requests due and the requests that were not handled (overdue) is presented to the maintenance department. The data contains the name of the resident, apartment address, submission date, due date, and a short description of the problem to be handled.
- The Housing department will need to considering remodeling some of the apartments. Priority will be given to the apartments and Suits that have had the largest number of maintenance problems during the last year.

Administrative

The system will have database administrative functions: residents, applicants, maintenance orders, etc. These will allow the user to add, update, and delete information about residents, applicants, maintenance orders, etc.

- At the end of each semester, the Housing department prepares a list of all the apartments that will be available for the next semester. The list provides details about the address, of all the apartments and suits/beds available.
- List all the information about the applicants that presents for each applicant his/her marital status, degree pursued, application date, and apartment preferences

Resident

- List all the information about the residents and for each resident the corresponding total amount of money paid to the Housing department so far.

Demographics Studies

The Housing department would like to performing a demographical study of their residents. They are interested to know about the changes in the following:

- The number of married residents (per year) during the last 5years
- The number of undergraduate married residents (per year) during the last 5 years
- The number of unmarried residents (per year) during the last 5 years
- The number of female residents (per year) during the last 5 years
- List for each apartment the maintenance orders that are still due.
- The status of maintenance requests for a particular day.

CS331- Project Assignment
(Part B)
Bellevue College Student Housing Reservation System

After Modifying your EER conceptual model based on the feedback. Please update your relational schema for the housing system.

For this part of the project you are as a team requested to implement the housing Booking Reservation System application using Java.

a. Load some initial data into the database tables that you created in Part A of the assignment. You need to write a loading program, use MYSQL. Your data should be kept in files so that it can easily be reloaded during debugging. The data format should be designed by you.
b. Write queries to retrieve and print all the requirements assigned for your team and listed below.
c. Record a video and upload to YouTube to demonstrate your project tasks (1 and 2) as a team. Submit your YouTube video link

What to submit:

1. URL for the YouTube that demonstrate your project. Please record the video as a team and make sure everyone is presenting.
2. updated documentation that includes your final EER model after all the changes and modifications, Final Relational Model and The SQL DDL statements for the final relational Model

d. The source code for your application.
e. Complete list of what has been completed by each team member.

Connect your database with Java and create a java UI for the application, which includes the options listed below. Java GUI is not required, Console is sufficient.

You will implement the housing Booking Reservation System in Java using JDBC.
There are 3 kinds of database users:

1. Resident
2. Applicant
3. Admin

For the application UI, I will not include complete screen shots here as I want each group to think on how they want to present their application
Here is a sample for the main screen

```
*************************************************************************
                                ***********
                        Welcome to the Housing System
                                ************
*************************************************************************
                              1. Resident Login
                      2. Applicant Registration/Apply
                                  3. Admin
                                  4. Quit
Type in your option: 1
Enter User ID: raj
Enter password: raj
```

Figure 1: Main Menu interface option

```
    ************************************************************
             Welcome to Bellevue College Housing System
                     Administrators Staff

     ************************************************************
                         1. Mange Residents
                        2. Mange Applicants
                      3. Demographic Studies
                    4. Mange Maintenance orders
                     5. Administrative Reports
                              6. Quit
  Type in your option:
```

Figure 2: Admin option interface

<u>Task 1-(Registration process)</u> Booking request is submitted, approval by admin or automatic match to become a resident.

The project requires the student to implement the following member functions:
If the option" Applicants registration/Apply" is chosen, the system will allow

a. Users to check the availability of apartments in a particular category. It shows a list with all the available apartments for users to check availability of apartments in a particular category.
b. Submit Booking request- fill up application and choose one of the available housing categories
c. Receive confirmation message
d. After receiving confirmation can login to Residents login option in the main menu

Please Implement the above described functions. Please demonstrate and recode how the registration process can be completed for a new applicant using your system.

Here are suggested scenarios steps that can be used as a guideline for your implementation and demonstration

1. New applicant will choose option 2 to apply for the housing
2. The applicant should fill up a form with all the required details including the options the available type of housing that he/ she can choose from. Also, he should be able to list 3 preferences and preferred roommate if he/she likes to.
3. The application will insert the new applicant information in the system
4. The application will show a message confirming the receiver of the application

Please note these coming steps might be different from one team to another based on the design.

Option 1(Automatic matching)

5. The applicant will get the match automatically and it will be printed on the screen with the new assigned address.
6. The user now should be a resident and can return to the main screen and Login as a resident. The login details should be available on the system now, using same student id.
7. Record your Demonstration for the above steps

Option 2(Administrator approval)

5. The applicant will receive a message that his application is waiting for an administrator approval.
6. Follow the scenario by going to the main screen. Login as an admin.
7. If the option "Manage Applicant" is chosen, it will open a list of activities that involve applicants, such as add a new applicant, update the information about a current applicant, delete the information about an applicant from the database, check the application status, approve application Status.
8. Choose check applications status. Admins can request a list for all the applicant who are in the waiting list or pending approval.
   Please Note: In assigning an apartment, the Housing department prioritizes based on application date. Current student has highest priority, then graduate students. The system should sort the information about the applicants based on their priority level. This option should return all the applications that needs approval.
9. Get the application number.
10. Go back to admin menu. Choose the option to update the status.
11. Insert the application number and update the status to approved.

The application now is approved. The applicant has been inserted/updated as a resident.
Go back to main menu and as an admin, choose Manage resident option

12. Select recent registered residents, it will show a list of people who are recently became residents in the last 2 days. The list should include the applicant who just been approved.

Please note these are just a suggested step to complete the registration process. You can modify the steps as required. Make the steps matching your design needs. Please explain your steps in the demonstration.

Task 2: The team need to implement the functionality for the option listed below

If the option "Administrative Reports" is chosen (Option 5 in the suggested GUI figure 2)
This should show a list of available reports options

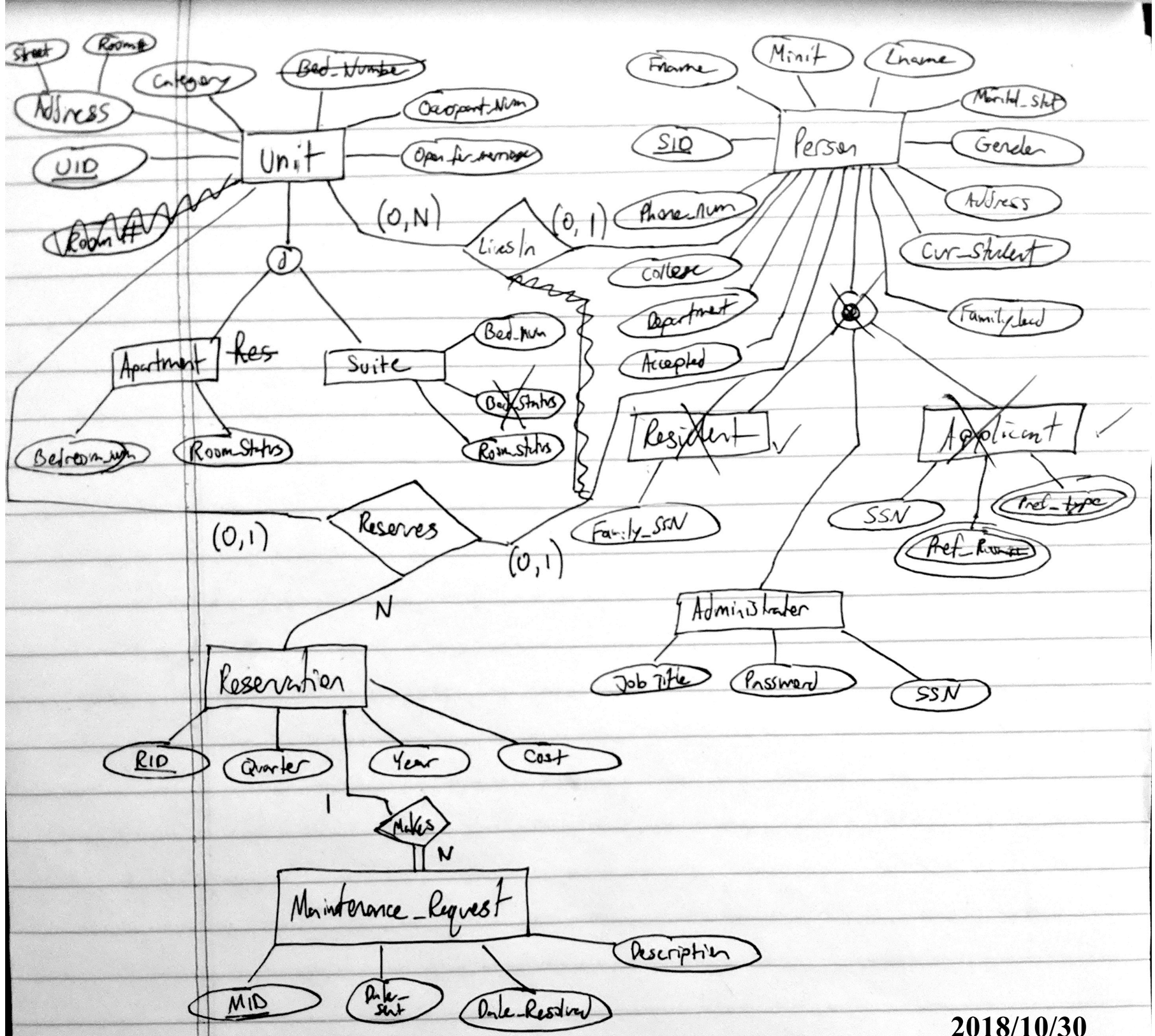
1. Housing department reports
2. Applicants Reports
3. Resident Reports
4. Maintenance department reports

Please implement and demonstrate

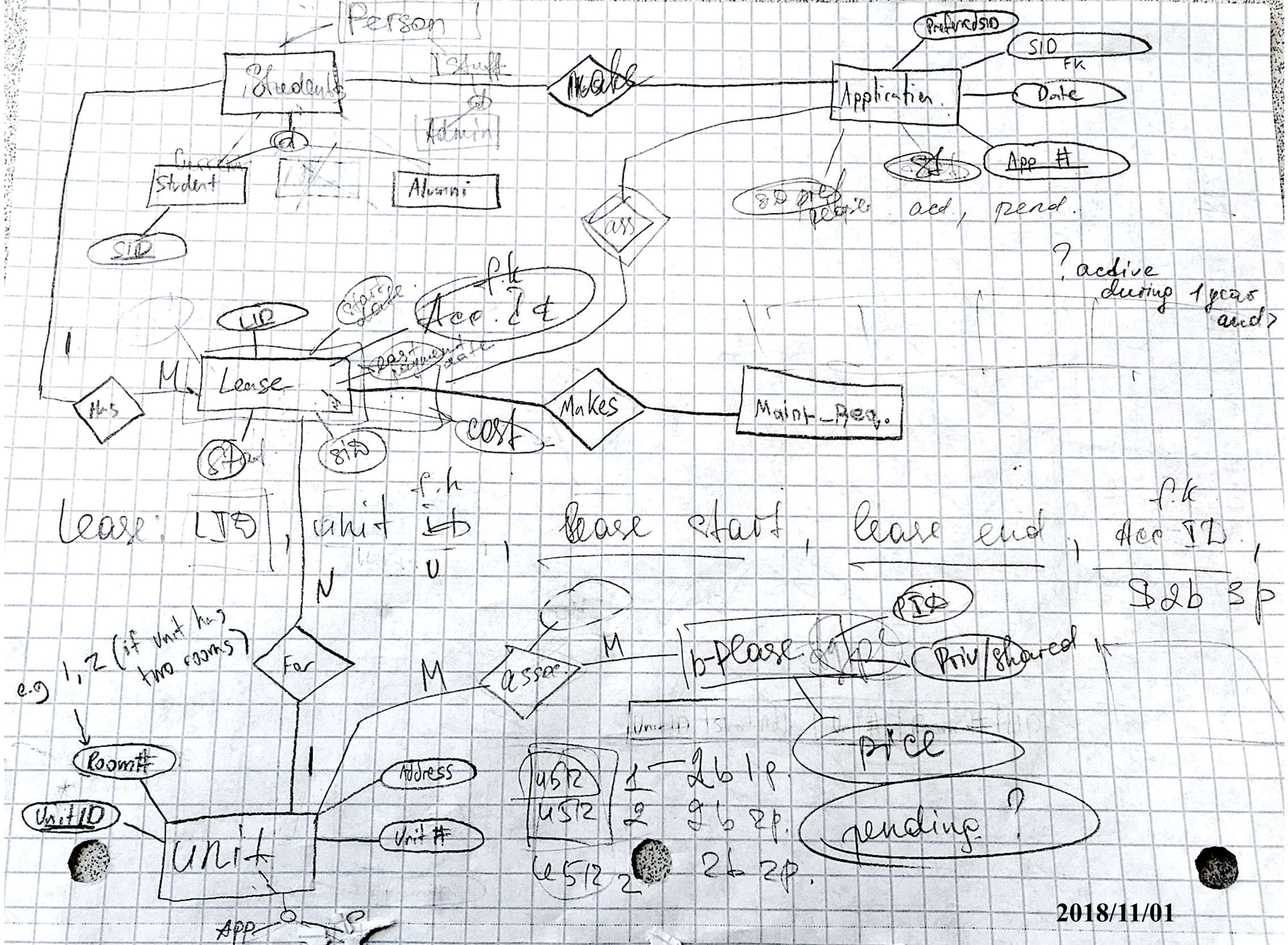
- When option 4 is chosen. Create a report for the case listed below

  Most of the apartments are vacated at the end of the quarter. On the checkout date, an employee should visit the apartment and check its status and inventory. The Housing department has a limited number of employees, and, therefore, for scheduling purposes it is necessary to know in advance which apartments will be vacant and when the checkout date is. **List the addresses of the apartments that will be vacated at the end of the quarter together with the checkout dates.**

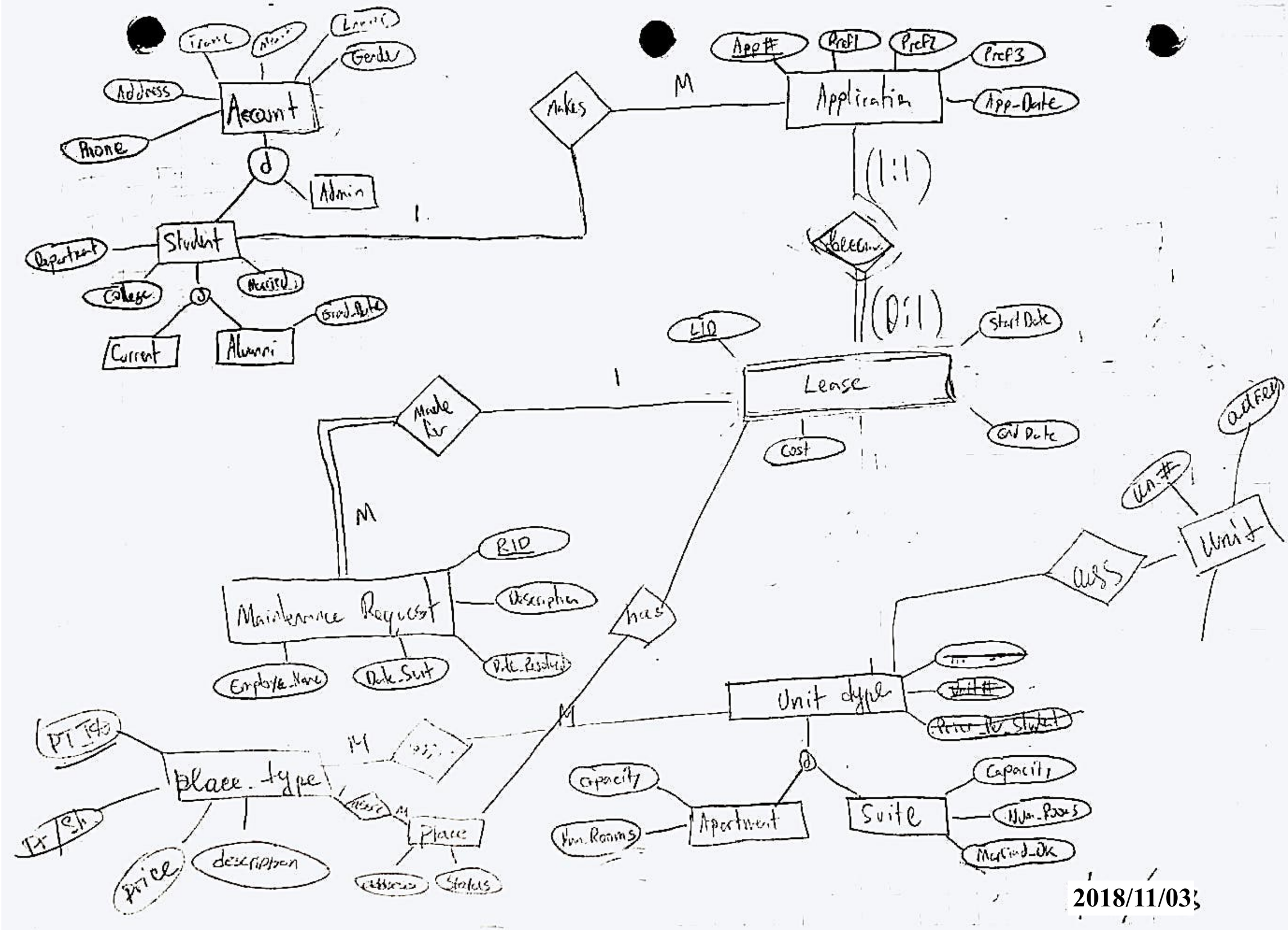
Appendix B –
EER Model Sketches


Street
Room#
Address
Category
Bed_Number
Occupant_Num
Open for_marriage
UID
Unit
Room #
(0,N)
Lives In
(0,1)
d
Apartment
Res
Suite
Bed_num
Bed_Status
Room_Status
Room_Status
Bedroom_num
(0,1)
Reserves
(0,1)
N
Reservation
RID
Quarter
Year
Cost
1
Makes
N
Maintenance_Request
Description
MID
Date_sent
Date_Resolved
Fname
Minit
Lname
Marital_Stat
SID
Person
Gender
Address
Phone_Num
College
Department
Accepted
Cur_Student
Family_Lead
Resident
Applicant
Family_SSN
SSN
Pref_type
Pref_Number
Administrator
Job Title
Password
SSN

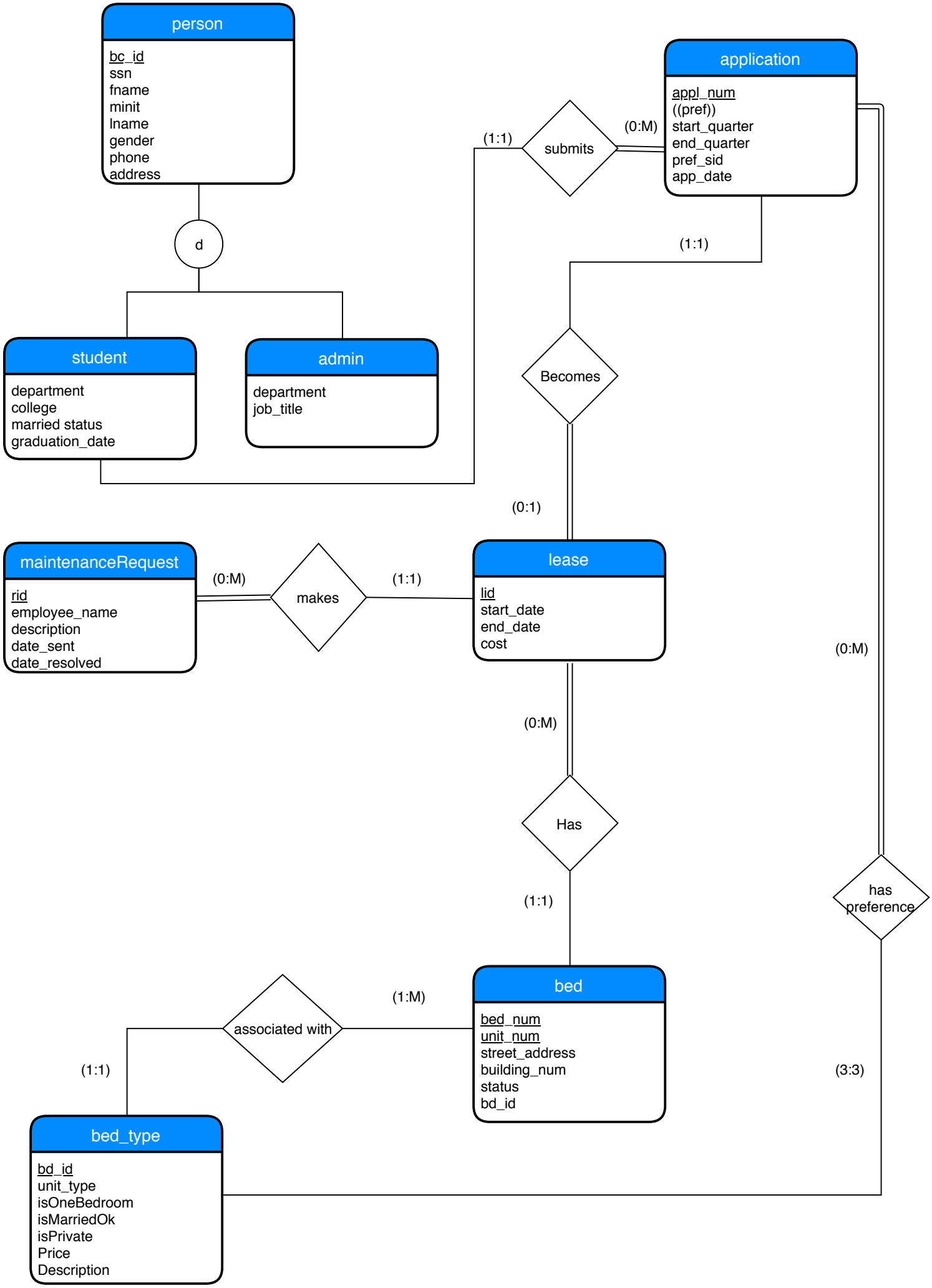
2018/10/30


Person
Student
Staff
Admin
Student
Alumni
SID
Makes
Application
PreferredSID
SID
FK
Date
App #
SD pref
acd, pend.
ass
? active during 1 year and >
f.k
Acc. Id
LID
Lease
M
Has
cost
Makes
Maint_Req.
f.k
lease: LID, unit ID, lease start, lease end, Acc ID,
$2b 3p
N
U
For
e.g 1, 2 (if unit has two rooms)
Room#
Unit ID
unit
Address
Unit #
assoc
M
M
b-Please
PID
Priv/shared
price
pending ?
4512 1 — 2b 1p.
4512 2 2b 2p.
4512 2 2b 2p.
APP

2018/11/01


Account
Address
Phone
Gender
d
Admin
Student
Department
College
Current
Alumni
Makes
M
1
Application
App#
Pref1
Pref2
Pref3
App-Date
(1:1)
(0:1)
Lease
LID
Start Date
End Date
Cost
Make for
M
Maintenance Request
RID
Description
Employee_Name
Date_Sent
Date_Resolved
has
M
Unit type
Place_type
PT_ID
price
description
Place
Status
Apartment
Capacity
Num_Rooms
Suite
Capacity
Num_Rooms
Married_OK
Unit
Unit #
assigned
assigned
2018/11/03

Appendix C – EER Model

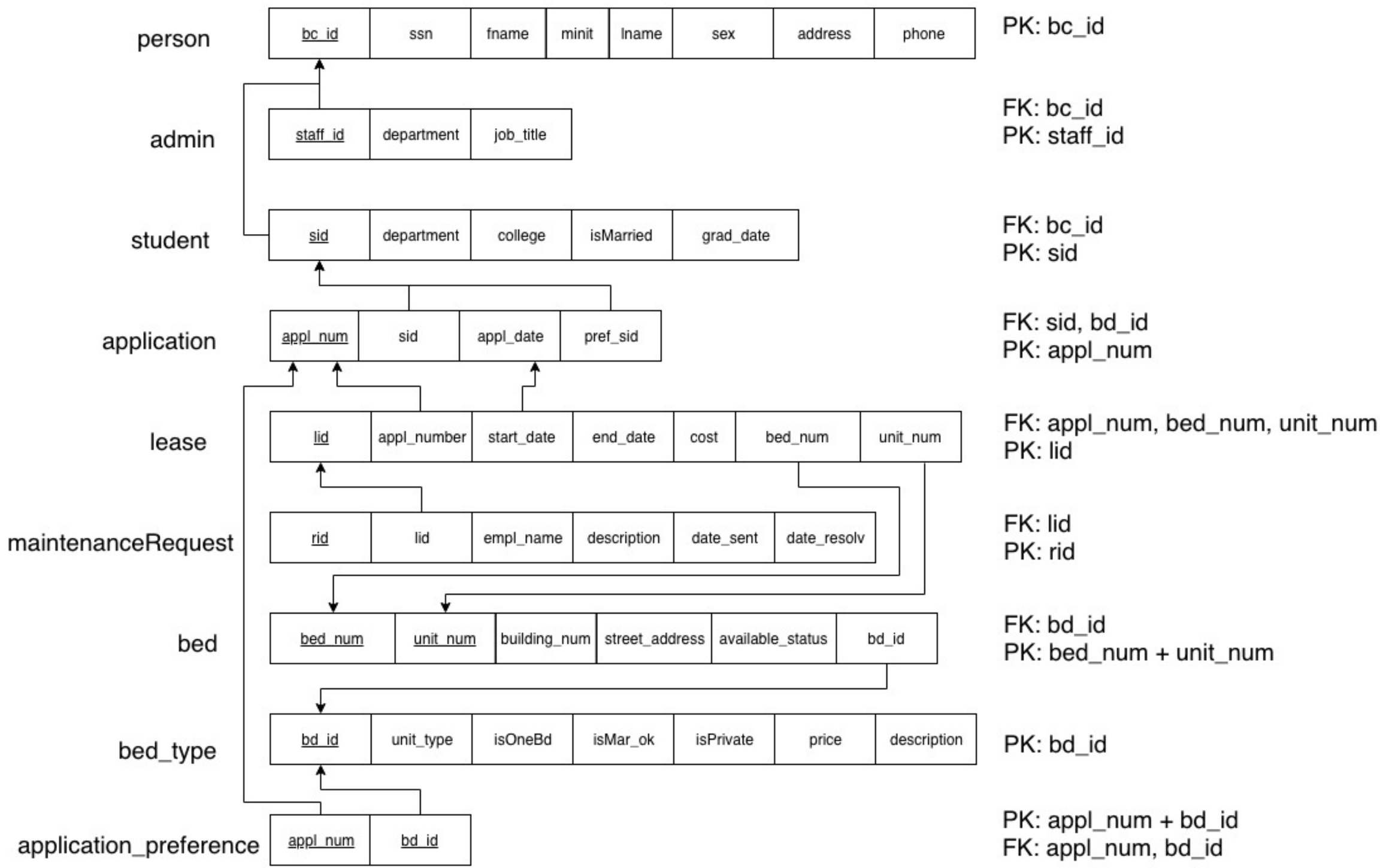

person
bc_id
ssn
fname
minit
lname
gender
phone
address
d
student
department
college
married status
graduation_date
admin
department
job_title
application
appl_num
((pref))
start_quarter
end_quarter
pref_sid
app_date
submits
(1:1)
(0:M)
(1:1)
Becomes
(0:1)
maintenanceRequest
rid
employee_name
description
date_sent
date_resolved
(0:M)
makes
(1:1)
lease
lid
start_date
end_date
cost
(0:M)
(0:M)
Has
(1:1)
has preference
bed
bed_num
unit_num
street_address
building_num
status
bd_id
(1:M)
associated with
(1:1)
(3:3)
bed_type
bd_id
unit_type
isOneBedroom
isMarriedOk
isPrivate
Price
Description

Appendix D –
Entity Attribute Table

person	
Attribute	***Description***
bc_id	Unique identifying number stored by Bellevue College for both staff and students. These numbers will be accessed and authenticated using the existing Bellevue College database (PK)
ssn	The social security number of each person
fname	The first name of each person
minit	The middle initial of each person (if applicable)
lname	The last name of each person
sex	The gender of each person
phone	The 10-digit phone number of each person
address	The full address of each person
student	
sid	The student id of the student; this is the same as the BC_ID in the 'Person' supertype (PK)
department	The department in which a student currently studies
college	The college in which a student currently studies
isMarried	A Boolean value representing whether or not a student is married
graduation_date	The date a student graduated (if applicable)
admin	
staff_id	The staff id of the administrator; this is the same as the BC_ID in the 'Person' super-type (PK)
department	The department in which the administrator works
job_title	The name of the administrator's job
application	
appl_num	Unique number identifying the application (PK)
sid	The BC_ID of the student who made the application
appl_date	The date a student submitted their application, or the date of the student's latest application revision (used to set priority)
pref	The student ID of a student's preferred roommate, if applicable (FK)
start_quarter	The quarter (F, W, or S) the applicant's lease would start
end_quarter	The quarter (F, W, or S) the applicant's lease would end
bed	
bed_num	A number assigned to uniquely define each bed within a multi-bed residence (partial PK)
unit_num	The number of the residence the bed is in (partial PK)

building_num	The number of the residence building the bed is located in
street_address	The address of the residence building the bed is in
status	A Boolean value representing whether or not the bed is still available
bd_id	A unique identifier which correlates with a specific bed_type (FK)
bed_type	
bd_id	A unique identifying number which corresponds to each bed_type (PK)
unit_type	Specifies the type of unit (suite or apartment)
isOneBd	A Boolean value specifying whether or not the residence is a single
isMarriedOk	A Boolean value specifying whether or not the residence can accommodate married couples
isPrivate	A Boolean value specifying whether or not a residence is private
price	The default price of a specific configuration (used to auto-populate the 'cost' field of each lease entity)
description	A description of each configuration
lease	
lid	A unique identifying number for each lease (PK)
appl_num	The number of the application which resulted in the lease (FK)
start_date	The date on which the lease period begins
end_date	The date on which the lease period ends
cost	The cost of the lease to the student
bed_num	The number of the bed in the unit which the student is leasing (FK)
unit_num	The number of the unit the student is partially/fully leasing (FK)
maintenance_request	
rid	A unique identifying number for each maintenance request (PK)
lid	The lease number of the student making the request (FK)
empl_name	The name of the employee who works to resolve the request
description	A description of the issue for which the request is made
date_sent	The date the maintenance request is made
date_resolv	The date the maintenance request is resolved

Appendix E –
Relational Model Schema


person
bc_id
ssn
fname
minit
lname
sex
address
phone
PK: bc_id
admin
staff_id
department
job_title
FK: bc_id
PK: staff_id
student
sid
department
college
isMarried
grad_date
FK: bc_id
PK: sid
application
appl_num
sid
appl_date
pref_sid
FK: sid, bd_id
PK: appl_num
lease
lid
appl_number
start_date
end_date
cost
bed_num
unit_num
FK: appl_num, bed_num, unit_num
PK: lid
maintenanceRequest
rid
lid
empl_name
description
date_sent
date_resolv
FK: lid
PK: rid
bed
bed_num
unit_num
building_num
street_address
available_status
bd_id
FK: bd_id
PK: bed_num + unit_num
bed_type
bd_id
unit_type
isOneBd
isMar_ok
isPrivate
price
description
PK: bd_id
application_preference
appl_num
bd_id
PK: appl_num + bd_id
FK: appl_num, bd_id

Appendix F –
Database SQL Script

```
CREATE TABLE person (
  bc_id    integer(9) primary key,
  ssn      char(9) unique not null,
  fname    varchar(15) not null,
  minit    varchar(1),
  lname    varchar(15) not null,
  address  varchar(50),
  sex      char not null,
  phone    integer(10)
);

alter table person add constraint sex check (sex in ('F', 'M'));

CREATE TABLE admin (
  staff_id    integer(9) primary key,
  department  varchar(25),
  job_title   varchar(25),
  foreign key (staff_id) references person(bc_id) on delete cascade
);

CREATE TABLE personVerification (
  bc_id    integer(9),
  password  varchar(8),
  primary key (bc_id, password),
  foreign key (bc_id) references person(bc_id) on delete cascade
);

CREATE TABLE student (
  sid      integer(9) primary key,
  department varchar(50),
  college    varchar(50),
  isMarried  bool,
  grad_date  date,
  foreign key (sid) references person(bc_id) on delete cascade
);


CREATE TABLE bed_type (
  bd_id       integer(2) primary key auto_increment,
  unit_type   varchar(1) not null,
  isOneBd     bool not null,
  isMar_ok    bool not null,
  isPrivate   bool not null,
  price       decimal(10,2),
  description varchar(50)
);

alter table bed_type add constraint unit_type check (unit_type in ('A', 'S'));

CREATE TABLE application(
  appl_num       integer(9) primary key auto_increment,
  sid            integer(9) not null,
  appl_date      date,
  start_quarter  varchar(1) not null,
  end_quarter    varchar (1) not null,
  pref_sid       integer(9),
  status         varchar(1),
  foreign key (sid) references student(sid) on delete cascade,
  foreign key (pref_sid) references student(sid) on delete cascade
);

alter table application add constraint start_quarter check (start_quarter in ('F', 'W', 'S'));
alter table application add constraint end_quarter check (end_quarter in ('F', 'W', 'S'));
alter table application add constraint Status check (status in ('A', 'C', 'L', 'P'));
-- alter table application add constraint pref_sid check (pref_sid != sid);    logic to java

CREATE TABLE application_preference(
  appl_num  integer(9),
  bd_id     integer(2),
  primary key(appl_num, bd_id),
  foreign key(appl_num) references application(appl_num) on delete cascade,
  foreign key(bd_id) references bed_type(bd_id) on delete cascade
);

CREATE TABLE bed (
  bed_num          varchar(1),
  unit_num         integer(3),
  building_num     integer(6) not null,
  street_address   varchar(25),
  bd_id            integer(2),
  primary key (bed_num, unit_num),
  foreign key (bd_id) references bed_type(bd_id) on delete cascade
);
```

```
CREATE TABLE lease (
  lid         integer(9) primary key auto_increment,
  appl_num    integer(9) not null,
  start_date  date not null,
  end_date    date not null,
  cost        decimal(10,2),
  bed_num     varchar(1) not null,
  unit_num    integer(3) not null,
  foreign key (appl_num) references application(appl_num) on delete cascade,
  foreign key (bed_num, unit_num) references bed(bed_num, unit_num) on delete cascade
);

CREATE TABLE maintenanceRequest(
  rid          integer(9) primary key auto_increment,
  lid          integer(9),
  empl_name    varchar(15),
  description  varchar(50),
  date_sent    date,
  date_resolv  date,
  foreign key (lid) references lease(lid)
);

INSERT INTO person VALUES(111111111,111111111,"Walter","H","White","308 Negra Arroyo Ln,
Albuquerque, NM 87111 USA",'M',2065551111);
INSERT INTO person VALUES(222222222,222222222,"Lyndon","B","Johnson","123 Dirt Path, Stonewall,
TX 98125 USA",'M',2065552222);
INSERT INTO person VALUES(333333333,333333333,"Jeffrey","P","Bezos","999 Bezos' Big Balls,
Seattle, WA 98125 USA",'M',2065553333);
INSERT INTO person VALUES(444444444,444444444,"William","H","Gates","400 Lk Wash Blvd, Seattle,
WA 98101 USA",'M',2065554444);
INSERT INTO person VALUES(555555555,555555555,"Alan","M","Turing","505 Maida Vale Rd, Maida Vale,
UK",'M',2065555555);
INSERT INTO person VALUES(666666666,666666666,"Grace","B","Hopper","255 Main St, New York City,
NY 10001",'F',2065556666);
INSERT INTO person VALUES(777777777,777777777,"Marie","S","Curie","404 Road Not Found, Warsaw,
PL",'F',2065557777);
INSERT INTO person VALUES(888888888,888888888,"Barbara","J","Liskov","200 OK Road, Los Angeles,
CA 90895",'F',2065558888);
INSERT INTO person VALUES(999999999,999999999,"Jane","A","Doe","123 Alaskan Way, Seattle, WA
98125",'F',2065559999);
INSERT INTO person VALUES(100000000,100000000,"Amelia","M","Earhart","500 Grass Field Rd,
Atchison, KS 66101",'F',2065550000);


-- Adding 5 more people to the table so we have 6 male and 6 female students
INSERT INTO person VALUES(110000000,110000000,"Danerys","B","Targaryen","Dragonstone, Westeross
71101",'F',2065551100);
INSERT INTO person VALUES(120000000,120000000,"Arya","C","Stark","Braavos, Free Cities
91101",'F',2065551200);
INSERT INTO person VALUES(130000000,130000000,"Egrett","F","Wild","North of the Wall, Westeross
00000",'F',2065551300);


INSERT INTO person VALUES(140000000,140000000,"John","B","Snow","Winterfell, Westeross
81101",'M',2065551400);
INSERT INTO person VALUES(150000000,150000000,"Jaime","B","Lannister","Kings Landing, Westeross
66101",'M',2065551500);
INSERT INTO person VALUES(160000000,160000000,"Bron","B","Blackwater","Tarth, Westeross
99101",'M',2065551600);



INSERT INTO admin VALUES(444444444,"Housing Services","Database Administrator");
INSERT INTO admin VALUES(666666666,"Housing Services","Supervisor");


INSERT INTO student VALUES(111111111,"Chemistry","Arts and Sciences",true,'2020-06-15');
INSERT INTO student VALUES(222222222,"Political Science","Arts and Sciences",false,'1973-01-22');
INSERT INTO student VALUES(333333333,"Computer Science","Engineering",false,'2021-06-15');
INSERT INTO student VALUES(777777777,"Computer Science","Engineering",true,'1934-07-04');
INSERT INTO student VALUES(888888888,"Computer Science","Engineering",false,'1969-06-15');
INSERT INTO student VALUES(999999999,"Architecture","Arts and Sciences",false,'2020-06-15');
INSERT INTO student VALUES(100000000,"Aeronautics","Engineering",false,'1937-07-02');

-- Additional students

INSERT INTO student VALUES(110000000,"Dragons","Arts and Sciences",false,'2019-06-15');
INSERT INTO student VALUES(120000000,"Swords","Engineering",false,'2023-06-15');
INSERT INTO student VALUES(130000000,"Fletching","Engineering",false,'2022-06-15');
INSERT INTO student VALUES(140000000,"PeaceKeeper","Arts and Sciences",false,'2018-06-15');
INSERT INTO student VALUES(150000000,"General","Engineering",false,'2020-04-15');
INSERT INTO student VALUES(160000000,"Mooching","Engineering",true,'2011-03-15');
INSERT INTO student VALUES(555555555,"Computer Science", "Engineering", false,'2019-06-15');
```

```
INSERT INTO bed_type VALUES(1,"s",true,true,true,4667,"One Bedroom Suite, One Person");
INSERT INTO bed_type VALUES(2,"s",false,false,false,3667,"One Bedroom Suite, Two Persons");
INSERT INTO bed_type VALUES(3,"s",false,false,true,3500,"Two Bedroom Suite, Two Persons");
INSERT INTO bed_type VALUES(4,"s",false,true,false,2500,"Two Bedroom Suite, Three Persons -
Shared Room");
INSERT INTO bed_type VALUES(5,"s",false,false,true,3500,"Two Bedroom Suite, Three Persons -
Private Room");
INSERT INTO bed_type VALUES(6,"s",false,true,false,2500,"Two Bedroom Suite, Four Persons");
INSERT INTO bed_type VALUES(7,"a",false,false,false,3334,"Two Bedroom Apartment, Four Persons");
INSERT INTO bed_type VALUES(8,"a",false,false,true,4000,"Four Bedroom Apartment, Four Persons");


INSERT INTO bed VALUES(1,100,400,"3000 Landerholm Cir SE",1);  -- 1b 1p aval
INSERT INTO bed VALUES(1,310,400,"3000 Landerholm Cir SE",1);  -- 1b 1p m  need appl + lease SID
111111111

INSERT INTO bed VALUES(1,110,400,"3000 Landerholm Cir SE",2);  -- 1b 2p aval
INSERT INTO bed VALUES(2,110,400,"3000 Landerholm Cir SE",2);  -- 1b 2p aval  the same room
INSERT INTO bed VALUES(1,320,400,"3000 Landerholm Cir SE",2);  -- 1b 2p 1f       need appl +
lease SID 777777777
INSERT INTO bed VALUES(1,520,400,"3000 Landerholm Cir SE",2);  -- 1b 2p 1m       need appl +
lease SID 222222222

INSERT INTO bed VALUES(1,120,400,"3000 Landerholm Cir SE",3);  -- 2b 2p aval
INSERT INTO bed VALUES(2,120,400,"3000 Landerholm Cir SE",3);  -- 2b 2p aval   same room
INSERT INTO bed VALUES(1,400,400,"3000 Landerholm Cir SE",3);  -- 2b 2p m SID 333333333
INSERT INTO bed VALUES(2,400,400,"3000 Landerholm Cir SE",3);  -- 2b 2p aval   same room
INSERT INTO bed VALUES(1,500,400,"3000 Landerholm Cir SE",3);  -- 2b 2p f SID 888888888
INSERT INTO bed VALUES(2,500,400,"3000 Landerholm Cir SE",3);  -- 2b 2p aval   same room

INSERT INTO bed VALUES(1,130,400,"3000 Landerholm Cir SE",4);  -- 2b 3p shared aval
INSERT INTO bed VALUES(2,130,400,"3000 Landerholm Cir SE",4);  -- 2b 3p shared aval  same room
INSERT INTO bed VALUES(3,130,400,"3000 Landerholm Cir SE",5);  -- 2b 3p priv f SID 999999999

INSERT INTO bed VALUES(1,410,400,"3000 Landerholm Cir SE",4);  -- 2b 3p shared aval
INSERT INTO bed VALUES(2,410,400,"3000 Landerholm Cir SE",4);  -- 2b 3p shared m     same room
SID 120000000
INSERT INTO bed VALUES(3,410,400,"3000 Landerholm Cir SE",5);  -- 2b 3p priv aval

INSERT INTO bed VALUES(1,510,400,"3000 Landerholm Cir SE",4);  -- 2b 3p shared f SID 100000000
INSERT INTO bed VALUES(2,510,400,"3000 Landerholm Cir SE",4);  -- 2b 3p shared aval     same room
INSERT INTO bed VALUES(3,510,400,"3000 Landerholm Cir SE",5);  -- 2b 3p priv aval

INSERT INTO bed VALUES(1,220,400,"3000 Landerholm Cir SE",6);  -- 2b 4p aval
INSERT INTO bed VALUES(2,220,400,"3000 Landerholm Cir SE",6);  -- 2b 4p aval   same room
INSERT INTO bed VALUES(3,220,400,"3000 Landerholm Cir SE",6);  -- 2b 4p m SID 140000000
INSERT INTO bed VALUES(4,220,400,"3000 Landerholm Cir SE",6);  -- 2b 4p aval   same room

INSERT INTO bed VALUES(1,140,400,"3000 Landerholm Cir SE",6);  -- 2b 4p aval
INSERT INTO bed VALUES(2,140,400,"3000 Landerholm Cir SE",6);  -- 2b 4p aval   same room
INSERT INTO bed VALUES(3,140,400,"3000 Landerholm Cir SE",6);  -- 2b 4p f SID 110000000
INSERT INTO bed VALUES(4,140,400,"3000 Landerholm Cir SE",6);  -- 2b 4p aval   same room

INSERT INTO bed VALUES(1,230,400,"3000 Landerholm Cir SE",7);  -- 2b 4p f SID 130000000
INSERT INTO bed VALUES(2,230,400,"3000 Landerholm Cir SE",7);  -- 2b 4p aval   same room
INSERT INTO bed VALUES(3,230,400,"3000 Landerholm Cir SE",7);  -- 2b 4p aval
INSERT INTO bed VALUES(4,230,400,"3000 Landerholm Cir SE",7);  -- 2b 4p aval   same room


INSERT INTO bed VALUES(1,300,400,"3000 Landerholm Cir SE",8);  -- 4b 4p m SID 150000000
INSERT INTO bed VALUES(2,300,400,"3000 Landerholm Cir SE",8);  -- 4b 4p aval
INSERT INTO bed VALUES(3,300,400,"3000 Landerholm Cir SE",8);  -- 4b 4p aval
INSERT INTO bed VALUES(4,300,400,"3000 Landerholm Cir SE",8);  -- 4b 4p aval


INSERT INTO personVerification VALUES(111111111,"Chem");
INSERT INTO personVerification VALUES(222222222,"Science");
INSERT INTO personVerification VALUES(333333333,"Computer");
INSERT INTO personVerification VALUES(777777777,"Engin");
INSERT INTO personVerification VALUES(888888888,"CompSE");
INSERT INTO personVerification VALUES(999999999,"Architec");
INSERT INTO personVerification VALUES(100000000,"Aeronaut");
INSERT INTO personVerification VALUES(110000000,"Dragons");
INSERT INTO personVerification VALUES(120000000,"Swords");
INSERT INTO personVerification VALUES(130000000,"Fletchin");
INSERT INTO personVerification VALUES(140000000,"PeaceK");
INSERT INTO personVerification VALUES(150000000,"General");
INSERT INTO personVerification VALUES(444444444,"Admin");
INSERT INTO personVerification VALUES(666666666,"Supervis");
INSERT INTO personVerification VALUES(160000000,"zzzzzzzz");
INSERT INTO personVerification VALUES(555555555,"Student");
```

```
INSERT INTO application VALUES(3, 111111111, '2018-09-04', 'F', 'S', 777777777, 'A'); -- m moved
in
INSERT INTO application VALUES(1, 222222222, '2017-08-30', 'F', 'S', null, 'L'); -- m moved in
past
INSERT INTO application VALUES(4, 333333333, '2018-09-10', 'F', 'W', null, 'L'); -- m moved in
INSERT INTO application VALUES(5, 777777777, '2018-09-13', 'F', 'S', 111111111, 'A'); -- f moved
in
INSERT INTO application VALUES(6, 888888888, '2018-10-13', 'W', 'S', null, 'L'); -- f moved in
INSERT INTO application VALUES(7, 999999999, '2018-11-01', 'W', 'W', null, 'L'); -- f moved in
INSERT INTO application VALUES(8, 100000000, '2018-11-10', 'W', 'W', null, 'L'); -- f moved in
INSERT INTO application VALUES(9, 110000000, '2018-11-10', 'W', 'W', null, 'L'); -- f moved in
INSERT INTO application VALUES(10, 120000000, '2018-11-01', 'W', 'S', null, 'A'); -- m moved in
INSERT INTO application VALUES(11, 130000000, '2018-11-20', 'S', 'S', null, 'L'); -- f moved in
INSERT INTO application VALUES(13, 140000000, '2019-11-20', 'W', 'S', null, 'L'); -- m moved in
INSERT INTO application VALUES(14, 150000000, '2019-11-20', 'S', 'S', null, 'L'); -- m moved in
INSERT INTO application VALUES(12, 222222222, '2018-11-20', 'W', 'S', null, 'L'); -- m moved in
INSERT INTO application VALUES(2, 555555555, '2018-08-30', 'F', 'S', null, 'L'); -- m moved in


INSERT INTO application_preference VALUES(3, 1);
INSERT INTO application_preference VALUES(3, 4);
INSERT INTO application_preference VALUES(3, 6);
INSERT INTO application_preference VALUES(5, 1);
INSERT INTO application_preference VALUES(5, 4);
INSERT INTO application_preference VALUES(5, 6);
INSERT INTO application_preference VALUES(10, 3);
INSERT INTO application_preference VALUES(10, 5);
INSERT INTO application_preference VALUES(10, 8);


INSERT INTO lease VALUES(1, 1, '2017-09-03', '2018-06-15', 14001.00, 1, 310);
INSERT INTO lease VALUES(2, 2, '2018-09-03', '2019-06-15', 11001.00, 1, 520);
INSERT INTO lease VALUES(3, 4, '2018-09-03', '2019-03-15', 9334.00, 1, 310);
INSERT INTO lease VALUES(4, 6, '2019-01-03', '2019-06-15', 7334.00, 1, 320);
INSERT INTO lease VALUES(5, 7, '2019-01-03', '2019-03-15', 3500.00, 1, 500);
INSERT INTO lease VALUES(6, 12, '2019-01-03', '2019-06-15', 5000.00, 2, 410);
INSERT INTO lease VALUES(7, 8, '2019-01-03', '2019-03-15', 3500.00, 3, 130);
INSERT INTO lease VALUES(8, 9, '2019-01-03', '2019-03-15', 2500.00, 1, 510);
INSERT INTO lease VALUES(9, 11, '2019-04-03', '2019-06-15', 3334.00, 1, 230);
INSERT INTO lease VALUES(10, 13, '2019-01-03', '2019-06-15', 8000.00, 1, 300);
INSERT INTO lease VALUES(11, 14, '2019-04-03', '2019-06-15', 2500.00, 3, 220);


-- need to check
INSERT INTO maintenanceRequest VALUES(1, 2, 'Grace Hopper', 'Kitchen lights burnt out',
'2018-10-03', null);
INSERT INTO maintenanceRequest VALUES(2, 4, 'Grace Hopper', 'Sink clogged', '2018-11-04',
'2018-11-05');
INSERT INTO maintenanceRequest VALUES(3, 2, 'Grace Hopper', 'Sink clogged', '2018-11-03', null);
INSERT INTO maintenanceRequest VALUES(4, 1, 'Grace Hopper', 'Microwave no longer works',
'2017-11-29', '2018-12-04');


DELIMITER //

-- DROP PROCEDURE IF EXISTS getActiveStudentApplication;

CREATE PROCEDURE getActiveStudentApplication (IN studentId integer(9))
BEGIN
    SELECT appl_num, appl_date, start_quarter, end_quarter, pref_sid
    FROM application
    WHERE sid = studentId AND
          status = 'A'
        LIMIT 1;
END

DELIMITER ;


DELIMITER //

-- DROP PROCEDURE IF EXISTS getApplicationPreferences;

CREATE PROCEDURE getApplicationPreferences (IN applNum integer(9))
BEGIN
    SELECT bd_id
    FROM application_preference
    WHERE appl_num = applNum
        LIMIT 3;
END
```

```
DELIMITER ;

DELIMITER //

-- DROP PROCEDURE IF EXISTS getActiveStudentLease;

CREATE PROCEDURE getActiveStudentLease (IN studentId integer(9))
BEGIN
    SELECT lid, lease.appl_num, lease.start_date, lease.end_date, cost, bed_num, unit_num
    FROM lease, application
    WHERE application.sid = studentId AND
                    lease.appl_num = application.appl_num AND
          lease.end_date > CURDATE()
        LIMIT 1;
END

DELIMITER ;

-- CALL getActiveStudentLease (555555555);  -- for check

DELIMITER //

--  DROP PROCEDURE IF EXISTS getActiveRequests;

CREATE PROCEDURE getActiveRequests (IN studentId integer(9))
BEGIN
    SELECT rid, description, date_sent
    FROM maintenanceRequest, lease, application
    WHERE application.sid = studentId AND
                    lease.appl_num = application.appl_num AND
          lease.lid = maintenanceRequest.lid AND
          date_resolv IS NULL;
END

DELIMITER ;

DELIMITER //

--  DROP PROCEDURE IF EXISTS updateCurrentApplication;

CREATE PROCEDURE updateCurrentApplication (IN applNum integer(9),
                                    startQ varchar(1), endQ varchar(1), prefSid integer(9), INOUT
upload bool)
BEGIN
    -- Update
    UPDATE application
    SET appl_date = CURDATE(),  start_quarter = startQ, end_quarter = endQ, pref_sid = prefSid
    WHERE appl_num = applNum;
    -- return
    set upload = true;
END

DELIMITER ;


DELIMITER //

--  DROP PROCEDURE IF EXISTS updateCurrentApplicationPrefer;

CREATE PROCEDURE updateCurrentApplicationPrefer (IN applNum integer(9), firstPr integer(2),
                                    secondPr integer(2), thirdPr integer(2), INOUT upload bool)
BEGIN
    -- delete
    DELETE FROM application_preference
    WHERE appl_num = applNum;

    -- insert
    INSERT INTO application_preference VALUES (applNum, firstPr);
    IF (firstPr != secondPr)
    THEN INSERT INTO application_preference VALUES (applNum, secondPr);
    END IF;
    IF (firstPr != thirdPr AND secondPr != thirdPr)
    THEN INSERT INTO application_preference VALUES (applNum, thirdPr);
    END IF;

    -- return
    set upload = true;
END

DELIMITER ;

DELIMITER //

--  DROP PROCEDURE IF EXISTS cancelCurrentApplication;
```

```
CREATE PROCEDURE changeApplicationStatus(IN applNum integer(9), newStatus varchar(1), INOUT
upload bool)
BEGIN
    -- Update
    UPDATE application
    SET status = newStatus
    WHERE appl_num = applNum;
    -- return
    set upload = true;
END

DELIMITER ;



DELIMITER //

-- DROP PROCEDURE IF EXISTS createNewApplication;

CREATE PROCEDURE createNewApplication(IN sId integer(9),
                                      startQ varchar(1), endQ varchar(1), prefSid integer(9), OUT num
integer)
BEGIN
    -- insert
    INSERT INTO application (sid, appl_date, start_quarter, end_quarter, pref_sid, status)
    VALUES (sId, CURDATE(), startQ, endQ, prefSid, 'A');

    -- return
    SELECT appl_num INTO num
    FROM application
    WHERE appl_num = LAST_INSERT_ID();
END

DELIMITER ;



DELIMITER //

-- DROP PROCEDURE IF EXISTS findOccupancy;

CREATE PROCEDURE findOccupancy(IN app_start TIMESTAMP, app_end TIMESTAMP )
BEGIN
    SELECT unit_num,
           bd_id,
           bed_num,
           CASE
             WHEN sum_occupancy_score = 0 THEN 'FREE'
             WHEN sum_occupancy_score < 100 THEN 'MALE'
             WHEN (sum_occupancy_score % 100) = 0 THEN 'FEMALE'
             ELSE 'BOTH'
                    END AS occupancy
        FROM
        (
                SELECT unit_num, bd_id, bed_num,
                           SUM(occupancy_score) AS sum_occupancy_score
                FROM
                (
                SELECT
                           bed.unit_num, bed.bd_id, bed.bed_num,
                           CASE
                                 WHEN sex = 'M' THEN 1
                                 WHEN sex = 'F' THEN 100
                                 ELSE 0
                           END AS occupancy_score
                FROM bed
                          LEFT JOIN lease
                          ON bed.bed_num = lease.bed_num AND
                                 bed.unit_num = lease.unit_num AND
                                 ((lease.end_date > app_start AND lease.end_date <
app_end) OR
                                  (lease.start_date > app_start AND lease.start_date <
app_end) OR
                                  (lease.start_date < app_start AND lease.end_date >
app_end))
                          LEFT JOIN application ON lease.appl_num = application.appl_num
                          LEFT JOIN person ON application.sid = person.bc_id
                     ) AS T
                GROUP BY unit_num, bd_id, bed_num
        ) AS T2;
END

DELIMITER ;
```

```
 DELIMITER //

-- DROP PROCEDURE IF EXISTS createNewLease;

CREATE PROCEDURE createNewLease(IN applNum integer(9), startL TIMESTAMP, endL TIMESTAMP,
                                                                    cost decimal(10,2), bedNum
integer, unitNum integer, INOUT upload bool)
BEGIN
    -- insert
    INSERT INTO lease (appl_num, start_date, end_date, cost, bed_num, unit_num)
    VALUES (applNum, startL, endL, cost, bedNum, unitNum);

   -- return
    set upload = true;
END

DELIMITER ;

DELIMITER //

-- DROP PROCEDURE IF EXISTS getVacantUnits;

CREATE PROCEDURE getVacantUnits(IN quarterDate date)
BEGIN
    SELECT unit_num, bed_num, end_date
        FROM lease
        WHERE lease.end_date = quarterDate
    ORDER BY unit_num, bed_num;
END

DELIMITER ;

DROP TABLE maintenanceRequest;
DROP TABLE lease;
DROP TABLE bed;
DROP TABLE application_preference;
DROP TABLE application;
DROP TABLE bed_type;
DROP TABLE personVerification;
DROP TABLE student;
DROP TABLE admin;
DROP TABLE person;
```