# Online Apartment Management System-Database Design

Fangwen Ge, Xueyang Li, Lin Tao, Yuyang Han, Pan Xu

## Business Problems Addressed:

The purpose of the database is to develop a leasing/tenant management system used to simplify work for tenants and renter, having their effort done efficiently and effectively. It will be used by both the two client terminals, the apartment agents or manager and tenants, modeling the process of a tenant renting an apartment, a tenant paying rent, and a tenant placing work orders for their apartments.

## The database aims to solve the following business problems:

- To track the status of potential tenants' applications.
- To track the status of leasing terms.
- To track the status of work orders submitted by the tenants.
- To track the status of maintenance services by the workers.
- To track tenants who have or have not paid their rent, utilities, or other billing listings for a given month.

## Entities:

| Entity Name | Why Entity is Included? | How Entity is Related to Other Entities |
|---|---|---|
| **User** | We track user's detail such as user's name (First + last name), email address, phone numbers, user type and user's unit if existed | User entity is connected with UserType and Unit entities. User and UserType have a one to many relationship while Unit and User also has a one to many relationship Use is also linked to entities such as EmergencyContact, |

| | | |
|---|---|---|
| | | LeasingApplication, Address, and Payment entities directly or through their respective linking table |
| **UserType** | Since the database can be used for both the tenants and the prospective applicants, we track the user's type such as if the user is a current tenant, an applicant, etc. | The UserType entity is connected to the User entity by the UserTypeID. One UserType can be associated with multiple or no User, a user will have one and only one UserType. |
| **UserAddress** | This is a linking table between User and Address entity to store user's address information. | Since the Address entity stores only the street address of a building, User and Address entity have a many to many relationship. This UserAddress entity normalizes this relationship by using an linking table to store this relationship |
| **Address** | We track the street address of a building in the Address entity. This means that all street addresses (excluding the user's apartment address) will be stored in this entity. It has the standard fields in the entity such as street, city, state, and zip code. | As mentioned above, the Address entity is linked to the User entity with a linking table UserAddress. Additionally, the Address entity has a one to one relationship with the Building entity, which is referred to as the AddressID in the Building entity |
| **Building** | We will locate the building of each user and each applicant. Detailed information such as floors and built year is included in this entity. | One or more leasing agents can work in one building.Each building includes one or more units but each building can only have one address. |
| **UserUnit** | This is a linking table between User and Unit entity to store user's Unit | A user can sign multiple leases for different units over time. We create a |

| | | |
|---|---|---|
| | information. | new associative entity between Unit and User. |
| **Unit** | Since we want to track the leasing information of the users, knowing which unit they live in is important. The unit entity also includes the unit number and description of the detail of the unit. | Unit has foreign key BuildingID that links to a specific building. One unit can make zero to many service orders and has zero to many utilities, therefore will have zero to many billings. |
| **EmergencyContact** | The Emergency Contact entity provides the emergency contact information of the users. Detailed information contains the name, phone number of the contacts and the relationship between them. | The EmergencyContact entity is linked to the User entity with a linking table UserEmergencyContact entity. |
| **UserEmergencyContact** | This is a table linking the users and their emergency contacts to avoid many to many relationships. So it includes the User and EmergencyContact as FKS and we add the UserEmergencyContactID as the PK. | One user has one or more user emergency contacts, and one contact can be emergency contacts of many users. We use this entity to link the User entity and the EmergencyContact entity. |
| **LeasingOffice** | The leasing office is responsible for managing the apartment building. It is responsible for managing the application for units and daily maintenance of apartments. If the applicants / residents meet problems, they could contact the leasing office. So we track their name and contact information (including email, phone and official website) | One leasing office can manage multiple buildings, but one building can only be managed by one leasing office, thus the relationship between them is one to many. One leasing office has many agents and also manages multiple applications. Their relationship is one to many. |

| | | |
|---|---|---|
| **LeasingAgent** | The leasing agent is the group of managers of a building, designated by the leasing office. | Leasing Agent is connected with the LeasingOfficeID and the BuildingID |
| **RentHistory** | In this entity, we collect the information about the applicant's rent history to evaluate whether to rent the unit to him/her. We manage a large company, so it is assumed that the building where the applicant previously lived is also managed by a leasing office recorded in our system. | Rent history is linked to the user entity, they have a one to many relationship. And it can be seen by the leasing office, so these two are also connected. |
| **Payment** | Since users need to pay application fee / rent / utilities fee, we set up entity ‘payment’ to track each payment of users. | There is a many to many relationship between payment and billing. One billing can be paid multiple times, and users can make the present payment pay multiple billing (rent or maintenance fee, etc.). Therefore, we established the entity ‘PaymentBilling’ to build a many to many relationship between payment and billing. |
| **PaymentBilling** | This is a linking table between user’s payment and billing, as one billing could be paid by multiple payment methods, and also one payment method can provide support for several billings. | One end, the payment entity only stores the details such as date, amount as well as method which is related to each user’s rent payment requirement, while the billing entity includes the maintenance service and utility-equipped placed by individual units which also need to be paid by multiple payment methods. This payment billing entity |

| | | |
|---|---|---|
| | | normalizes this kind of many-to-many payment methods-and-billing relationship by using a linking table, generating the paymentbillID. |
| **Billing** | We use the billing entity to store all general fees including utilities and service order collected from each individual unit. | As each unit may generate several maintenance service orders and apartment utility fees during different time phases accordingly, the billing entity combines the two fees and attaches them to the unit room. In addition, the billing entity gets a payment method by being connected to the payment billing entity. The utilities entity and billing has a many-to-one relationship, and so as the service order entity and billing entity. The unit entity and billing entity has a one-to-many relationship. |
| **Utilities** | The utility entity exhibits the billing information attached to each type of utility based on unitrooms. Additionally, to have an appropriate management of basic utilities, we've marked all of the utilities by utilityID number and categorized them into the appropriate infrastructure category, which includes the basic four category like water, electricity, heating as well as insurance, and other equipments like Community Amenities(fitness center, | We attach the utility to each unit connected by the unitID, and generate the according utility billing through billingID. As mentioned above, utility and billing have a one-to-many relationship. Also, utility and Unit have a one-to-many relationship. |

| | | |
|---|---|---|
| | elevator, business center, etc) , Property Services(Wi-Fi at pool, controlled Access , etc), of course the Shared Community(loung, conference rooms, etc) | |
| **ServiceOrder** | The service order entity contains information related to maintenance of utility and paid by units, which generates the service order, categorizes the service type into fix, replace, clean, etc. Besides, the service order will be assigned to related workers and this order will be appointed to the exact date. Finally, our system will collect the customer feedback to complete the service cycle and do help to our subsequent improvements and staff rating. | The service order entity is connected to other entities by the service completion process from retrieving the address where the tenants give a maintenance service request, connected to the address entity. And then, select the exact maintenance service type, allocate corresponding maintenance workers to implement the maintenance task. After completing the maintenance service, the billing will be generated and attached to the unit, requiring the tenants to pay the service amount within the due date. ServiceOrder and Unit have a one-to-many relationship. ServiceOrder and Billing have a one-to-many relationship. ServiceOrder and MaintenanceWorker have a one-to-many relationship. ServiceOrder and MaintenanceCompany have a one-to-one relationship. |
| **LeasingApplication** | The LeasingApplication contains the application information. The user starts a new application and the leasing office | LeasingApplication is connected to three entities, including User, Unit and LeasingOffice. One user may have zero to many |

| | | |
|---|---|---|
| | receives the application. It has the basic information of the office and user and the specific information of the details of the application. Once the office approves the application, we can track the application status by this entity. | applications; One Unit may have zero to many applications; One LeasingOffice may receive zero to many applications. |
| **MaintenanceCompany** | The maintenance company is responsible for the building maintenance. One company provides one specific type of service. | The MaintenanceCompany is connected to two entities. One maintenance company provides one type of maintenance service; One maintenance company may have one to many workers. |
| **MaintenanceWorkers** | This is a group of workers who work for the maintenance company. When a user requires a maintenance service, the company could allocate a worker to the order. | The MaintenanceWorkers entity is connected to two entities. One maintenance company may have one to many workers; One maintenance worker may be responsible for many service orders. |
| **MaintenanceServiceType** | This is a table about the service type that the maintenance company provided. | The MaintenanceService Type is connected to two entities. One service order includes one or many types of service and one company is responsible for one type of service. |

## Key Design Decisions:

- Billing vs. Payment vs. Utility vs. ServiceOrder: To fit the business need of this database, we determined that each type of payment should be its own entity

because each payment type requires different data to be stored. Payment entity stands for the user's rent payment in a given month; Utility entity directly deals with the user's utility charges, while ServiceOrder is all about the room charges generated from a request and completion of a service order. All three payment types are related to the Billing entity, which is directly connected to the user's unit (room) through the UnitID field. Therefore, each time any type of payment is generated, the user would be billed for that specific type of payment, but can all be queried by the UnitID of their unit.

- Identifying vs. non-identifying relationships: This database primarily uses non-identifying relationships. In order to maintain data integrity in non-identifying relationships, some precautions need to be taken during implementation. First of all, we need to implement foreign key constraints on the non-primary key IDs. Additionally, we need to make sure that the fields that contain foreign keys in all entities are not null.