REPORT WORKSHOP #1

DataBase Foundations

Author:

Villalba Delgadillo Esteban Alejandro - 20212020064

Professor:

Eng. Carlos Andrés Sierra, M.Sc.

Francisco José de Caldas District University

Faculty of Engineering

Systems Engineering

September 2024

Content Table

User Stories

- As a potential resident, I want to view apartments and prices, so that I can decide which apartment to rent or buy.
- As an administrator, I want to track public services consumption and resident complaints, so that I can manage services and respond to issues efficiently.
- As a resident, I want direct communication with administration and the ability to reserve common spaces, so that I can handle services and reservations easily.
- As a resident, I want my parking space to be assigned and see residents with overdue administration fees, so that I can know my space and payment responsibilities.
- As a resident, I want to see meeting dates, submit anonymous complaints, and participate in surveys, so that I can stay informed and voice my opinion on decisions.
- As a resident, I want information about the security personnel, so that I know who is responsible for safety in the complex.
- As a resident, I want a reservation system for common areas and to pay for services, so that I can manage my bookings and payments online.
- As a resident, I want to reserve communal halls, so that I can ensure availability for events.
- As a resident, I want to know administration office hours, information about administrators, and complex rules, so that I can follow the guidelines and contact the right person.
- As an Administrator, I want an app to authorize guest entries and receive notifications for bills and payments, so that I can manage the security and finances from my phone.
- As a resident, I want a security module to preauthorize visitors and view visitor history, so that I can manage who enters the complex and keep track of past visits.
- As a resident, I want a maintenance management system to report issues in common areas or my apartment, so that I can notify problems quickly and ensure they are resolved.
- As a resident, I want reminders about service deadlines or upcoming events, so that I can stay informed and avoid missing important dates.
- As a resident, I want customer service and communication features, and the ability to hold virtual assemblies, so that I can handle issues and attend meetings remotely.
- As an Administrator, I want communication with security, other apartments, and shared agendas, so that I can coordinate with the community and security staff easily.

Technical and design considerations/decisions

1. Prioritization of Core Entities

We prioritized entities based on the user feedback and the essential functionalities required for the application. The **Apartment**, **Block**, **Payment**, **Reservation**, **Incident**, and **User** entities were identified as the core elements because they directly support the main features of the system:

- Listing apartments and blocks.
- Managing payments and financial transactions.
- Allowing reservations of common spaces.
- Logging incidents related to maintenance and management.

These core entities were implemented first to ensure that the minimum viable product (MVP) addresses the most requested functionalities.

2. One-to-One Relationships for User Management

To simplify user roles and access control, a **one-to-one relationship** between **User** and both **Resident** and **Staff** was implemented. This decision allows the system to manage all users under a single User entity, which is then linked to specific roles (resident or staff). The advantage of this design is that:

- It centralizes authentication and role management within the **User** entity.
- It allows the system to easily differentiate between different types of users, while still leveraging a unified login and permission system.

3. User-Resident and User-Staff Role Distinction

In line with user feedback, only residents should be able to make reservations or report incidents, while staff members are responsible for managing incidents, payments, and overall administration. This design reflects the following:

- **User-Resident Relationship**: A resident is tied to a user, and the system verifies this association before allowing reservations or incident reporting.
- **User-Staff Relationship**: Staff accounts handle administrative tasks and receive permissions based on their assigned role in the system, such as managing incidents and payments.

4. Handling Reservations through User

Only **Users** who are also **Residents** can make reservations for common spaces, and this relationship is validated at the application level. Instead of creating a direct relationship between **Resident** and **Reservation**, the **UserID** serves as the foreign key in the **Reservation** entity. This decision was driven by the need to manage all actions (reservations, payments, incidents) through the **User** entity, maintaining a consistent structure across the system.

5. Common Space Reservations and Timing

The **Reservation** entity is designed to handle the scheduling of common spaces, ensuring that no overlaps occur between reservations. Constraints such as DateEndTime > DateStartTime ensure that the system only accepts valid time frames, preventing double bookings and conflicts within the system.

Database Design

Step 1 – Define Components

Component 1 (High Priority - Most Requested):

1. **Apartments and Blocks:**
   Display a list of all available blocks and apartments, including details such as number, status, and availability.
2. **Payments:**
   Receive payments for administration services and all related expenses, including maintenance and reservations.
3. **Common Space Reservations:**
   A module for reserving common spaces such as halls, gyms, and recreational areas.
4. **Maintenance and Incidents:**
   Allow residents to report issues in common areas or their apartments and track incidents until resolved.
5. **Visitor Authorization:**
   A module for residents to preauthorize visitors and view visitor history.
6. **Staff:**
   Detailed information about all staff working in the complex, including administrators, security, cleaning, and maintenance staff. This module will include role, work schedule, and contact details when necessary.

Component 2 (Low Priority - Additional Features):

1. **Communication with Administration:**
   A module to send and receive direct messages from the administration, allowing residents to handle administrative matters more efficiently.
2. **Surveys and Complaints:**
   Enable the creation of community surveys and allow residents to submit complaints, whether anonymous or not.
3. **Meeting Dates and Rules:**
   Display important meeting dates and provide access to the complex’s rules and regulations.

Step 2 – Define entities

- Apartment – E1
- Block – E2
- Payment – E3
- Common Space – E4
- Reservation – E5
- Incident – E6
- Resident – E7
- Visitor – E8
- Staff – E9
- User – E10

Step 3 – Define attributes per entity

E1. Apartment:

- **ApartmentID (PK):** Unique identifier for the apartment.
- **BlockID (FK):** Identifier for the block where the apartment is located.
- **Number:** Apartment number or identifier.
- **Status:** Current status of the apartment (e.g., available, occupied, in maintenance).
- **RentValue:** Monthly rent amount.

E2. Block:

- **BlockID (PK):** Unique identifier for the block.
- **Name:** Name or number of the block.
- **Description:** Description of the block.
- **Address:** Address or location details of the block.
- **NumberOfApartments:** Total number of apartments in the block.

E3. Payment:

- **ReferenceNumberID (PK):** Unique number identifier for the payment.
- **ApartmentID (FK):** Identifier for the apartment related to the payment.
- **UserID (FK):** Identifier for the user making the payment.
- **StaffID (FK):** Identifier for the staff member processing or approving the payment.
- **Amount:** Amount paid.

- **PaymentType:** Type of payment (e.g., rent, administration, maintenance).
- **Date:** Date of the payment.

E4. Common Space:

- **SpaceID (PK):** Unique identifier for the common space.
- **Name:** Name of the common space (e.g., gym, hall).
- **Description:** Description of the common space.
- **Capacity:** Maximum capacity of the space.
- **Location:** Location details within the complex.

E5. Reservation:

- **ReservationID (PK):** Unique identifier for the reservation.
- **UserID (FK):** Identifier for the user making the reservation (usually a resident).
- **SpaceID (FK):** Identifier for the common space being reserved.
- **DateStartTime:** Start time of the reservation.
- **DateEndTime:** End time of the reservation.

E6. Incident:

- **IncidentID (PK):** Unique identifier for the incident.
- **UserID (FK):** Identifier for the user who reported the incident.
- **ApartmentID (FK):** Identifier for the apartment where the incident occurred (if applicable).
- **SpaceID (FK):** Identifier for the common space where the incident occurred (if applicable).
- **StaffID (FK):** Identifier for the staff member assigned to handle the incident.
- **Description:** Description of the incident or maintenance issue.
- **Status:** Current status of the incident (e.g., reported, in progress, resolved).
- **DateReported:** Date when the incident was reported.
- **DateResolved:** Date when the incident was resolved (if applicable).

E7. Resident:

- **ResidentID (PK):** Unique identifier for the resident.
- **UserID (FK):** Identifier for the user account of the resident.
- **ApartmentID (FK):** Identifier for the apartment the resident occupies.
- **Name:** Name of the resident.

- **ContactNumber:** Phone number or contact details.
- **Address:** Address of the resident.

E8. Visitor:

- **VisitorID (PK):** Unique identifier for the visitor.
- **ResidentID (FK):** Identifier for the resident who preauthorized the visitor.
- **Name:** Name of the visitor.
- **EntryTime:** Time of entry.
- **ExitTime:** Time of exit (if applicable).
- **Purpose:** Purpose of the visit.

E9. Staff:

- **StaffID (PK):** Unique identifier for the staff member.
- **UserID (FK):** Identifier for the user account of the staff member.
- **Name:** Name of the staff member.
- **Role:** Role or job title (e.g., administrator, security, cleaning).
- **Schedule:** Work schedule or shifts.
- **ContactDetails:** Contact information (e.g., phone number, email).

E10. User:

- **UserID (PK):** Unique identifier for the user.
- **Username:** Username for login purposes.
- **Password:** Encrypted password for authentication.
- **Role:** Role or type of user (e.g., resident, staff).
- **Status:** Account status (e.g., active, inactive).

Step 4 – Define Relationships

- Apartment – E1
- Block – E2
- Payment – E3
- Common Space – E4
- Reservation – E5
- Incident – E6
- Resident – E7
- Visitor – E8
- Staff – E9
- User – E10

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
E1	■	X	X			X	X			
E2	X	■								
E3	X		■						X	X
E4				■	X	X				
E5				X	■					X
E6	X			X		■			X	X
E7	X						■	X		X
E8							X	■		
E9			X			X			■	X
E10			X		X	X	X		X	■

Step 5 – Define Relationships Types

E1 n…1 E2 / E1 1…n E3 / E1 1…n E6 / E1 1…1 E7

E3 n…1 E9 / E3 n…1 E10

E4 1…n E5 / E4 1…n E6

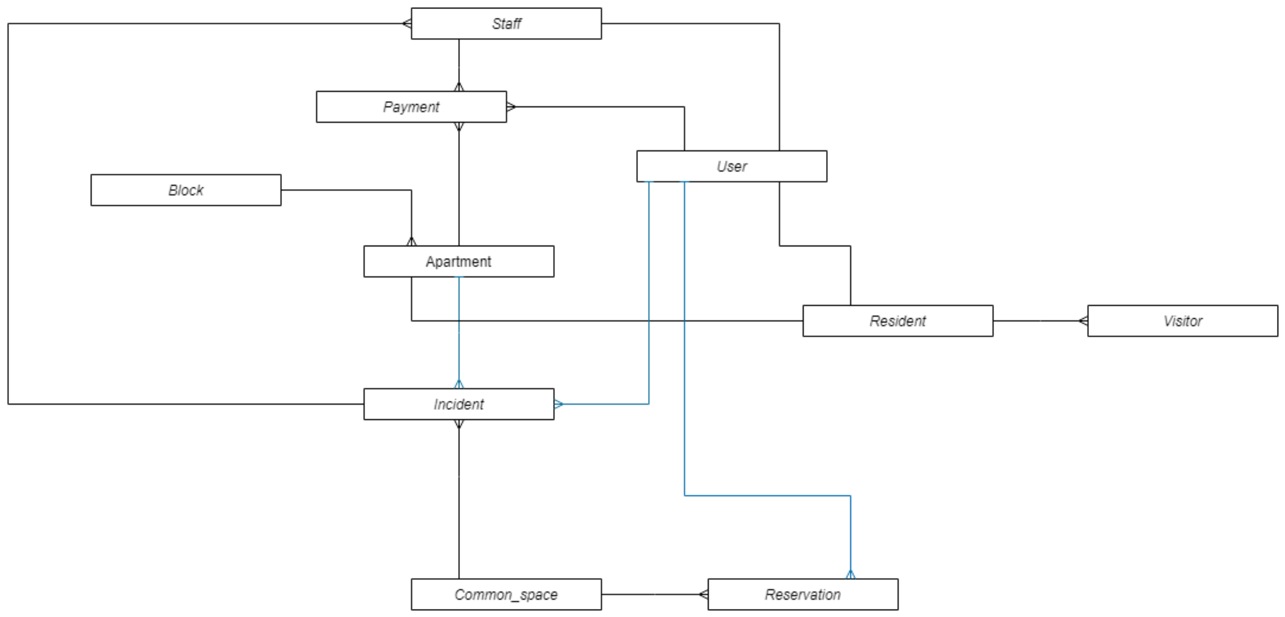
E5 n…1 E10

E6 1…n E9 / E6 n…1 E10

E7 1…n E8 / E7 1…1 E10

E9 1…1 E10

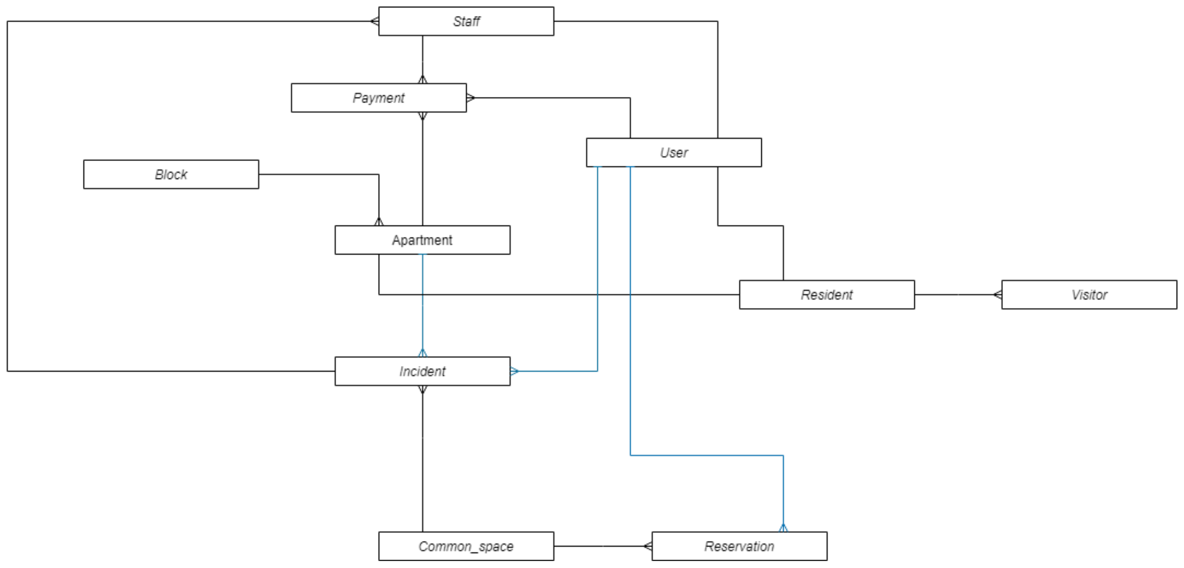
Step 6 – First Entity-Relationship Draw



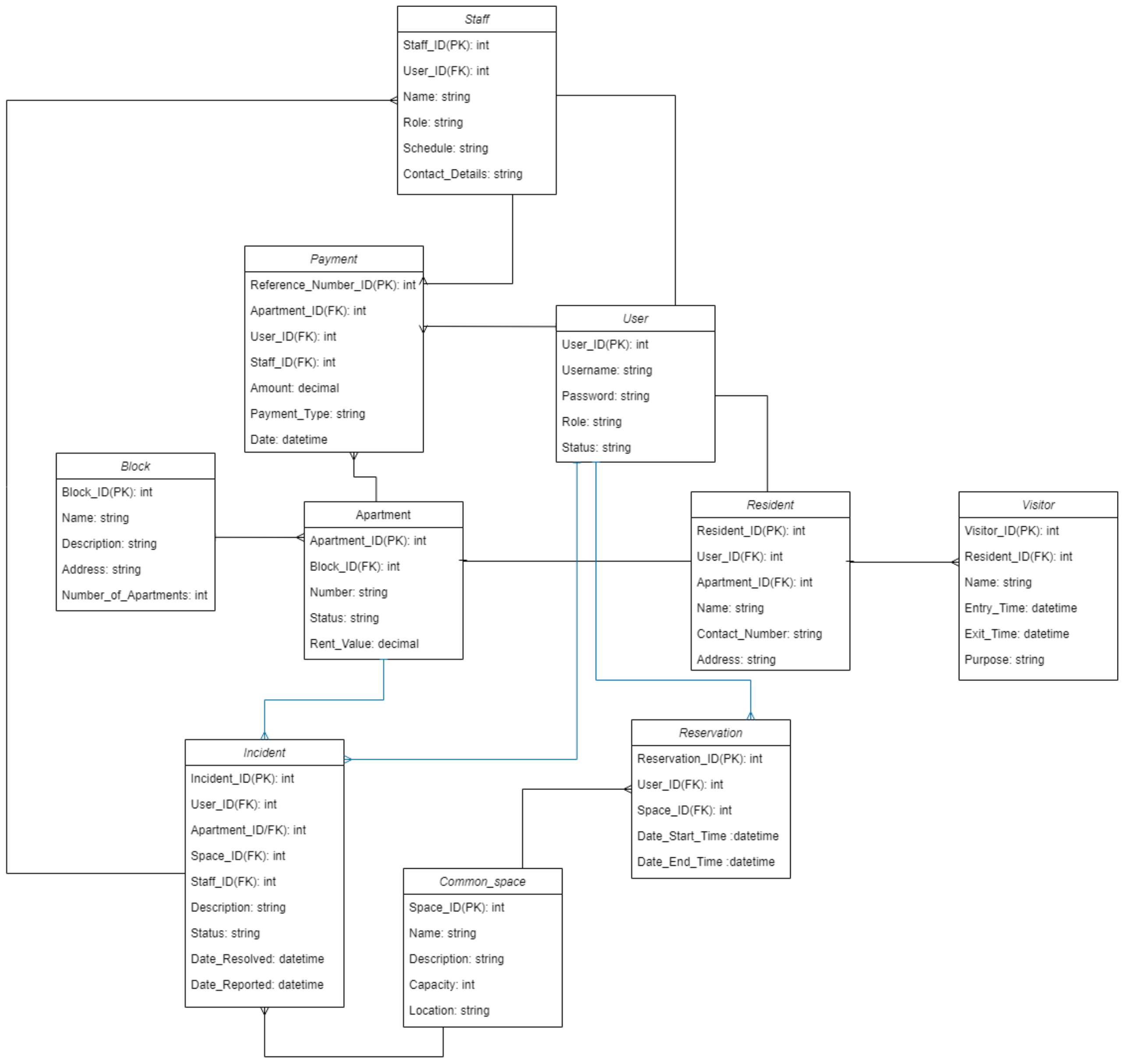
Step 7 - First Split Many-to-Many Relationships

Because no relationship resulted n...n this step is not necessary

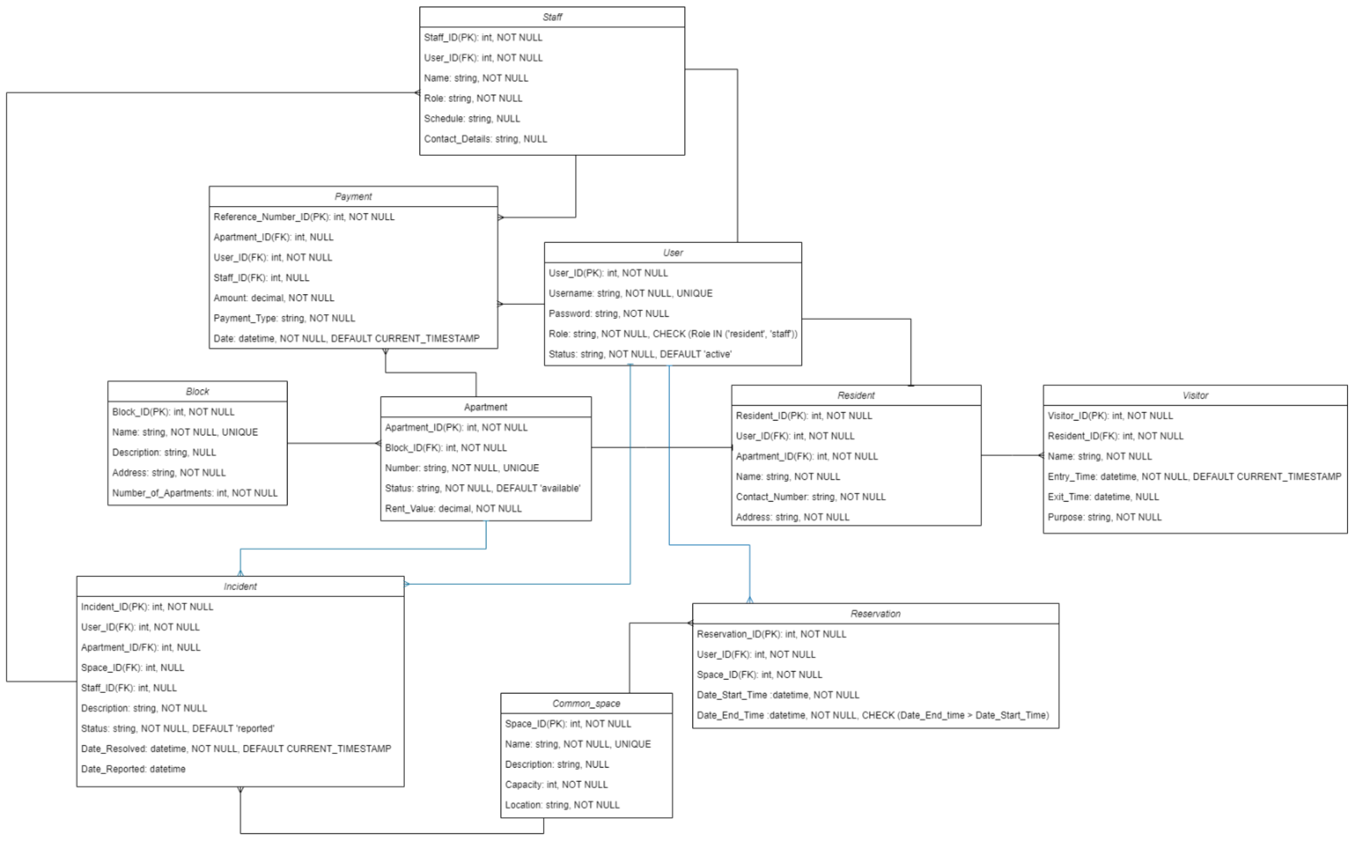
Step 8. Second Entity-Relationship Draw



Step 9 - Get Data-Structure E-R M



Step 10 - Define Constraints and Properties of Data


Staff
Staff_ID(PK): int, NOT NULL
User_ID(FK): int, NOT NULL
Name: string, NOT NULL
Role: string, NOT NULL
Schedule: string, NULL
Contact_Details: string, NULL
Payment
Reference_Number_ID(PK): int, NOT NULL
Apartment_ID(FK): int, NULL
User_ID(FK): int, NOT NULL
Staff_ID(FK): int, NULL
Amount: decimal, NOT NULL
Payment_Type: string, NOT NULL
Date: datetime, NOT NULL, DEFAULT CURRENT_TIMESTAMP
User
User_ID(PK): int, NOT NULL
Username: string, NOT NULL, UNIQUE
Password: string, NOT NULL
Role: string, NOT NULL, CHECK (Role IN ('resident', 'staff'))
Status: string, NOT NULL, DEFAULT 'active'
Block
Block_ID(PK): int, NOT NULL
Name: string, NOT NULL, UNIQUE
Description: string, NULL
Address: string, NOT NULL
Number_of_Apartments: int, NOT NULL
Apartment
Apartment_ID(PK): int, NOT NULL
Block_ID(FK): int, NOT NULL
Number: string, NOT NULL, UNIQUE
Status: string, NOT NULL, DEFAULT 'available'
Rent_Value: decimal, NOT NULL
Resident
Resident_ID(PK): int, NOT NULL
User_ID(FK): int, NOT NULL
Apartment_ID(FK): int, NOT NULL
Name: string, NOT NULL
Contact_Number: string, NOT NULL
Address: string, NOT NULL
Visitor
Visitor_ID(PK): int, NOT NULL
Resident_ID(FK): int, NOT NULL
Name: string, NOT NULL
Entry_Time: datetime, NOT NULL, DEFAULT CURRENT_TIMESTAMP
Exit_Time: datetime, NULL
Purpose: string, NOT NULL
Incident
Incident_ID(PK): int, NOT NULL
User_ID(FK): int, NOT NULL
Apartment_ID(FK): int, NULL
Space_ID(FK): int, NULL
Staff_ID(FK): int, NULL
Description: string, NOT NULL
Status: string, NOT NULL, DEFAULT 'reported'
Date_Resolved: datetime, NOT NULL, DEFAULT CURRENT_TIMESTAMP
Date_Reported: datetime
Reservation
Reservation_ID(PK): int, NOT NULL
User_ID(FK): int, NOT NULL
Space_ID(FK): int, NOT NULL
Date_Start_Time :datetime, NOT NULL
Date_End_Time :datetime, NOT NULL, CHECK (Date_End_time > Date_Start_Time)
Common_space
Space_ID(PK): int, NOT NULL
Name: string, NOT NULL, UNIQUE
Description: string, NULL
Capacity: int, NOT NULL
Location: string, NOT NULL