# SuiteMate

**No More Gloomy Roomie :)**

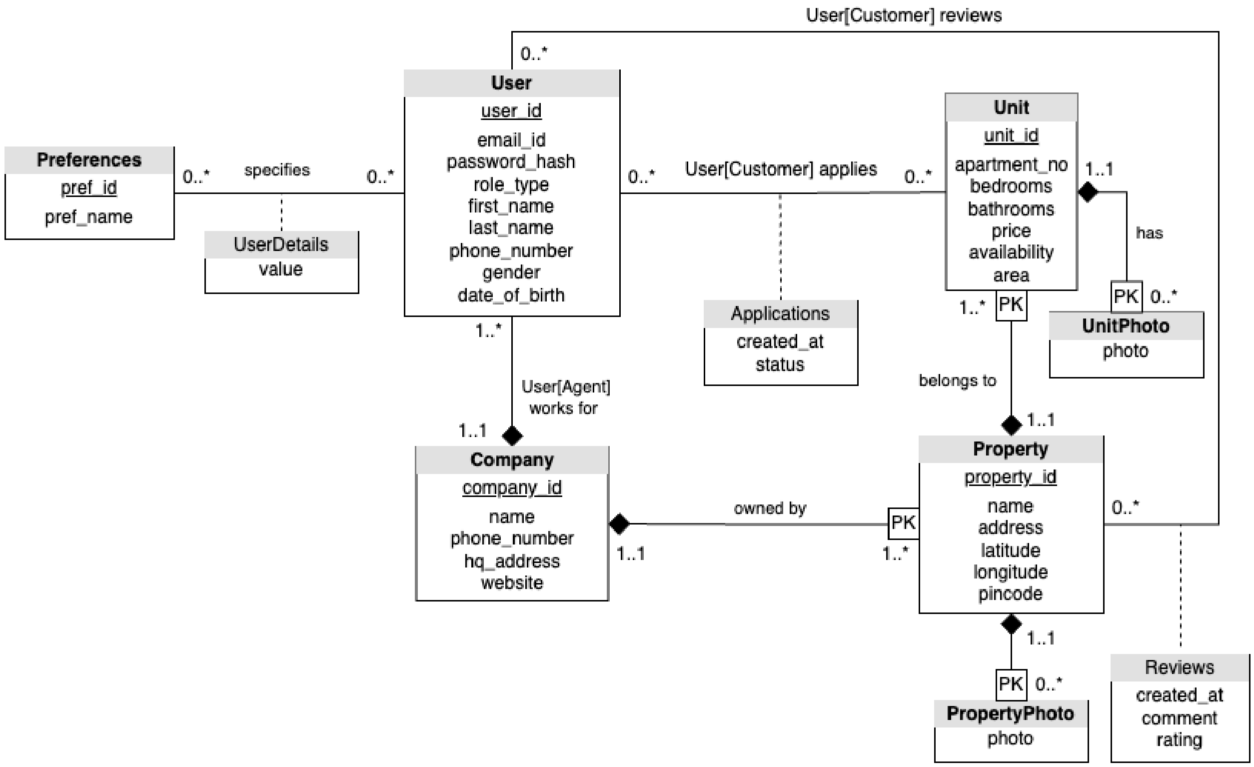
**Project Track 1 : Stage 2**

Database Design

**Team034 - TeamNescafe**

Mehul Oswal - mehuljo2
Daksh Gandhi - dakshg3
Daksh Pokar - dakshp2
Dhruv Kathpalia - dhruvk5

# UML Diagram




# Entities

## 1. Users

User is an entity which corresponds to all the users of SuiteMate.

### Assumptions

- A User can either be a Customer or an Agent working for a leasing company.

### Relationships and Cardinality

- A User[Customer] can *review* 0 or more Properties.
- A User[Customer] can *apply* to 0 or more Units.
- A User[Customer] *can have* 0 or more preferences for potential roommates.
- A User[Agent] *works for* exactly 1 company.

```
User (
    user_id: INT [PK],
    email_id: VARCHAR(255) UNIQUE,
    password_hash: VARCHAR(255),
    role_type: ENUM(Customer, Agent),
    first_name: VARCHAR(255),
    last_name: VARCHAR(255),
    phone_number: VARCHAR(20),
    gender: VARCHAR(10),
    date_of_birth: DATE
)
```

## **2.** Company

It is an entity representing leasing companies which own the rental properties and employ Agents to handle rental applications.

### Assumptions

- A Company needs to own at least one rental property.
- These are *prefilled* rows with the company information.

### Relationships and Cardinality

- A Company can *employ* 0 or more Agents.
- A Company can *own* 1 or more properties.

```
Company(
    company_id: INT [PK],
    name: VARCHAR(255),
    phone_number: VARCHAR(20),
    hq_address: VARCHAR(255),
    website: VARCHAR(255),
)
```

## **3.** Property

Property is an entity representing a rental building consisting of individual units/apartments for rent. A property belongs to a leasing company.

### Assumptions

- A property can be created, updated and deleted by a User[Agent] who works for the same company as the property.

### Relationships and Cardinality

- A property must be *owned* by exactly 1 company.
- A property can *have* 0 or more photos.
- A property can *contain* 1 or more units.
- A property can be *reviewed* by 0 or more Users[Customer].

```
Property(
    property_id: INT [PK],
    name: VARCHAR(255),
    address: VARCHAR(255),
    latitude: REAL,
    longitude: REAL,
    company_id: INT [FK to Company.company_id],
    pincode: INT
)
```

## **4.** PropertyPhoto

PropertyPhoto is an entity which represents the photos of a property.
PropertyPhoto is a weak entity which cannot exist without the property

### Assumptions

- Property and Unit can have separate photos.
- A `photo` is a VARCHAR type because it will store a URL (and not bytes).

### Relationships and Cardinality:

- A PropertyPhoto can *belong* to 1 Property only.
- A Property *can* have 0 or more photos.

```
PropertyPhotos(
        property_id: INT [FK to Property.property_id],
        photo: VARCHAR(255)
)
```

## **5.** Unit

A unit is an <u>entity</u> which represents a specific unit in a property. A unit is a weak entity which cannot exist without the property.

### Assumptions

- All the units in a given property will be managed by User[Agent] of the same company.
- A specific unit can be applied to by a specific User[Customer] only once.
- `bathroom` is a REAL datatype since it can also be added as 1.5.

### Relationships and Cardinality

- A Unit can *belong* to exactly 1 Property.
- A Unit can be *applied* to by 0 or more User[Customer].
- A Unit can *have* 0 or more photos.

```
Unit(
        unit_id: INT [PK],
        property_id: INT [FK to Property.property_id],
        apartment_no: INT,
        bedrooms: INT,
        bathrooms: REAL,
        price: REAL,
        availability: BOOLEAN,
        area: INT
)
```

## **6.** UnitPhoto

UnitPhoto is an <u>entity</u> that represents the photos of a unit.
UnitPhoto is a weak entity that cannot exist without the unit.

### Assumptions

- Property and Unit can have separate photos.
- A `photo` is a VARCHAR type because it will store a URL (and not bytes).

### Relationships and Cardinality

- A UnitPhoto can *belong* to 1 Unit only.
- A Unit *can* have 0 or more photos.

| |
|---|
| **UnitPhoto**(<br>unit_id: INT [FK to Unit.unit_id],<br>photo: VARCHAR(255),<br>) |

## **7.** Preferences

Preferences is an <u>entity</u> which describes the types of preferences that a user can have for potential roommates. It is designed in such a way that it can handle *addition / updation / deletion* of preferences in future.

### Assumptions

- These are *<u>prefilled</u>* rows with the preferences like *Gender, Food Choices (Veg/Non-Veg)* and many others.

### Relationships and Cardinality

- A Preference Type can be *specified by* 0 or more Users[Customer].

| |
|---|
| **Preferences**(<br>pref_id: INT [PK],<br>pref_name: VARCHAR(255),<br>) |

# Relationships

## **1.** UserDetails

UserDetails is a **many-to-many** relationship between the Users and Preferences table where a user can define the preferences that he has for searching for roommates. The `value` field in relationship signifies the choice or weightage of the preference. Let's say if it's a preference like “Dietary Preference", we can store a value which can be either Veg, Non-Veg or Vegan, or for “Cleanliness” the values will range from “1-5”.

### Assumptions

- A preference may or may not be specified by a User[Customer].
- These values will be used by the preference matching algorithm to find potential roommates.

```
UserDetails(
    user_id: INT [PK] [FK to User.user_id],
    pref_id: INT [PK] [FK to Preferences.pref_id],
    value: VARCHAR(255),
)
```

## **2.** AgentCompanyRelationship

User[Agent] is a special type of User that is employed by a leasing agency. This is a **many-to-one** relationship. As we have a different type of User[Customer] it is not ideal to store the `company_id` inside the User table.

### Assumptions

- An Agent can work for exactly one company.
- A company can have multiple agents.

```
AgentCompanyRelationship(
    user_id: INT [PK] [FK to User.user_id],
    company_id: INT [PK] [FK to Company.company_id]
)
```

## **3.** Applications

Application is a relation between Users and Units. Since multiple Users can apply for multiple Units, we have a separate table for this **many-to-many** relation.

### Assumptions

- An application can be approved/rejected by any agent belonging to the same company as the property in the application.
- If an application for a unit is accepted by an agent, then other applications for the same unit will be automatically rejected.

```
Applications(
    user_id: INT [PK] [FK to User.user_id],
    unit_id: INT [PK] [FK to Unit.unit_id],
    created_at: DATE,
    status: VARCHAR(255)
)
```

## **4.** Reviews

Review is a relation between Users and Properties. Since multiple users can review multiple properties, we have a separate table for this **many-to-many** relation.

### Assumptions

- Users can only review the Properties that they have leased.
- A user can review a property exactly once.

**Reviews**(
    user_id: INT [PK] [FK to User.user_id],
    property_id: INT [PK] [FK to Property.property_id],
    created_at: DATE,
    comment: VARCHAR(255),
    rating: INT,
)

## **5.** Unit - UnitPhoto Relationship

This is a relation between Unit and UnitPhoto. This is a **one-to-many** relationship as each unit can have 0 or many photos. This relation has no additional attributes. `unit_id` is a foreign key in the UnitPhoto table which references the `unit_id` of the Unit table.

## **6.** Property - PropertyPhoto Relationship

This is a relation between Property and PropertyPhotos. This is a **one-to-many** relationship as each property can have 0 or many photos. This relation has no additional attributes. `property_id` is a foreign key in the PropertyPhotos table which references the `property_id` of the Property table.

## **7**. Company - Property Relationship

This relation is between Company and Property. This is a **one-to-many** relationship as each company can have multiple properties. This relation has no additional attributes. `company_id` is a foreign key in the Property table which references the `company_id` of the Company table.

## **8**. Unit - Property Relationship

This relation is between Unit and Property. This is a **many-to-one** relationship as each property can have multiple units. This relation has no additional attributes. `property_id` is a foreign key in the Unit table which references the `property_id` of the Property table.

# Normalization

The database design will be based on the following 11 normalized tables. A relation R is in 3rd Normalization Form (3NF): if whenever there is a nontrivial dependency A1, A2, ..., An → B for R, then *{A1, A2, ..., An}* is a super-key for R, or B is part of a key. This holds for our tables.

All our tables follow the following:

- No partial dependencies exist, so it satisfies the first normal form (1NF).
- There are no transitive dependencies, so it also satisfies the second normal form (2NF).
- Since all non-key attributes are functionally dependent on the primary key, it satisfies the third normal form (3NF).

## 1. Users

**Dependencies:** No transitive dependencies are present. All non-key attributes are functionally dependent on the primary key (`user_id`).
**Conditions for 3NF:** It satisfies 3NF since it doesn't have any transitive dependencies.

## 2. Company

**Dependencies:** No transitive dependencies are present. All non-key attributes are functionally dependent on the primary key (`company_id`).
**Conditions for 3NF:** It satisfies 3NF since it doesn't have any transitive dependencies.

## 3. Property

**Dependencies:** The attributes (`name, address, latitude, longitude`) are functionally dependent on the primary key (`property_id`). The `company_id` foreign key establishes a relationship with the Company table.
**Conditions for 3NF:** It satisfies 3NF since it doesn't have any transitive dependencies.

## 4. PropertyPhotos

**Dependencies:** All attributes are functionally dependent on the primary key `(photo_id, property_id)`.
**Conditions for 3NF:** It satisfies 3NF since it doesn't have any transitive dependencies.

## 5. Unit

**Dependencies:** All attributes are functionally dependent on the primary key (`unit_id`). The company_id foreign key establishes a relationship with the Property table.
**Conditions for 3NF:** It satisfies 3NF since it doesn't have any transitive dependencies.

## 6. UnitPhoto

**Dependencies:** All attributes are functionally dependent on the primary key (`photo_id, unit_id`).

**Conditions for 3NF:** It satisfies 3NF since it doesn't have any transitive dependencies.

## 7. Preferences

**Dependencies:** All attributes are functionally dependent on the primary key (`pref_id`).
**Conditions for 3NF:** It satisfies 3NF since it doesn't have any transitive dependencies.

## 8. UserDetails

**Dependencies:** All attributes are functionally dependent on the composite primary key (`user_id, pref_id`).
**Conditions for 3NF:** It satisfies 3NF since it doesn't have any transitive dependencies.

## 9. AgentCompanyRelationship

**Dependencies:** All attributes are functionally dependent on the composite primary key (`user_id, company_id`).
**Conditions for 3NF:** It satisfies 3NF since it doesn't have any transitive dependencies.

## 10. Applications

**Dependencies:** All attributes are functionally dependent on the composite primary key `(user_id, unit_id).`
**Conditions for 3NF:** It satisfies 3NF since it doesn't have any transitive dependencies.

## 11. Reviews

**Dependencies:** All attributes are functionally dependent on the composite primary key `(user_id, property_id).`
**Conditions for 3NF:** It satisfies 3NF since it doesn't have any transitive dependencies.

From our analysis, all tables are in 3NF. All non-primary attributes in each table are directly dependent on the primary key of their respective tables, and no non-primary attribute determines another non-primary attribute. We chose 3NF to avoid the loss of information and preserve the dependency.