# Introduction

The Real Estate Management System (REMS) is a comprehensive solution designed to streamline the management of properties, tenants, leases, clients, agents, and payments. Our system aims to simplify the operational processes of real estate management, ensuring data accuracy, consistency, and accessibility.

This report outlines the conceptual and logical design phases of the database development process. It describes the entities, attributes, and relationships modeled, highlights the challenges encountered during the design process, and explains the decisions made to ensure a robust, scalable, and efficient database structure.

# Entities, Attributes, and Relationships

- Entities and Attributes:

* Properties: id (PK), property\_name, property\_address, property\_type, property\_size, property\_price, property\_status, user\_id (FK), tenant\_id (FK), lease\_id (FK), image\_id (FK), featured.
* Tenants: tenant\_id (PK), tenant\_name, tenant\_contact, tenant\_email, tenant\_address, property\_id (FK), lease\_start (FK), lease\_end (FK), rent\_amount (FK).
* Leases: lease\_id (PK), tenant\_id (FK), property\_id (FK), lease\_start, lease\_end, rent\_amount.
* Payments: payment\_id (PK), lease\_id (FK), payment\_date, amount, payment\_method.
* Users (Clients): id (PK), username, password, created\_at.
* Images: id (PK), image\_name, image\_path, uploaded\_at, property\_id (FK).
* Agents: Yet under development

- Relationships:

* A **property** can have multiple leases, but a **lease** is associated to one **property** (1 - \*).
* A **tenant** can have one active lease at a time, and a **lease** must be lined with a single **tenant** (1 - 1).
* A **lease** can have multiple **payments**, but a **payment** belongs to one **lease** (1 - \*).
* An **agent** can have multiple **clients**, and a **client** can have multiple **agents** (\* - \*).

# Challenge and Considerations

* **Data Integrity and Consistency:** Maintaining data integrity was crucial, especially when defining foreign key constraints to ensure valid relationships between tables. This required careful planning to avoid potential orphaned records and ensure data consistency during updates and deletions.
* **Avoiding Data Redundancy:** The database structure was designed to ensure that information, such as tenant details or property data, is stored only once and referenced wherever necessary. This approach minimizes duplication and simplifies updates.
* **Scalability:** The database was designed with future scalability in mind, allowing for the addition of new entities, such as inspection records or utility management, without disrupting the existing structure. For instance, fields like status in properties or leases were included to accommodate future enhancements.
* **Performance Optimization:** Ensuring the database could handle complex queries efficiently required careful selection of indexes and data types for key attributes such as property\_id, tenant\_id, and lease\_id. These choices ensure faster query execution and better handling of larger datasets.
* **User Requirements:** The database structure was tailored to meet practical user requirements, such as easy access to lease histories, maintenance requests, and payment records. Reports and query generation were prioritized to support day-to-day operations effectively.

# Tables, Fields, and Data Types

- Properties

* property\_id (INT, PK, AUTO\_INCREMENT)
* property\_name (VARCHAR(100), NOT NULL)
* property\_address (VARCHAR(255), NOT NULL)
* property\_type (VARCHAR(100), NOT NULL)
* property\_size (VARCHAR(100), NOT NULL)
* property\_price (INT(100), NOT NULL)
* property\_status (VARCHAR(100), NOT NULL)
* user\_id (INT(6), FK)
* tenant\_id (INT(11), FK)
* lease\_id (INT(11), FK)
* image\_id (INT(11), FK)
* featured (VARCHAR(10), DEFAULT ‘no’)

- Tenants

* tenant\_id (INT, PK, AUTO\_INCREMENT)
* tenant\_name (VARCHAR(100), NOT NULL)
* tenant\_contact (VARCHAR(15))
* tenant\_email (VARCHAR(100))
* tenant\_address (TEXT)
* property\_id (INT(11), FK)
* lease\_start (DATE, FK)
* lease\_end (DATE, FK)
* rent\_amount (INT(11), FK)

- Leases

* lease\_id (INT(11), PK, AUTO\_INCREMENT)
* tenant\_id (INT(11), FK, NOT NULL)
* property\_id (INT(11), FK, NOT NULL)
* lease\_start (DATE, NOT NULL)
* lease\_end (DATE, NOT NULL)
* rent\_amount (DECIMAL(10, 2), NOT NULL)

- Payments

* payment\_id (INT(11), PK, AUTO\_INCREMENT)
* lease\_id (INT(11), FK, NOT NULL)
* payment\_date (DATE, NOT NULL)
* amount (DECIMAL(10, 2), NOT NULL)
* payment\_method (VARCHAR(50))

- Users (Clients)

* id (INT(6), PK, AUTO\_INCREMENT)
* username (VARCHAR(50), NOT NULL)
* password (VARCHAR(255), NOT NULL)
* created\_at (TIMESTAMP, NOT NULL)

- Images

* id (INT(11), PK, AUTO\_INCREMENT)
* image\_name (VARCHAR(255), NOT NULL)
* image\_path (VARCHAR(255), NOT NULL)
* uploaded\_at (DATETIME)
* property\_id (INT(11), FK)

- Agents

* Yet under production

# Relationships Between Tables

- Primary Keys (PK)

* property\_id
* user\_id
* tenant\_id
* lease\_id
* payment\_id
* image\_id

- Foreign Keys (FK)

* **user\_id** in **Properties** references **id** in **Users**
* **tenant\_id** in **Properties** references **tenant\_id** in **Tenants**
* **lease\_id** in **Properties** references **lease\_id** in **Leases**
* **image\_id** in **Properties** references **id** in **Images**
* **property\_id** in **Tenants** references **property\_id** in **Properties**
* **lease\_start** in **Tenant** references **lease\_start** in **Leases**
* **lease\_end** in **Tenant** references **lease\_end** in **Leases**
* **rent\_amount** in **Tenant** references **rent\_amount** in **Leases**
* **tenant\_id** in **Leases** references **tenant\_id** in **Tenants**
* **property\_id** in **Leases** references **property\_id** in **Properties**
* **lease\_id** in **Payments** references **lease\_id** in **Leases**

# Transforming ERD into RDM

The ERD was transformed into a relational schema by mapping each entity to a table and establishing foreign key relationships based on identified associations. This ensures logical consistency and adherence to normalization standards.

# Design Decisions

* **Entity Segmentation:** Each entity was clearly defined to represent a real-world concept, minimizing data redundancy.
* **Attribute Selection:** Attributes were chosen to support system functionalities while avoiding unnecessary complexity.
* **Data Types:** Data types were selected based on expected data values, ensuring efficiency and accuracy.
* **Normalization:** The design adheres to 3NF, eliminating redundant data and ensuring attribute dependencies are on the primary key.
* **Indexes:** Primary keys and foreign keys were indexed to enhance data retrieval speed.

# Conclusion

The database design for the Real Estate Management System provides a robust and scalable framework for managing properties, tenants, leases, maintenance requests, and payments. By addressing challenges such as data integrity, scalability, and user requirements, the design ensures efficient operations and supports future growth. The structured approach taken in the conceptual and logical design phases guarantees that the system meets current needs while being adaptable for additional features and enhancements.