PROPERTY RENTAL MANAGEMENT SYSTEM

Team Details:

Vidyullata- PES1UG23CS683
Prerana M P- PES1UG23CS

Abstract:

The *Property Rental Management System* is a full-stack web application built using **MySQL** as the backend database and **Streamlit** as the front-end. The system provides a structured way for property owners to list houses, tenants to browse and book them, employees to manage maintenance tasks, and admins to oversee the entire system. The application performs all core database operations such as creating rental agreements, managing maintenance requests, assigning employees, processing payments, and generating analytics. CRUD operations, complex SQL functions, triggers, and stored procedures are integrated into the system to handle business logic efficiently. The goal of the application is to simplify communication between Owners, Tenants, Employees, and Admin while demonstrating end-to-end database handling, UI integration, and query-based insights.

User Requirement Specification (URS):

1. Purpose of the Project

The purpose of the *Property Rental Management System* is to provide a digital platform where property owners, tenants, employees, and administrators can manage the complete rental process efficiently. The system reduces manual paperwork by allowing tenants to browse houses, request bookings, raise maintenance complaints, and track their payments through a unified interface. Owners can manage their properties, approve agreements, assign maintenance tasks, and view revenue analytics. Admins can handle all CRUD operations for houses, owners, tenants, employees, agreements, and payments.
The project demonstrates an end-to-end database-backed application using MySQL and Streamlit, showcasing how real-world rental workflows can be automated using SQL queries, functions, triggers, and structured UI interactions.

2. Scope of the Project

The scope of the project includes designing and developing a full-stack application integrated with a relational database. The system supports multiple user roles—Admin, Owner, Tenant, and Employee—each having clearly defined functionalities. It includes CRUD operations, authentication, rental agreement handling, maintenance workflows, payment recording, and analytics for owners and employees.

The application will cover tasks such as:

- Maintaining house listings, user profiles, agreements, and payments
- Enabling tenants to browse and book houses
- Providing dashboards for each role
- Handling maintenance assignments and task completion
- Displaying revenue and performance analytics

The project focuses only on features implemented inside the MySQL database and Streamlit application. External payment gateways, OTP login, mobile apps, and automation beyond SQL logic are excluded from the scope.

3. Detailed Description of the Project

The Property Rental Management System is a complete database-driven web application involving four types of users:

Admin

The Admin oversees the entire system and manages all the data. They can add, edit, delete, and view Owners, Houses, Tenants, Employees, Agreements, Maintenance Requests, and Payments. Admin acts as the root controller of the system.

Owner

Owners use the system to manage the properties they have listed. They can add houses, update house information, check rental requests, approve or reject agreements, track payments received, handle maintenance issues raised by tenants, assign employees to fix problems, and view total revenue generated from rented houses.

Tenant

Tenants can browse through all available houses and filter them by rent, city, or house type. They can submit a booking request for a house, view their rental agreement status, raise maintenance requests for issues in the property, and view/record payments made to the owner.

Employee

Employees receive maintenance tasks assigned by the Owner or Admin. They can view the assigned tasks, update completion status, enter the cost incurred, and track their performance such as total tasks completed and average completion time.

Database Layer

The system uses MySQL with tables such as Owner, Tenant, Employee, House, RentalAgreement, MaintenanceRequest, Assignment, and Payment. SQL functions, procedures, joins, and triggers are used to generate analytics like revenue, maintenance cost, agreement count, and task performance.
Every CRUD action in the front-end interacts with the backend database through SQL queries.

4. Functional Requirements (System Functionalities)

System Functionality 1: User Registration & Login

Supports login for Admin, Owner, Tenant, and Employee. Tenants and Owners can register with username, password, and basic details.

System Functionality 2: House Browsing (Tenant)

Tenants can browse available houses, apply filters, and view detailed information such as rent, city, type, and furnishing.

System Functionality 3: Rental Agreement Request (Tenant)

Tenants can request to book a house which creates a pending rental agreement in the system.

System Functionality 4: Agreement Approval (Owner)

Owners can approve or reject rental agreement requests. Approval activates the agreement and updates house status.

System Functionality 5: Maintenance Request Creation (Tenant)

Tenants can raise maintenance requests for houses with active agreements.

System Functionality 6: Maintenance Assignment (Owner/Admin)

Maintenance requests can be assigned to an employee by the owner or admin. Status changes from "Open" to "InProgress".

System Functionality 7: Task Completion (Employee)

Employees can update completion details, including final cost and completion date. Status becomes “Closed”.

System Functionality 8: Payment Recording

Payments can be added by owners or tenants with amount, mode, and status. Owner/admin can update payment status (Pending, Completed, Failed, Refunded).

System Functionality 9: Owner Analytics Dashboard

Owners can view total revenue, active agreements count, maintenance cost, and house availability analytics powered by SQL functions.

System Functionality 10: Employee Performance Dashboard

Employees can view tasks completed, pending tasks, average completion time, and revenue generated from maintenance charges.

System Functionality 11: CRUD Operations (Admin)

Admin has full Create, Read, Update, Delete access to Houses, Tenants, Owners, Employees, Agreements, Maintenance Requests, and Payments.

System Functionality 12: Viewing Tenant Agreements

Tenants can view their agreement details, start/end dates, monthly rent, and agreement status.

System Functionality 13: Viewing Maintenance & Payment History

Tenants and Owners can view lists of previous requests and payments made, with filtering and sorting options

List of Softwares / Tools / Programming Languages Used

1. Database

- **MySQL 8.x**
    - Used for creating tables, defining constraints, writing DDL/DML queries.
    - Used to implement triggers, functions, stored procedures, joins, and analytics queries.

2. Backend Integration / Database Connectivity

- **Python (3.x)**
    - Used to connect the Streamlit front-end with the MySQL database.
    - Executes SQL queries using Python's MySQL connector.
- **MySQL Connector / PyMySQL**
    - Python library for executing CRUD operations from the application.

3. Front-End Framework

- **Streamlit**
    - Used to develop the interactive user interface.

- Provides dashboards for Admin, Owner, Tenant, and Employee.
- Supports forms, tables, charts, and dynamic UI elements.

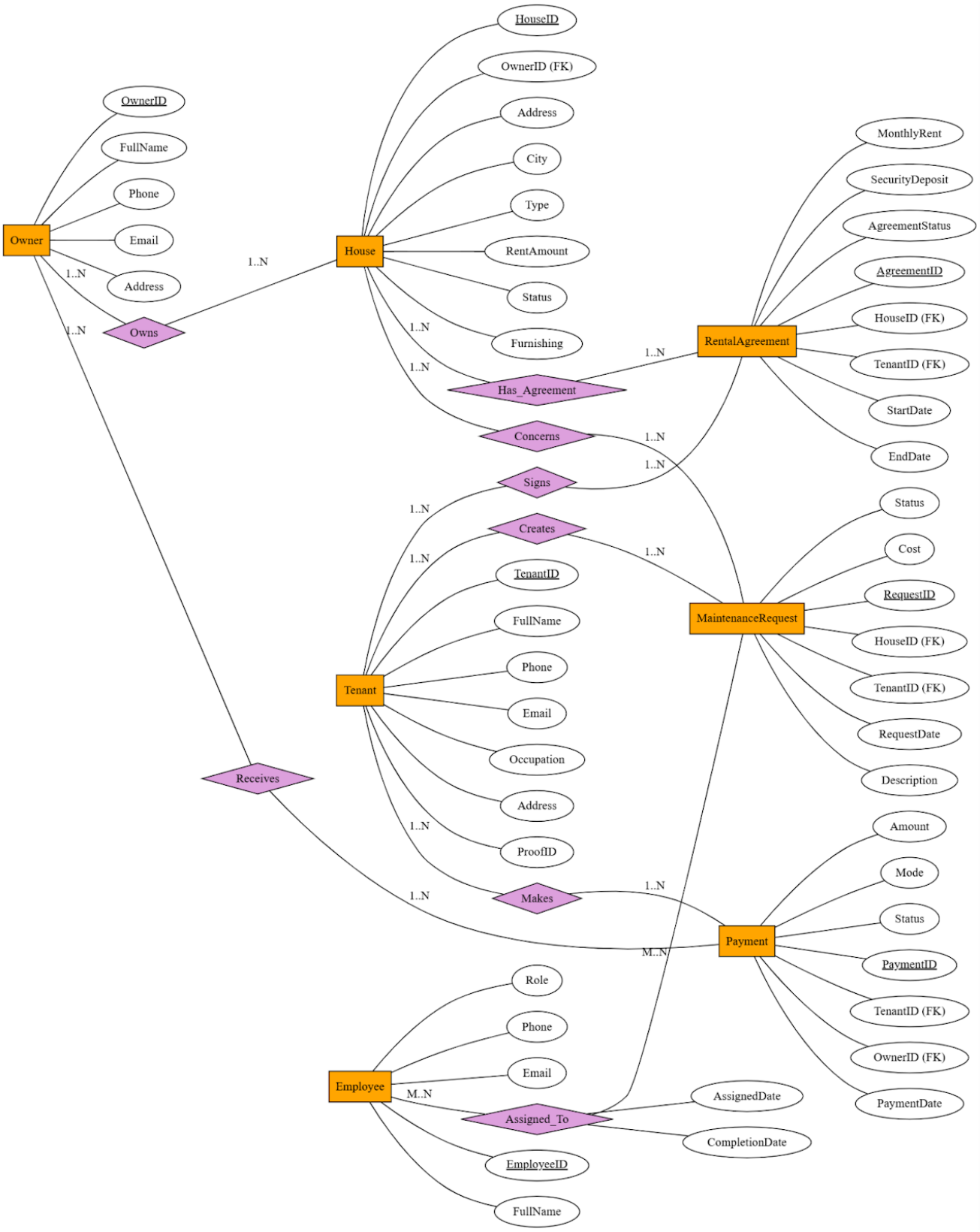
4. Programming Languages

- **Python**
  - Core application logic.
  - Handles user input, database interaction, dashboards, calculations.
- **SQL**
  - Used to create tables, insert data, write procedures, triggers, functions, and analytical queries.

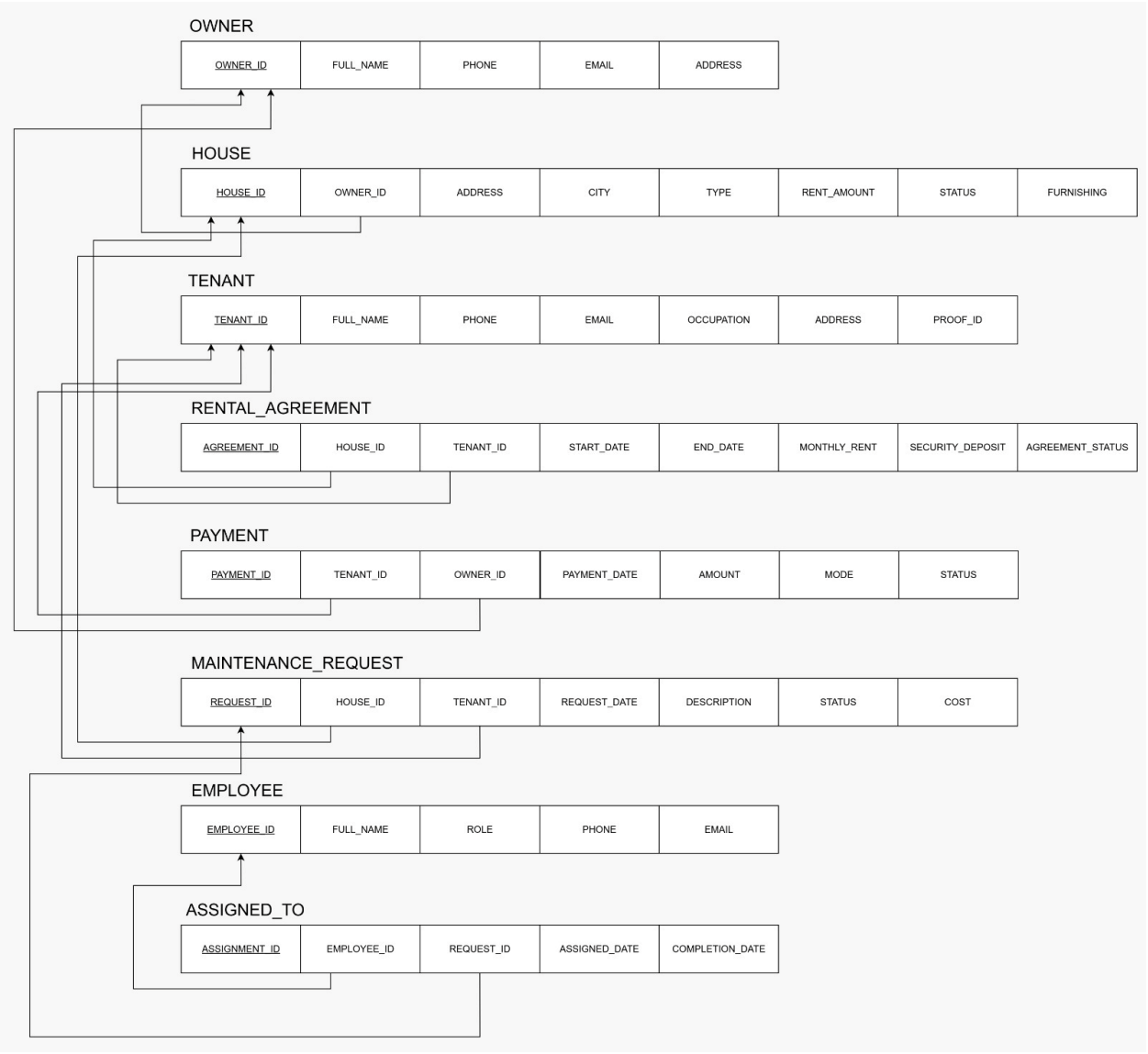
5. Tools & Environment

- **MySQL Workbench**
  - Used for managing the MySQL database
- **Visual Studio Code**
  - Code editor for writing Python and SQL code.
- **Git & GitHub**
  - Version control and repository hosting for the final project submission.

ER DIAGRAM:



RELATIONAL SCHEMA:



DDL COMMANDS:

```sql
-- ============================
-- 1) Create database + use
-- ============================
CREATE DATABASE IF NOT EXISTS `rental_db`
  DEFAULT CHARACTER SET utf8mb4
  DEFAULT COLLATE utf8mb4_general_ci;
USE `rental_db`;

-- ============================
-- 2) Drop existing tables
-- ============================
DROP TABLE IF EXISTS `Assignment`;
DROP TABLE IF EXISTS `Payment`;
DROP TABLE IF EXISTS `RentalAgreement`;
DROP TABLE IF EXISTS `MaintenanceRequest`;
DROP TABLE IF EXISTS `House`;
DROP TABLE IF EXISTS `Employee`;
DROP TABLE IF EXISTS `Tenant`;
DROP TABLE IF EXISTS `Owner`;

 -- ============================
 -- 3) Create tables (DDL)
 -- Note: InnoDB for FK support, utf8mb4 charset
 -- ============================

 -- Owner table
CREATE TABLE `Owner` (
  `OwnerID` INT NOT NULL AUTO_INCREMENT,
  `FullName` VARCHAR(100) NOT NULL,
  `Phone` VARCHAR(20),
  `Email` VARCHAR(150) UNIQUE,
  `Address` VARCHAR(255),
  PRIMARY KEY (`OwnerID`)
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
OwnerID	int	NO	PRI	NULL	auto_increment
FullName	varchar(100)	NO		NULL	
Phone	varchar(20)	YES		NULL	
Email	varchar(150)	YES	UNI	NULL	
Address	varchar(255)	YES		NULL	

```sql
-- Tenant table
CREATE TABLE `Tenant` (
  `TenantID` INT NOT NULL AUTO_INCREMENT,
  `FullName` VARCHAR(100) NOT NULL,
  `Phone` VARCHAR(20),
  `Email` VARCHAR(150) UNIQUE,
  `Occupation` VARCHAR(100),
  `Address` VARCHAR(255),
  `ProofID` VARCHAR(50),
  PRIMARY KEY (`TenantID`)
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
TenantID	int	NO	PRI	NULL	auto_increment
FullName	varchar(100)	NO		NULL	
Phone	varchar(20)	YES		NULL	
Email	varchar(150)	YES	UNI	NULL	
Occupation	varchar(100)	YES		NULL	
Address	varchar(255)	YES		NULL	
ProofID	varchar(50)	YES		NULL	

```sql
-- Employee table
CREATE TABLE `Employee` (
  `EmployeeID` INT NOT NULL AUTO_INCREMENT,
  `FullName` VARCHAR(100) NOT NULL,
  `Role` VARCHAR(50),
  `Phone` VARCHAR(20),
  `Email` VARCHAR(150) UNIQUE,
  PRIMARY KEY (`EmployeeID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
EmployeeID	int	NO	PRI	NULL	auto_increment
FullName	varchar(100)	NO		NULL	
Role	varchar(50)	YES		NULL	
Phone	varchar(20)	YES		NULL	
Email	varchar(150)	YES	UNI	NULL	

```sql
-- House table (FK -> Owner)
CREATE TABLE `House` (
  `HouseID` INT NOT NULL AUTO_INCREMENT,
  `OwnerID` INT NOT NULL,
  `Address` VARCHAR(255),
  `City` VARCHAR(100),
  `Type` VARCHAR(50),
  `RentAmount` DECIMAL(10,2) DEFAULT 0.00,
  `Status` ENUM('Available','Rented','Maintenance','Reserved') DEFAULT 'Available',
  `Furnishing` VARCHAR(50),
  PRIMARY KEY (`HouseID`),
  INDEX `idx_house_owner` (`OwnerID`),
  CONSTRAINT `fk_house_owner` FOREIGN KEY (`OwnerID`) REFERENCES `Owner`(`OwnerID`)
    ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
HouseID	int	NO	PRI	NULL	auto_increment
OwnerID	int	NO	MUL	NULL	
Address	varchar(255)	YES		NULL	
City	varchar(100)	YES		NULL	
Type	varchar(50)	YES		NULL	
RentAmount	decimal(10,2)	YES		0.00	
Status	enum('Available','Rented','Maintenance','Reserv...	YES		Available	
Furnishing	varchar(50)	YES		NULL	

```sql
-- RentalAgreement table (FK -> House, Tenant)
CREATE TABLE `RentalAgreement` (
  `AgreementID` INT NOT NULL AUTO_INCREMENT,
  `HouseID` INT NOT NULL,
  `TenantID` INT NOT NULL,
  `StartDate` DATE,
  `EndDate` DATE,
  `MonthlyRent` DECIMAL(10,2) NOT NULL,
  `SecurityDeposit` DECIMAL(10,2),
  `AgreementStatus` ENUM('Pending','Active','Terminated') DEFAULT 'Pending',
  PRIMARY KEY (`AgreementID`),
  INDEX `idx_ra_house` (`HouseID`),
  INDEX `idx_ra_tenant` (`TenantID`),
  CONSTRAINT `fk_ra_house` FOREIGN KEY (`HouseID`) REFERENCES `House`(`HouseID`)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  CONSTRAINT `fk_ra_tenant` FOREIGN KEY (`TenantID`) REFERENCES `Tenant`(`TenantID`)
    ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
AgreementID	int	NO	PRI	NULL	auto_increment
HouseID	int	NO	MUL	NULL	
TenantID	int	NO	MUL	NULL	
StartDate	date	YES		NULL	
EndDate	date	YES		NULL	
MonthlyRent	decimal(10,2)	NO		NULL	
SecurityDeposit	decimal(10,2)	YES		NULL	
AgreementStatus	enum('Pending','Active','Terminated')	YES		Pending	

```sql
-- MaintenanceRequest (FK -> House, Tenant)
CREATE TABLE `MaintenanceRequest` (
  `RequestID` INT NOT NULL AUTO_INCREMENT,
  `HouseID` INT NOT NULL,
  `TenantID` INT NOT NULL,
  `RequestDate` DATE,
  `Description` TEXT,
  `Status` ENUM('Open','InProgress','Closed','Cancelled') DEFAULT 'Open',
  `Cost` DECIMAL(10,2),
  PRIMARY KEY (`RequestID`),
  INDEX `idx_mr_house` (`HouseID`),
  INDEX `idx_mr_tenant` (`TenantID`),
  CONSTRAINT `fk_mr_house` FOREIGN KEY (`HouseID`) REFERENCES `House`(`HouseID`)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  CONSTRAINT `fk_mr_tenant` FOREIGN KEY (`TenantID`) REFERENCES `Tenant`(`TenantID`)
    ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
RequestID	int	NO	PRI	NULL	auto_increment
HouseID	int	NO	MUL	NULL	
TenantID	int	NO	MUL	NULL	
RequestDate	date	YES		NULL	
Description	text	YES		NULL	
Status	enum('Open','InProgress','Closed','Cancelled')	YES		Open	
Cost	decimal(10,2)	YES		NULL	

```sql
-- Payment table (Tenant -> Owner)
CREATE TABLE `Payment` (
  `PaymentID` INT NOT NULL AUTO_INCREMENT,
  `TenantID` INT NOT NULL,
  `OwnerID` INT NOT NULL,
  `PaymentDate` DATE,
  `Amount` DECIMAL(10,2) NOT NULL,
  `Mode` ENUM('Cash','Card','BankTransfer','UPI','Cheque','Other') DEFAULT 'UPI',
  `Status` ENUM('Completed','Pending','Failed','Refunded') DEFAULT 'Completed',
  PRIMARY KEY (`PaymentID`),
  INDEX `idx_payment_tenant` (`TenantID`),
  INDEX `idx_payment_owner` (`OwnerID`),
  CONSTRAINT `fk_payment_tenant` FOREIGN KEY (`TenantID`) REFERENCES `Tenant`(`TenantID`)
    ON DELETE RESTRICT ON UPDATE CASCADE,
  CONSTRAINT `fk_payment_owner` FOREIGN KEY (`OwnerID`) REFERENCES `Owner`(`OwnerID`)
    ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
PaymentID	int	NO	PRI	NULL	auto_increment
TenantID	int	NO	MUL	NULL	
OwnerID	int	NO	MUL	NULL	
PaymentDate	date	YES		NULL	
Amount	decimal(10,2)	NO		NULL	
Mode	enum('Cash','Card','BankTransfer','UPI','Cheque...	YES		UPI	
Status	enum('Completed','Pending','Failed','Refunded')	YES		Completed	

```
128
129     -- Assignment table to implement the Assigned_To relationship (Employee <-> MaintenanceRequest)
130     -- Relationship has attributes (AssignedDate, CompletionDate) so it becomes a table
131 • CREATE TABLE `Assignment` (
132       `AssignmentID` INT NOT NULL AUTO_INCREMENT,
133       `EmployeeID` INT NOT NULL,
134       `RequestID` INT NOT NULL,
135       `AssignedDate` DATE,
136       `CompletionDate` DATE,
137       PRIMARY KEY (`AssignmentID`),
138       INDEX `idx_assign_emp` (`EmployeeID`),
139       INDEX `idx_assign_req` (`RequestID`),
140       CONSTRAINT `fk_assign_emp` FOREIGN KEY (`EmployeeID`) REFERENCES `Employee`(`EmployeeID`)
141         ON DELETE RESTRICT ON UPDATE CASCADE,
142       CONSTRAINT `fk_assign_req` FOREIGN KEY (`RequestID`) REFERENCES `MaintenanceRequest`(`RequestID`)
143         ON DELETE CASCADE ON UPDATE CASCADE
144     ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
145
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
AssignmentID	int	NO	PRI	NULL	auto_increment
EmployeeID	int	NO	MUL	NULL	
RequestID	int	NO	MUL	NULL	
AssignedDate	date	YES		NULL	
CompletionDate	date	YES		NULL	

```
-- ============================
-- 4) Show table structure (DESC) right after creation
-- ============================
DESC `Owner`;
DESC `Tenant`;
DESC `Employee`;
DESC `House`;
DESC `RentalAgreement`;
DESC `MaintenanceRequest`;
DESC `Payment`;
DESC `Assignment`;

159
160     -- Also list all tables
161 •   SHOW TABLES;
162
163     -- ----------------------------
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Tables_in_rental_db
assignment
employee
house
maintenancerequest
owner
payment
rentalagreement
tenant

DML COMMANDS:

```
163     -- ============================
164     -- 5) Insert sample data (DML) - 3 entries per table
165     -- ============================
166     -- Owners
167 •   INSERT INTO `Owner` (`OwnerID`, `FullName`, `Phone`, `Email`, `Address`) VALUES
168       (1, 'Ravi Kumar', '9876543210', 'ravi.kumar@example.com', '12 MG Road, Bangalore'),
169       (2, 'Priya Sharma', '9876501234', 'priya.sharma@example.com', '45 Residency, Chennai'),
170       (3, 'Suresh Rao', '9445556677', 'suresh.rao@example.com', '8 Park St, Hyderabad');
171
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

OwnerID	FullName	Phone	Email	Address
1	Ravi Kumar	9876543210	ravi.kumar@example.com	12 MG Road, Bangalore
2	Priya Sharma	9876501234	priya.sharma@example.com	45 Residency, Chennai
3	Suresh Rao	9445556677	suresh.rao@example.com	8 Park St, Hyderabad
NULL	NULL	NULL	NULL	NULL

```
172     -- Tenants
173 •   INSERT INTO `Tenant` (`TenantID`, `FullName`, `Phone`, `Email`, `Occupation`, `Address`, `ProofID`) VALUES
174       (1, 'Amit Shah', '9988776655', 'amit.shah@example.com', 'Engineer', 'Flat 101, Tower A', 'ID1001'),
175       (2, 'Neha Gupta', '9876612345', 'neha.gupta@example.com', 'Researcher', 'Flat 202, Tower B', 'ID1002'),
176       (3, 'Rohan Verma', '9900099000', 'rohan.verma@example.com', 'Student', 'PG House, 5th Cross', 'ID1003');
177
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

TenantID	FullName	Phone	Email	Occupation	Address	ProofID
1	Amit Shah	9988776655	amit.shah@example.com	Engineer	Flat 101, Tower A	ID1001
2	Neha Gupta	9876612345	neha.gupta@example.com	Researcher	Flat 202, Tower B	ID1002
3	Rohan Verma	9900099000	rohan.verma@example.com	Student	PG House, 5th Cross	ID1003
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Result 12 | Tenant 13 ×

Output

Action Output

#	Time	Action	Message
49	23:00:06	INSERT INTO `Tenant` (`TenantID`, `FullName`, `Phone`, `Email`, `Occupation`, `Address`, `ProofID`) VALUES (1, 'Amit Shah', '9988776655', 'amit.sha...	3 row(s) affected Rec
50	23:00:12	INSERT INTO `Employee` (`EmployeeID`, `FullName`, `Role`, `Phone`, `Email`) VALUES (1, 'Manoj Das', 'Plumber', '9441112233', 'manoj.das@services....	3 row(s) affected Rec
51	23:00:18	INSERT INTO `House` (`HouseID`, `OwnerID`, `Address`, `City`, `Type`, `RentAmount`, `Status`, `Furnishing`) VALUES (101, 1, '12 MG Road Apt 3B', '...	3 row(s) affected Rec
52	23:00:24	INSERT INTO `RentalAgreement` (`AgreementID`, `HouseID`, `TenantID`, `StartDate`, `EndDate`, `MonthlyRent`, `SecurityDeposit`, `AgreementStatus`) ...	3 row(s) affected Rec
53	23:00:30	INSERT INTO `MaintenanceRequest` (`RequestID`, `HouseID`, `TenantID`, `RequestDate`, `Description`, `Status`, `Cost`) VALUES (2001, 102, 1, '20...	3 row(s) affected Rec
54	23:00:35	INSERT INTO `Payment` (`PaymentID`, `TenantID`, `OwnerID`, `PaymentDate`, `Amount`, `Mode`, `Status`) VALUES (3001, 1, 1, '2024-07-05', 35000....	3 row(s) affected Rec
55	23:00:40	INSERT INTO `Assignment` (`AssignmentID`, `EmployeeID`, `RequestID`, `AssignedDate`, `CompletionDate`) VALUES (4001, 1, 2001, '2024-08-06', N...	3 row(s) affected Rec
56	23:01:16	DESC `Owner`	5 row(s) returned
57	23:01:16	SELECT * FROM `Owner` LIMIT 0, 1000	3 row(s) returned
58	23:01:44	DESC `Tenant`	7 row(s) returned
59	23:01:44	SELECT * FROM `Tenant` LIMIT 0, 1000	3 row(s) returned

```sql
-- Employees
INSERT INTO `Employee` (`EmployeeID`, `FullName`, `Role`, `Phone`, `Email`) VALUES
  (1, 'Manoj Das', 'Plumber', '9441112233', 'manoj.das@services.com'),
  (2, 'Kavita Roy', 'Electrician', '9442223344', 'kavita.roy@services.com'),
  (3, 'Sajan K', 'Carpenter', '9443334455', 'sajan.k@services.com');
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

EmployeeID	FullName	Role	Phone	Email
1	Manoj Das	Plumber	9441112233	manoj.das@services.com
2	Kavita Roy	Electrician	9442223344	kavita.roy@services.com
3	Sajan K	Carpenter	9443334455	sajan.k@services.com
NULL	NULL	NULL	NULL	NULL

```sql
-- Houses (referencing owners)
INSERT INTO `House` (`HouseID`, `OwnerID`, `Address`, `City`, `Type`, `RentAmount`, `Status`, `Furnishing`) VALUES
  (101, 1, '12 MG Road Apt 3B', 'Bangalore', 'Apartment', 25000.00, 'Available', 'Semi-Furnished'),
  (102, 1, '78 Park Lane', 'Bangalore', 'Independent', 35000.00, 'Rented', 'Furnished'),
  (201, 2, '45 Residency Apt 5C', 'Chennai', 'Apartment', 18000.00, 'Available', 'Unfurnished');
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

HouseID	OwnerID	Address	City	Type	RentAmount	Status	Furnishing
101	1	12 MG Road Apt 3B	Bangalore	Apartment	25000.00	Available	Semi-Furnished
102	1	78 Park Lane	Bangalore	Independent	35000.00	Rented	Furnished
201	2	45 Residency Apt 5C	Chennai	Apartment	18000.00	Available	Unfurnished
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```sql
-- RentalAgreements
INSERT INTO `RentalAgreement` (`AgreementID`, `HouseID`, `TenantID`, `StartDate`, `EndDate`, `MonthlyRent`, `SecurityDeposit`, `AgreementStatus`) VALUES
  (1001, 102, 1, '2024-07-01', '2025-06-30', 35000.00, 70000.00, 'Active'),
  (1002, 101, 3, '2025-01-01', '2025-12-31', 25000.00, 50000.00, 'Pending'),
  (1003, 201, 2, '2024-10-01', '2025-09-30', 18000.00, 36000.00, 'Active');
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

AgreementID	HouseID	TenantID	StartDate	EndDate	MonthlyRent	SecurityDeposit	AgreementStatus
1001	102	1	2024-07-01	2025-06-30	35000.00	70000.00	Active
1002	101	3	2025-01-01	2025-12-31	25000.00	50000.00	Pending
1003	201	2	2024-10-01	2025-09-30	18000.00	36000.00	Active
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```sql
-- MaintenanceRequests
INSERT INTO `MaintenanceRequest` (`RequestID`, `HouseID`, `TenantID`, `RequestDate`, `Description`, `Status`, `Cost`) VALUES
  (2001, 102, 1, '2024-08-05', 'Leaking pipe in bathroom', 'Open', NULL),
  (2002, 101, 3, '2025-02-10', 'AC not cooling', 'InProgress', 500.00),
  (2003, 201, 2, '2024-11-20', 'Window grill repair', 'Closed', 1200.00);
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

RequestID	HouseID	TenantID	RequestDate	Description	Status	Cost
2001	102	1	2024-08-05	Leaking pipe in bathroom	Open	NULL
2002	101	3	2025-02-10	AC not cooling	InProgress	500.00
2003	201	2	2024-11-20	Window grill repair	Closed	1200.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
202     -- Payments (Tenant pays Owner)
203 •  INSERT INTO `Payment` (`PaymentID`, `TenantID`, `OwnerID`, `PaymentDate`, `Amount`, `Mode`, `Status`) VALUES
204       (3001, 1, 1, '2024-07-05', 35000.00, 'UPI', 'Completed'),
205       (3002, 3, 1, '2025-01-05', 25000.00, 'BankTransfer', 'Completed'),
206       (3003, 2, 2, '2024-10-05', 18000.00, 'Card', 'Completed');
207
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

PaymentID	TenantID	OwnerID	PaymentDate	Amount	Mode	Status
3001	1	1	2024-07-05	35000.00	UPI	Completed
3002	3	1	2025-01-05	25000.00	BankTransfer	Completed
3003	2	2	2024-10-05	18000.00	Card	Completed
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
207
208     -- Assignments (link Employee <-> MaintenanceRequest)
209 •  INSERT INTO `Assignment` (`AssignmentID`, `EmployeeID`, `RequestID`, `AssignedDate`, `CompletionDate`) VALUES
210       (4001, 1, 2001, '2024-08-06', NULL),
211       (4002, 2, 2002, '2025-02-11', NULL),
212       (4003, 3, 2003, '2024-11-21', '2024-11-22');
213
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

AssignmentID	EmployeeID	RequestID	AssignedDate	CompletionDate
4001	1	2001	2024-08-06	NULL
4002	2	2002	2025-02-11	NULL
4003	3	2003	2024-11-21	2024-11-22
NULL	NULL	NULL	NULL	NULL

CRUD OPERATIONS:

CREATE:

```
241
242     -- ============================
243     -- 7) CRUD examples (show before/after)
244     -- ============================
245
246     -- CREATE:
247 •  SELECT * FROM `Tenant`;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

TenantID	FullName	Phone	Email	Occupation	Address	ProofID
1	Amit Shah	9988776655	amit.shah@example.com	Engineer	Flat 101, Tower A	ID1001
2	Neha Gupta	9876612345	neha.gupta@example.com	Researcher	Flat 202, Tower B	ID1002
3	Rohan Verma	9900099000	rohan.verma@example.com	Student	PG House, 5th Cross	ID1003
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
246     -- CREATE:
247 •   SELECT * FROM `Tenant`;
248 •   INSERT INTO `Tenant` (`TenantID`, `FullName`, `Phone`, `Email`, `Occupation`, `Address`, `ProofID`)
249       VALUES (4, 'Sneha Iyer', '9900112233', 'sneha.iyer@example.com', 'Designer', 'Flat 404, Tower C', 'ID1004');
250 •   SELECT * FROM `Tenant`;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

TenantID	FullName	Phone	Email	Occupation	Address	ProofID
1	Amit Shah	9988776655	amit.shah@example.com	Engineer	Flat 101, Tower A	ID1001
2	Neha Gupta	9876612345	neha.gupta@example.com	Researcher	Flat 202, Tower B	ID1002
3	Rohan Verma	9900099000	rohan.verma@example.com	Student	PG House, 5th Cross	ID1003
4	Sneha Iyer	9900112233	sneha.iyer@example.com	Designer	Flat 404, Tower C	ID1004
NULL	NULL	NULL	NULL	NULL	NULL	NULL

READ:

```
252     -- READ: Show all tenants
253 •   SELECT * FROM Tenant;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

TenantID	FullName	Phone	Email	Occupation	Address	ProofID
1	Amit Shah	9988776655	amit.shah@example.com	Engineer	Flat 101, Tower A	ID1001
2	Neha Gupta	9876612345	neha.gupta@example.com	Researcher	Flat 202, Tower B	ID1002
3	Rohan Verma	9900099000	rohan.verma@example.com	Student	PG House, 5th Cross	ID1003
4	Sneha Iyer	9900112233	sneha.iyer@example.com	Designer	Flat 404, Tower C	ID1004
NULL	NULL	NULL	NULL	NULL	NULL	NULL

UPDATE:

```
255     -- UPDATE:
256 •   SELECT TenantID, FullName, Phone FROM `Tenant` WHERE TenantID = 4;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Co

TenantID	FullName	Phone
4	Sneha Iyer	9900112233
NULL	NULL	NULL

```
255     -- UPDATE:
256 •   SELECT TenantID, FullName, Phone FROM `Tenant` WHERE TenantID = 4;
257 •   UPDATE `Tenant` SET Phone = '9887766554' WHERE TenantID = 4;
258 •   SELECT TenantID, FullName, Phone FROM `Tenant` WHERE TenantID = 4;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell

TenantID	FullName	Phone
4	Sneha Iyer	9887766554
NULL	NULL	NULL

DELETE:

```sql
260     -- DELETE:
261 •   SELECT * FROM `Payment`;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Conte

PaymentID	TenantID	OwnerID	PaymentDate	Amount	Mode	Status
3001	1	1	2024-07-05	35000.00	UPI	Completed
3002	3	1	2025-01-05	25000.00	BankTransfer	Completed
3003	2	2	2024-10-05	18000.00	Card	Completed
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```sql
260     -- DELETE:
261 •   SELECT * FROM `Payment`;
262 •   DELETE FROM `Payment` WHERE `PaymentID` = 3003;
263 •   SELECT * FROM `Payment`;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

PaymentID	TenantID	OwnerID	PaymentDate	Amount	Mode	Status
3001	1	1	2024-07-05	35000.00	UPI	Completed
3002	3	1	2025-01-05	25000.00	BankTransfer	Completed
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```sql
265     -- ============================
266     -- 8) JOIN queries demonstrating relationships
267     -- ============================
268     -- Agreements + Tenant + House
269 •   SELECT ra.AgreementID, ra.StartDate, ra.EndDate, ra.MonthlyRent,
270            t.TenantID, t.FullName AS TenantName,
271            h.HouseID, h.Address AS HouseAddress, h.City
272     FROM `RentalAgreement` ra
273     JOIN `Tenant` t ON ra.TenantID = t.TenantID
274     JOIN `House` h ON ra.HouseID = h.HouseID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

AgreementID	StartDate	EndDate	MonthlyRent	TenantID	TenantName	HouseID	HouseAddress	City
1001	2024-07-01	2025-06-30	35000.00	1	Amit Shah	102	78 Park Lane	Bangalore
1002	2025-01-01	2025-12-31	25000.00	3	Rohan Verma	101	12 MG Road Apt 3B	Bangalore
1003	2024-10-01	2025-09-30	18000.00	2	Neha Gupta	201	45 Residency Apt 5C	Chennai

```sql
276     -- Which payments did OwnerID = 1 receive?
277 •   SELECT p.PaymentID, p.PaymentDate, p.Amount, t.FullName AS Payer
278     FROM `Payment` p
279     JOIN `Tenant` t ON p.TenantID = t.TenantID
280     WHERE p.OwnerID = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

PaymentID	PaymentDate	Amount	Payer
3001	2024-07-05	35000.00	Amit Shah
3002	2025-01-05	25000.00	Rohan Verma

```
282     -- Maintenance requests with assigned employee(s)
283 •   SELECT mr.RequestID, mr.Description, mr.Status,
284           a.AssignmentID, e.EmployeeID, e.FullName AS EmployeeName, a.AssignedDate, a.CompletionDate
285     FROM `MaintenanceRequest` mr
286     LEFT JOIN `Assignment` a ON mr.RequestID = a.RequestID
287     LEFT JOIN `Employee` e ON a.EmployeeID = e.EmployeeID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

RequestID	Description	Status	AssignmentID	EmployeeID	EmployeeName	AssignedDate	CompletionDate
2001	Leaking pipe in bathroom	Open	4001	1	Manoj Das	2024-08-06	NULL
2002	AC not cooling	InProgress	4002	2	Kavita Roy	2025-02-11	NULL
2003	Window grill repair	Closed	4003	3	Sajan K	2024-11-21	2024-11-22

```
289     -- ============================
290     -- 9) ALTER table
291     -- ============================
292 •   DESC `Tenant`;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
TenantID	int	NO	PRI	NULL	auto_increment
FullName	varchar(100)	NO		NULL	
Phone	varchar(20)	YES		NULL	
Email	varchar(150)	YES	UNI	NULL	
Occupation	varchar(100)	YES		NULL	
Address	varchar(255)	YES		NULL	
ProofID	varchar(50)	YES		NULL	

```
289     -- ============================
290     -- 9) ALTER table
291     -- ============================
292 •   DESC `Tenant`;
293 •   ALTER TABLE `Tenant` ADD COLUMN `IsActive` TINYINT(1) NOT NULL DEFAULT 1;
294 •   DESC `Tenant`;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
TenantID	int	NO	PRI	NULL	auto_increment
FullName	varchar(100)	NO		NULL	
Phone	varchar(20)	YES		NULL	
Email	varchar(150)	YES	UNI	NULL	
Occupation	varchar(100)	YES		NULL	
Address	varchar(255)	YES		NULL	
ProofID	varchar(50)	YES		NULL	
IsActive	tinyint(1)	NO		1	

```
289     -- ============================
290     -- 9) ALTER table
291     -- ============================
292     DESC `Tenant`;
293     ALTER TABLE `Tenant` ADD COLUMN `IsActive` TINYINT(1) NOT NULL DEFAULT 1;
294     DESC `Tenant`;
295     SELECT TenantID, FullName, IsActive FROM `Tenant` LIMIT 5;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Cont

TenantID	FullName	IsActive
1	Amit Shah	1
2	Neha Gupta	1
3	Rohan Verma	1
4	Sneha Iyer	1
NULL	NULL	NULL

FOREIGN KEY ENFORCEMENT CHECK:

```
297     -- ============================
298     -- 10) Foreign key enforcement check
299     -- ============================
300     -- The following is expected to FAIL because OwnerID 9999 does not exist.
301     INSERT INTO `House` (`HouseID`, `OwnerID`, `Address`, `City`) VALUES (999, 9999, 'Nowhere Road', 'Nowhere');
302
303     -- ============================
304
```

Context Help | Snippets

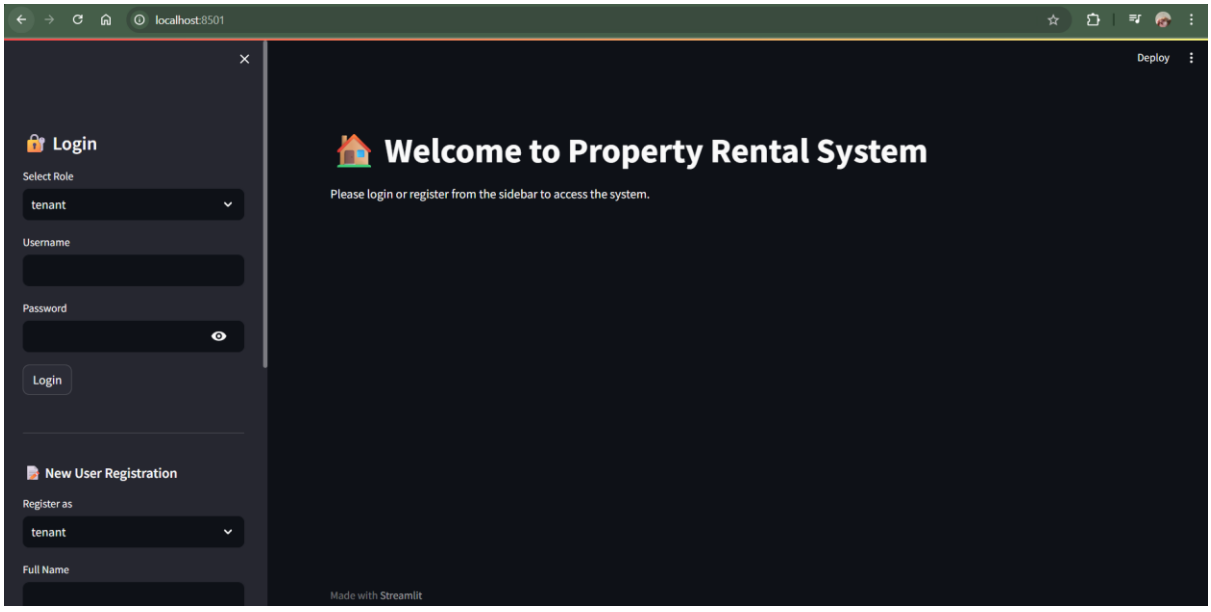
Output

Action Output

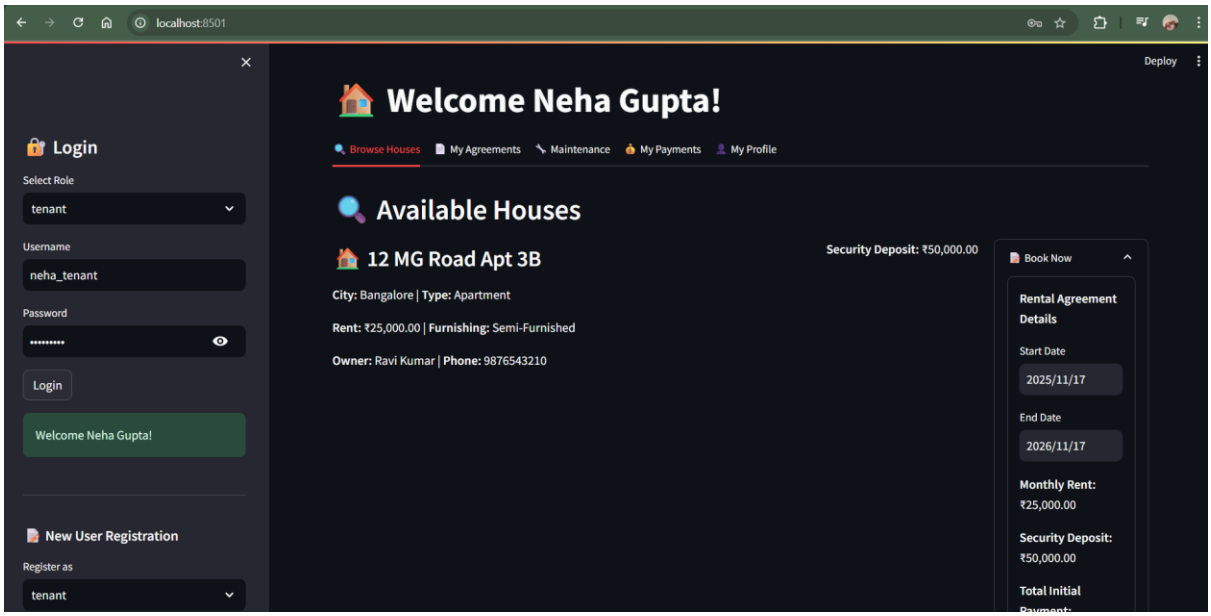
#	Time	Action	Message	Duration / Fetch
87	23:15:14	DESC `Tenant`	8 row(s) returned	0.000 sec / 0.000 sec
88	23:15:35	SELECT TenantID, FullName, IsActive FROM `Tenant` LIMIT 5	4 row(s) returned	0.000 sec / 0.000 sec
89	23:17:42	INSERT INTO `House` (`HouseID`, `OwnerID`, `Address`, `City`) VALUES (999, 9999, 'Nowhere Road', 'Nowhere')	Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`rental_db`.`house`, CONSTRAINT `fk_house_owner` FOREIGN KEY...	0.016 sec

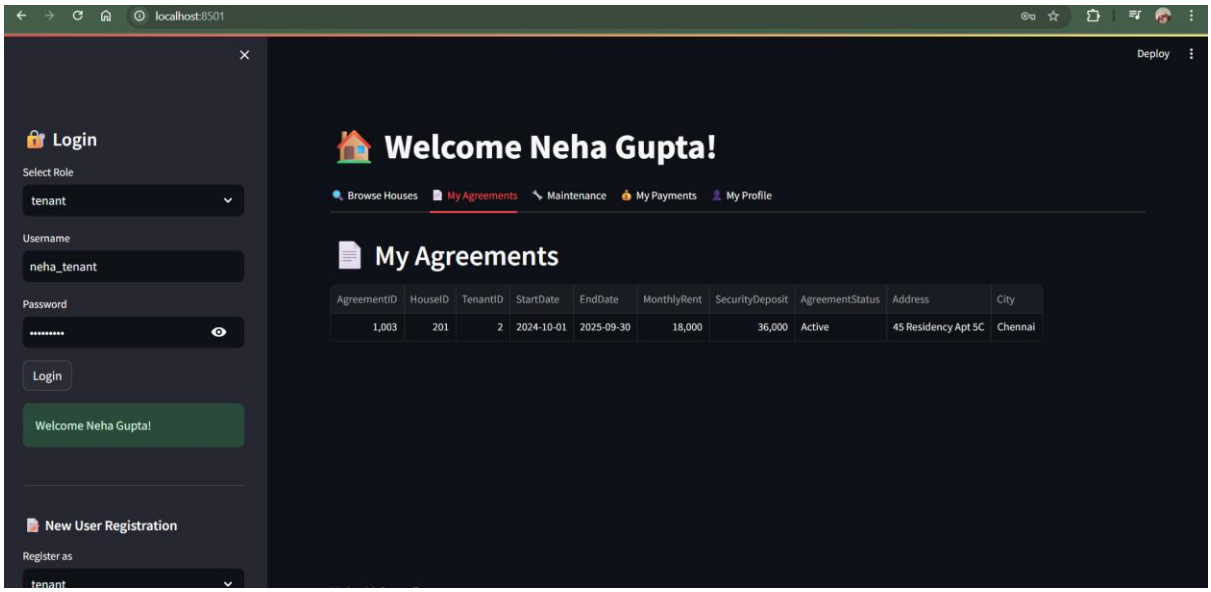
FRONTEND FUNCTIONALITY SCREENSHOTS

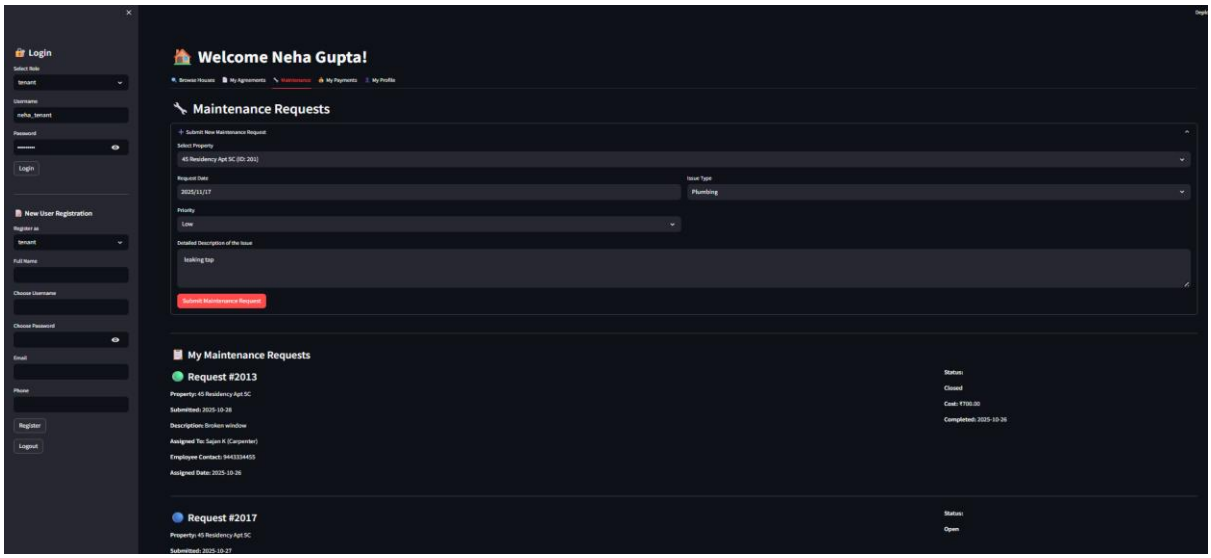
Initial login page:

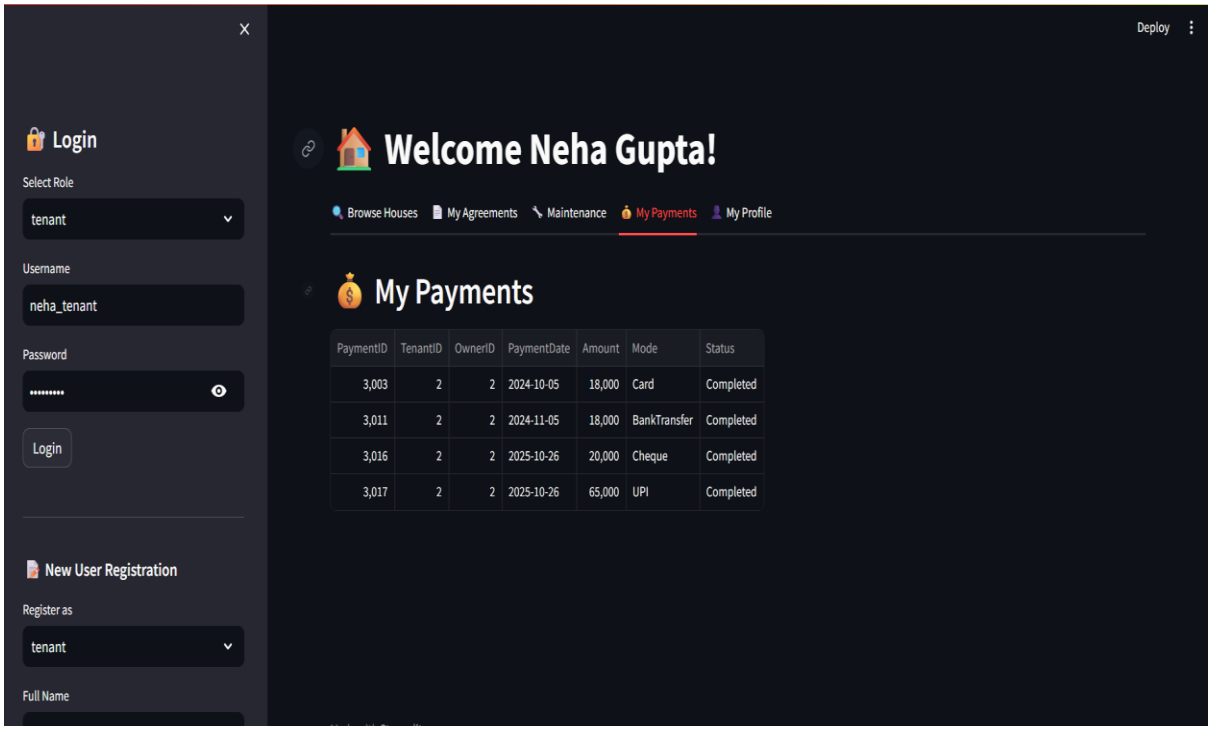


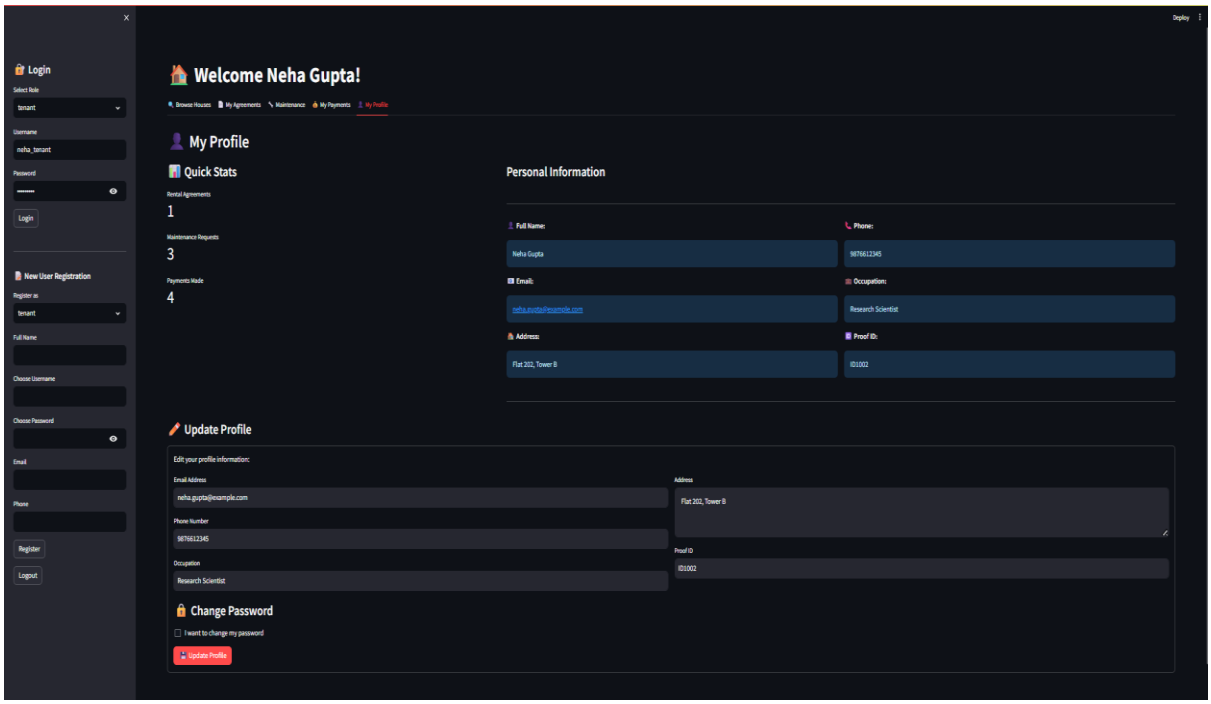
Tenant features:



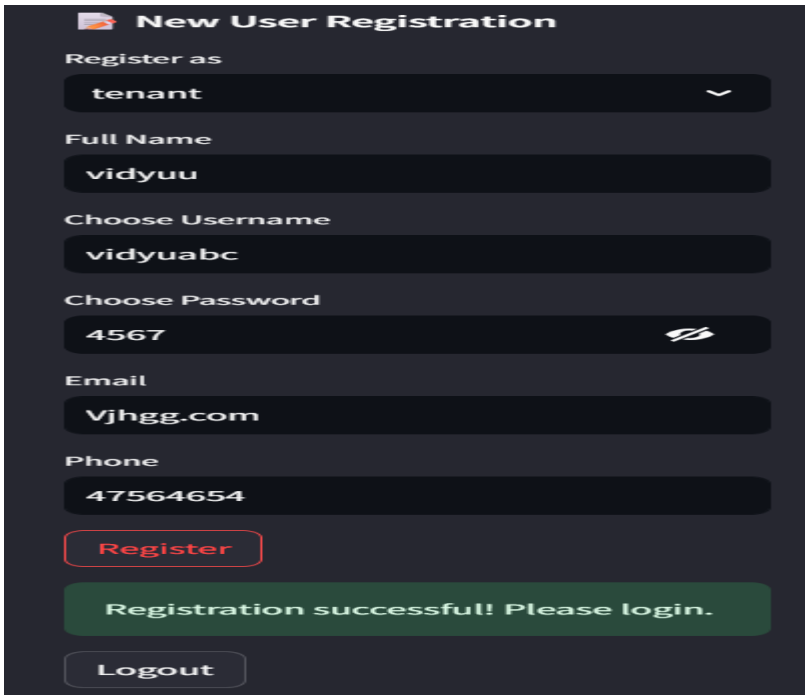




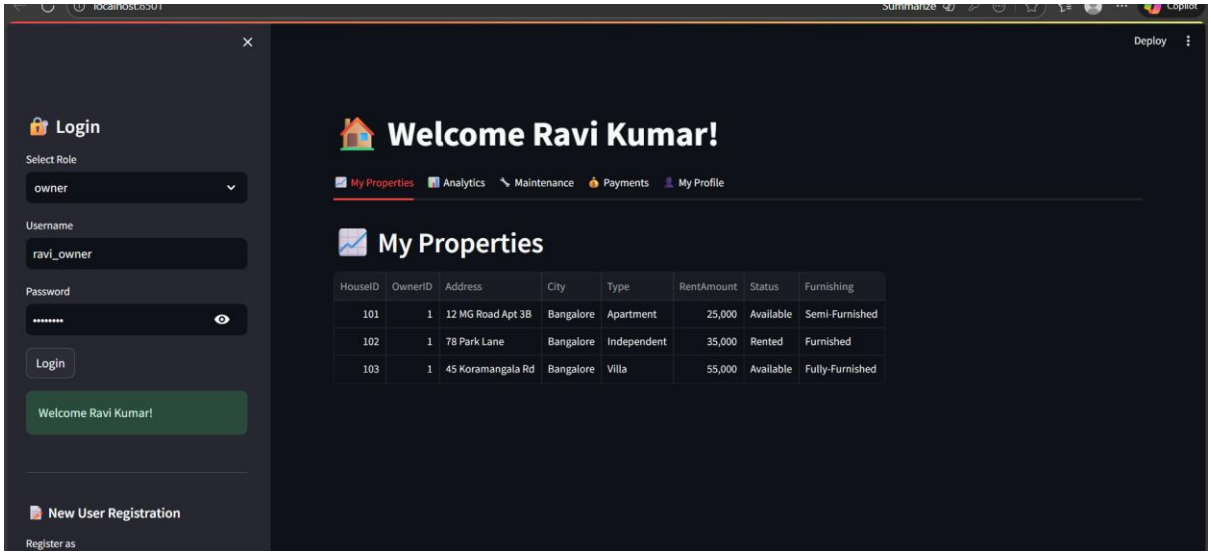

Deploy
Login
Select Role
tenant
Username
neha_tenant
Password
Login
New User Registration
Register as
tenant
Full Name
Welcome Neha Gupta!
Browse Houses
My Agreements
Maintenance
My Payments
My Profile
My Payments
PaymentID TenantID OwnerID PaymentDate Amount Mode Status
3,003 2 2 2024-10-05 18,000 Card Completed
3,011 2 2 2024-11-05 18,000 BankTransfer Completed
3,016 2 2 2025-10-26 20,000 Cheque Completed
3,017 2 2 2025-10-26 65,000 UPI Completed

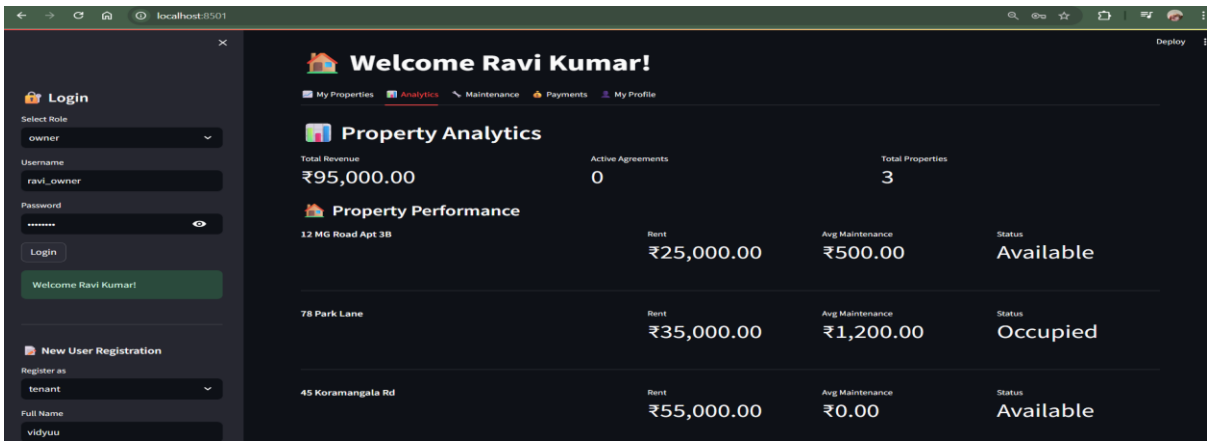

Welcome Neha Gupta!
My Profile
Quick Stats
Rental Agreements
1
Maintenance Requests
3
Payments Made
4
Personal Information
Full Name:
Neha Gupta
Phone:
9876612345
Email:
Occupation:
Research Scientist
Address:
Flat 202, Tower B
Proof ID:
ID1002
Update Profile
Edit your profile information:
Email Address
neha.gupta@example.com
Address
Flat 202, Tower B
Phone Number
9876612345
Proof ID
ID1002
Occupation
Research Scientist
Change Password
I want to change my password
Update Profile

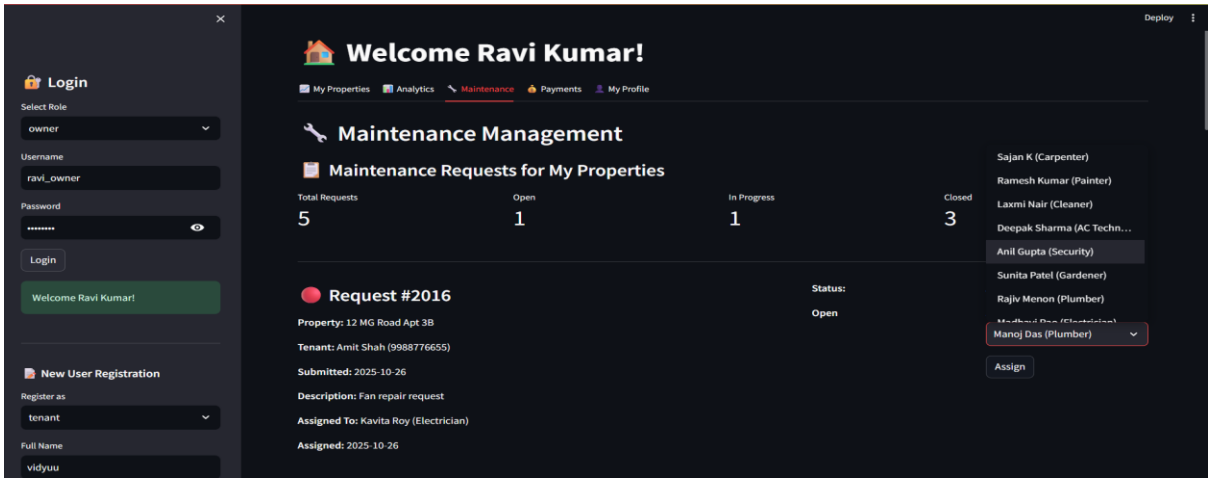
New User Registration

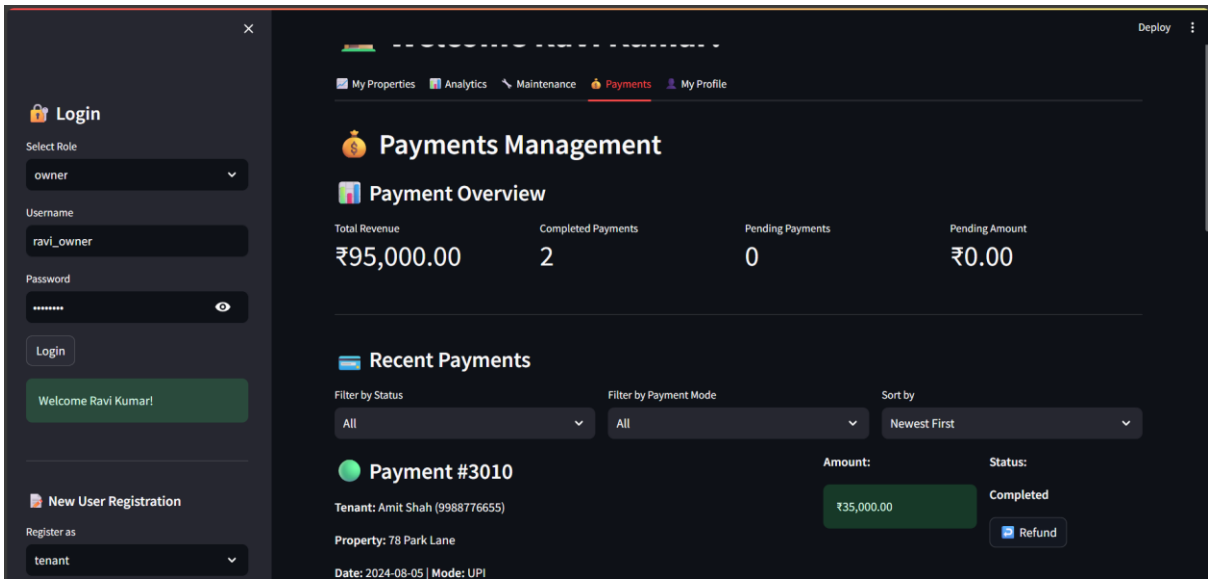


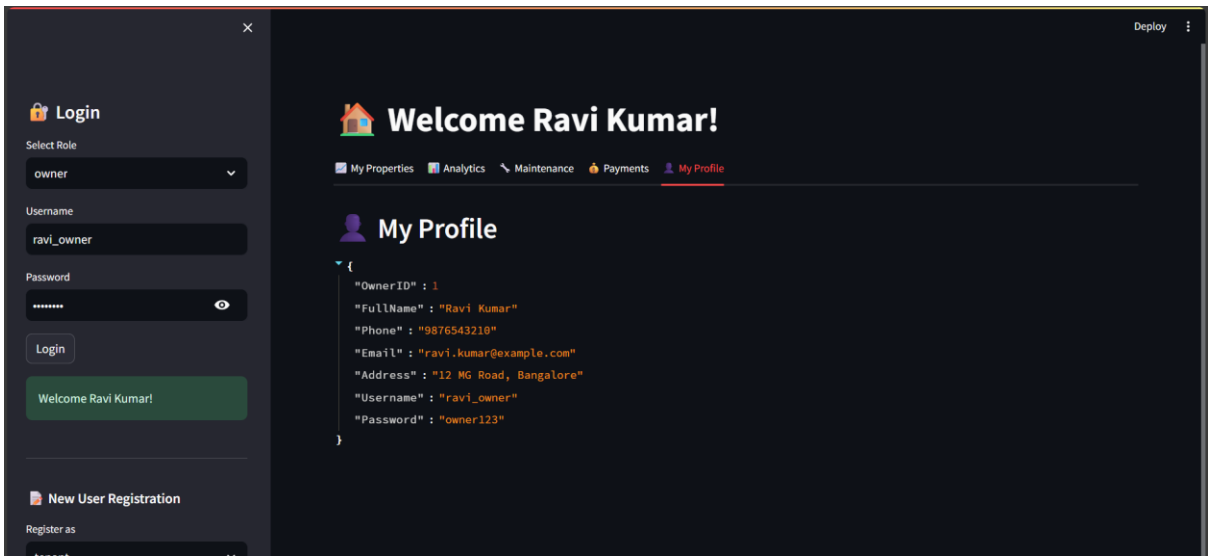
Owner features:



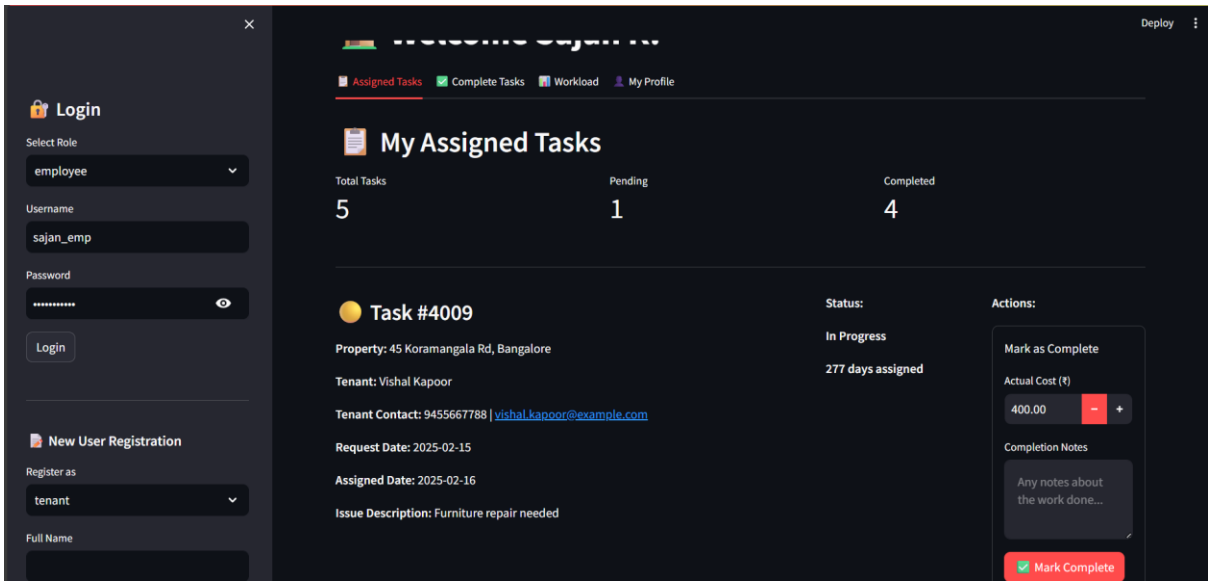


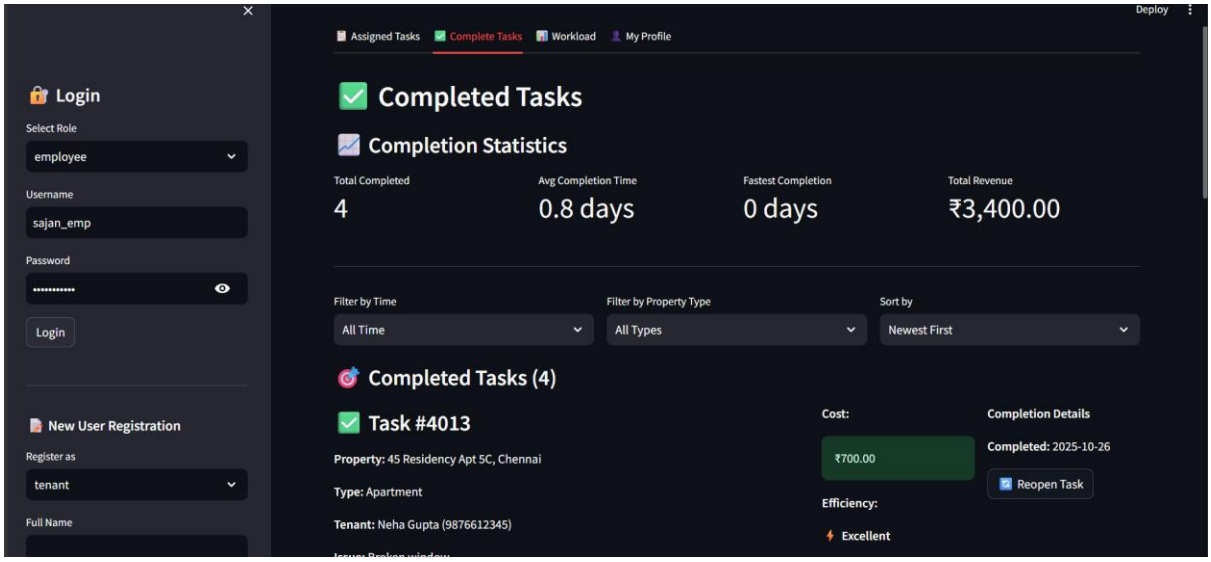

Login
Select Role
owner
Username
ravi_owner
Password
Login
Welcome Ravi Kumar!
New User Registration
Register as
tenant
Full Name
vidyuu
Deploy
Welcome Ravi Kumar!
My Properties
Analytics
Maintenance
Payments
My Profile
Maintenance Management
Maintenance Requests for My Properties
Total Requests
5
Open
1
In Progress
1
Closed
3
Request #2016
Property: 12 MG Road Apt 3B
Tenant: Amit Shah (9988776655)
Submitted: 2025-10-26
Description: Fan repair request
Assigned To: Kavita Roy (Electrician)
Assigned: 2025-10-26
Status:
Open
Sajan K (Carpenter)
Ramesh Kumar (Painter)
Laxmi Nair (Cleaner)
Deepak Sharma (AC Techn...
Anil Gupta (Security)
Sunita Patel (Gardener)
Rajiv Menon (Plumber)
Manoj Das (Plumber)
Assign

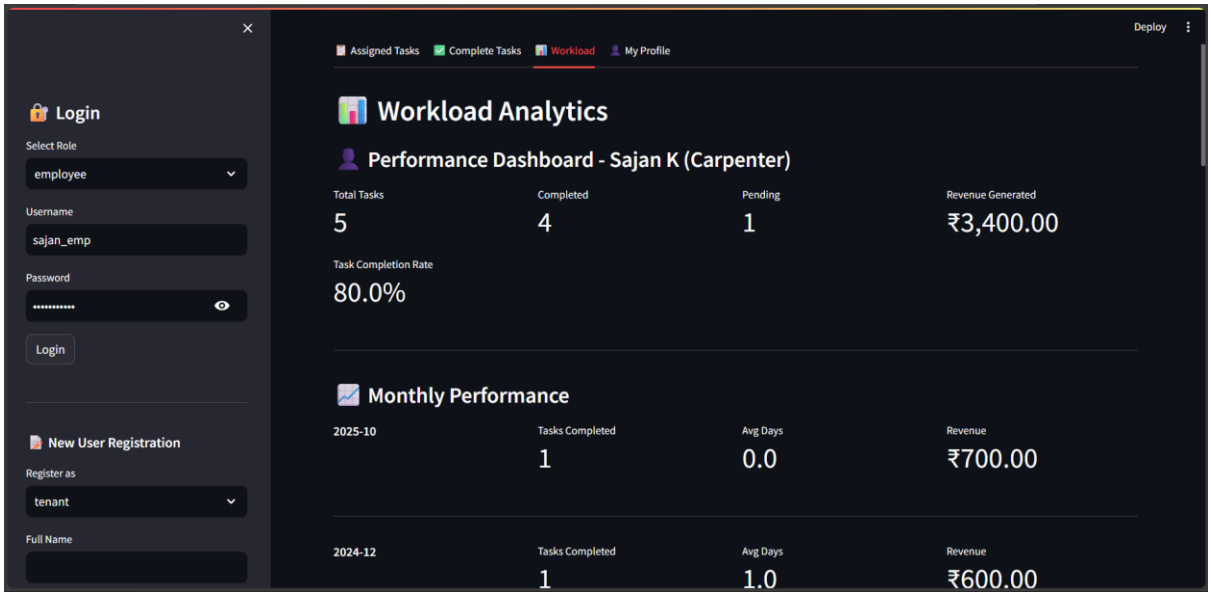

Login
Select Role
owner
Username
ravi_owner
Password
Login
Welcome Ravi Kumar!
New User Registration
Register as
tenant
Deploy
My Properties
Analytics
Maintenance
Payments
My Profile
Payments Management
Payment Overview
Total Revenue
₹95,000.00
Completed Payments
2
Pending Payments
0
Pending Amount
₹0.00
Recent Payments
Filter by Status
All
Filter by Payment Mode
All
Sort by
Newest First
Payment #3010
Tenant: Amit Shah (9988776655)
Property: 78 Park Lane
Date: 2024-08-05 | Mode: UPI
Amount:
₹35,000.00
Status:
Completed
Refund

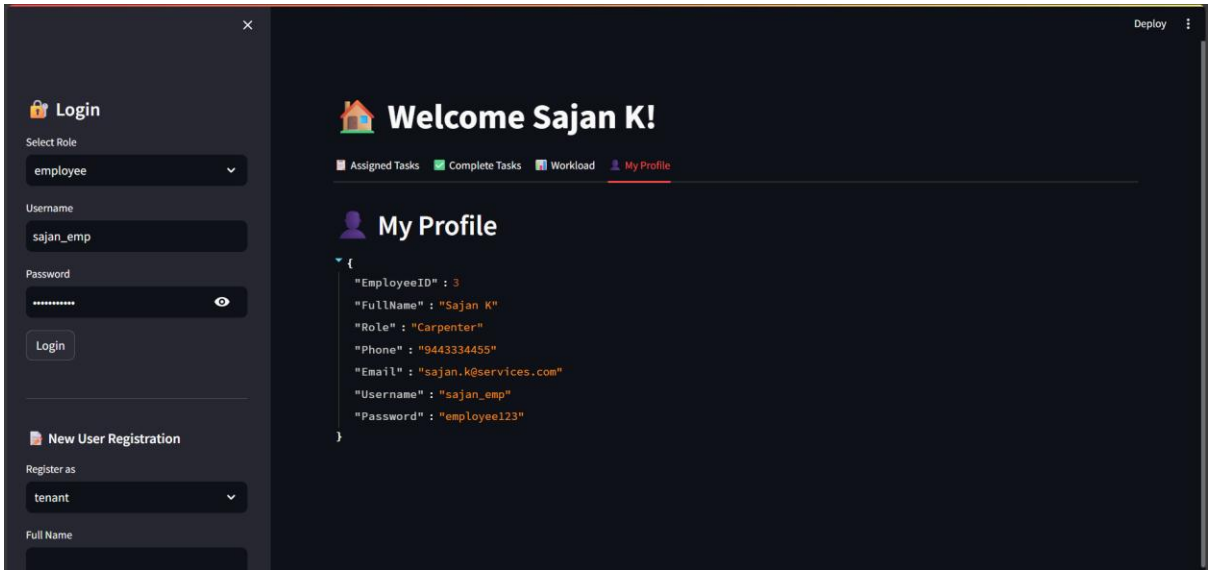

Login
Select Role
owner
Username
ravi_owner
Password
Login
Welcome Ravi Kumar!
New User Registration
Register as
Deploy
Welcome Ravi Kumar!
My Properties
Analytics
Maintenance
Payments
My Profile
My Profile
{
"OwnerID" : 1
"FullName" : "Ravi Kumar"
"Phone" : "9876543210"
"Email" : "ravi.kumar@example.com"
"Address" : "12 MG Road, Bangalore"
"Username" : "ravi_owner"
"Password" : "owner123"
}

Employee Features:

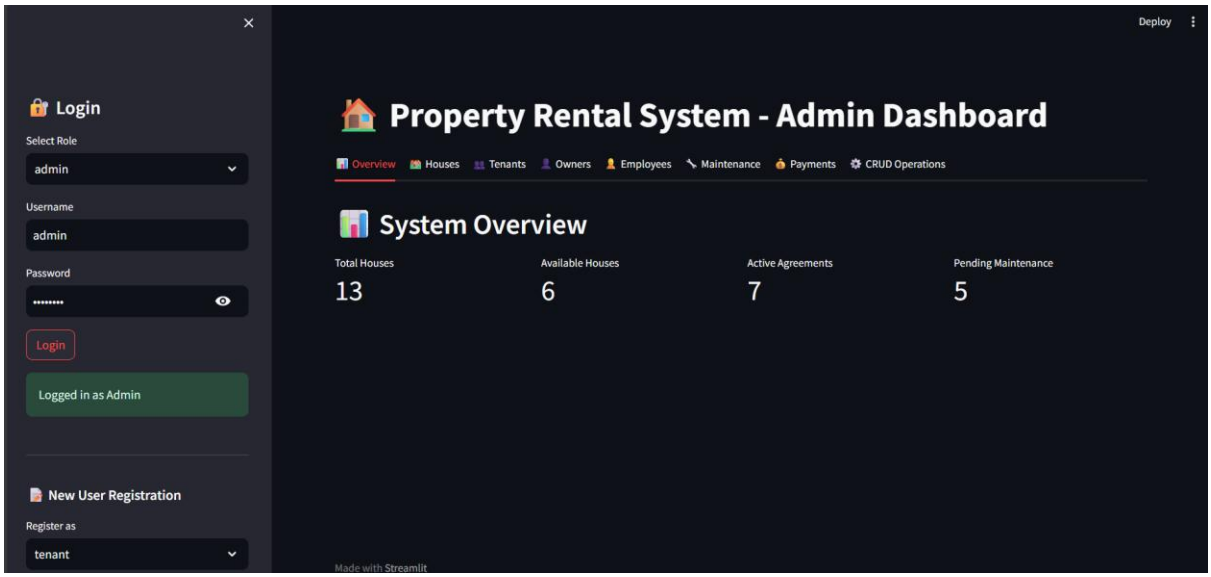


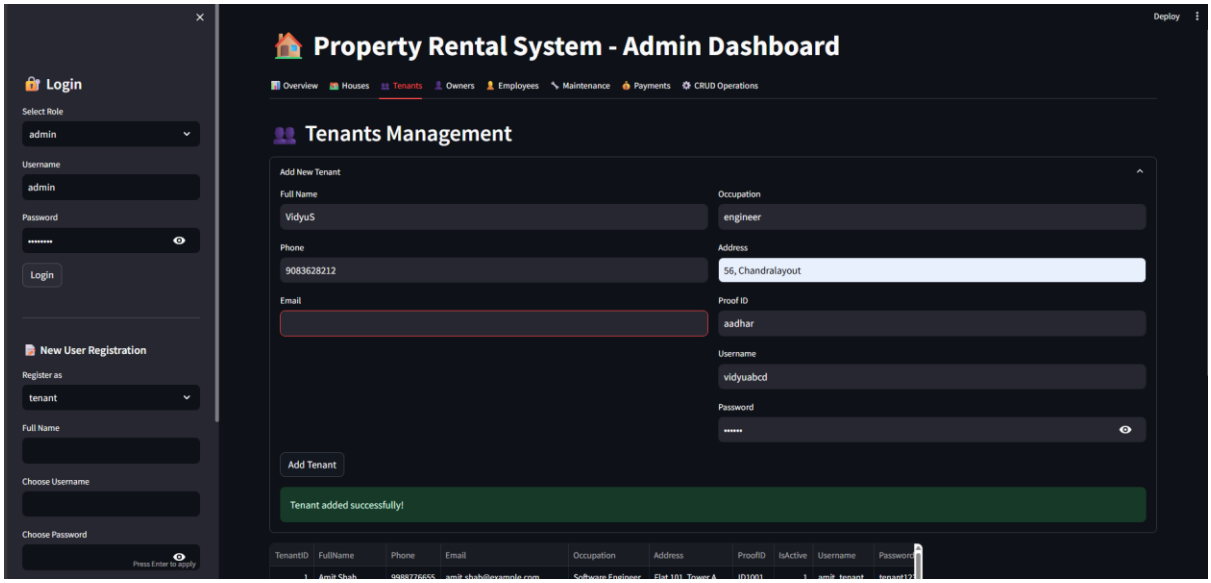




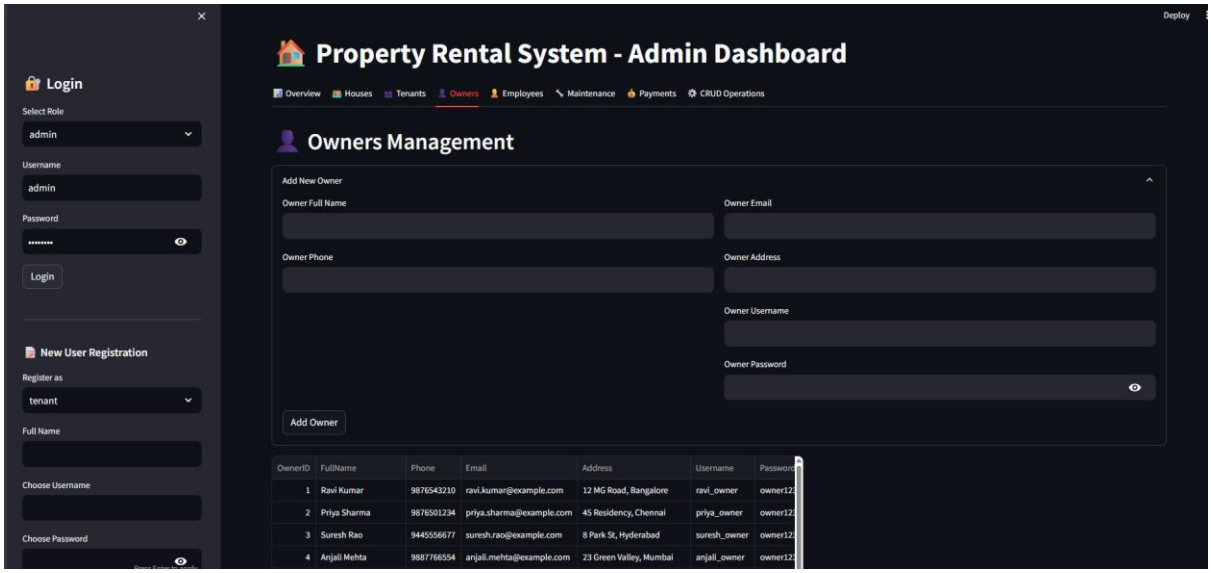


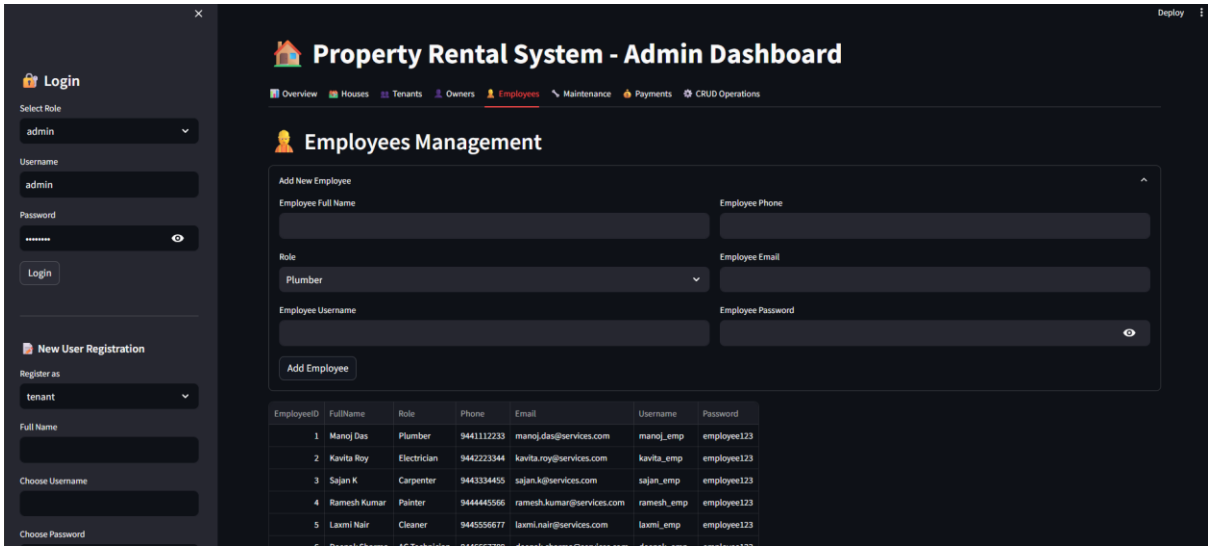
Admin features:

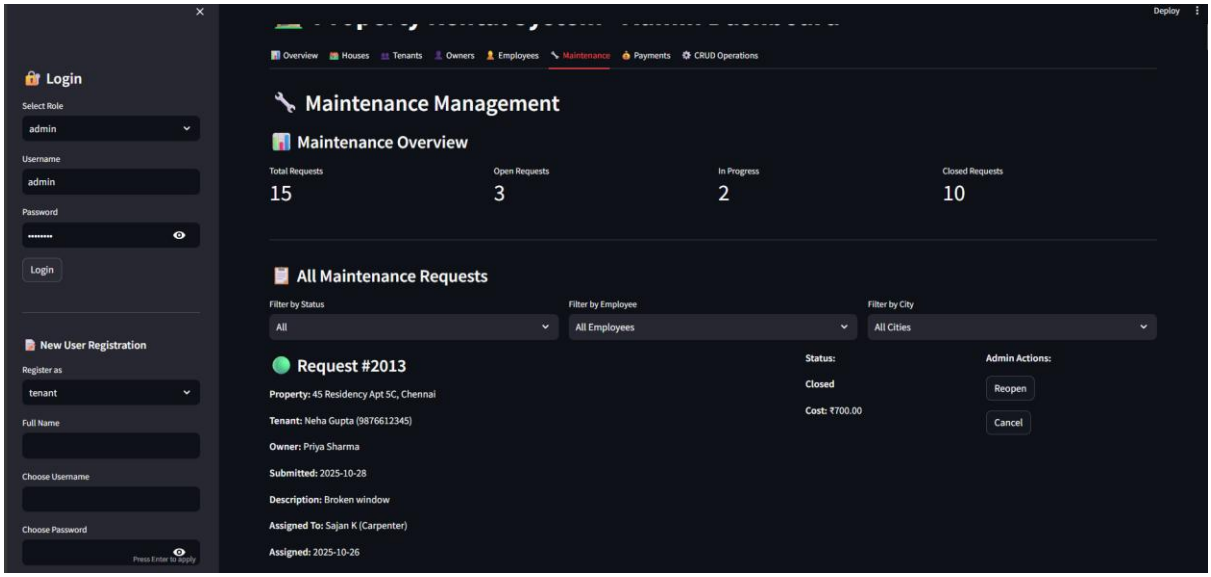


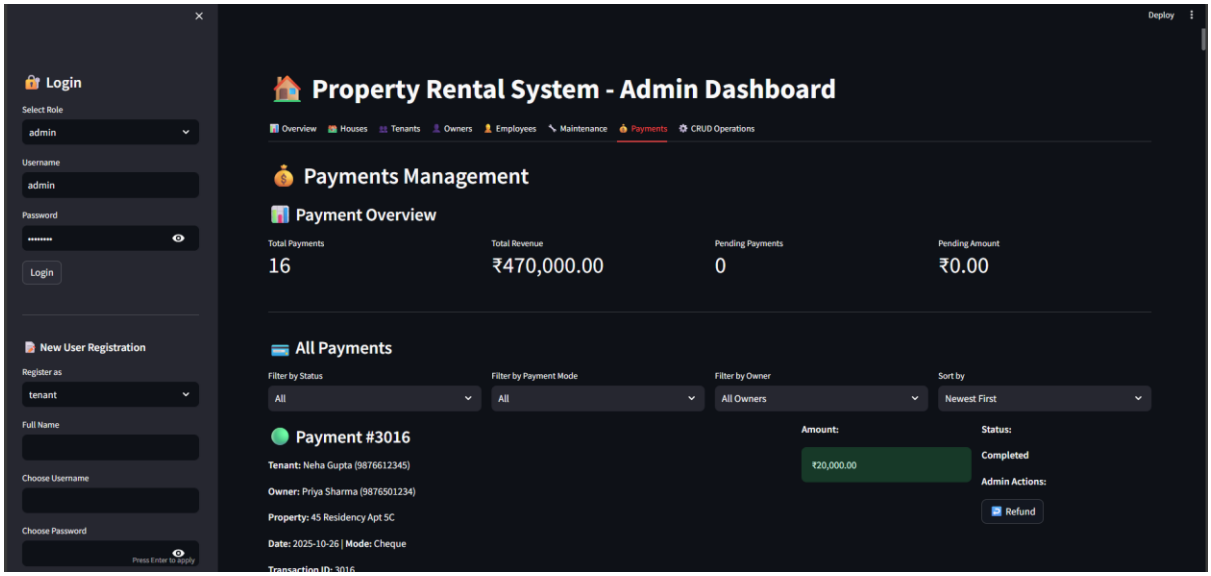


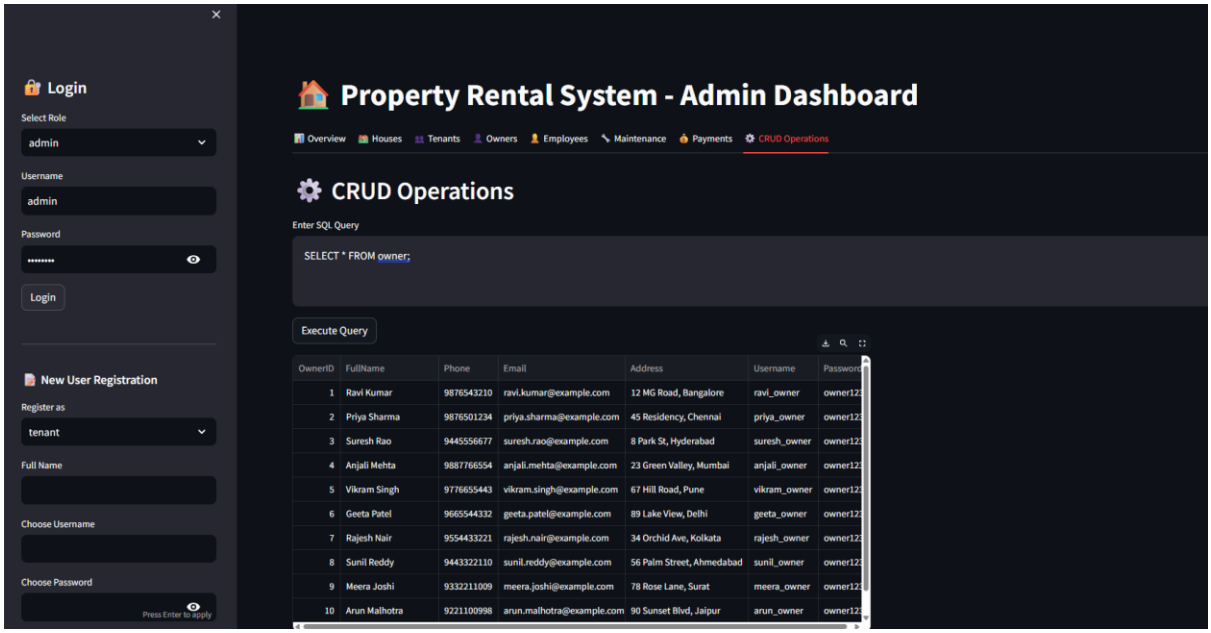
10	Pooja Mehta	9366778899	pooja.mehta@example.com	Architect	Flat 1010, Tower F	ID1010	1	pooja_tenant	tenant123
12	vidyuu	47564654	Vjhgg.com	Not specified	Not specified	Pending	1	vidyuabc	db2e7f1b
13	VidyuS	9083628212		engineer	56, Chandralayout	aadhar	1	vidyuabcd	bb7c5b29




Login
Select Role
admin
Username
admin
Password
Login
New User Registration
Register as
tenant
Full Name
Choose Username
Choose Password
Deploy
Property Rental System - Admin Dashboard
Overview
Houses
Tenants
Owners
Employees
Maintenance
Payments
CRUD Operations
Employees Management
Add New Employee
Employee Full Name
Employee Phone
Role
Plumber
Employee Email
Employee Username
Employee Password
Add Employee
EmployeeID
FullName
Role
Phone
Email
Username
Password
1 Manoj Das Plumber 9441112233 manoj.das@services.com manoj_emp employee123
2 Kavita Roy Electrician 9442223344 kavita.roy@services.com kavita_emp employee123
3 Sajan K Carpenter 9443334455 sajan.k@services.com sajan_emp employee123
4 Ramesh Kumar Painter 9444445566 ramesh.kumar@services.com ramesh_emp employee123
5 Laxmi Nair Cleaner 9445556677 laxmi.nair@services.com laxmi_emp employee123


Login
Select Role
admin
Username
admin
Password
Login
New User Registration
Register as
tenant
Full Name
Choose Username
Choose Password
Deploy
Overview
Houses
Tenants
Owners
Employees
Maintenance
Payments
CRUD Operations
Maintenance Management
Maintenance Overview
Total Requests
15
Open Requests
3
In Progress
2
Closed Requests
10
All Maintenance Requests
Filter by Status
All
Filter by Employee
All Employees
Filter by City
All Cities
Request #2013
Property: 45 Residency Apt SC, Chennai
Tenant: Neha Gupta (9876612345)
Owner: Priya Sharma
Submitted: 2025-10-28
Description: Broken window
Assigned To: Sajan K (Carpenter)
Assigned: 2025-10-26
Status:
Closed
Cost: ₹700.00
Admin Actions:
Reopen
Cancel


Login
Select Role
admin
Username
admin
Password
Login
New User Registration
Register as
tenant
Full Name
Choose Username
Choose Password
Deploy
Property Rental System - Admin Dashboard
Overview
Houses
Tenants
Owners
Employees
Maintenance
Payments
CRUD Operations
Payments Management
Payment Overview
Total Payments
16
Total Revenue
₹470,000.00
Pending Payments
0
Pending Amount
₹0.00
All Payments
Filter by Status
All
Filter by Payment Mode
All
Filter by Owner
All Owners
Sort by
Newest First
Payment #3016
Tenant: Neha Gupta (9876612345)
Owner: Priya Sharma (9876501234)
Property: 45 Residency Apt SC
Date: 2025-10-26 | Mode: Cheque
Transaction ID: 3016
Amount:
₹20,000.00
Status:
Completed
Admin Actions:
Refund



Triggers, Procedures/Functions, Nested query, Join, Aggregate queries

Triggers

```
-- Trigger 1: Auto-update House status when RentalAgreement becomes Active
DELIMITER //
CREATE TRIGGER UpdateHouseStatusOnAgreementActivation
AFTER UPDATE ON RentalAgreement
FOR EACH ROW
BEGIN
    IF NEW.AgreementStatus = 'Active' AND OLD.AgreementStatus != 'Active' THEN
        UPDATE House
        SET Status = 'Rented'
        WHERE HouseID = NEW.HouseID;
    END IF;

    IF NEW.AgreementStatus = 'Terminated' AND OLD.AgreementStatus != 'Terminated' THEN
        UPDATE House
        SET Status = 'Available'
        WHERE HouseID = NEW.HouseID;
    END IF;
END //
DELIMITER ;
```

```sql
-- Trigger 2: Auto-calculate Security Deposit when inserting RentalAgreement
DELIMITER //
CREATE TRIGGER CalculateSecurityDepositOnInsert
BEFORE INSERT ON RentalAgreement
FOR EACH ROW
BEGIN
    IF NEW.SecurityDeposit IS NULL THEN
        SET NEW.SecurityDeposit = NEW.MonthlyRent * 2;
    END IF;
END //
DELIMITER ;


-- Trigger 3: Log maintenance cost changes
CREATE TABLE MaintenanceCostAudit (
    AuditID INT AUTO_INCREMENT PRIMARY KEY,
    RequestID INT,
    OldCost DECIMAL(10,2),
    NewCost DECIMAL(10,2),
    ChangedBy VARCHAR(100),
    ChangeDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

DELIMITER //
CREATE TRIGGER AuditMaintenanceCostChanges
AFTER UPDATE ON MaintenanceRequest
FOR EACH ROW
BEGIN
    IF OLD.Cost != NEW.Cost THEN
        INSERT INTO MaintenanceCostAudit (RequestID, OldCost, NewCost, ChangedBy)
        VALUES (NEW.RequestID, OLD.Cost, NEW.Cost, USER());
    END IF;
END //
DELIMITER ;
```

```sql
-- Trigger 4: Prevent duplicate active agreements for same house
DELIMITER //
CREATE TRIGGER PreventDuplicateActiveAgreements
BEFORE INSERT ON RentalAgreement
FOR EACH ROW
BEGIN
    DECLARE active_count INT;

    SELECT COUNT(*) INTO active_count
    FROM RentalAgreement
    WHERE HouseID = NEW.HouseID
    AND AgreementStatus = 'Active'
    AND CURDATE() BETWEEN StartDate AND EndDate;

    IF active_count > 0 AND NEW.AgreementStatus = 'Active' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot create active agreement for a house that already has an active agreement';
    END IF;
END //
DELIMITER ;
```

Procedures:

```sql
-- Procedure 1: Create new rental agreement with validation
DELIMITER //
CREATE PROCEDURE CreateRentalAgreement(
    IN p_HouseID INT,
    IN p_TenantID INT,
    IN p_StartDate DATE,
    IN p_EndDate DATE,
    IN p_MonthlyRent DECIMAL(10,2),
    IN p_SecurityDeposit DECIMAL(10,2)
)
BEGIN
    DECLARE house_status VARCHAR(20);
    DECLARE existing_agreement INT;

    -- Check if house exists and is available
    SELECT Status INTO house_status FROM House WHERE HouseID = p_HouseID;
    IF house_status != 'Available' THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'House is not available for rent';
    END IF;

    -- Check for overlapping agreements
    SELECT COUNT(*) INTO existing_agreement
    FROM RentalAgreement
    WHERE HouseID = p_HouseID
    AND AgreementStatus = 'Active'
    AND (p_StartDate BETWEEN StartDate AND EndDate OR p_EndDate BETWEEN StartDate AND EndDate);

    IF existing_agreement > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'House has overlapping rental agreement';
    END IF;

    -- Insert the agreement
    INSERT INTO RentalAgreement (HouseID, TenantID, StartDate, EndDate, MonthlyRent, SecurityDeposit, AgreementStatus)
    VALUES (p_HouseID, p_TenantID, p_StartDate, p_EndDate, p_MonthlyRent, p_SecurityDeposit, 'Pending');

    SELECT LAST_INSERT_ID() AS NewAgreementID;
END //
```

```sql
-- Procedure 2: Process monthly payment
DELIMITER //
CREATE PROCEDURE ProcessMonthlyPayment(
    IN p_TenantID INT,
    IN p_OwnerID INT,
    IN p_Amount DECIMAL(10,2),
    IN p_PaymentMode ENUM('Cash','Card','BankTransfer','UPI','Cheque','Other')
)
BEGIN
    DECLARE active_agreement INT;

    -- Check if tenant has active agreement
    SELECT AgreementID INTO active_agreement
    FROM RentalAgreement
    WHERE TenantID = p_TenantID
    AND AgreementStatus = 'Active'
    AND CURDATE() BETWEEN StartDate AND EndDate;

    IF active_agreement IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Tenant does not have an active rental agreement';
    END IF;

    -- Insert payment record
    INSERT INTO Payment (TenantID, OwnerID, PaymentDate, Amount, Mode, Status)
    VALUES (p_TenantID, p_OwnerID, CURDATE(), p_Amount, p_PaymentMode, 'Completed');

    SELECT LAST_INSERT_ID() AS PaymentID;
END //
DELIMITER ;
```

Functions:

```sql
-- Function 1: Calculate total revenue for an owner
DELIMITER //
CREATE FUNCTION CalculateOwnerRevenue(owner_id INT)
RETURNS DECIMAL(10,2)
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE total_revenue DECIMAL(10,2) DEFAULT 0;

    SELECT COALESCE(SUM(Amount), 0) INTO total_revenue
    FROM Payment
    WHERE OwnerID = owner_id
    AND Status = 'Completed';

    RETURN total_revenue;
END //
DELIMITER ;
```

```sql
-- Function 2: Calculate average maintenance cost for a house
DELIMITER //
CREATE FUNCTION GetAverageMaintenanceCost(house_id INT)
RETURNS DECIMAL(10,2)
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE avg_cost DECIMAL(10,2);

    SELECT COALESCE(AVG(Cost), 0) INTO avg_cost
    FROM MaintenanceRequest
    WHERE HouseID = house_id
    AND Cost IS NOT NULL;

    RETURN avg_cost;
END //
DELIMITER ;
```

```sql
-- Function 3: Check if house is available for rent
DELIMITER //
CREATE FUNCTION IsHouseAvailable(house_id INT)
RETURNS BOOLEAN
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE house_status VARCHAR(20);

    SELECT Status INTO house_status
    FROM House
    WHERE HouseID = house_id;

    RETURN (house_status = 'Available');
END //
DELIMITER ;
```

```sql
-- Function 4: Get tenant's current active agreement
DELIMITER //
CREATE FUNCTION GetTenantActiveAgreement(tenant_id INT)
RETURNS INT
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE active_agreement_id INT;

    SELECT AgreementID INTO active_agreement_id
    FROM RentalAgreement
    WHERE TenantID = tenant_id
    AND AgreementStatus = 'Active'
    AND CURDATE() BETWEEN StartDate AND EndDate
    LIMIT 1;

    RETURN active_agreement_id;
END //
DELIMITER ;
```

```sql
-- Function 5: Calculate total maintenance cost for a house
DELIMITER //
CREATE FUNCTION GetTotalMaintenanceCost(house_id INT)
RETURNS DECIMAL(10,2)
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE total_cost DECIMAL(10,2);

    SELECT COALESCE(SUM(Cost), 0) INTO total_cost
    FROM MaintenanceRequest
    WHERE HouseID = house_id
    AND Cost IS NOT NULL;

    RETURN total_cost;
END //
DELIMITER ;


-- Function 6: Get number of active agreements for an owner
DELIMITER //
CREATE FUNCTION CountOwnerActiveAgreements(owner_id INT)
RETURNS INT
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE active_count INT;

    SELECT COUNT(*) INTO active_count
    FROM RentalAgreement ra
    JOIN House h ON ra.HouseID = h.HouseID
    WHERE h.OwnerID = owner_id
    AND ra.AgreementStatus = 'Active'
    AND CURDATE() BETWEEN ra.StartDate AND ra.EndDate;

    RETURN active_count;
END //
DELIMITER ;
```

Nested queries:

```sql
-- Nested Query 1: Find tenants who have paid more than average rent
SELECT t.TenantID, t.FullName, p.Amount
FROM Tenant t
JOIN Payment p ON t.TenantID = p.TenantID
WHERE p.Amount > (
    SELECT AVG(Amount)
    FROM Payment
    WHERE Status = 'Completed'
)
AND p.Status = 'Completed';

-- Nested Query 2: Find houses that have never had maintenance requests
SELECT h.HouseID, h.Address, h.City
FROM House h
WHERE h.HouseID NOT IN (
    SELECT DISTINCT HouseID
    FROM MaintenanceRequest
);
```

Joins:

```sql
-- 8) JOIN queries demonstrating relationships
-- ===========================
-- Agreements + Tenant + House
SELECT ra.AgreementID, ra.StartDate, ra.EndDate, ra.MonthlyRent,
       t.TenantID, t.FullName AS TenantName,
       h.HouseID, h.Address AS HouseAddress, h.City
FROM `RentalAgreement` ra
JOIN `Tenant` t ON ra.TenantID = t.TenantID
JOIN `House` h ON ra.HouseID = h.HouseID;

-- Which payments did OwnerID = 1 receive?
SELECT p.PaymentID, p.PaymentDate, p.Amount, t.FullName AS Payer
FROM `Payment` p
JOIN `Tenant` t ON p.TenantID = t.TenantID
WHERE p.OwnerID = 1;

-- Maintenance requests with assigned employee(s)
SELECT mr.RequestID, mr.Description, mr.Status, mr.Cost,
       a.AssignmentID, e.EmployeeID, e.FullName AS EmployeeName, e.Role,
       a.AssignedDate, a.CompletionDate
FROM `MaintenanceRequest` mr
LEFT JOIN `Assignment` a ON mr.RequestID = a.RequestID
LEFT JOIN `Employee` e ON a.EmployeeID = e.EmployeeID;
```

Aggregate queries:

```sql
-- Aggregate Query 1: Monthly revenue trend
SELECT
    YEAR(PaymentDate) AS Year,
    MONTH(PaymentDate) AS Month,
    COUNT(*) AS TotalPayments,
    SUM(Amount) AS TotalRevenue,
    AVG(Amount) AS AveragePayment,
    MIN(Amount) AS MinPayment,
    MAX(Amount) AS MaxPayment
FROM Payment
WHERE Status = 'Completed'
GROUP BY YEAR(PaymentDate), MONTH(PaymentDate)
ORDER BY Year DESC, Month DESC;
```

```sql
-- Aggregate Query 2: Maintenance statistics by employee role
SELECT
    e.Role,
    COUNT(DISTINCT a.EmployeeID) AS TotalEmployees,
    COUNT(a.AssignmentID) AS TotalAssignments,
    AVG(DATEDIFF(a.CompletionDate, a.AssignedDate)) AS AvgCompletionDays,
    SUM(mr.Cost) AS TotalMaintenanceCost,
    AVG(mr.Cost) AS AvgMaintenanceCost
FROM Employee e
LEFT JOIN Assignment a ON e.EmployeeID = a.EmployeeID
LEFT JOIN MaintenanceRequest mr ON a.RequestID = mr.RequestID
WHERE a.CompletionDate IS NOT NULL
GROUP BY e.Role
ORDER BY TotalAssignments DESC;
```

INVOKING TRIGGERS, PROCEDURES AND FUNCTIONS

```sql
026
027     -- Test Trigger 1: Update agreement status to activate house status change
028 •   SELECT * FROM House WHERE HouseID = 101;
029 •   UPDATE RentalAgreement SET AgreementStatus = 'Active' WHERE AgreementID = 1002;
030 •   SELECT * FROM House WHERE HouseID = 101;
031
032     -- Test Trigger 2: Insert rental agreement without security deposit
033 •   INSERT INTO RentalAgreement (HouseID, TenantID, StartDate, EndDate, MonthlyRent, AgreementStatus)
034     VALUES (501, 11, '2025-03-01', '2026-02-28', 28000.00, 'Pending');
035 •   SELECT * FROM RentalAgreement WHERE AgreementID = LAST_INSERT_ID();
036
037     -- Test Procedure 1: Create new rental agreement
038 •   CALL CreateRentalAgreement(701, 11, '2025-03-01', '2026-02-28', 15000.00, NULL);
039
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

AgreementID	HouseID	TenantID	StartDate	EndDate	MonthlyRent	SecurityDeposit	AgreementStatus
1011	501	11	2025-03-01	2026-02-28	28000.00	NULL	Pending
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
1036
1037    -- Test Procedure 1: Create new rental agreement
1038    CALL CreateRentalAgreement(701, 11, '2025-03-01', '2026-02-28', 15000.00, NU
1039
1040    -- Test Procedure 2: Process monthly payment
1041    CALL ProcessMonthlyPayment(1, 1, 35000.00, 'UPI');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

NewAgreementID
1012

```
569    -- Test Function 1: Calculate owner revenue
570    SELECT OwnerID, FullName, CalculateOwnerRevenue(OwnerID) AS TotalRevenue
571    FROM Owner;
572
573    -- Test Function 2: Average maintenance cost per house
574    SELECT HouseID, Address, GetAverageMaintenanceCost(HouseID) AS AvgMaintenanceCost
575    FROM House;
576
577    -- Test Function 3: Check house availability
578    SELECT HouseID, Address, Status, IsHouseAvailable(HouseID) AS IsAvailable
579    FROM House;
580
581    -- Test Function 4: Get tenant's active agreement
582    SELECT TenantID, FullName, GetTenantActiveAgreement(TenantID) AS ActiveAgreementID
583    FROM Tenant;
584
585    -- Test Function 5: Total maintenance cost per house
586    SELECT HouseID, Address, GetTotalMaintenanceCost(HouseID) AS TotalMaintenanceCost
587    FROM House;
588
589    -- Test Function 6: Count active agreements per owner
590    SELECT OwnerID, FullName, CountOwnerActiveAgreements(OwnerID) AS ActiveAgreements
591    FROM Owner;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

OwnerID	FullName	TotalRevenue
1	Ravi Kumar	95000.00
2	Priya Sharma	58000.00
3	Suresh Rao	80000.00
4	Anjali Mehta	90000.00
5	Vikram Singh	0.00

Result 44 | Result 45 | Result 46 | Result 47 | Result 48 | Result 49

GITHUB REPO LINK

https://github.com/prerana2005/Property-Rental-Management-System