Real Estate Management System

A PROJECT REPORT

Submitted by

Sarthak Sharma [Reg No:RA2211003010744]

Under the Guidance of

Dr. S Ramesh

Assistant Professor, Department of Computing Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

RA2211003010744, Certified to be the bonafide work done by **Sarthak Sharma** of II year/IV sem B.Tech Degree Course in the Project Course – **21CSC205P Database Management Systems** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**, Kattankulathur for the academic year 2023-2024.

Date:2nd May,2024

Faculty in Charge
Dr. S Ramesh
Assistant Professor
Department of computing technologies
SRMSIT -KTR

HEAD OF THE DEPARTMENT
Dr. M Pushpalatha
Professor & Head
Department of computing technologies
SRMIST - KTR

ABSTRACT

The Real Estate Management System is a robust solution designed to streamline the complexities of real estate operations through a user-friendly interface. Leveraging Python's Tkinter library for the frontend and MySQL as the backend database, the system offers a seamless experience for adding, organizing, and retrieving critical real estate data. Upon initiation, the system establishes a secure connection to the MySQL database, ensuring data integrity and reliability. It dynamically creates essential tables like properties, owners, buyers, agents, and transactions, adapting to evolving business needs. Through an intuitive GUI, users can effortlessly input property details, including address, price, bedrooms, and bathrooms, while also managing owner, buyer, and agent information with ease. Each input is meticulously validated to prevent errors and maintain database consistency. By providing comprehensive tools for managing property listings, tracking ownership information, facilitating transactions, and maintaining detailed records, the system empowers real estate professionals to optimize their operations, enhance efficiency, and deliver superior service to clients.

PROBLEM STATEMENT

The Real Estate Management System (REMS) seeks to revolutionize the way real estate assets are handled and transactions are conducted by offering an all-encompassing platform tailored to the needs of property professionals. With its suite of features, REMS addresses the complexities of property management, ensuring seamless operations and optimal decision-making.

At the core of REMS is its capability to efficiently handle property listings. By providing a user-friendly interface, property owners can effortlessly add, update, and view property details, including crucial information such as address, pricing, and property specifications like the number of bedrooms and bathrooms. This centralized repository of property listings serves as a valuable resource for agents and buyers alike, facilitating informed decision-making and expediting the property search process.

Moreover, REMS places a strong emphasis on ownership tracking, enabling users to maintain a comprehensive record of property ownership details. Through robust ownership management functionalities, users can accurately monitor ownership changes, track property history, and ensure compliance with legal and regulatory requirements. This feature enhances transparency and accountability in property transactions, fostering trust and confidence among stakeholders.

Transaction management is another key aspect of REMS, empowering users to efficiently oversee and execute property transactions from initiation to completion. By facilitating seamless communication between buyers, agents, and property owners, REMS streamlines the entire transaction process, reducing administrative overhead and minimizing the risk of errors or delays. Additionally, REMS provides real-time transaction status updates, allowing stakeholders to stay informed and engaged throughout the transaction lifecycle.

Central to REMS is its comprehensive agent representation module, which serves as a pivotal resource for real estate agents to manage their client portfolios, track leads, and coordinate property viewings and negotiations. Through advanced agent representation functionalities, REMS empowers agents to deliver personalized services, nurture client relationships, and drive business growth.

Overall, REMS embodies a holistic approach to real estate management, leveraging cutting-edge technology and innovative features to optimize efficiency, enhance transparency, and elevate the real estate experience for all stakeholders involved. Whether it's property listing, ownership tracking, transaction management, or agent representation, REMS stands as a beacon of innovation in the ever-evolving real estate landscape, poised to redefine the standards of excellence in property management.

TABLE OF CONTENTS

1. Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the project:

Understanding and identifying entities and relationships is a crucial step in designing a database using an Entity-Relationship (ER) model for the Real Estate Management System project. Let's break down the entities and relationships that are likely to exist in this system:

Entities:

1. Property: Represents real estate properties with attributes like ID, address, price, number of bedrooms, and number of bathrooms.
2. Owner: Represents the owners of properties with attributes like ID, name, and contact number.
3. Buyer: Represents the potential buyers of properties with attributes like ID, name, and contact number.
4. Agent: Represents the real estate agents involved in transactions with attributes like ID, name, and contact number.
5. Transaction: Represents the transactions that occur between properties, buyers, and agents with attributes like ID, property ID, buyer ID, agent ID, and transaction date.

Relationships:

1. Ownership: Links properties to their respective owners. (One-to-Many - One owner can own multiple properties, but each property is owned by one owner).
2. Representation: Associates properties with agents who represent them. (Many-to-Many – An agent can represent multiple properties, and a property can be represented by multiple agents).
3. Purchase: Connects properties with buyers and agents involved in transactions. (Many-to-Many - A property can be purchased by multiple buyers, and a buyer can purchase multiple properties. Similarly, an agent can facilitate multiple transactions, and a transaction involves one buyer and one property).

Now, using this understanding, we can construct a database schema based on the ER model:

1. Tables:
   - Property (id, address, price, bedrooms, bathrooms, owner_id)
   - Owner (id, name, contact_number)
   - Buyer (id, name, contact_number)
   - Agent (id, name, contact_number)
   - Transaction (id, property_id, buyer_id, agent_id, transaction_date)

2. Relationships:
   - Property(owner_id) -> Owner(id)
   - Property(agent_id) <- Agent(id)
   - Property(id) <- Transaction(property_id)
   - Buyer(id) <- Transaction(buyer_id)
   - Agent(id) <- Transaction(agent_id)

2. Design of Relational Schemas, Creation of Database Tables for project.

- **Design of Relational Schemas:**

In designing the relational schema for the Real Estate Management System, careful consideration is given to the entities involved and their relationships. The Property entity serves as a core component, representing real estate listings. Each property is uniquely identified by an ID an possesses attributes such as address, price, bedrooms, and bathrooms. To establish ownership associations, a foreign key references the Owner entity, which stores details about property owners, including their name and contact information. Additionally, the Buyer and Agent entities play pivotal roles in real estate transactions. Buyers express interest in properties and are identified by their name and contact information, while agents facilitate transactions and are similarly characterized by their name and contact details.

The Transaction entity serves to capture the dynamic interactions between properties, buyers, and agents, recording essential information like the property ID, buyer ID, agent ID, and the date of the transaction.

- **Creation of Database Tables:**

The translation of the relational schema into concrete database tables is achieved through SQL commands. Each entity corresponds to a distinct table, with columns representing their attributes. The Property table is constructed with columns for ID, address, price, bedrooms, bathrooms, and a foreign key linking it to the Owner table. Owners are represented in a separate table, with columns for ID, name, and contact number. Similarly, the Buyer and Agent tables are created to store information about buyers and agents, respectively. The Transaction table encapsulates the complex relationships between entities, featuring columns for ID, property ID, buyer ID, agent ID, and transaction date. To ensure data integrity and consistency, foreign key constraints are established to enforce referential integrity, guaranteeing that relationships between entities remain accurate and reliable. This systematic approach to database table creation forms a robust foundation for the Real Estate Management System, facilitating seamless data storage, retrieval, and manipulation to support efficient real estate management operations.

3. Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors.

Here are some complex queries based on the concepts of constraints, sets, joins, views, Triggers and cursors:

I. Constraints:
Query to retrieve properties with prices higher than a certain threshold:

```
SELECT * FROM Property WHERE price > 1000000;
```

II. Sets:
Query to find properties that are either owned by a specific owner or have more than 3 bedrooms:

```
SELECT * FROM Property WHERE owner_id = 1 OR bedrooms > 3;
```

III. Joins:

Query to fetch details of properties along with their owners' names:

```
SELECT p.*, o.name AS owner_name
FROM Property p
INNER JOIN Owner o ON p.owner_id = o.id;
```

IV. Views:

Create a view to show the total number of transactions each agent has handled:

```
CREATE VIEW AgentTransactionCount AS
SELECT agent_id, COUNT(*) AS transaction_count
FROM Transaction
GROUP BY agent_id;
```

V. Triggers:

Trigger to update the price of a property whenever a new transaction occurs:

```
CREATE TRIGGER UpdatePropertyPrice AFTER INSERT ON Transaction
FOR EACH ROW
BEGIN
    UPDATE Property
    SET price = price * 1.1  -- Increase price by 10%
    WHERE id = NEW.property_id;
END;
```

VI. Cursors:

Cursor to fetch details of all transactions and calculate the total amount spent by each buyer:

```
CREATE PROCEDURE CalculateTotalSpent()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE buyer_id INT;
    DECLARE total_spent DECIMAL(10, 2);

    DECLARE cur CURSOR FOR
        SELECT DISTINCT buyer_id FROM Transaction;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;

    read_loop: LOOP
        FETCH cur INTO buyer_id;
        IF done THEN
            LEAVE read_loop;
        END IF;

        SET total_spent = 0;
        SELECT SUM(price) INTO total_spent
        FROM Transaction t
        INNER JOIN Property p ON t.property_id = p.id
        WHERE t.buyer_id = buyer_id;

        INSERT INTO BuyerTotalSpent (buyer_id, total_spent) VALUES
    END LOOP;

    CLOSE cur;
END;
```

4. Analyzing the pitfalls, identifying the dependencies, and applying normalizations:

Analyzing the pitfalls, identifying dependencies, and applying normalizations are crucial steps in database design to ensure data integrity, minimize redundancy, and optimize performance. Let's go through these steps for the Real Estate Management System:

1) **Analyzing the Pitfalls**:
   - Redundancy: Storing the same information multiple times can lead to inconsistencies and wasted storage space.
   - Update Anomalies: Modifying data in one place but not in others can result in inconsistencies.
   - Insertion Anomalies: Inability to add certain information without having related data already present.
   - Deletion Anomalies: Accidental loss of valuable information due to the deletion of unrelated data.

2) **Identifying Dependencies:**
   - Functional Dependencies: Identify functional dependencies between attributes within tables.
   - Transitive Dependencies: Identify transitive dependencies where one attribute determines another indirectly through a third attribute.

3) **Applying Normalizations:**

   - First Normal Form (1NF): Ensure atomicity by eliminating repeating groups and multivalued attributes.
   - Second Normal Form (2NF): Eliminate partial dependencies by making sure that non-key attributes depend on the entire primary key.

- Third Normal Form (3NF): Eliminate transitive dependencies by ensuring that non-key attributes depend only on the primary key, not on other non-key attributes.

4) Normalization Steps for Real Estate Management System:

- Step 1: First Normal Form (1NF):

    a) Ensure that each table has a primary key.
    b) Split multi-valued attributes into separate columns.
    c) Ensure each column contains atomic values.
    d) Example: Address may need to be split into street, city, state, and zip code.

- Step 2: Second Normal Form (2NF):

    a) Eliminate partial dependencies by ensuring that non-key attributes depend on the entire primary key.
    b) Split tables to ensure that each table represents a single theme.
    c) Example: If Property has attributes like price and owner_name, move owner_ name to the Owner table.

- Step 3: Third Normal Form (3NF):

    a) Eliminate transitive dependencies by ensuring that non-key attributes depend only on the primary key.
    b) Break down tables further if necessary to achieve this.
    c) Example: If there's a transitive dependency between Property and Owner through the Agent, create separate tables for Property_Owner and Agent.

5) Dependencies Identified in Real Estate Management System:

- Property depends on Owner (owner_id).

- Transaction depends on Property (property_id), Buyer (buyer_id), and Agent (agent_id).
- Agent might depend on Property (property_id) if agents are assigned to specific properties.

6) Normalization Considerations:

- Ensure each table represents a single entity or relationship.
- Eliminate redundant data by splitting tables appropriately.
- Use foreign keys to establish relationships between tables.
- Normalize until the database meets at least the Third Normal Form (3NF).

By carefully analyzing the pitfalls, identifying dependencies, and applying normalization techniques, you can design a robust and efficient database schema for the Real Estate Management System that ensures data integrity and minimizes redundancy.

5. Implementation of concurrency control and recovery mechanisms

Implementing concurrency control and recovery mechanisms is essential for ensuring the reliability, consistency, and durability of the database in the Real Estate Management System.

1) Concurrency Control:

- Locking Mechanisms: Use locking mechanisms to prevent multiple transactions from accessing or modifying the same data concurrently. Implement various types of locks such as shared locks and exclusive locks to manage concurrency effectively.

- Transaction Isolation Levels: Define appropriate transaction isolation levels to control

the visibility of data changes made by concurrent transactions. Common isolation include Read Uncommitted, Read Committed, Repeatable Read, and Serializable.

- Timestamp-Based Concurrency Control: Implement timestamp-based concurrency control mechanisms to manage concurrency by assigning timestamps to transaction and ensuring serializability of transactions based on their timestamps.

- Deadlock Detection and Resolution: Implement deadlock detection algorithms to id and resolve deadlocks that may occur due to conflicting lock acquisitions by multiple transactions. Use techniques such as deadlock detection graphs and timeouts to hand deadlocks efficiently.

2) Recovery Mechanisms:

- Transaction Logging: Maintain transaction logs to record all changes made by transact-tions to the database. Log entries should include information about the transaction ID, operation type (insert, update, delete), old and new values, and timestamps.

- Write-Ahead Logging (WAL): Implement write-ahead logging to ensure that records are written to stable storage before corresponding data modifications are applied to database. This ensures durability and recoverability of transactions in case of system failures.

- Checkpointing: Periodically perform checkpointing to write dirty data pages from Memory to disk and update the checkpoint record in the transaction log. This helps reducing the time required for database recovery by minimizing the number records that need to be replayed during recovery.

- Transaction Rollback and Redo: During database recovery, use transaction rollback redo mechanisms to undo incomplete transactions and reapply committed transactions from the transaction log to restore the database to a consistent state.

- Point-in-Time Recovery: Implement point-in-time recovery mechanisms to restore the database to a specific consistent state based on a specified timestamp or log sequence number (LSN). This allows recovering the database to a known good state before a system failure occurred.

6. Code for the project (In Python):

```
import mysql.connector
import tkinter as tk
from tkinter import messagebox

# Function to connect to the MySQL database
def connect_to_database():
    try:
        connection = mysql.connector.connect(
            user='ss6801',
            password='**********',
            host='localhost',
            database="realestate"
        )
        return connection
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to connect to database: {err}")
        return None

# Function to create tables if they do not exist
def create_tables(connection):
    try:
        cursor = connection.cursor()
```

```
# Table: properties
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS properties (
                id INT AUTO_INCREMENT PRIMARY KEY,
                address VARCHAR(255) NOT NULL,
                price DECIMAL(10, 2) NOT NULL,
                bedrooms INT,
                bathrooms INT
            )
        """)

        # Table: owners
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS owners (
                id INT AUTO_INCREMENT PRIMARY KEY,
                name VARCHAR(255) NOT NULL,
                contact_number VARCHAR(15)
            )
        """)

        # Table: buyers
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS buyers (
                id INT AUTO_INCREMENT PRIMARY KEY,
                name VARCHAR(255) NOT NULL,
                contact_number VARCHAR(15)
            )
        """)
```

```
        # Table: agents
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS agents (
                id INT AUTO_INCREMENT PRIMARY KEY,
                name VARCHAR(255) NOT NULL,
                contact_number VARCHAR(15)
            )
        """)

        # Table: transactions
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS transactions (
                id INT AUTO_INCREMENT PRIMARY KEY,
                property_id INT,
                buyer_id INT,
                agent_id INT,
                transaction_date DATE,
                FOREIGN KEY (property_id) REFERENCES properties(id),
                FOREIGN KEY (buyer_id) REFERENCES buyers(id),
                FOREIGN KEY (agent_id) REFERENCES agents(id)
            )
        """)

        connection.commit()
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to create tables: {err}")

# Function to add a new property to the database
def add_property(connection, address, price, bedrooms, bathrooms):
    try:
```

```
     cursor = connection.cursor()
        cursor.execute("""
            INSERT INTO properties (address, price, bedrooms, bathrooms)
            VALUES (%s, %s, %s, %s)
        """, (address, price, bedrooms, bathrooms))
        connection.commit()
        messagebox.showinfo("Success", "Property added successfully!")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to add property: {err}")

# Function to add a new owner to the database
def add_owner(connection, name, contact_number):
    try:
        cursor = connection.cursor()
        cursor.execute("""
            INSERT INTO owners (name, contact_number)
            VALUES (%s, %s)
        """, (name, contact_number))
        connection.commit()
        messagebox.showinfo("Success", "Owner added successfully!")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to add owner: {err}")

# Function to add a new buyer to the database
def add_buyer(connection, name, contact_number):
    try:
        cursor = connection.cursor()
        cursor.execute("""
            INSERT INTO buyers (name, contact_number)
            VALUES (%s, %s)
```

```
    """, (name, contact_number))
    connection.commit()
        messagebox.showinfo("Success", "Buyer added successfully!")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to add buyer: {err}")

# Function to add a new agent to the database
def add_agent(connection, name, contact_number):
    try:
        cursor = connection.cursor()
        cursor.execute("""
            INSERT INTO agents (name, contact_number)
            VALUES (%s, %s)
        """, (name, contact_number))
        connection.commit()
        messagebox.showinfo("Success", "Agent added successfully!")
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to add agent: {err}")

# Function to add a new transaction to the database
def add_transaction(connection, property_id, buyer_id, agent_id, transaction_date):
    try:
        cursor = connection.cursor()
        cursor.execute("""
            INSERT INTO transactions (property_id, buyer_id, agent_id, transaction_date)
            VALUES (%s, %s, %s, %s)
        """, (property_id, buyer_id, agent_id, transaction_date))
        connection.commit()
        messagebox.showinfo("Success", "Transaction added successfully!")
```

```
 except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Failed to add transaction: {err}")

# Function to handle GUI interactions
def handle_choice(choice):
    if choice == 'Add Property':
        address = address_entry.get()
        price = float(price_entry.get())
        bedrooms = int(bedrooms_entry.get())
        bathrooms = int(bathrooms_entry.get())
        add_property(connection, address, price, bedrooms, bathrooms)
    elif choice == 'Add Owner':
        owner_name = owner_name_entry.get()
        owner_contact = owner_contact_entry.get()
        add_owner(connection, owner_name, owner_contact)
    elif choice == 'Add Buyer':
        buyer_name = buyer_name_entry.get()
        buyer_contact = buyer_contact_entry.get()
        add_buyer(connection, buyer_name, buyer_contact)
    elif choice == 'Add Agent':
        agent_name = agent_name_entry.get()
        agent_contact = agent_contact_entry.get()
        add_agent(connection, agent_name, agent_contact)
    elif choice == 'Add Transaction':
        prop_id = int(prop_id_entry.get())
        buyer_id = int(buyer_id_entry.get())
        agent_id = int(agent_id_entry.get())
        trans_date = transaction_date_entry.get()
        add_transaction(connection, prop_id, buyer_id, agent_id, trans_date)
```

```
# Main function
def main():
    global connection
    connection = connect_to_database()

    if connection:
        create_tables(connection)

        root = tk.Tk()
        root.title("Real Estate Management System")

        choices = ['Add Property', 'Add Owner', 'Add Buyer', 'Add Agent', 'Add Transaction']

        for i, choice in enumerate(choices):
            tk.Button(root, text=choice, command=lambda ch=choice:
handle_choice(ch)).grid(row=i, column=0, pady=5)

        # Entry fields for adding property
        global address_entry, price_entry, bedrooms_entry, bathrooms_entry
        address_label = tk.Label(root, text="Address:")
        address_label.grid(row=0, column=1)
        address_entry = tk.Entry(root)
        address_entry.grid(row=0, column=2)

        price_label = tk.Label(root, text="Price:")
        price_label.grid(row=1, column=1)
        price_entry = tk.Entry(root)
        price_entry.grid(row=1, column=2)

        bedrooms_label = tk.Label(root, text="Bedrooms:")
```

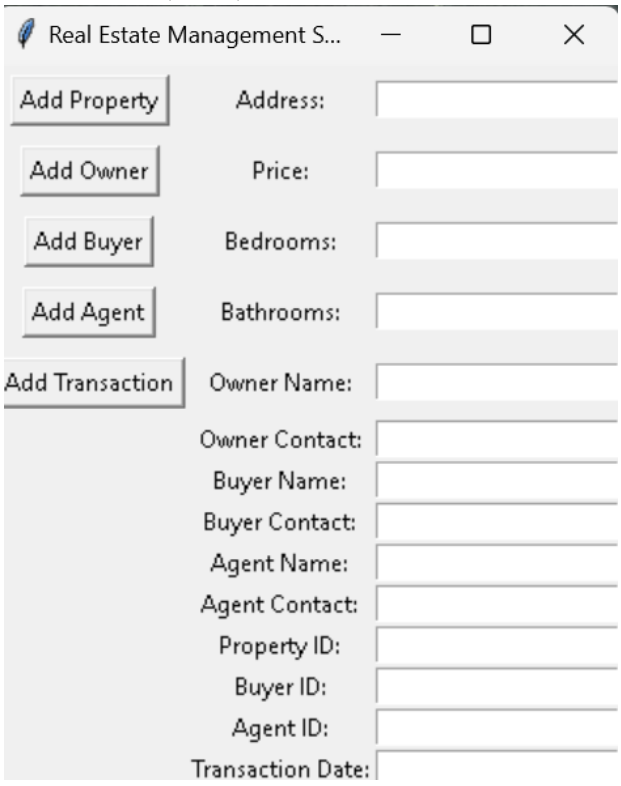
```
bedrooms_label.grid(row=2, column=1)
bedrooms_entry = tk.Entry(root)
bedrooms_entry.grid(row=2, column=2)

bathrooms_label = tk.Label(root, text="Bathrooms:")
bathrooms_label.grid(row=3, column=1)
bathrooms_entry = tk.Entry(root)
bathrooms_entry.grid(row=3, column=2)

# Entry fields for adding owner
global owner_name_entry, owner_contact_entry
owner_name_label = tk.Label(root, text="Owner Name:")
owner_name_label.grid(row=4, column=1)
owner_name_entry = tk.Entry(root)
owner_name_entry.grid(row=4, column=2)

owner_contact_label = tk.Label(root, text="Owner Contact:")
owner_contact_label.grid(row=5, column=1)
owner_contact_entry = tk.Entry(root)
owner_contact_entry.grid(row=5, column=2)

# Entry fields for adding buyer
global buyer_name_entry, buyer_contact_entry
buyer_name_label = tk.Label(root, text="Buyer Name:")
buyer_name_label.grid(row=6, column=1)
buyer_name_entry = tk.Entry(root)
buyer_name_entry.grid(row=6, column=2)

buyer_contact_label = tk.Label(root, text="Buyer Contact:")
buyer_contact_label.grid(row=7, column=1)
```

```
buyer_contact_entry = tk.Entry(root)
buyer_contact_entry.grid(row=7, column=2)

# Entry fields for adding agent
global agent_name_entry, agent_contact_entry
agent_name_label = tk.Label(root, text="Agent Name:")
agent_name_label.grid(row=8, column=1)
agent_name_entry = tk.Entry(root)
agent_name_entry.grid(row=8, column=2)

agent_contact_label = tk.Label(root, text="Agent Contact:")
agent_contact_label.grid(row=9, column=1)
agent_contact_entry = tk.Entry(root)
agent_contact_entry.grid(row=9, column=2)

# Entry fields for adding transaction
global prop_id_entry, buyer_id_entry, agent_id_entry, transaction_date_entry
prop_id_label = tk.Label(root, text="Property ID:")
prop_id_label.grid(row=10, column=1)
prop_id_entry = tk.Entry(root)
prop_id_entry.grid(row=10, column=2)

buyer_id_label = tk.Label(root, text="Buyer ID:")
buyer_id_label.grid(row=11, column=1)
buyer_id_entry = tk.Entry(root)
buyer_id_entry.grid(row=11, column=2)

agent_id_label = tk.Label(root, text="Agent ID:")
agent_id_label.grid(row=12, column=1)
agent_id_entry = tk.Entry(root)
```

```
        agent_id_entry.grid(row=12, column=2)

        transaction_date_label = tk.Label(root, text="Transaction Date:")
        transaction_date_label.grid(row=13, column=1)
        transaction_date_entry = tk.Entry(root)
        transaction_date_entry.grid(row=13, column=2)

        root.mainloop()

        connection.close()

if __name__ == "__main__":
    main()
```

7. Result and Discussion (Screen shots of the implementation with front end:

- **Front end (GUI):**



- **Back end (MySQL):**

```
mysql> use realestate;
Database changed
mysql> show tables;
+---------------------+
| Tables_in_realestate |
+---------------------+
| agents               |
| buyers               |
| owners               |
| properties           |
| transactions         |
+---------------------+
5 rows in set (0.01 sec)

mysql> select * from properties;
+----+----------------------------+-----------+----------+-----------+
| id | address                    | price     | bedrooms | bathrooms |
+----+----------------------------+-----------+----------+-----------+
|  1 | 123 Main St                |  250000.00 |        3 |         2 |
|  2 | 456 Oak Ave                |  350000.00 |        4 |         3 |
|  3 | 789 Pine Ln                |  200000.00 |        2 |         1 |
|  4 | T1-1022, Estancia,chennai  | 1100000.00 |        3 |         2 |
|  5 | srm oori                   |  300000.00 |        2 |         1 |
|  6 | R-401,abode valley, potheri | 7500000.00 |        3 |         2 |
|  7 | a-304, maya garden, punjab |  450000.00 |        3 |         3 |
|  8 | r-302, abode               | 4000000.00 |        3 |         2 |
+----+----------------------------+-----------+----------+-----------+
8 rows in set (0.01 sec)
```

```
mysql> select * from agents;
+----+---------+----------------+
| id | name    | contact_number |
+----+---------+----------------+
|  1 | Agent1  | 555-4444       |
|  2 | Agent2  | 555-5555       |
|  3 | Agent3  | 555-6666       |
|  4 | sarthak | 9876993443     |
+----+---------+----------------+
4 rows in set (0.01 sec)

mysql> select * from buyers;
+----+---------------+----------------+
| id | name          | contact_number |
+----+---------------+----------------+
|  1 | Alice Brown   | 555-1111       |
|  2 | Charlie Davis | 555-2222       |
|  3 | Eva White     | 555-3333       |
|  4 | aryan         | 1234523456     |
+----+---------------+----------------+
4 rows in set (0.01 sec)
```

```
mysql> select * from owners;
+----+--------------+----------------+
| id | name         | contact_number |
+----+--------------+----------------+
|  1 | John Doe     | 555-1234       |
|  2 | Jane Smith   | 555-5678       |
|  3 | Bob Johnson  | 555-9876       |
|  4 | sarthak      | 9876993443     |
|  5 | pranshu      | 5432678912     |
|  6 | dr. s. ramesh | 9999988888    |
+----+--------------+----------------+
6 rows in set (0.01 sec)
```

8. The Real Time project certificate / Online course certificate:

- **SARTHAK SHARMA (RA2211003010744):**

Sarthak sharma

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advanced Concepts**

Following are the the learning items, which are covered in this tutorial

74 Video Tutorials 16 Modules 16 Challenges 29 February 2024

Anshuman Singh

Co-founder **SCALER**

CERTIFICATE OF EXCELLENCE BY SCALER

THANK YOU!