2024

PROJECT REPORT

Presented to:
Sir Rohail Qamar

Group Members:
1. Ebaad Khan (DT-22045)
2. M. Ezaan Khan (DT-22046)
3. Syed Ahmed Ali (DT-22301)
4. Khuzaima Hassan (DT-22302)

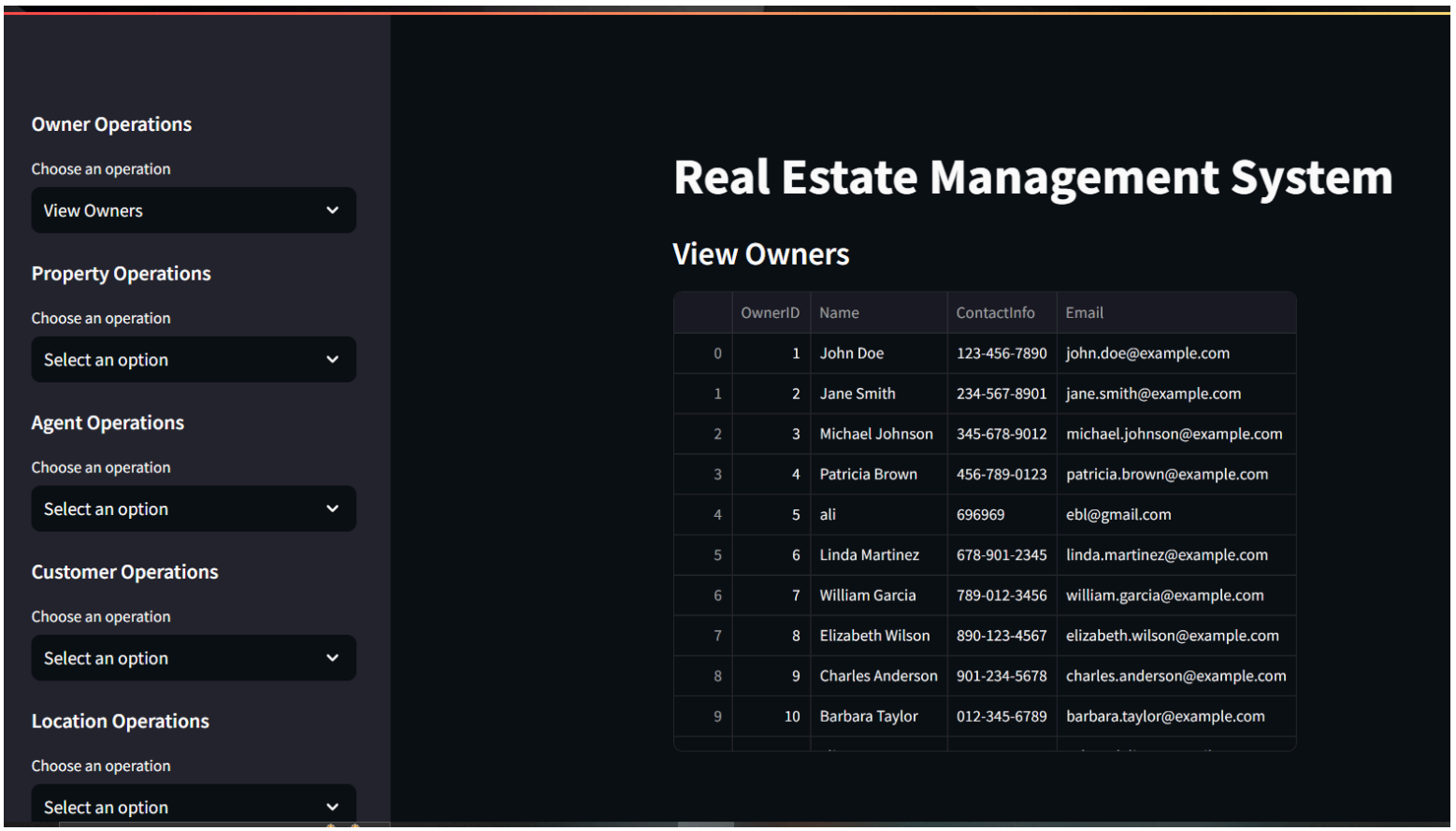
DATABASE MANAGEMENT SYSTEM

COURSE CODE: CT – 261

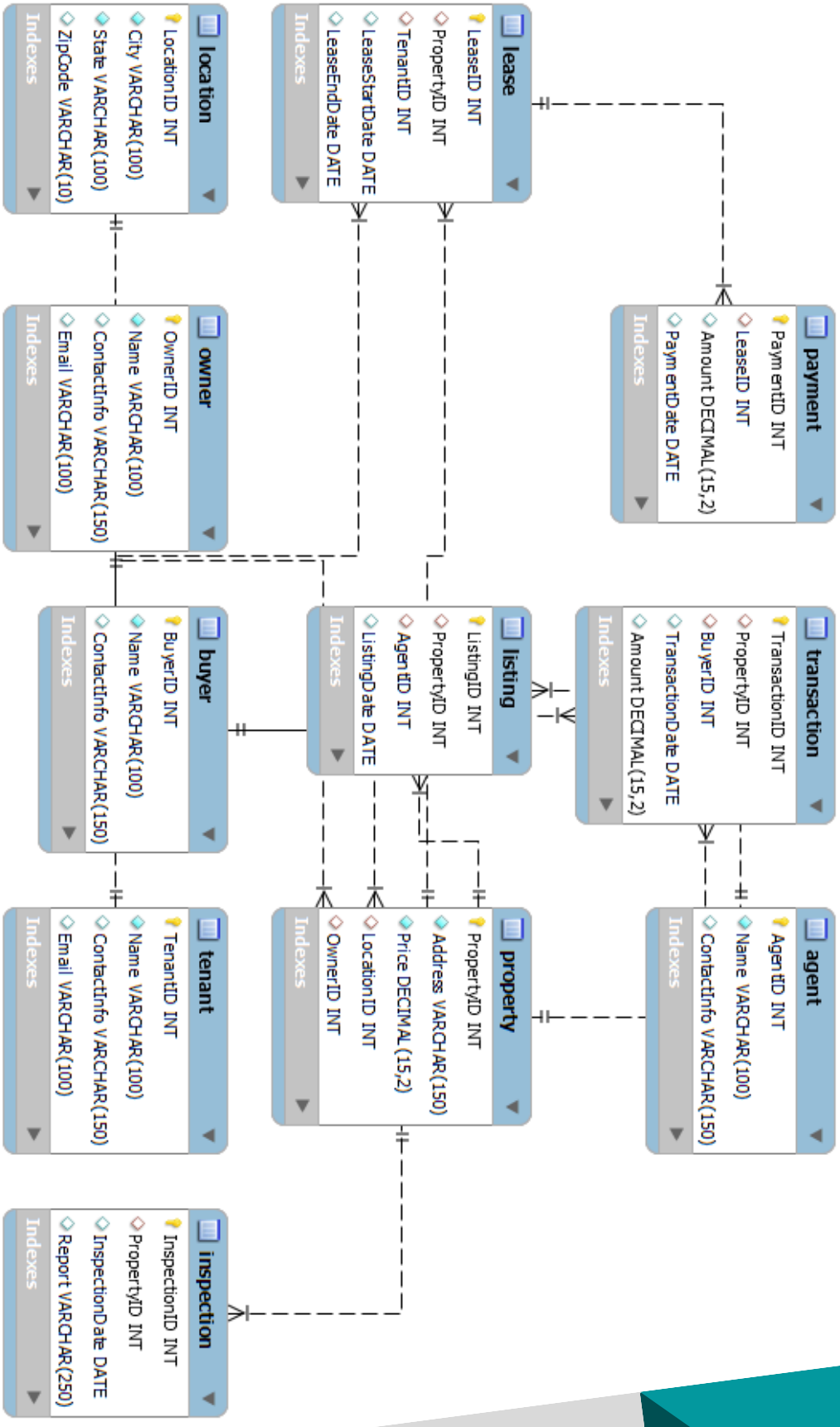
ABSTRACT

This project involves designing and implementing a database for a real estate management system. The database facilitates the management of property listings, transactions, tenants, and related activities. The objective is to create a robust, normalized database schema up to the Third Normal Form (3NF) to ensure data integrity and reduce redundancy. The front end of the application, developed using Python, interacts with this database to provide a user-friendly interface for managing real estate operations.

UI OF DATABASE



Entity Relationship Diagram



NORMALIZATION

Normalization involves organizing the attributes and tables of a relational database to minimize redundancy and dependency. The process typically involves the following steps:

1NF(First Normal Form): Ensuring that each table has atomic (indivisible) values and each column contains values of a single type.

2NF(Second Normal Form): Ensuring that each non primary key attribute is fully functionally dependent on the primary key.

3NF(Third Normal Form): Ensuring that there are no transitive dependencies, i.e., non primary key attributes are not dependent on other non primary key attributes.

Owner Table

1NF: All values are atomic and there is a primary key.

2NF All non key attributes depend on owner_ id.

3NF There are no transitive dependencies.

OwnerID	Name	ContactInfo	Email
1	John Doe	123-456-7890	john.doe@example.com
2	Jane Smith	234-567-8901	jane.smith@example.com
3	Michael Johnson	345-678-9012	michael.johnson@example.com
4	Patricia Brown	456-789-0123	patricia.brown@example.com
5	Robert Davis	567-890-1234	robert.davis@example.com
6	Linda Martinez	678-901-2345	linda.martinez@example.com
7	William Garcia	789-012-3456	william.garcia@example.com
8	Elizabeth Wilson	890-123-4567	elizabeth.wilson@example.com
9	Charles Anderson	901-234-5678	charles.anderson@example.com
10	Barbara Taylor	012-345-6789	barbara.taylor@example.com
NULL	NULL	NULL	NULL

Location Table

1NF: All values are atomic and there is a primary key.

2NF All non key attributes depend on location_ id.

3NF There are no transitive dependencies.

LocationID	Address	City	State	ZipCode
1	123 Elm St	Springfield	IL	62701
2	456 Oak St	Chicago	IL	60601
3	789 Pine St	Los Angeles	CA	90001
4	101 Maple St	New York	NY	10001
5	202 Birch St	Houston	TX	77001
6	303 Cedar St	Phoenix	AZ	85001
7	404 Walnut St	Philadelphia	PA	19019
8	505 Ash St	San Antonio	TX	78201
9	606 Willow St	San Diego	CA	92101
10	707 Hickory St	Dallas	TX	75201
NULL	NULL	NULL	NULL	NULL

Property Table

1NF: All values are atomic and there is a primary key.

2NF All non key attributes depend on property_ id.

3NF There are no transitive dependencies.

PropertyID	Address	Price	LocationID	OwnerID
1	123 Elm St	250000.00	1	1
2	456 Oak St	300000.00	2	2
3	789 Pine St	150000.00	3	3
4	101 Maple St	350000.00	4	4
5	202 Birch St	275000.00	5	5
6	303 Cedar St	325000.00	6	6
7	404 Walnut St	200000.00	7	7
8	505 Ash St	225000.00	8	8
9	606 Willow St	180000.00	9	9
10	707 Hickory St	400000.00	10	10
11	123 Elm St	250000.00	1	1
NULL	NULL	NULL	NULL	NULL

QUERIES:

DDL

```sql
CREATE TABLE Owner (
    OwnerID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    ContactInfo VARCHAR(150),
    Email VARCHAR(100) UNIQUE
);
```

```sql
CREATE TABLE Location (
    LocationID INT AUTO_INCREMENT PRIMARY KEY,
    Address VARCHAR(150) NOT NULL,
    City VARCHAR(100) NOT NULL,
    State VARCHAR(100) NOT NULL,
    ZipCode VARCHAR(10) NOT NULL
);
```

```sql
CREATE TABLE Property (
    PropertyID INT AUTO_INCREMENT PRIMARY KEY,
    Address VARCHAR(150) NOT NULL,
    Price DECIMAL(15, 2) NOT NULL,
    LocationID INT,
    OwnerID INT,
    FOREIGN KEY (OwnerID) REFERENCES Owner(OwnerID) ON DELETE CASCADE,
    FOREIGN KEY (LocationID) REFERENCES Location(LocationID) ON DELETE CASCADE
);
```

DML

Insert

```
116   -- Inserting sample data into Owner table
117 • INSERT INTO Owner (Name, ContactInfo, Email) VALUES
118   ('John Doe', '123-456-7890', 'john.doe@example.com'),
119   ('Jane Smith', '234-567-8901', 'jane.smith@example.com'),
120   ('Michael Johnson', '345-678-9012', 'michael.johnson@example.com'),
121   ('Patricia Brown', '456-789-0123', 'patricia.brown@example.com'),
122   ('Robert Davis', '567-890-1234', 'robert.davis@example.com'),
123   ('Linda Martinez', '678-901-2345', 'linda.martinez@example.com'),
124   ('William Garcia', '789-012-3456', 'william.garcia@example.com'),
125   ('Elizabeth Wilson', '890-123-4567', 'elizabeth.wilson@example.com'),
126   ('Charles Anderson', '901-234-5678', 'charles.anderson@example.com'),
127   ('Barbara Taylor', '012-345-6789', 'barbara.taylor@example.com');

129   -- Inserting sample data into Location table
130 • INSERT INTO Location (Address, City, State, ZipCode) VALUES
131   ('123 Elm St', 'Springfield', 'IL', '62701'),
132   ('456 Oak St', 'Chicago', 'IL', '60601'),
133   ('789 Pine St', 'Los Angeles', 'CA', '90001'),
134   ('101 Maple St', 'New York', 'NY', '10001'),
135   ('202 Birch St', 'Houston', 'TX', '77001'),
136   ('303 Cedar St', 'Phoenix', 'AZ', '85001'),
137   ('404 Walnut St', 'Philadelphia', 'PA', '19019'),
138   ('505 Ash St', 'San Antonio', 'TX', '78201'),
139   ('606 Willow St', 'San Diego', 'CA', '92101'),
140   ('707 Hickory St', 'Dallas', 'TX', '75201');


142   -- Inserting sample data into Property table
143 • INSERT INTO Property (Address, Price, OwnerID, LocationID) VALUES
144   ('123 Elm St', 250000.00, 1, 1),
145   ('456 Oak St', 300000.00, 2, 2),
146   ('789 Pine St', 150000.00, 3, 3),
147   ('101 Maple St', 350000.00, 4, 4),
148   ('202 Birch St', 275000.00, 5, 5),
149   ('303 Cedar St', 325000.00, 6, 6),
150   ('404 Walnut St', 200000.00, 7, 7),
151   ('505 Ash St', 225000.00, 8, 8),
152   ('606 Willow St', 180000.00, 9, 9),
153   ('707 Hickory St', 400000.00, 10, 10);
```

Update

```
354    -- UPDATE OPERATIONS
355
356 •  UPDATE Owner
357    SET ContactInfo = '999-999-9999'
358    WHERE OwnerID = 1;
359
360 •  UPDATE Property SET Price = 500000 WHERE PropertyID = 1;
361
```

Delete

```
367    -- Delete OPERATIONS
368
369 •  DELETE FROM Tenant WHERE TenantID = 1;
370
371    -- Select OPERATIONS
372 •  SELECT * FROM Agent WHERE City = 'New York';
373
```

Joins

```sql
-- Now the JOINS part

-- Retrieve all details for properties, including owner, location, and agent details for current listings:

SELECT
    p.PropertyID,
    p.Address,
    p.Price,
    o.Name AS OwnerName,
    l.City,
    l.State,
    a.Name AS AgentName,
    lg.ListingDate
FROM
    Property p
JOIN
    Owner o ON p.OwnerID = o.OwnerID
JOIN
    Location l ON p.LocationID = l.LocationID
JOIN
    Listing lg ON p.PropertyID = lg.PropertyID
JOIN
    Agent a ON lg.AgentID = a.AgentID;
```

```sql
-- Retrieve listing information along with agent and property details:

SELECT
    l.ListingID,
    a.Name AS AgentName,
    p.Address,
    l.ListingDate
FROM
    Listing l
JOIN
    Agent a ON l.AgentID = a.AgentID
JOIN
    Property p ON l.PropertyID = p.PropertyID;
```

```sql
-- Retrieve all leases including tenant and property details, and include lease payments:

SELECT
    l.LeaseID,
    t.Name AS TenantName,
    p.Address,
    l.LeaseStartDate,
    l.LeaseEndDate,
    pm.PaymentID,
    pm.Amount AS PaymentAmount,
    pm.PaymentDate
FROM
    Lease l
JOIN
    Tenant t ON l.TenantID = t.TenantID
JOIN
    Property p ON l.PropertyID = p.PropertyID
LEFT JOIN
    Payment pm ON l.LeaseID = pm.LeaseID;
```

```sql
-- Retrieve inspection details along with property information:

SELECT
    i.InspectionID,
    p.Address,
    i.InspectionDate,
    i.Report
FROM
    Inspection i
JOIN
    Property p ON i.PropertyID = p.PropertyID;
```

```sql
-- Retrieve all properties, including those without a listing (use RIGHT JOIN to include all properties, even those not listed).

SELECT
    p.PropertyID,
    p.Address,
    l.ListingID,
    a.Name AS AgentName
FROM
    Property p
RIGHT JOIN
    Listing l ON p.PropertyID = l.PropertyID
LEFT JOIN
    Agent a ON l.AgentID = a.AgentID
ORDER BY
    p.PropertyID;
```

```sql
--  Retrieve all tenants and their corresponding lease details, including tenants without a lease.

SELECT
    t.TenantID,
    t.Name AS TenantName,
    l.LeaseID,
    l.PropertyID,
    l.LeaseStartDate,
    l.LeaseEndDate
FROM
    Tenant t
LEFT JOIN
    Lease l ON t.TenantID = l.TenantID
ORDER BY
    t.TenantID;
```

CONCLUSION

In this project, we successfully designed and implemented a database management system for a real estate transaction application. The initial denormalized table provided a comprehensive view but included redundancy and potential anomalies. Through the normalization process, we ensured data integrity and minimized redundancy.

Key Accomplishments:
1. Database Design: Created a denormalized table combining properties, owners, and agents, and normalized it to 3NF.
2. Normalization:
   - 1NF: Ensured atomic values and eliminated repeating groups.
   - 2NF: Removed partial dependencies by decomposing into `Property`, `Owner`,and`Agent` tables.
   - 3NF: Removed transitive dependencies, ensuring non-key attributes fully depended on the primary key.
3. SQL Queries: Demonstrated DDL and DML commands, showcasing creating, altering, dropping tables, and data manipulation.
4. Joins: Used various types of joins to retrieve related data from multiple tables.

Benefits:
- Data Integrity: Minimizing redundancy and preventing anomalies.
- Efficiency:Improved query performance and database management.
- Scalability: A robust design that can handle additional data and relationships.

Future Enhancements:
- Additional Functionalities: Implementing advanced features like stored procedures and triggers.
- Optimization: Continuous performance monitoring and optimization.
- Security: Implementing robust security measures.

This project provided practical experience in database design, emphasizing the importance of best practices in database management. The final database schema is robust, scalable, and ready to support real-world applications.