EFREI

Advanced Databases Project

Luxury Property Management System

Samy BOUAISSA
Amine M'ZALI

18 janvier 2025

Table des matières

1 Introduction . . . 2
2 Requirements . . . 2
2.1 Database Design : Conceptual Model to Logical Model . . . 2
2.2 PL/SQL . . . 2
2.3 Object Relational Databases (ORD) . . . 3
2.4 The Normal Forms . . . 3
2.5 NoSQL Databases . . . 3
2.6 Conclusion . . . 4
3 Entity-Relationship Diagram . . . 4
3.1 E/R Diagram . . . 4
4 Logical Model . . . 5
4.1 SQL Script . . . 6
5 PL/SQL Procedures . . . 14
5.1 Package Definition and Implementation . . . 14
5.2 Example Usage of Procedures . . . 17
5.3 Procedures vs. Triggers : Why Choose Procedures ? . . . 18
5.4 Advanced Procedures . . . 19
6 Object-Relational Database (ORD) . . . 21
6.1 Custom Types . . . 22
6.2 Inheritance . . . 22
6.3 Tables and Data . . . 22
6.4 Conclusion . . . 23
7 Normalization . . . 23
7.1 First Normal Form (1NF) . . . 23
7.2 Second Normal Form (2NF) . . . 24
7.3 Third Normal Form (3NF) . . . 24
7.4 Boyce-Codd Normal Form (BCNF) . . . 24
7.5 Fourth Normal Form (4NF) . . . 24
7.6 Fifth Normal Form (5NF) . . . 24
7.7 Conclusion . . . 24
8 NoSQL Databases . . . 25
8.1 MongoDB . . . 25
8.2 Neo4J . . . 27
9 Conclusion . . . 29

1 Introduction

This report details the development of a database system for a luxury property management firm as part of the Advanced Databases module. The project explores advanced database concepts, including relational and NoSQL systems, PL/SQL, and normalization. The goal is to provide a robust solution for managing properties, clients, and transactions while ensuring data integrity and performance.

2 Requirements

This section outlines how the project fulfills each of the requirements specified in the guidelines, demonstrating the application of advanced database concepts and technologies.

2.1 Database Design : Conceptual Model to Logical Model

2.1.1 E/R Model Development

An Entity-Relationship (E/R) model was developed to represent the conceptual structure of the luxury property management system. Key entities include `Property`, `Owner`, `Transaction`, `Tour`, and `Facility`, with attributes such as :

- `Property` : Address, Type, Rooms, Price, Status, Availability Date.
- `Owner` : Name, Contact Information.
- `Transaction` : Total Amount, Commission.
- `Tour` : Client Information, Date and Time.
- `Facility` : Type, Description.

Primary keys were identified for each entity, ensuring unique identification of records.

2.1.2 Logical Model Translation

The E/R model was translated into a logical model, resulting in a relational database schema with tables, primary keys, foreign keys, and relationships. The schema respects normalization principles and serves as the foundation for SQL-based queries.

2.2 PL/SQL

2.2.1 Constraints

At least five constraints were implemented to ensure data integrity :

- `PRIMARY KEY` : For example, `Property.ID` uniquely identifies each property.
- `FOREIGN KEY` : Relationships between tables are enforced, such as `Property.OwnerID` referencing `Owner.ID`.
- `NOT NULL` : Columns like `Property.Address` are required.
- `CHECK` : Ensures valid values, such as `Property.Price > 10000`.
- `UNIQUE` : For example, `Owner.ContactInfo` ensures no duplicate contact information.

2.2.2 Triggers

At least five triggers were implemented to automate tasks :
— Automatically calculate commission after a transaction.
— Validate exclusivity rules for tours.
— Update property status to `Sold` after a transaction.
— Prevent scheduling conflicts for tours.
— Validate minimum property prices before insertion or update.

2.2.3 Organization in Packages

These triggers were complemented by encapsulated procedures within the `LuxuryPropertyUtils` package. The package contains advanced procedures such as dynamic commission assignment, multi-day tour validation, and monthly performance report generation.

2.3 Object Relational Databases (ORD)

2.3.1 Custom Types

At least five custom types were created to extend database functionality, such as :
— `AddressType` : Represents a structured address.
— `PropertyType` : Includes attributes like rooms, condition, and availability.

2.3.2 Inheritance Relations

Two inheritance relations were implemented :
— `BaseProperty` as a parent type.
— `Mansion` and `Apartment` as child types with unique attributes (e.g., swimming pool for mansions, floor number for apartments).
These relations were demonstrated using data insertion queries.

2.4 The Normal Forms

2.4.1 Normalization Adherence

The database design adheres to all six normal forms :
— **1NF** : Atomic values in columns, no multi-valued fields.
— **2NF** : Non-key attributes are fully dependent on the primary key.
— **3NF** : No transitive dependencies.
— **BCNF** : Every determinant is a candidate key.
— **4NF** : Multi-valued dependencies are eliminated using junction tables.
— **5NF** : No join dependencies exist, ensuring data integrity during decomposition.

2.5 NoSQL Databases

2.5.1 MongoDB

MongoDB was used to manage document-based data, such as property descriptions and client feedback. Collections like `properties` and `transactions` were created to handle unstructured and semi-structured data efficiently.

2.5.2 Neo4J

Neo4J was employed to model and analyze graph-based relationships, such as :
— `Client` nodes connected to `Property` nodes via `INTERESTED_IN` relationships.
— Graph queries demonstrated insights into client-property interactions.

2.5.3 Relational vs. NoSQL Comparison

A comparison was made to highlight the strengths of each approach :
— Relational databases ensure data integrity and structured querying.
— NoSQL databases excel in scalability and handling unstructured or graph-based data.

2.6 Conclusion

The project fulfills all the requirements by integrating advanced database technologies and concepts, providing a robust and scalable solution for luxury property management.

3 Entity-Relationship Diagram

The Entity-Relationship (E/R) diagram represents the conceptual model of the luxury property management system. It defines the entities, their attributes, and the relationships between them.

3.1 E/R Diagram

The following E/R diagram was created using the information provided in the project description. It highlights the main entities such as `Property`, `Owner`, `Transaction`, `Tour`, and `Facility`, along with their relationships.

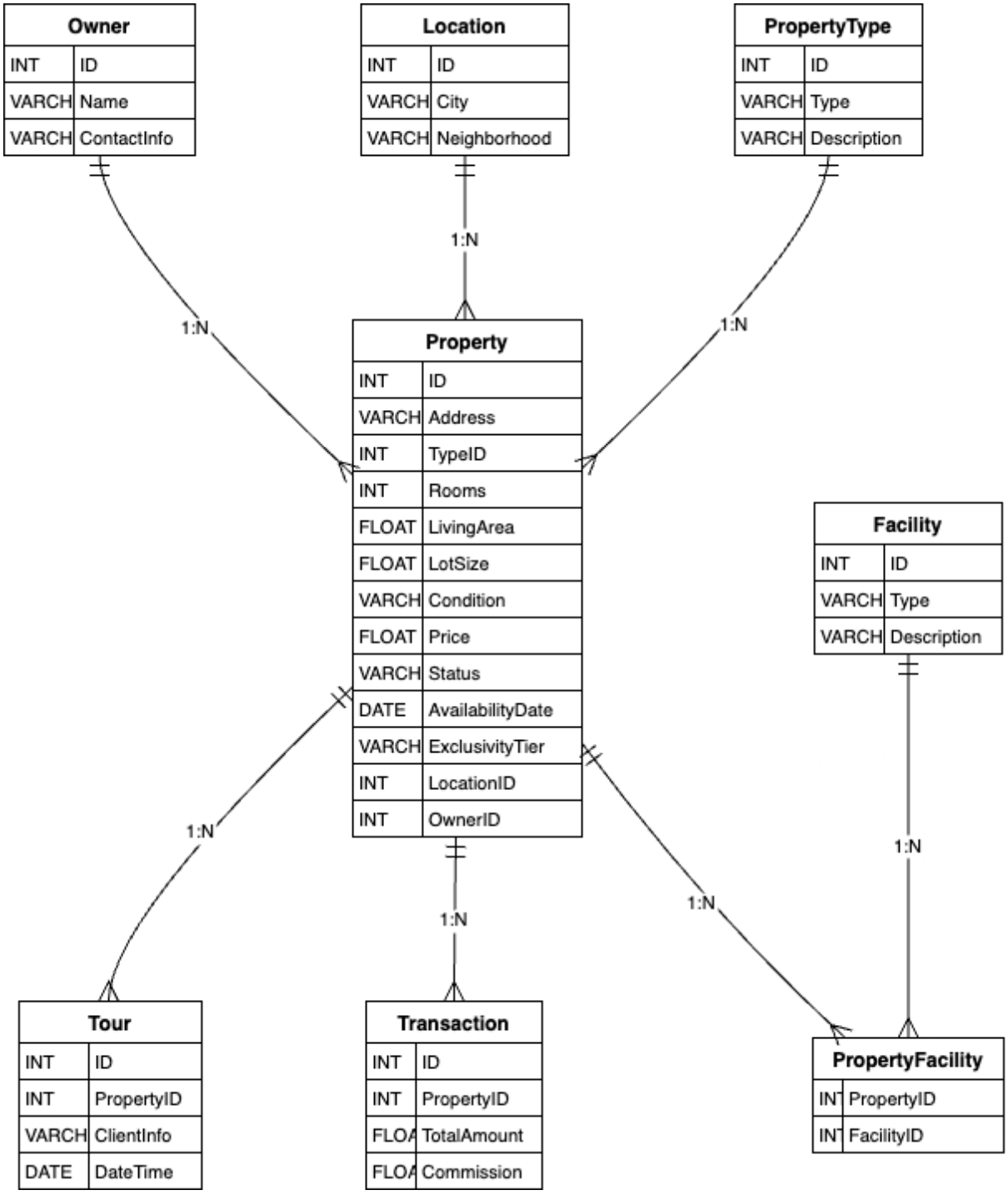


Figure 1 – E/R Diagram for Luxury Property Management System

4 Logical Model

Based on the E/R diagram, the logical model defines the database schema using SQL. It includes tables, attributes, primary and foreign keys, and relationships.

4.1 SQL Script

The final SQL script (including table creation and sample data insertion) is shown below.

4.1.1 Table Creation

```
-------------------------------------------------------------
-- TABLE CREATION
-------------------------------------------------------------

-- Table: PropertyType
CREATE TABLE PropertyType (
    ID INT PRIMARY KEY,
    Type VARCHAR(50),
    Description VARCHAR(255)
);

-- Table: Location
CREATE TABLE Location (
    ID INT PRIMARY KEY,
    City VARCHAR(100),
    Neighborhood VARCHAR(100)
);

-- Table: Owner
CREATE TABLE Owner (
    ID INT PRIMARY KEY,        -- Identifiant unique pour chaque
       propri taire
    Name VARCHAR(100),         -- Nom du propri taire
    ContactInfo VARCHAR(255)   -- Coordonn es (email, t l phone
       , etc.)
);

-- Table: Property
CREATE TABLE Property (
    ID INT PRIMARY KEY,                -- Identifiant unique pour
        chaque propri t
    Address VARCHAR(255),              -- Adresse de la
       propri t
    TypeID INT,                        -- R f rence vers
       PropertyType
    Rooms INT,                         -- Nombre de chambres
    LivingArea FLOAT,                  -- Surface habitable
    LotSize FLOAT,                     -- Taille du terrain
    Condition VARCHAR(50),             --  tat de la propri t
    Price FLOAT,                       -- Prix de la propri t
    Status VARCHAR(50),                -- Statut de la
       propri t  (e.g., Available, Sold)
    AvailabilityDate DATE,             -- Date de disponibilit
```

```
    ExclusivityTier VARCHAR(50),        -- Niveau d'exclusivit  (
       e.g., Premium, Standard)
    LocationID INT,                     -- R f rence vers
       Location
    OwnerID INT,                        -- R f rence vers Owner
    FOREIGN KEY (TypeID) REFERENCES PropertyType(ID),
    FOREIGN KEY (LocationID) REFERENCES Location(ID),
    FOREIGN KEY (OwnerID) REFERENCES Owner(ID)
);

-- Table: Facility
CREATE TABLE Facility (
    ID INT PRIMARY KEY,
    Type VARCHAR(50),
    Description VARCHAR(255)
);

-- Table: PropertyFacility
CREATE TABLE PropertyFacility (
    PropertyID INT,
    FacilityID INT,
    PRIMARY KEY (Property

'''latex
\documentclass[12pt,a4paper]{report}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[french]{babel}
\usepackage{lmodern}
\usepackage{graphicx}
\usepackage{hyperref}
\usepackage{geometry}
\usepackage{listings}
\usepackage{xcolor}
\usepackage{float}
\usepackage{placeins}

\geometry{margin=1in}

\hypersetup{
    colorlinks=true,
    linkcolor=blue,
    urlcolor=blue,
    citecolor=blue
}
\setcounter{secnumdepth}{3} % Contr le la profondeur de la
   num rotation
\renewcommand\thesection{\arabic{section}} % Num rotation
   normale pour les sections
\renewcommand\thesubsection{\thesection.\arabic{subsection}} %
   Num rotation des sous-sections
```

```
\renewcommand\thesubsubsection{\thesubsection.\arabic{
    subsubsection}} % Sous-sous-sections

\begin{document}

% Configuration for code listings
\lstset{
    basicstyle=\ttfamily\small,
    keywordstyle=\color{blue}\bfseries,
    commentstyle=\color{olive},
    stringstyle=\color{red},
    showstringspaces=false,
    tabsize=4,
    breaklines=true,
}

% Page de titre
\begin{titlepage}
    \centering
    {\scshape\LARGE EFREI \par}
    \vspace{1.5cm}
    {\scshape\Large Advanced Databases Project \par}
    \vspace{2cm}
    {\huge\bfseries Luxury Property Management System \par}
    \vspace{2cm}
    {\Large\itshape Samy BOUAISSA \\ Amine M'ZALI \par}
    \vfill
    {\large \today \par}
    \vspace{1cm}
    \includegraphics[width=0.4\textwidth]{logo.png} % Remplacez '
        logo.png' par le chemin vers le logo de l'EFREI
\end{titlepage}

\tableofcontents
\newpage

\section{Introduction}
This report details the development of a database system for a
    luxury property management firm as part of the Advanced
    Databases module. The project explores advanced database
    concepts, including relational and NoSQL systems, PL/SQL, and
    normalization. The goal is to provide a robust solution for
    managing properties, clients, and transactions while ensuring
    data integrity and performance.

\section{Entity-Relationship Diagram}
The Entity-Relationship (E/R) diagram represents the conceptual
    model of the luxury property management system. It defines the
     entities, their attributes, and the relationships between
    them.
```

```
\subsection{E/R Diagram}
The following E/R diagram was created using the information
   provided in the project description. It highlights the main
   entities such as \texttt{Property}, \texttt{Owner}, \texttt{
   Transaction}, \texttt{Tour}, and \texttt{Facility}, along with
    their relationships.

\begin{figure}[h!]
    \centering
    \includegraphics[width=\textwidth]{E:R.png}
    \caption{E/R Diagram for Luxury Property Management System}
    \label{fig:er-diagram}
\end{figure}

\newpage

\section{Logical Model}
Based on the E/R diagram, the logical model defines the database
   schema using SQL. It includes tables, attributes, primary and
   foreign keys, and relationships.

\subsection{SQL Script}
The final SQL script (including table creation and sample data
   insertion) is shown below.

\subsubsection{Table Creation}
\begin{lstlisting}[language=SQL]
------------------------------------------------------------
-- TABLE CREATION
------------------------------------------------------------

-- Table: PropertyType
CREATE TABLE PropertyType (
    ID INT PRIMARY KEY,
    Type VARCHAR(50),
    Description VARCHAR(255)
);

-- Table: Location
CREATE TABLE Location (
    ID INT PRIMARY KEY,
    City VARCHAR(100),
    Neighborhood VARCHAR(100)
);

-- Table: Owner
CREATE TABLE Owner (
    ID INT PRIMARY KEY,        -- Identifiant unique pour chaque
       propri taire
    Name VARCHAR(100),         -- Nom du propri taire
```

```
    ContactInfo VARCHAR(255)  -- Coordonn es (email, t l phone
       , etc.)
);

-- Table: Property
CREATE TABLE Property (
    ID INT PRIMARY KEY,                 -- Identifiant unique pour
        chaque propri t
    Address VARCHAR(255),               -- Adresse de la
       propri t
    TypeID INT,                         -- R f rence vers
       PropertyType
    Rooms INT,                          -- Nombre de chambres
    LivingArea FLOAT,                   -- Surface habitable
    LotSize FLOAT,                      -- Taille du terrain
    Condition VARCHAR(50),              --  tat  de la propri t
    Price FLOAT,                        -- Prix de la propri t
    Status VARCHAR(50),                 -- Statut de la
       propri t  (e.g., Available, Sold)
    AvailabilityDate DATE,              -- Date de disponibilit
    ExclusivityTier VARCHAR(50),        -- Niveau d'exclusivit  (
       e.g., Premium, Standard)
    LocationID INT,                     -- R f rence vers
       Location
    OwnerID INT,                        -- R f rence vers Owner
    FOREIGN KEY (TypeID) REFERENCES PropertyType(ID),
    FOREIGN KEY (LocationID) REFERENCES Location(ID),
    FOREIGN KEY (OwnerID) REFERENCES Owner(ID)
);

-- Table: Facility
CREATE TABLE Facility (
    ID INT PRIMARY KEY,
    Type VARCHAR(50),
    Description VARCHAR(255)
);

-- Table: PropertyFacility
CREATE TABLE PropertyFacility (
    PropertyID INT,
    FacilityID INT,
    PRIMARY KEY (PropertyID, FacilityID),
    FOREIGN KEY (PropertyID) REFERENCES Property(ID),
    FOREIGN KEY (FacilityID) REFERENCES Facility(ID)
);

-- Table: Transaction
CREATE TABLE Transaction (
    ID INT PRIMARY KEY,
    PropertyID INT,
    TotalAmount FLOAT,
```

```
    Commission FLOAT,
    FOREIGN KEY (PropertyID) REFERENCES Property(ID)
);

-- Table: Tour
CREATE TABLE Tour (
    ID INT PRIMARY KEY,
    PropertyID INT,
    ClientInfo VARCHAR(255),
    DateTime DATE,
    FOREIGN KEY (PropertyID) REFERENCES Property(ID)
);
```

```
62      ID INT PRIMARY KEY,
63      PropertyID INT,
64      TotalAmount FLOAT,
65      Commission FLOAT,
66      FOREIGN KEY (PropertyID) REFERENCES Property(ID)
67  );
68
69  -- Table: Tour
70  CREATE TABLE Tour (
71      ID INT PRIMARY KEY,
72      PropertyID INT,
73      ClientInfo VARCHAR(255),
74      DateTime DATE,  -- or TIMESTAMP if needed
75      FOREIGN KEY (PropertyID) REFERENCES Property(ID)
76  );
```

```
Table created.

Table created.

Table created.

Table created.

Table created.

Table created.

Table created.
```

FIGURE 2 – Capture d'écran : Tables créées

4.1.2 Data Insertion

```
-------------------------------------------------------------
-- SAMPLE DATA INSERTION (Updated)
-------------------------------------------------------------

-- Insertion dans PropertyType
INSERT INTO PropertyType (ID, Type, Description)
VALUES (1, 'Mansion', 'A luxurious large house with many
   amenities');
INSERT INTO PropertyType (ID, Type, Description)
```

```
VALUES (2, 'Apartment', 'A unit in a residential building');
INSERT INTO PropertyType (ID, Type, Description)
VALUES (3, 'Villa', 'A luxury countryside house');
INSERT INTO PropertyType (ID, Type, Description)
VALUES (4, 'Penthouse', 'An exclusive apartment on the top floor'
   );

-- Insertion dans Location
INSERT INTO Location (ID, City, Neighborhood)
VALUES (1, 'Paris', 'Champs-  lyses  ');
INSERT INTO Location (ID, City, Neighborhood)
VALUES (2, 'London', 'Chelsea');
INSERT INTO Location (ID, City, Neighborhood)
VALUES (3, 'New York', 'Manhattan');
INSERT INTO Location (ID, City, Neighborhood)
VALUES (4, 'Tokyo', 'Shibuya');

-- Insertion dans Owner
INSERT INTO Owner (ID, Name, ContactInfo)
VALUES (1, 'John Doe', 'john.doe@example.com');
INSERT INTO Owner (ID, Name, ContactInfo)
VALUES (2, 'Jane Smith', 'jane.smith@example.com');
INSERT INTO Owner (ID, Name, ContactInfo)
VALUES (3, 'Alice Johnson', 'alice.johnson@example.com');

-- Insertion dans Property
INSERT INTO Property (
    ID, Address, TypeID, Rooms, LivingArea, LotSize, Condition,
       Price, Status,
    AvailabilityDate, ExclusivityTier, LocationID, OwnerID
)
VALUES (
    1, '123 Luxury St', 1, 10, 450.5, 1200.3, 'Excellent',
    15000000, 'Available', TO_DATE('2025-01-15','YYYY-MM-DD'),
    'Premium', 1, 1
);

INSERT INTO Property (
    ID, Address, TypeID, Rooms, LivingArea, LotSize, Condition,
       Price, Status,
    AvailabilityDate, ExclusivityTier, LocationID, OwnerID
)
VALUES (
    2, '456 Elegant Rd', 2, 5, 200.0, NULL, 'New',
    5000000, 'Available', TO_DATE('2025-02-01','YYYY-MM-DD'),
    'Standard', 2, 2
);

INSERT INTO Property (
    ID, Address, TypeID, Rooms, LivingArea, LotSize, Condition,
       Price, Status,
```

```
    AvailabilityDate, ExclusivityTier, LocationID, OwnerID
)
VALUES (
    3, '789 Villa Way', 3, 7, 350.0, 900.0, 'Good',
    8000000, 'Available', TO_DATE('2025-03-01','YYYY-MM-DD'),
    'Limited Access', 3, 3
);

-- Insertion dans Facility
INSERT INTO Facility (ID, Type, Description)
VALUES (1, 'Pool', 'A private swimming pool');
INSERT INTO Facility (ID, Type, Description)
VALUES (2, 'Garage', 'A spacious garage for vehicles');
INSERT INTO Facility (ID, Type, Description)
VALUES (3, 'Guest House', 'An additional guest house on the
   property');

-- Insertion dans PropertyFacility
INSERT INTO PropertyFacility (PropertyID, FacilityID)
VALUES (1, 1);
INSERT INTO PropertyFacility (PropertyID, FacilityID)
VALUES (1, 2);
INSERT INTO PropertyFacility (PropertyID, FacilityID)
VALUES (1, 3);
INSERT INTO PropertyFacility (PropertyID, FacilityID)
VALUES (2, 2);
INSERT INTO PropertyFacility (PropertyID, FacilityID)
VALUES (3, 1);

-- Insertion dans Transaction
INSERT INTO Transaction (ID, PropertyID, TotalAmount, Commission)
VALUES (1, 1, 15000000, NULL);
INSERT INTO Transaction (ID, PropertyID, TotalAmount, Commission)
VALUES (2, 2, 5000000, NULL);
INSERT INTO Transaction (ID, PropertyID, TotalAmount, Commission)
VALUES (3, 3, 8000000, NULL);

-- Insertion dans Tour
INSERT INTO Tour (ID, PropertyID, ClientInfo, DateTime)
VALUES (
    1, 1, 'VIP Client',
    TO_DATE('2025-01-10 14:00:00','YYYY-MM-DD HH24:MI:SS')
);

INSERT INTO Tour (ID, PropertyID, ClientInfo, DateTime)
VALUES (
    2, 2, 'Potential Buyer',
    TO_DATE('2025-02-05 11:00:00','YYYY-MM-DD HH24:MI:SS')
);

INSERT INTO Tour (ID, PropertyID, ClientInfo, DateTime)
```

```
VALUES (
    3, 3, 'Exclusive Client',
    TO_DATE('2025-03-15 16:00:00','YYYY-MM-DD HH24:MI:SS')
);
```

```
95      1, 1, 'VIP Client',
96      TO_DATE('2025-01-10 14:00:00','YYYY-MM-DD HH24:MI:SS')
97  );
98
99  INSERT INTO Tour (ID, PropertyID, ClientInfo, DateTime)
100 VALUES (
101     2, 2, 'Potential Buyer',
102     TO_DATE('2025-02-05 11:00:00','YYYY-MM-DD HH24:MI:SS')
103 );
104
105 INSERT INTO Tour (ID, PropertyID, ClientInfo, DateTime)
106 VALUES (
107     3, 3, 'Exclusive Client',
108     TO_DATE('2025-03-15 16:00:00','YYYY-MM-DD HH24:MI:SS')
109 );

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.
```

FIGURE 3 – Capture d'écran : Données insérées avec succès

5 PL/SQL Procedures

According to the updated instruction, we propose at least five different **procedures** and organize them into a **package**, then illustrate them with examples.

5.1 Package Definition and Implementation

```
-------------------------------------------------------------
-- PACKAGE DEFINITION
-------------------------------------------------------------
CREATE OR REPLACE PACKAGE LuxuryPropertyUtils AS
    -- Procedure 1: Validate Property Price
    PROCEDURE validate_property_price(p_property_id INT, p_price
       FLOAT);

    -- Procedure 2: Check Tour Conflict
    PROCEDURE check_tour_conflict(p_property_id INT, p_datetime
       DATE);
```

```
    -- Procedure 3: Update Property Status
    PROCEDURE update_property_status(p_property_id INT);

    -- Procedure 4: Calculate Commission
    PROCEDURE calculate_commission(p_transaction_id INT,
       p_total_amount FLOAT);

    -- Procedure 5: Validate Exclusive Tour
    PROCEDURE validate_exclusive_tour(p_property_id INT,
       p_client_info VARCHAR2);
END LuxuryPropertyUtils;
/
-------------------------------------------------------
-- PACKAGE BODY
-------------------------------------------------------
CREATE OR REPLACE PACKAGE BODY LuxuryPropertyUtils AS

    -------------------------------------------------------
    -- Procedure 1: Validate Property Price
    -------------------------------------------------------
    PROCEDURE validate_property_price(p_property_id INT, p_price
       FLOAT) IS
    BEGIN
        IF p_price < 10000 THEN
            RAISE_APPLICATION_ERROR(-20001, 'The property price
               must be at least   10  ,000.');
        END IF;
    END validate_property_price;

    -------------------------------------------------------
    -- Procedure 2: Check Tour Conflict
    -------------------------------------------------------
    PROCEDURE check_tour_conflict(p_property_id INT, p_datetime
       DATE) IS
        v_count INTEGER;
    BEGIN
        SELECT COUNT(*)
        INTO v_count
        FROM Tour
        WHERE PropertyID = p_property_id
          AND DateTime = p_datetime;

        IF v_count > 0 THEN
            RAISE_APPLICATION_ERROR(-20002,
                'A tour is already scheduled for this property at
                    the specified time.');
        END IF;
    END check_tour_conflict;

    -------------------------------------------------------
```

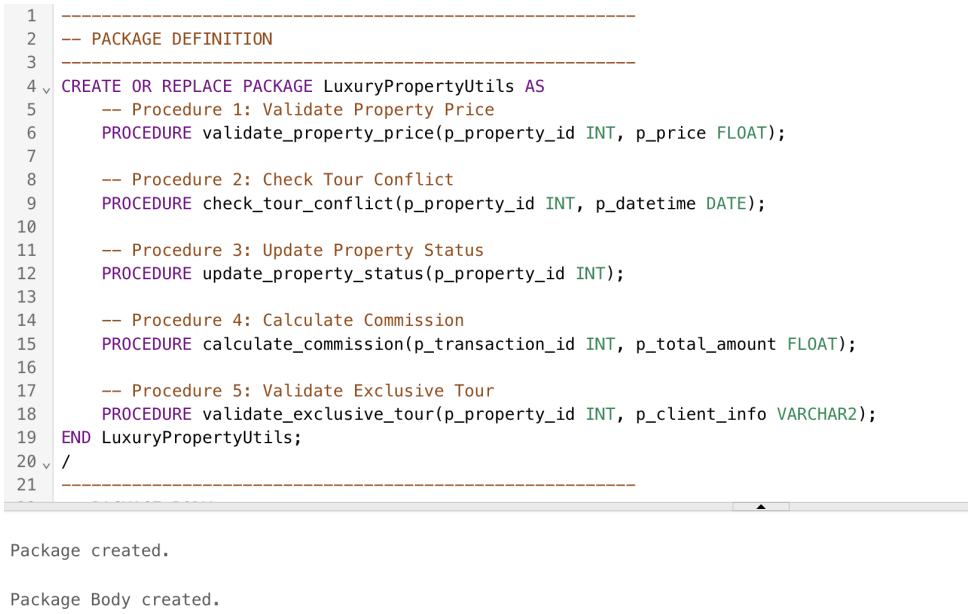
```
    -- Procedure 3: Update Property Status
    ----------------------------------------------------
    PROCEDURE update_property_status(p_property_id INT) IS
    BEGIN
        UPDATE Property
        SET Status = 'Sold'
        WHERE ID = p_property_id;
    END update_property_status;

    ----------------------------------------------------
    -- Procedure 4: Calculate Commission
    ----------------------------------------------------
    PROCEDURE calculate_commission(p_transaction_id INT,
       p_total_amount FLOAT) IS
        v_commission FLOAT;
    BEGIN
        IF p_total_amount > 0 THEN
            v_commission := 5000 + (p_total_amount * 0.05);
            UPDATE Transaction
            SET Commission = v_commission
            WHERE ID = p_transaction_id;
        ELSE
            RAISE_APPLICATION_ERROR(-20003, 'Total amount must be
                greater than zero.');
        END IF;
    END calculate_commission;

    ----------------------------------------------------
    -- Procedure 5: Validate Exclusive Tour
    ----------------------------------------------------
    PROCEDURE validate_exclusive_tour(p_property_id INT,
       p_client_info VARCHAR2) IS
        v_exclusivity VARCHAR2(50);
    BEGIN
        SELECT ExclusivityTier
        INTO v_exclusivity
        FROM Property
        WHERE ID = p_property_id;

        IF v_exclusivity != 'Premium'
           AND p_client_info LIKE '%VIP%' THEN
            RAISE_APPLICATION_ERROR(-20004,
                'Only premium properties can have VIP tours.');
        END IF;
    END validate_exclusive_tour;

END LuxuryPropertyUtils;
/
```



```
------------------------------------------------------------
-- PACKAGE DEFINITION
------------------------------------------------------------
CREATE OR REPLACE PACKAGE LuxuryPropertyUtils AS
    -- Procedure 1: Validate Property Price
    PROCEDURE validate_property_price(p_property_id INT, p_price FLOAT);

    -- Procedure 2: Check Tour Conflict
    PROCEDURE check_tour_conflict(p_property_id INT, p_datetime DATE);

    -- Procedure 3: Update Property Status
    PROCEDURE update_property_status(p_property_id INT);

    -- Procedure 4: Calculate Commission
    PROCEDURE calculate_commission(p_transaction_id INT, p_total_amount FLOAT);

    -- Procedure 5: Validate Exclusive Tour
    PROCEDURE validate_exclusive_tour(p_property_id INT, p_client_info VARCHAR2);
END LuxuryPropertyUtils;
/
------------------------------------------------------------
```

```
Package created.

Package Body created.
```

FIGURE 4 – Capture d'écran : Package créé avec succès

5.2 Example Usage of Procedures

Below is an example of how to call these procedures from an anonymous PL/SQL block, with `DBMS_OUTPUT.PUT_LINE` messages :

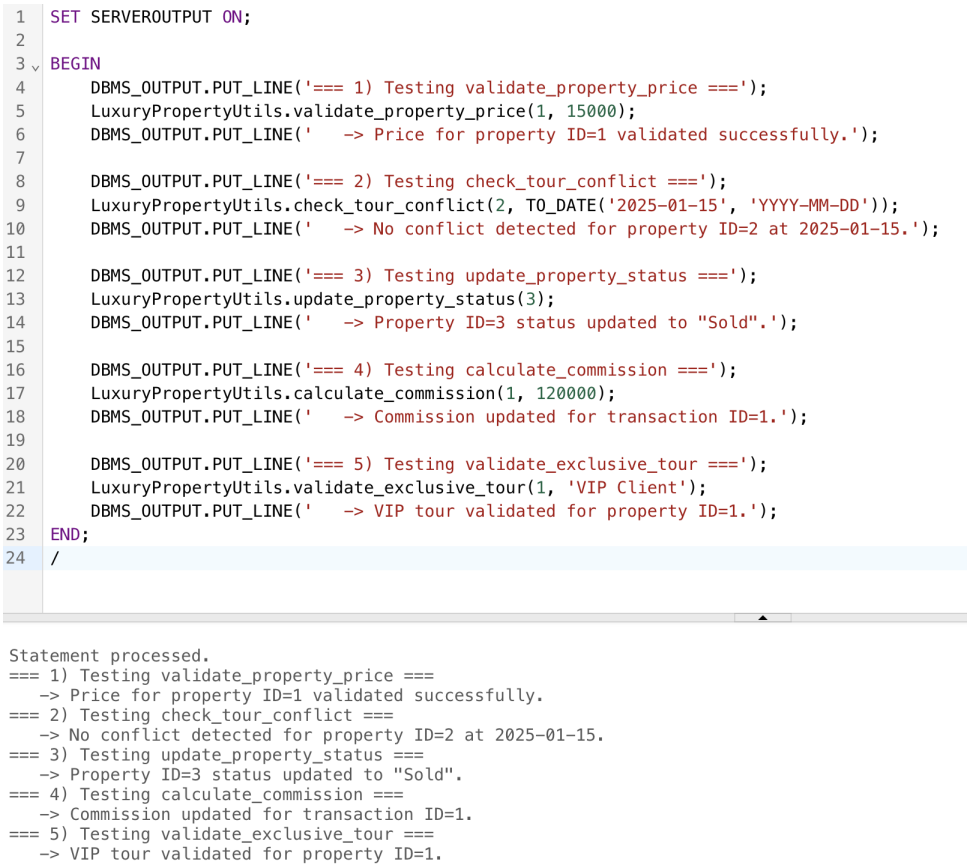
```
BEGIN
    DBMS_OUTPUT.PUT_LINE('=== 1) Testing validate_property_price
       ===');
    LuxuryPropertyUtils.validate_property_price(1, 9000);
    DBMS_OUTPUT.PUT_LINE('   -> Price for property ID=1 validated
        successfully.');

    DBMS_OUTPUT.PUT_LINE('=== 2) Testing check_tour_conflict ===')
       ;
    LuxuryPropertyUtils.check_tour_conflict(2, TO_DATE('
       2025-01-15','YYYY-MM-DD'));
    DBMS_OUTPUT.PUT_LINE('   -> No conflict detected for property
        ID=2 at 2025-01-15.');

    DBMS_OUTPUT.PUT_LINE('=== 3) Testing update_property_status
       ===');
    LuxuryPropertyUtils.update_property_status(3);
    DBMS_OUTPUT.PUT_LINE('   -> Property ID=3 status updated to "
       Sold".');

    DBMS_OUTPUT.PUT_LINE('=== 4) Testing calculate_commission ===
       ');
    LuxuryPropertyUtils.calculate_commission(1, 120000);
    DBMS_OUTPUT.PUT_LINE('   -> Commission updated for
       transaction ID=1.');
```

```
    DBMS_OUTPUT.PUT_LINE('=== 5) Testing validate_exclusive_tour
        ===');
    LuxuryPropertyUtils.validate_exclusive_tour(1, 'VIP Client');
    DBMS_OUTPUT.PUT_LINE('   -> VIP tour validated for property
        ID=1.');
END;
/
```



```
SET SERVEROUTPUT ON;

BEGIN
    DBMS_OUTPUT.PUT_LINE('=== 1) Testing validate_property_price ===');
    LuxuryPropertyUtils.validate_property_price(1, 15000);
    DBMS_OUTPUT.PUT_LINE('   -> Price for property ID=1 validated successfully.');

    DBMS_OUTPUT.PUT_LINE('=== 2) Testing check_tour_conflict ===');
    LuxuryPropertyUtils.check_tour_conflict(2, TO_DATE('2025-01-15', 'YYYY-MM-DD'));
    DBMS_OUTPUT.PUT_LINE('   -> No conflict detected for property ID=2 at 2025-01-15.');

    DBMS_OUTPUT.PUT_LINE('=== 3) Testing update_property_status ===');
    LuxuryPropertyUtils.update_property_status(3);
    DBMS_OUTPUT.PUT_LINE('   -> Property ID=3 status updated to "Sold".');

    DBMS_OUTPUT.PUT_LINE('=== 4) Testing calculate_commission ===');
    LuxuryPropertyUtils.calculate_commission(1, 120000);
    DBMS_OUTPUT.PUT_LINE('   -> Commission updated for transaction ID=1.');

    DBMS_OUTPUT.PUT_LINE('=== 5) Testing validate_exclusive_tour ===');
    LuxuryPropertyUtils.validate_exclusive_tour(1, 'VIP Client');
    DBMS_OUTPUT.PUT_LINE('   -> VIP tour validated for property ID=1.');
END;
/
```

```
Statement processed.
=== 1) Testing validate_property_price ===
   -> Price for property ID=1 validated successfully.
=== 2) Testing check_tour_conflict ===
   -> No conflict detected for property ID=2 at 2025-01-15.
=== 3) Testing update_property_status ===
   -> Property ID=3 status updated to "Sold".
=== 4) Testing calculate_commission ===
   -> Commission updated for transaction ID=1.
=== 5) Testing validate_exclusive_tour ===
   -> VIP tour validated for property ID=1.
```

FIGURE 5 – Capture d'écran : Exécution des procédures PL/SQL

5.3 Procedures vs. Triggers : Why Choose Procedures ?

In this project, we implemented a total of five PL/SQL procedures, grouped into a package called `LuxuryPropertyUtils`. Instead of relying on triggers, we chose procedures for several reasons :

- **Explicit Invocation :** Procedures must be explicitly called by the application or user, granting full control over their execution flow. Triggers fire automatically on table events, which can lead to unexpected behavior if not carefully managed.
- **Debugging and Testing :** Procedures can be tested independently with defined input parameters, making debugging more straightforward. Triggers are tied to table events, complicating their test scenarios.
- **Performance Optimization :** Procedures only run when needed, avoiding the overhead triggers might introduce if they fire frequently or if multiple triggers exist for the same event.

— **Reusability :** Procedures allow code to be reused in different parts of the application, improving modularity and consistency. Triggers remain bound to a specific table and event.
— **Predictable Execution Flow :** Procedures offer a clear, predictable flow because the logic is executed upon a direct call rather than implicitly through a table operation.

Overall, using procedures promotes better maintainability, reusability, and performance compared to triggers for the business logic implemented in this system.

5.4 Advanced Procedures

This section introduces advanced procedures designed to tackle complex scenarios and automate intricate tasks within the luxury property management system. These procedures enhance the functionality of the database by enabling dynamic operations and in-depth analysis.

5.4.1 1. Validate Multi-Day Tour

This procedure ensures that a property can be booked for multiple consecutive days without conflicts with existing reservations.

```
PROCEDURE validate_multi_day_tour(
    p_property_id INT,
    p_start_date DATE,
    p_end_date DATE
) IS
    v_count INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO v_count
    FROM Tour
    WHERE PropertyID = p_property_id
      AND (DateTime BETWEEN p_start_date AND p_end_date);

    IF v_count > 0 THEN
        RAISE_APPLICATION_ERROR(
            -20005,
            'The property has conflicts with existing tours
               within the selected date range.'
        );
    END IF;
END validate_multi_day_tour;
```

5.4.2 2. Calculate Total Revenue per Property

This procedure calculates the total revenue generated by a specific property based on completed transactions.

```
PROCEDURE calculate_total_revenue(
    p_property_id INT,
    p_total_revenue OUT FLOAT
```

```
) IS
BEGIN
    SELECT SUM(TotalAmount)
    INTO p_total_revenue
    FROM Transaction
    WHERE PropertyID = p_property_id;

    IF p_total_revenue IS NULL THEN
        p_total_revenue := 0;
    END IF;
END calculate_total_revenue;
```

5.4.3 3. Assign Dynamic Commission Rate

This procedure dynamically assigns a commission rate based on the transaction amount, providing flexibility in commission calculations.

```
PROCEDURE assign_dynamic_commission_rate(
    p_transaction_id INT,
    p_total_amount FLOAT
) IS
    v_commission_rate FLOAT;
BEGIN
    IF p_total_amount >= 10000000 THEN
        v_commission_rate := 0.03; -- 3% for high-value
           properties
    ELSIF p_total_amount >= 5000000 THEN
        v_commission_rate := 0.05; -- 5% for mid-range properties
    ELSE
        v_commission_rate := 0.07; -- 7% for low-value properties
    END IF;

    UPDATE Transaction
    SET Commission = 5000 + (p_total_amount * v_commission_rate)
    WHERE ID = p_transaction_id;
END assign_dynamic_commission_rate;
```

5.4.4 4. Generate Monthly Performance Report

This procedure generates a monthly report detailing the number of sold properties and total revenue for a given month and year.

```
PROCEDURE generate_monthly_report(
    p_month IN NUMBER,
    p_year IN NUMBER,
    p_sold_properties OUT NUMBER,
    p_total_revenue OUT FLOAT
) IS
BEGIN
    -- Count sold properties
    SELECT COUNT(*)
    INTO p_sold_properties
```

```
    FROM Property
    WHERE Status = 'Sold'
      AND EXTRACT(MONTH FROM AvailabilityDate) = p_month
      AND EXTRACT(YEAR FROM AvailabilityDate) = p_year;

    -- Calculate total revenue
    SELECT SUM(TotalAmount)
    INTO p_total_revenue
    FROM Transaction
    WHERE EXTRACT(MONTH FROM TransactionDate) = p_month
      AND EXTRACT(YEAR FROM TransactionDate) = p_year;

    IF p_total_revenue IS NULL THEN
        p_total_revenue := 0;
    END IF;
END generate_monthly_report;
```

5.4.5 5. Update Property Status Based on Duration

This procedure automatically updates the status of a property to "Expired" if it remains "Available" for more than 90 days.

```
PROCEDURE update_status_based_on_duration(
    p_property_id INT
) IS
    v_availability_date DATE;
BEGIN
    SELECT AvailabilityDate
    INTO v_availability_date
    FROM Property
    WHERE ID = p_property_id;

    IF v_availability_date < SYSDATE - 90 THEN
        UPDATE Property
        SET Status = 'Expired'
        WHERE ID = p_property_id;
    END IF;
END update_status_based_on_duration;
```

5.4.6 Conclusion

These advanced procedures significantly enhance the database's capabilities, enabling the automation of complex tasks and providing insights critical to the efficient management of luxury properties.

6 Object-Relational Database (ORD)

This section presents the implementation of object-relational features for the Luxury Property Management System. Types are defined to model complex data structures, and inheritance is used to represent hierarchical relationships.

6.1 Custom Types

AddressType : Defines the structure of property addresses.

```
CREATE OR REPLACE TYPE AddressType AS OBJECT (
    Street VARCHAR(255),
    City VARCHAR(100),
    Neighborhood VARCHAR(100),
    PostalCode VARCHAR(20)
);
```

PropertyType : Represents a property with its attributes, including address and status.

```
CREATE OR REPLACE TYPE PropertyType AS OBJECT (
    ID INT,
    Address AddressType,
    Rooms INT,
    LivingArea FLOAT,
    LotSize FLOAT,
    Condition VARCHAR(50),
    Price FLOAT,
    Status VARCHAR(50),
    AvailabilityDate DATE,
    ExclusivityTier VARCHAR(50)
);
```

6.2 Inheritance

Inheritance is used to differentiate between property types such as mansions and apartments.

BaseProperty : A parent type for properties.

```
CREATE OR REPLACE TYPE BaseProperty AS OBJECT (
    ID INT,
    Price FLOAT
) NOT FINAL;
```

Mansion : A child type with additional attributes for mansions.

```
CREATE OR REPLACE TYPE Mansion UNDER BaseProperty (
    SwimmingPool BOOLEAN,
    GardenArea FLOAT
);
```

Apartment : A child type with additional attributes for apartments.

```
CREATE OR REPLACE TYPE Apartment UNDER BaseProperty (
    FloorNumber INT,
    Elevator BOOLEAN
);
```

6.3 Tables and Data

```sql
-- Creating tables based on types
CREATE TABLE BaseProperties OF BaseProperty (
    PRIMARY KEY (ID)
);

CREATE TABLE Mansions OF Mansion (
    PRIMARY KEY (ID)
);

CREATE TABLE Apartments OF Apartment (
    PRIMARY KEY (ID)
);

-- Inserting example data
INSERT INTO Mansions VALUES (
    1, -- ID
    1500000, -- Price
    TRUE, -- SwimmingPool
    200.5 -- GardenArea
);

INSERT INTO Apartments VALUES (
    2, -- ID
    500000, -- Price
    10, -- FloorNumber
    TRUE -- Elevator
);
```

6.4 Conclusion

The use of object-relational features simplifies the representation of complex and hierarchical data, making it more flexible and maintainable.

7 Normalization

Normalization is a critical aspect of database design that reduces redundancy and ensures data integrity. This section explains how our database design adheres to each normal form.

7.1 First Normal Form (1NF)

Requirement : All columns must contain atomic values, and each column should store a single type of data. **Explanation :** - In our database, each table stores atomic values. For example, the `Property` table has columns like `Address`, `Rooms`, and `Price`, where each field contains a single, indivisible value. - Multi-valued fields are avoided by creating separate tables, such as `PropertyFacility` to manage the many-to-many relationship between properties and facilities.

7.2 Second Normal Form (2NF)

Requirement : All non-key attributes must be fully dependent on the entire primary key. **Explanation :** - In the `Transaction` table, attributes like `TotalAmount` and `Commission` depend solely on the primary key (`ID`), and there are no partial dependencies. - Similarly, the `PropertyFacility` table ensures that the relationship between properties and facilities depends on the composite primary key (`PropertyID`, `FacilityID`).

7.3 Third Normal Form (3NF)

Requirement : There should be no transitive dependencies, meaning non-key attributes must not depend on other non-key attributes. **Explanation :** - For the `Property` table, attributes like `Condition` are directly related to the primary key (`ID`) and not dependent on other non-key attributes. - To avoid transitive dependencies, attributes such as `Condition` could be moved to a separate `PropertyCondition` table if more detailed descriptions are needed.

7.4 Boyce-Codd Normal Form (BCNF)

Requirement : Every determinant must be a candidate key. **Explanation :** - The `PropertyFacility` table uses a composite primary key (`PropertyID`, `FacilityID`), ensuring that all relationships are fully dependent on these keys. - Similarly, all tables ensure that no non-trivial functional dependencies exist other than those involving a candidate key.

7.5 Fourth Normal Form (4NF)

Requirement : Multi-valued dependencies must be eliminated. **Explanation :** - Multi-valued attributes like `Facilities` are separated into the `PropertyFacility` table, linking each property to its associated facilities. This prevents duplication of property data for each facility.

7.6 Fifth Normal Form (5NF)

Requirement : There should be no join dependencies, ensuring data integrity even after decomposition. **Explanation :** - The `Tour` table connects properties to clients through individual tours, ensuring that complex relationships can be reconstructed without redundancy or anomalies.

7.7 Conclusion

The database design adheres to the principles of normalization up to the 5NF, ensuring :

— Minimal redundancy through decomposition into smaller tables.
— Improved data integrity and consistency by eliminating anomalies.
— Enhanced scalability and ease of maintenance.

This robust design lays the foundation for efficient data management in the Luxury Property Management System.

8 NoSQL Databases

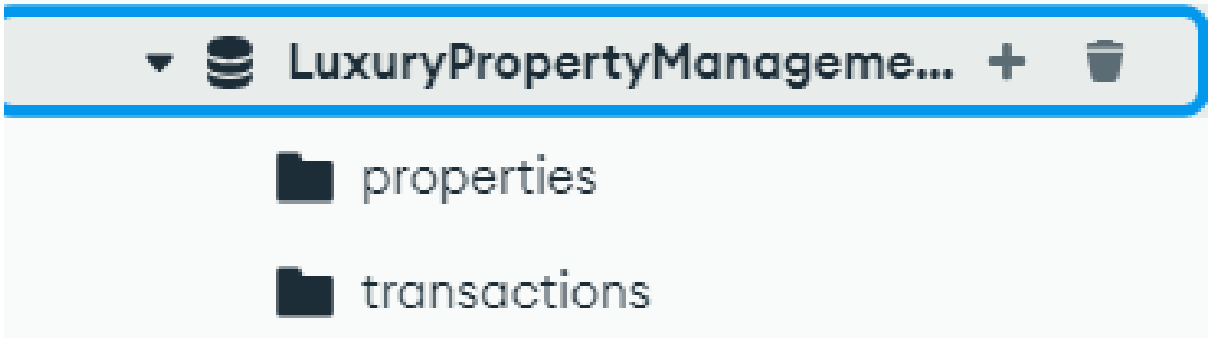
This section explores the use of NoSQL databases, specifically MongoDB for document-based data and Neo4J for graph-based relationships.

8.1 MongoDB

MongoDB is used to manage unstructured data such as multimedia property descriptions and client feedback.

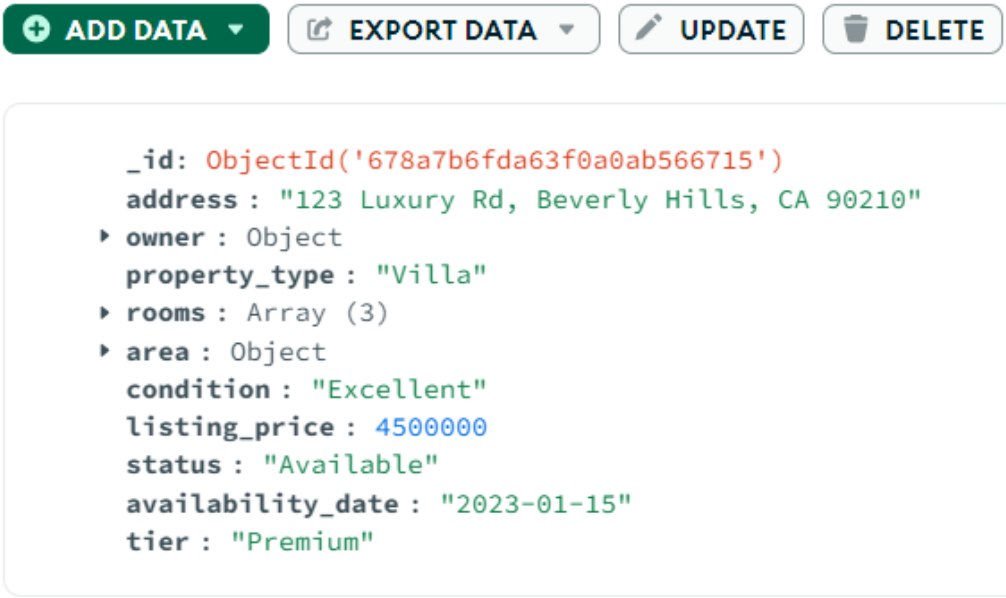
8.1.1 Database and Collection Creation

We created a database and two collections using MongoDB Compass. Below is the structure of a document inserted into the `properties` collection in JSON format :



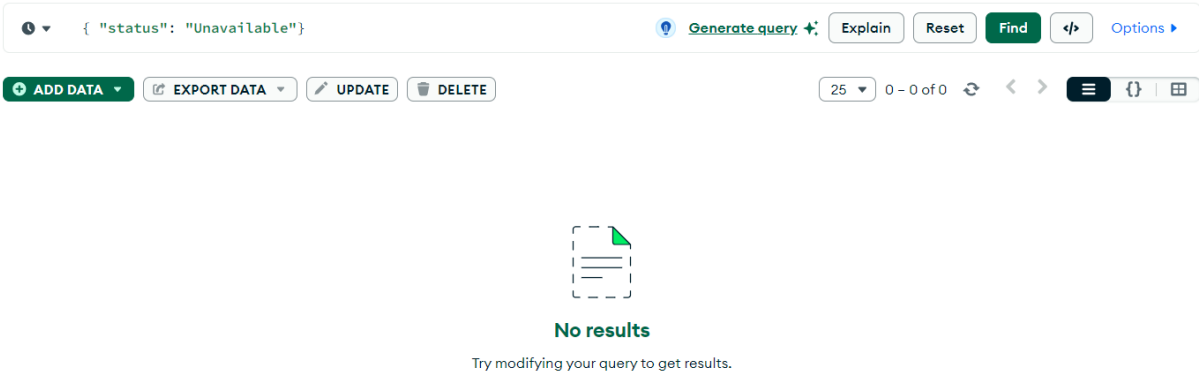
Listing 1 – Document structure in the `properties` collection

```
{
  "address": "123 Luxury Rd, Beverly Hills, CA 90210",
  "owner": {
    "name": "John Doe",
    "contact_info": "john.doe@example.com"
  },
  "property_type": "Villa",
  "rooms": [
    {"type": "bedroom", "number": 4},
    {"type": "living area", "number": 2},
    {"type": "study", "number": 1}
  ],
  "area": {"living_area": 3500, "lot_size": 5000},
  "condition": "Excellent",
  "listing_price": 4500000,
  "status": "Available",
  "availability_date": "2023-01-15",
  "tier": "Premium"
}
```

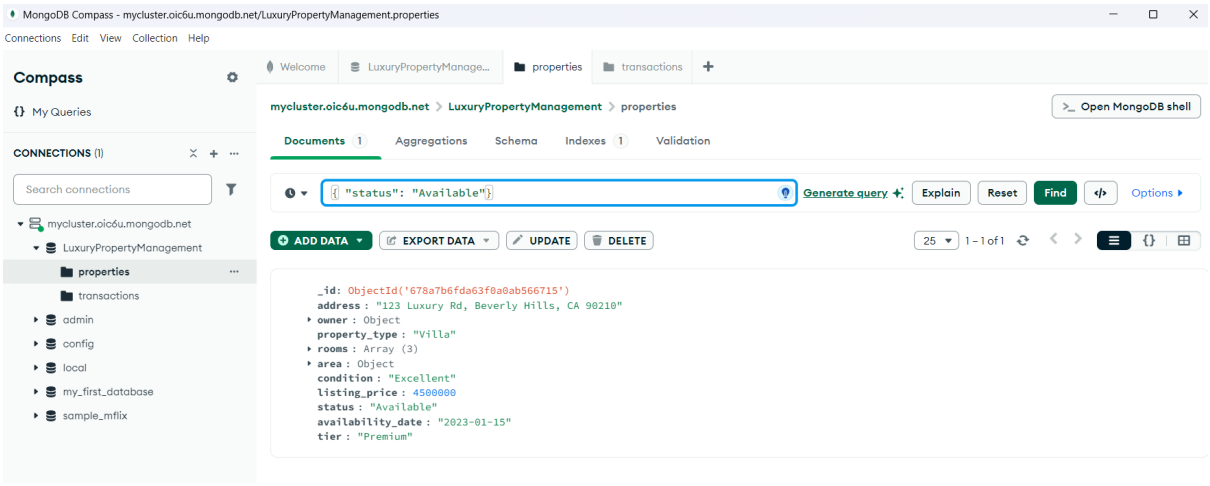


8.1.2 Using the Search Bar in MongoDB Compass

MongoDB Compass allows filtering of documents through its search bar.



For instance, searching for properties with a status of `Unavailable` yields no results (as only properties with `Available` status were inserted). Conversely, searching for `Available` properties works as expected.



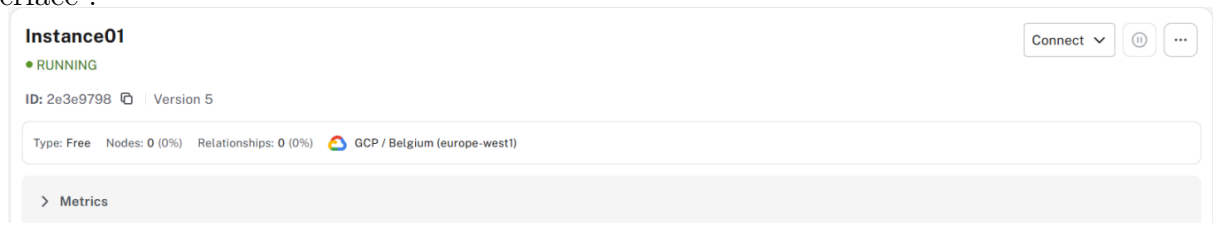
The Importance of MongoDB Compass

MongoDB Compass has been a key tool for building a robust database for our project. Using Compass, we created specific collections like `properties` and `transactions`, which are essential for managing detailed information about luxury properties.

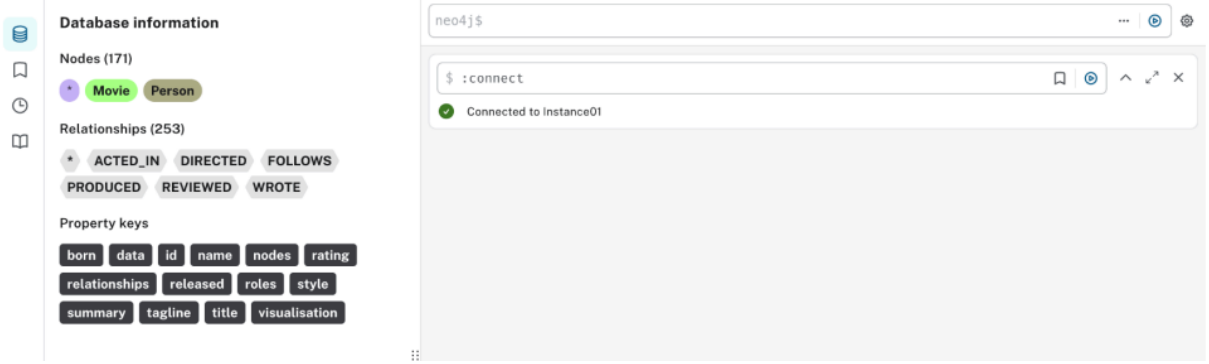
The graphical interface of MongoDB Compass made data visualization and manipulation intuitive, enabling efficient CRUD operations on properties and transactions. This streamlined the process of testing and optimizing queries, improving database performance. It not only laid the groundwork for a solid database structure but also provided an intuitive platform for data management and analysis, essential for making informed decisions in luxury property management.

8.2 Neo4J

Neo4J is used to represent and analyze relationships between entities such as properties and their owners. We created an instance to host our database using the following interface :



Next, we created our nodes in the following interface and verified that the relationships were correctly established by performing a few tests :



8.2.1 Creating and Testing Nodes

We created a client node and a property node to demonstrate the data structure. For example :

8.2.2 Graph Creation

```
// Create nodes for properties
CREATE (:Property {id: 1, type: "Mansion", price: 15000000,
   exclusivityTier: "Premium"});
CREATE (:Property {id: 2, type: "Apartment", price: 5000000,
   exclusivityTier: "Standard"});
```

So we created the following nodes :

FIGURE 6 – Nodes created in Neo4j.

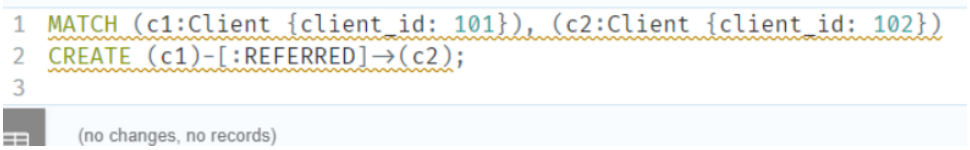


FIGURE 7 – Additional node example.

We then verified the connection between two nodes, illustrating the relationship between a client and a property. The `Client` node named `John Doe` is connected to a `Property` node described as a `Luxurious 3-bedroom apartment` via a relationship `INTERESTED_IN`. This visually demonstrates how data is linked in Neo4j, enabling intuitive analysis of relationships between entities in the luxury property management system. This graphical representation helps understand client preferences and interactions

8.2.3 Using Neo4j Aura for Luxury Property Management

Using Neo4j Aura for our luxury property management project allowed us to effectively model complex relationships between clients, owners, and properties. By structuring our data this way, we were able to analyze these interactions. This approach is particularly useful as it helps identify client preference trends and optimize property portfolio management strategies.

FIGURE 8 – Connection verification in Neo4j.

9 Conclusion

The Advanced Databases Project demonstrates the effective use of relational and NoSQL database technologies for a real-world luxury property management system. By implementing concepts such as normalization, PL/SQL procedures, object-relational features, and NoSQL approaches, the project achieves the following :

- Ensured data integrity through normalization and constraints.
- Automated critical operations such as commission calculations and property status updates using PL/SQL **procedures**.
- Enhanced data representation and flexibility using object-relational features.
- Explored the use of NoSQL technologies like MongoDB for document-based data and Neo4J for graph-based relationships, enabling advanced data analytics and efficient handling of complex relationships.

The comparison between relational and NoSQL approaches highlights the strengths and use cases of each technology. While relational databases provide robust data integrity and structured querying, NoSQL databases excel in scalability and handling unstructured or semi-structured data.

This project provides a foundation for future improvements, such as integrating machine learning models for property recommendations or leveraging cloud-based database solutions for scalability. The skills and insights gained through this project are invaluable for addressing complex database challenges in various domains.