EFREI

Advanced Databases Project

Luxury Property Management System

Samy BOUAISSA
Amine M'ZALI

16 janvier 2025

Table des matières

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 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.

2.1 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.

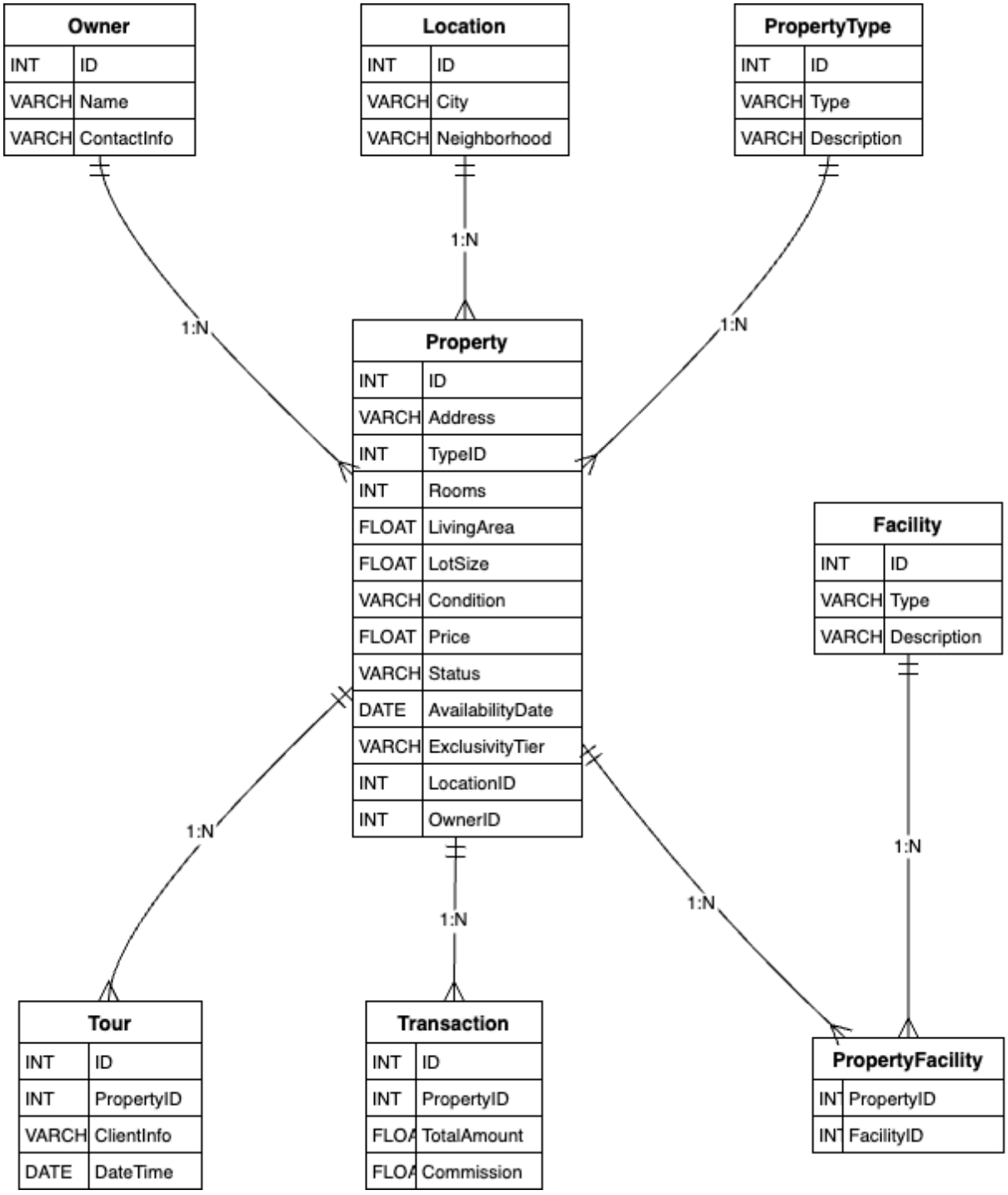


Figure 1 – E/R Diagram for Luxury Property Management System

3 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.

3.1 SQL Script

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

3.1.1 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 (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)
);
```

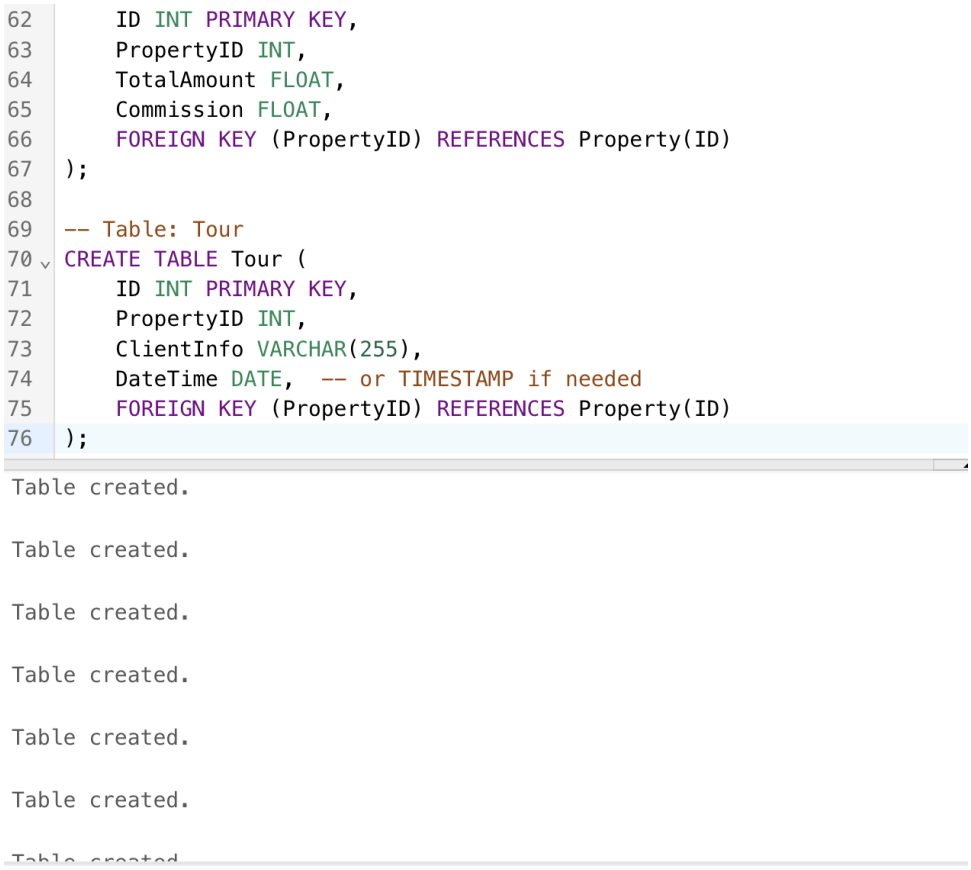


FIGURE 2 – Capture d'écran : Tables créées

3.1.2 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');
```

```sql
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')
);
```

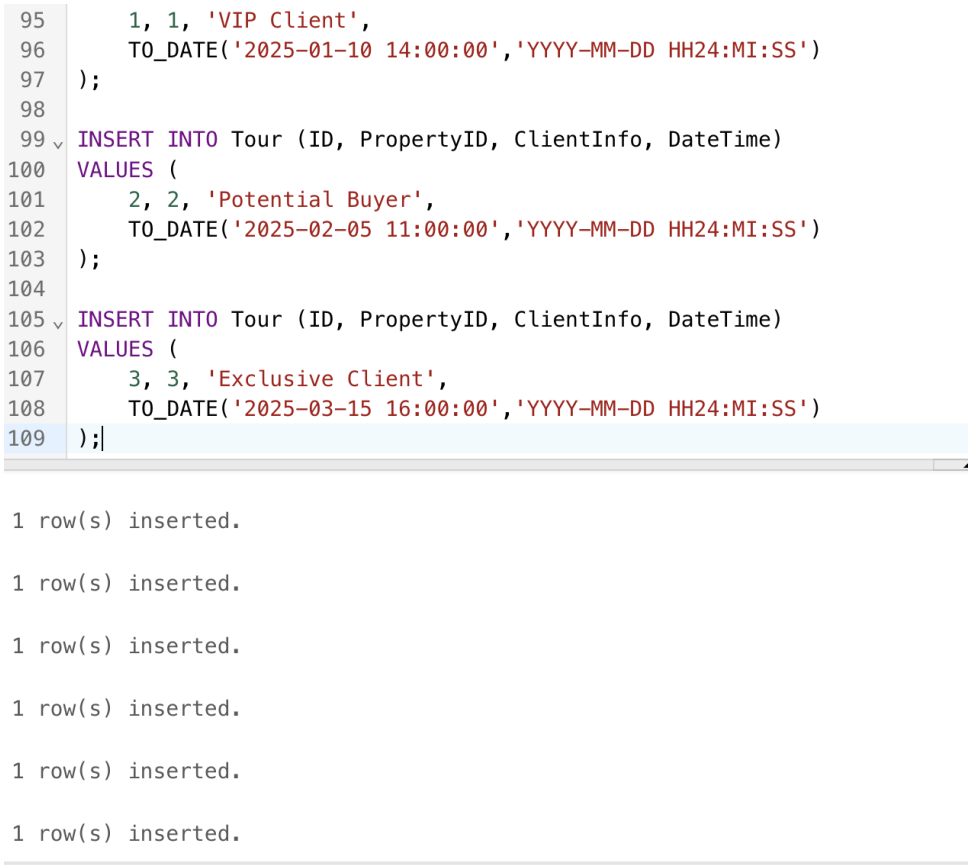


FIGURE 3 – Capture d'écran : Données insérées avec succès

4 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.

4.1 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;
/
```

```
1  ------------------------------------------------------------
2  -- PACKAGE DEFINITION
3  ------------------------------------------------------------
4  CREATE OR REPLACE PACKAGE LuxuryPropertyUtils AS
5      -- Procedure 1: Validate Property Price
6      PROCEDURE validate_property_price(p_property_id INT, p_price FLOAT);
7
8      -- Procedure 2: Check Tour Conflict
9      PROCEDURE check_tour_conflict(p_property_id INT, p_datetime DATE);
10
11     -- Procedure 3: Update Property Status
12     PROCEDURE update_property_status(p_property_id INT);
13
14     -- Procedure 4: Calculate Commission
15     PROCEDURE calculate_commission(p_transaction_id INT, p_total_amount FLOAT);
16
17     -- Procedure 5: Validate Exclusive Tour
18     PROCEDURE validate_exclusive_tour(p_property_id INT, p_client_info VARCHAR2);
19 END LuxuryPropertyUtils;
20 /
21 ------------------------------------------------------------
```

```
Package created.

Package Body created.
```

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

4.2 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 :

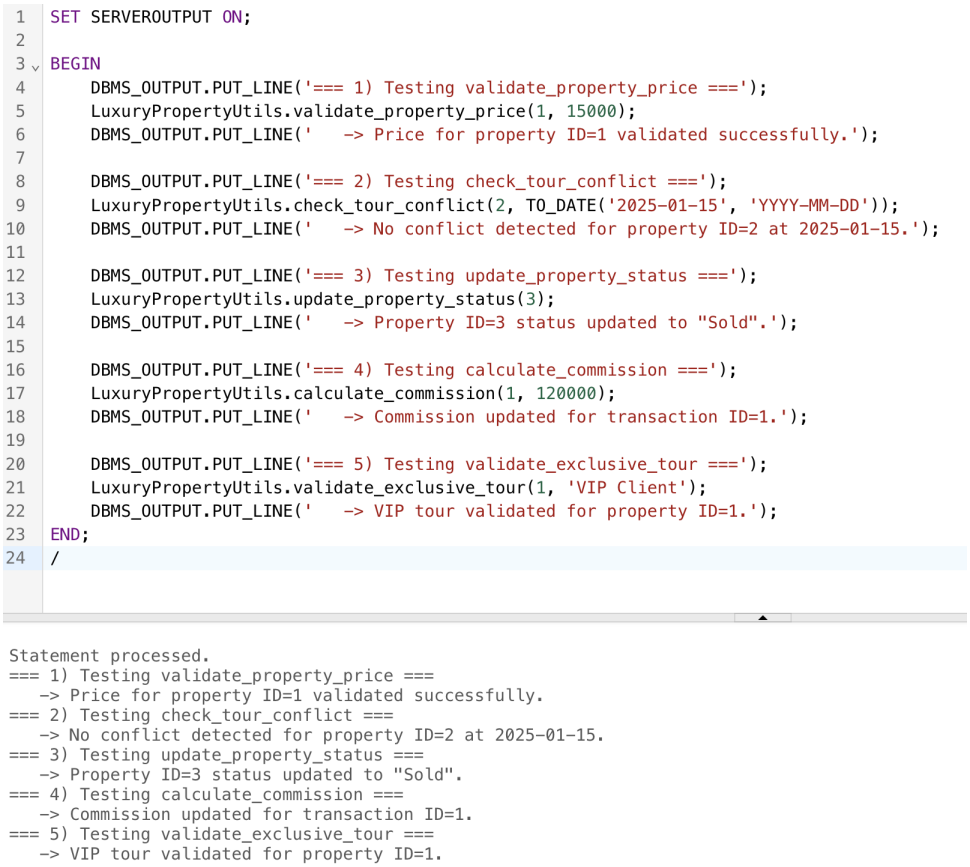
```
BEGIN
    DBMS_OUTPUT.PUT_LINE('=== 1) Testing validate_property_price
       ===');
    LuxuryPropertyUtils.validate_property_price(1, 9000);
    DBMS_OUTPUT.PUT_LINE('   -> Price for property ID=1 validated
        successfully.');

    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 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.

5.1 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)
);
```

5.2 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
);
```

5.3 Tables and Data

```
-- 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
);
```

5.4 Conclusion

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

6 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.

6.1 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.

6.2 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‘).

6.3 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.

6.4 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.

6.5 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.

6.6 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.

6.7 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.

7 NoSQL Databases

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

7.1 MongoDB

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

7.1.1 Database and Collection Creation

```
use LuxuryPropertyDB;

db.properties.insertMany([
    {
        "propertyID": 1,
        "address": {
            "street": "123 Luxury St",
            "city": "Paris",
            "neighborhood": "Champs-  lyses  ",
            "postalCode": "75008"
        },
        ...
    }
]);
```

7.1.2 Example Queries

```
// Find all premium properties
db.properties.find({ exclusivityTier: "Premium" });

// Find properties with a pool
db.properties.find({ facilities: "Pool" });
```

7.2 Neo4J

Neo4J is used to represent and analyze relationships between entities such as properties and their owners.

7.2.1 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"});
```

7.2.2 Example Queries

```
// Find all premium properties
MATCH (p:Property {exclusivityTier: "Premium"})
RETURN p;
```

```
// Find the owner of a property
MATCH (o:Owner)-[:OWNS]->(p:Property {id: 1})
RETURN o, p;
```

8 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.