EST Project Report Submitted for

DATABASE MANAGEMENT SYSTEM

Submitted by:

Keshav Bansal	(102203615)
Bhavya Goyal	(102203638)
Somya Bansal	(102203682)
Rohit Deepchandani	(102203986)

Submitted to:

Dr. Sanjeev Rao

Subgroup : 2CO29

Topic : Real Estate Management System

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Department of Computer Science and Engineering TIET, Patiala

Jan-May 2024

Index

Introduction

The real estate project under consideration seeks to address the challenges faced by real estate agencies, property managers, and individual property owners in managing their portfolios effectively. By leveraging the capabilities of modern technology, the project aims to provide a comprehensive solution that facilitates seamless management of real estate properties, from acquisition to disposition.

Scope of the Project

The project encompasses the development and deployment of a custom-built DBMS tailored to the specific requirements of the real estate industry. Key functionalities include:

- Property Management: Tracking property details, ownership history, rental agreements, and maintenance records.
- Client Relationship Management (CRM): Managing client profiles, preferences, inquiries, and communications.
- Financial Transactions: Recording sales, purchases, leases, rental payments, and expenses.

Significance of Using a DBMS

Utilising a DBMS in the context of real estate management offers several advantages, including:

- Data Centralisation : Consolidating disparate data sources into a single unified database for easy access and retrieval.
- Data Integrity : Enforcing data integrity constraints and validation rules to maintain the accuracy and consistency of information.
- Scalability : Accommodating the growing volume of real estate data and adapting to changing business needs.
- Security : Implementing role-based access control and encryption techniques to protect sensitive data from unauthorised access and breaches.
- Efficiency: Streamlining business processes, reducing redundancy, and improving overall operational efficiency.

Requirement Analysis

The Real Estate Management System is designed to address the challenges faced by real estate agencies, property managers, and individual property owners in managing their portfolios effectively. Leveraging modern technology, the system aims to provide a comprehensive solution facilitating seamless management of real estate properties from acquisition to disposition.

Functional Requirements:

1. Property Table:
   The core of the real estate DBMS is the Property table, which stores detailed information about individual properties such as houses, apartments, commercial buildings, or land parcels. Each property entry includes attributes like address, type, size, amenities, and geographic information.

2. Transaction Table:
   Transactions are recorded in the Transaction table, documenting real estate transactions such as property sales or rentals. Each transaction entry includes details like transaction price, date, associated listing ID, client ID, and agent ID.

3. Client Table:
   The Client table stores information about clients interested in buying, selling, or renting properties. Client records include attributes like name, contact information, preferences, and transaction history.

4. Agent Table:
   Real estate agents are managed in the Agent table, which contains details about individual agents such as name, contact information, license number, and associated agency ID.

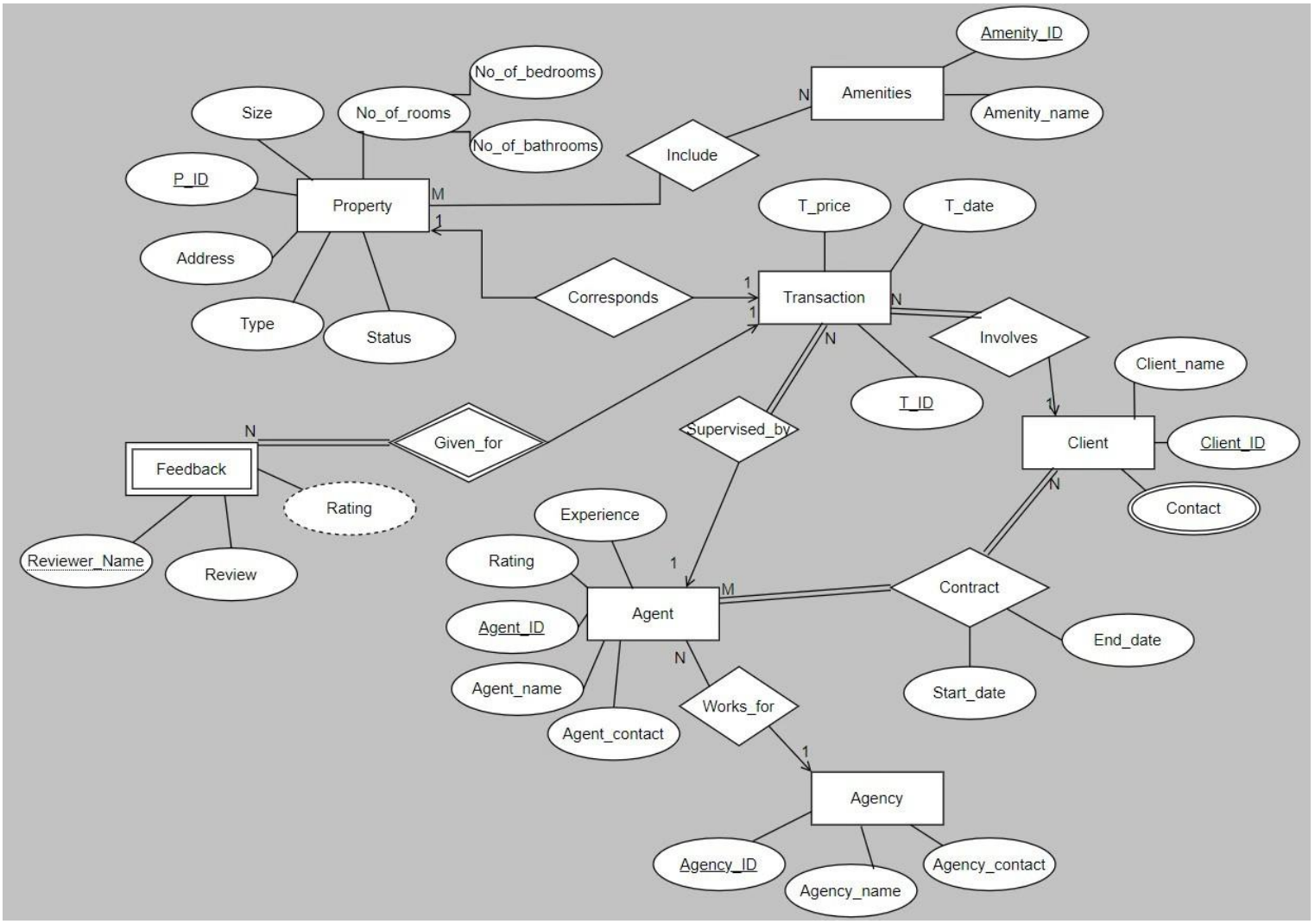
5. Agency Table:
   The Agency table tracks information about real estate agencies or brokerages, including agency name, address, contact details, and license information.

6. Contract Table:
   Contracts governing real estate transactions are stored in the Contract table, capturing details such as contract type, terms, start date, and end date. Each contract entry is associated with one or more transactions.

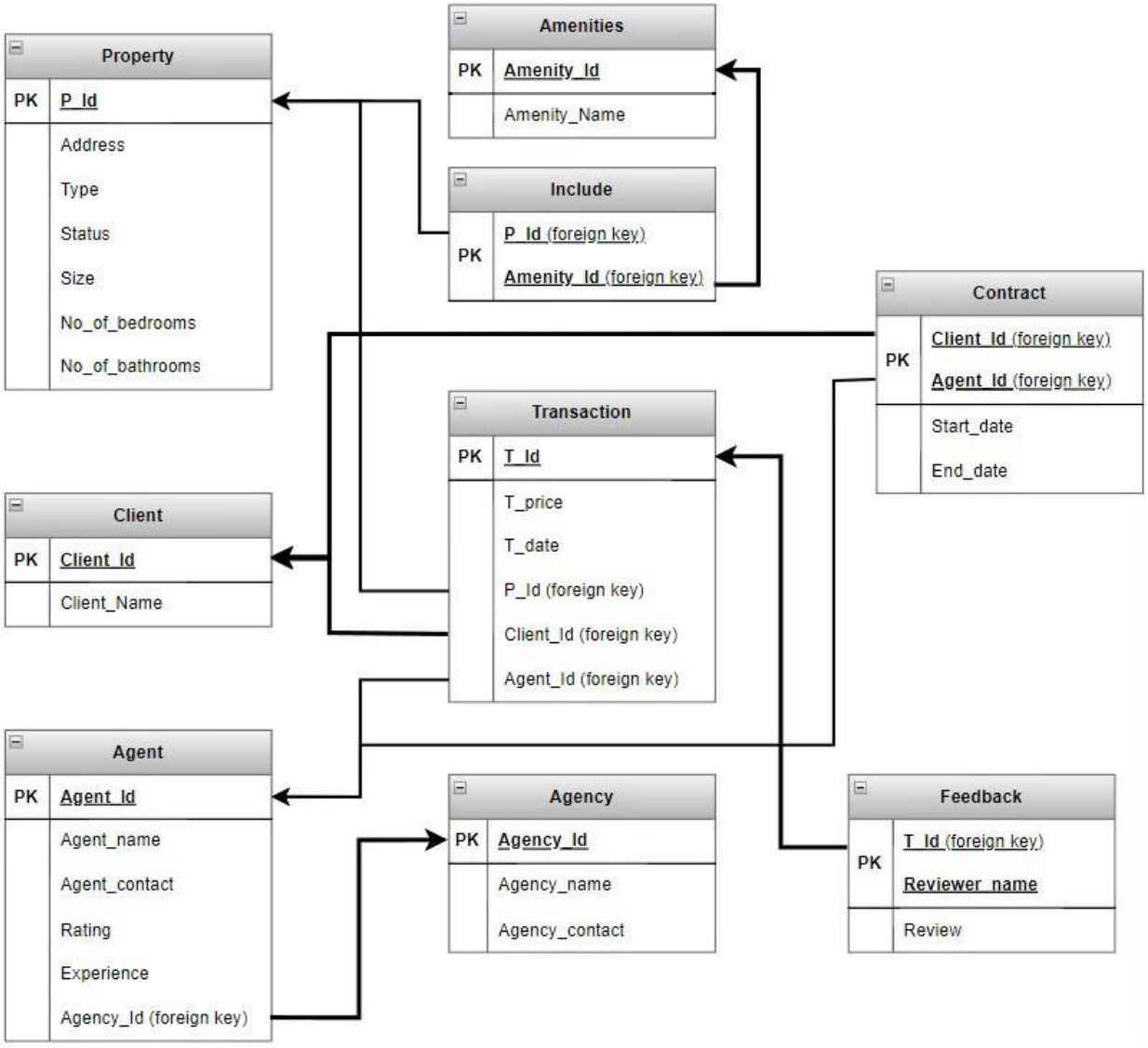
7. Amenities Table:
   Property amenities are catalogued in this table, like malls, swimming pool, cinema, market place, etc.

8. Feedback Table:
   Feedback provided by clients or agents regarding property transactions is recorded in the Feedback table. Each feedback entry includes text feedback, rating, and associated transaction ID.

ER Table


Amenity_ID
Amenities
Amenity_name
No_of_bedrooms
Size
No_of_rooms
No_of_bathrooms
Include
P_ID
Property
Address
Type
Status
T_price
T_date
Corresponds
Transaction
Involves
T_ID
Client_name
Client
Client_ID
Contact
Supervised_by
Given_for
Feedback
Rating
Reviewer_Name
Review
Experience
Rating
Agent
Agent_ID
Agent_name
Agent_contact
Contract
End_date
Start_date
Works_for
Agency
Agency_ID
Agency_name
Agency_contact

ER to Table (Before Normalisation)



Normalisation

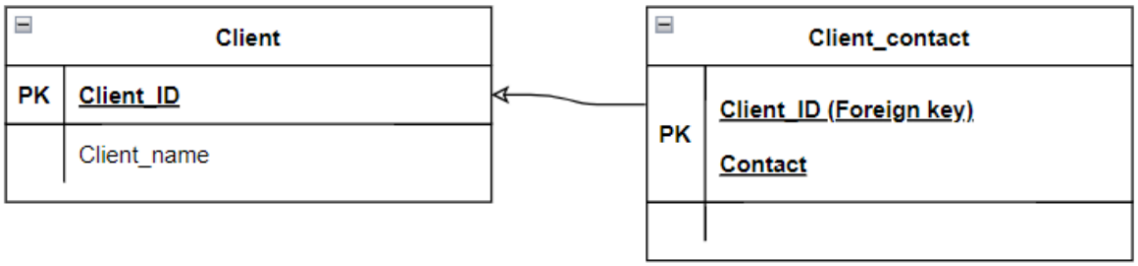
DBMS Normalisation is a systematic approach to decompose tables to eliminate data redunduency and undesirable characteristics like insertion, update and delete anomaly DBMS.

First Normal Form (1NF)

A Table is in 1NF if:

- Each column of table should be single valued
- Attribute domain should not change!
- Each column in table should have a unique name
- Order doesn't matter of data in table.

There is a multivalued attribute Contact in client table (as seen in ER diagram). So, we will break client table into two parts **Client** and **Client contact**,



Now tables are in 1NF.

Second Normal Form (2NF)

A Table is in 2NF if:
- The table should be in INF.
- There should be no Partial Dependency.

Partial dependency exists, when for a composite primary key, any, attribute in the table depends only on a part of primary by and not on complete primary key.

{Agent_ID,Client_ID} → **{Start_Date,End_Date}**

{Transaction_ID,Feedback_given} → **{Review}**

Third Normal Form (3NF)

A table is in 3NF if:

- It should be in 2NF.
- It should not have Transitive Dependancy.

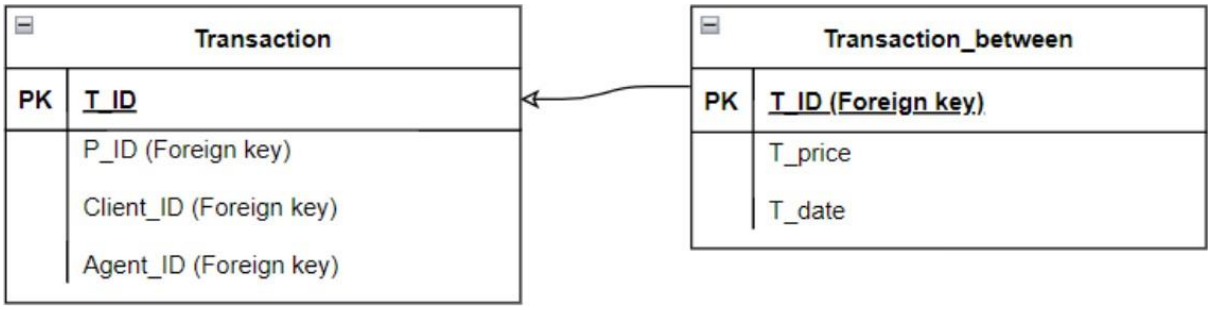
We see that there is a transitive dependancy in the table Transaction

T_ID → {P_ID,Client_ID,Agent_ID}

{P_ID,Client_ID,Agent_ID} → {T_Price,T_Date}

T_ID → {T_Price,T_Date}

Hence we decompose transaction into two tables:



BCNF Normal Form

A table is in BCNF if:

- It should be in the 3NF.
- And, for any dependency A → B, A should be a super key.

For a table in our RDBMS, for a dependency A → B, A is not a non-prime attribute, if B is a prime attribute. And our tables are already in 3NF.

We have our database schema already normalized to the BCNF Normal Form

Fourth Normal Form (4NF)

A table is in 4NF if:

- It should be in the Boyce-Codd Normal Form.
- And, the table should not have any Multi-valued Dependency.

For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.

We have our database schema already normalized to the 4NF Normal Form

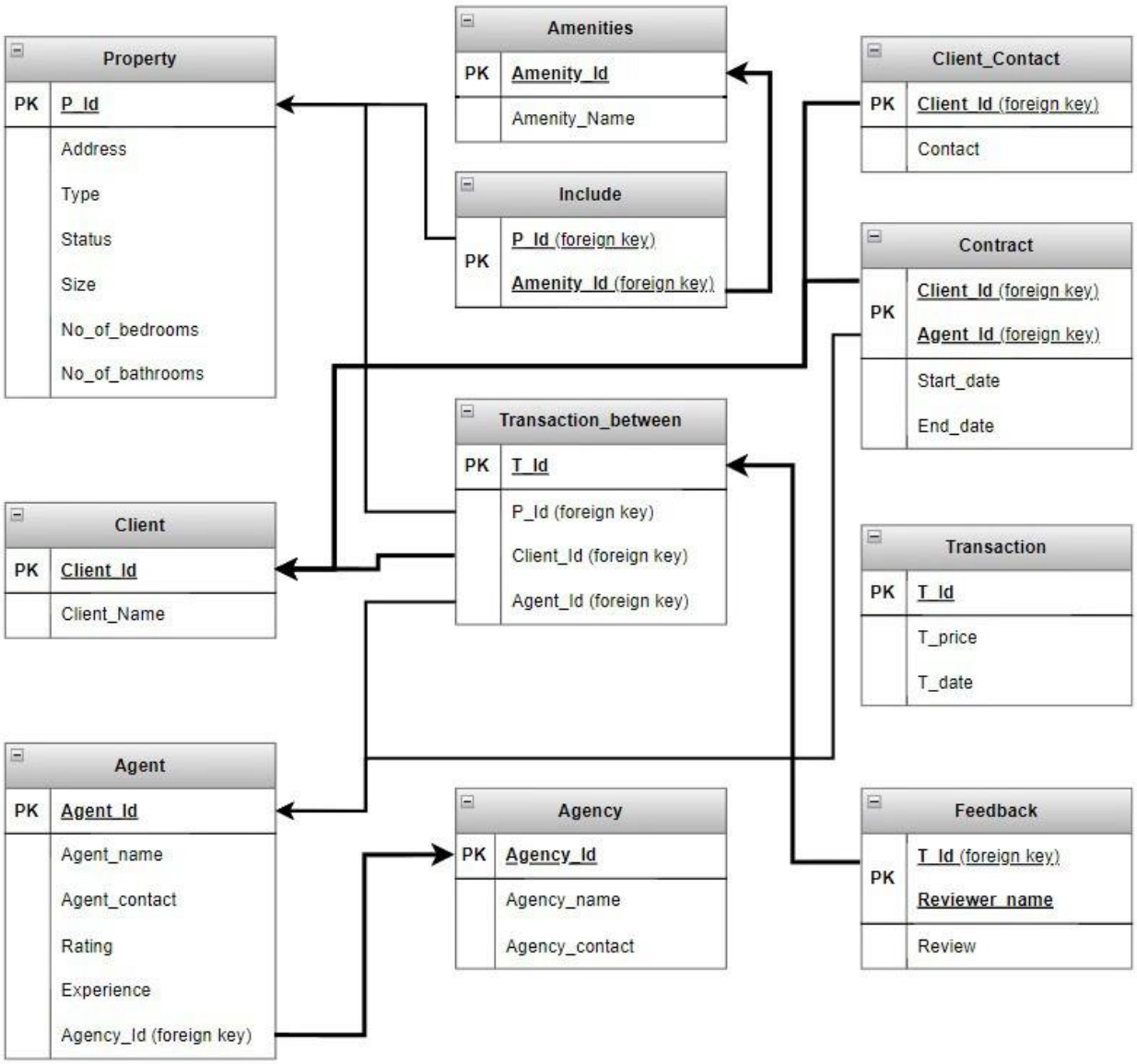
Fifth Normal Form (5NF)

A table is in 5NF if:

- It should be in the 4NF.
- And, the table cannot have a lossless decomposition in to any number of smaller tables (relations).

We have our database schema already normalized to the 5NF Normal Form

ER to Table (After Normalisation)



SQL Commands

Table Creation

```
create table property(
    p_id number(3) primary key,
    address varchar(30) not null unique,
    p_type varchar(30) not null check (p_type in ('residential','commercial','land
parcels')) ,
    status varchar(30) not null check (status in ('available','under contract','sold')) ,
    p_size varchar(20) not null,
    no_of_bed number(2) ,
    no_of_bathroom number(2)
);
```

TABLE PROPERTY

Column	Null?	Type
P_ID	NOT NULL	NUMBER(3,0)
ADDRESS	NOT NULL	VARCHAR2(30)
P_TYPE	NOT NULL	VARCHAR2(30)
STATUS	NOT NULL	VARCHAR2(30)
P_SIZE	NOT NULL	VARCHAR2(20)
NO_OF_BED	-	NUMBER(2,0)
NO_OF_BATHROOM	-	NUMBER(2,0)

```
create table agency(
    agency_id number(3) primary key ,
    agency_name varchar(20) not null,
    agency_contact varchar(20) not null   --email address
);
```

TABLE AGENCY

Column	Null?	Type
AGENCY_ID	NOT NULL	NUMBER(3,0)
AGENCY_NAME	NOT NULL	VARCHAR2(20)
AGENCY_CONTACT	NOT NULL	VARCHAR2(20)

```
create table agent(
   agent_id number(3) primary key,
   agency_id number(3) ,
   agent_name varchar(20) not null,
   agent_contact number(10) not null unique check (length(agent_contact)=10) ,
   Experience_review_num number not null,
   Rating varchar(20) not null,
   foreign key (agency_id) references agency(agency_id) on delete cascade
);
```

TABLE AGENT

Column	Null?	Type
AGENT_ID	NOT NULL	NUMBER(3,0)
AGENCY_ID	-	NUMBER(3,0)
AGENT_NAME	NOT NULL	VARCHAR2(20)
AGENT_CONTACT	NOT NULL	NUMBER(10,0)
EXPERIENCE_REVIEW_NUM	NOT NULL	NUMBER
RATING	NOT NULL	VARCHAR2(20)

```
create table client(
   client_id number(3) primary key ,
   client_name varchar(20)
);
```

TABLE CLIENT

Column	Null?	Type
CLIENT_ID	NOT NULL	NUMBER(3,0)
CLIENT_NAME	-	VARCHAR2(20)

```
create table client_contact (
   client_id NUMBER(3),
   contact NUMBER(10) not null unique check (length(contact) = 10),
   foreign key (client_id) references client(client_id) on delete cascade,
   primary key (client_id, contact)
);
```

TABLE CLIENT_CONTACT

Column	Null?	Type
CLIENT_ID	NOT NULL	NUMBER(3,0)
CONTACT	NOT NULL	NUMBER(10,0)

```
create table contract(
    client_id number(3) ,
    agent_id number(3) ,
    start_date date ,
    end_date date ,
    foreign key (client_id) references client(client_id) on delete cascade ,
    primary key(client_id,agent_id)
);
```

TABLE CONTRACT

Column	Null?	Type
CLIENT_ID	NOT NULL	NUMBER(3,0)
AGENT_ID	NOT NULL	NUMBER(3,0)
START_DATE	-	DATE
END_DATE	-	DATE

```
create table amenities(
    amenity_id number(3) primary key ,
    amenity_name varchar(20)
);
```

TABLE AMENITIES

Column	Null?	Type
AMENITY_ID	NOT NULL	NUMBER(3,0)
AMENITY_NAME	-	VARCHAR2(20)

```
create table include(
    p_id number(3) ,
    amenity_id number(3) ,
    foreign key (p_id) references property(p_id) on delete cascade ,
    foreign key (amenity_id) references amenities(amenity_id) on delete cascade ,
    primary key(p_id,amenity_id)
);
```

TABLE INCLUDE

Column	Null?	Type
P_ID	NOT NULL	NUMBER(3,0)
AMENITY_ID	NOT NULL	NUMBER(3,0)

```
create table transact(
    t_id number(3) primary key ,
    p_id number(3) ,
    client_id number(3) ,
    agent_id number(3) ,
    foreign key (p_id) references property(p_id) on delete cascade ,
    foreign key (client_id) references client(client_id) on delete cascade ,
    foreign key (agent_id) references agent(agent_id) on delete cascade
);
```

TABLE TRANSACT

Column	Null?	Type
T_ID	NOT NULL	NUMBER(3,0)
P_ID	-	NUMBER(3,0)
CLIENT_ID	-	NUMBER(3,0)
AGENT_ID	-	NUMBER(3,0)

```
create table transaction_between(
    t_id number(3) primary key,
    t_price number(30) not null ,
    t_date date not null ,
    foreign key (t_id) references transact(t_id) on delete cascade
```

TABLE TRANSACTION_BETWEEN

Column	Null?	Type
T_ID	NOT NULL	NUMBER(3,0)
T_PRICE	NOT NULL	NUMBER(30,0)
T_DATE	NOT NULL	DATE

```
);
create table feedback_t(
    t_id number(3) ,
    reviewer_name varchar(20) ,
    primary key(t_id,reviewer_name) ,
    foreign key (t_id) references transact(t_id) on delete cascade
);
```

TABLE FEEDBACK_T

Column	Null?	Type
T_ID	NOT NULL	NUMBER(3,0)
REVIEWER_NAME	NOT NULL	VARCHAR2(20)

Insert Into Tables

Insert values into the property table

INSERT INTO property (p_id, address, p_type, status, p_size, no_of_bed, no_of_bathroom) VALUES (1, '123 Main St', 'residential', 'available', '2000 sqft', 3, 2);
INSERT INTO property (p_id, address, p_type, status, p_size, no_of_bed, no_of_bathroom) VALUES (2, '456 Elm St', 'commercial', 'under contract', '5000 sqft', NULL, NULL);
INSERT INTO property (p_id, address, p_type, status, p_size, no_of_bed, no_of_bathroom) VALUES (3, '789 Oak St', 'land parcels', 'sold', '100 acres', NULL, NULL);
INSERT INTO property (p_id, address, p_type, status, p_size, no_of_bed, no_of_bathroom) VALUES (4, '101 Pine St', 'residential', 'available', '1500 sqft', 2, 1);

P_ID	ADDRESS	P_TYPE	STATUS	P_SIZE	NO_OF_BED	NO_OF_BATHROOM
1	123 Main St	residential	available	2000 sqft	3	2
2	456 Elm St	commercial	under contract	5000 sqft	-	-
3	789 Oak St	land parcels	sold	100 acres	-	-
4	101 Pine St	residential	available	1500 sqft	2	1

Insert values into the agency table

INSERT INTO agency (agency_id, agency_name, agency_contact) VALUES (1, 'ABC Realty', 'abc@realty.com');
INSERT INTO agency (agency_id, agency_name, agency_contact) VALUES (2, 'XYZ Properties', 'xyz@properties.com');
INSERT INTO agency (agency_id, agency_name, agency_contact) VALUES (3, '123 Real Estate', '123@realestate.com');
INSERT INTO agency (agency_id, agency_name, agency_contact) VALUES (4, '456 Realty Group', '456@realtygroup.com');

AGENCY_ID	AGENCY_NAME	AGENCY_CONTACT
1	ABC Realty	abc@realty.com
2	XYZ Properties	xyz@properties.com
3	123 Real Estate	123@realestate.com
4	456 Realty Group	456@realtygroup.com

Insert values into the agent table

INSERT INTO agent (agent_id, agency_id, agent_name, agent_contact, Experience_review_num, Rating) VALUES (1, 1, 'John Doe', 1234567890, 48, 'ok');
INSERT INTO agent (agent_id, agency_id, agent_name, agent_contact, Experience_review_num, Rating) VALUES (2, 1, 'Jane Smith', 9876543210, 102, 'very good');
INSERT INTO agent (agent_id, agency_id, agent_name, agent_contact, Experience_review_num, Rating) VALUES (3, 2, 'Bob Johnson', 5555555555, 0, 'not good');
INSERT INTO agent (agent_id, agency_id, agent_name, agent_contact, Experience_review_num, Rating) VALUES (4, 3, 'Alice Williams', 7777777777, 30, ‘ok');

AGENT_ID	AGENCY_ID	AGENT_NAME	AGENT_CONTACT	EXPERIENCE_REVIEW_NUM	RATING
1	1	John Doe	1234567890	48	ok
2	1	Jane Smith	9876543210	102	very good
3	2	Bob Johnson	5555555555	0	not good
4	3	Alice Williams	7777777777	30	ok

Insert values into the client table

INSERT INTO client (client_id, client_name) VALUES (1, 'Michael');
INSERT INTO client (client_id, client_name) VALUES (2, 'Emily');
INSERT INTO client (client_id, client_name) VALUES (3, 'David');
INSERT INTO client (client_id, client_name) VALUES (4, ‘Sophia’);

CLIENT_ID	CLIENT_NAME
1	Michael
2	Emily
3	David
4	Sophia

Insert values into the client_contact table

INSERT INTO client_contact (client_id, contact) VALUES (1, 1111111111);
INSERT INTO client_contact (client_id, contact) VALUES (2, 2222222222);
INSERT INTO client_contact (client_id, contact) VALUES (3, 3333333333);
INSERT INTO client_contact (client_id, contact) VALUES (4, 4444444444);

CLIENT_ID	CONTACT
1	1111111111
2	2222222222
3	3333333333
4	4444444444

Insert values into the contract table

INSERT INTO contract (client_id, agent_id, start_date, end_date) VALUES (1, 1, '01-JAN-2024', '31-DEC-2024');
INSERT INTO contract (client_id, agent_id, start_date, end_date) VALUES (2, 2, '01-FEB-2024', '31-DEC-2024');
INSERT INTO contract (client_id, agent_id, start_date, end_date) VALUES (3, 3, '01-MAR-2024', '31-DEC-2024');
INSERT INTO contract (client_id, agent_id, start_date, end_date) VALUES (4, 4, '01-APR-2024', ’31-DEC-2024');

CLIENT_ID	AGENT_ID	START_DATE	END_DATE
1	1	01-JAN-24	31-DEC-24
2	2	01-FEB-24	31-DEC-24
3	3	01-MAR-24	31-DEC-24
4	4	01-APR-24	31-DEC-24

Insert values into the amenities table

INSERT INTO amenities (amenity_id, amenity_name) VALUES (1, 'Swimming Pool');
INSERT INTO amenities (amenity_id, amenity_name) VALUES (2, 'Gym');

INSERT INTO amenities (amenity_id, amenity_name) VALUES (3, 'Tennis Court');
INSERT INTO amenities (amenity_id, amenity_name) VALUES (4, 'Parking Lot');

AMENITY_ID	AMENITY_NAME
1	Swimming Pool
2	Gym
3	Tennis Court
4	Parking Lot

Insert values into the include table

INSERT INTO include (p_id, amenity_id) VALUES (1, 1);
INSERT INTO include (p_id, amenity_id) VALUES (2, 2);
INSERT INTO include (p_id, amenity_id) VALUES (3, 3);
INSERT INTO include (p_id, amenity_id) VALUES (4, 4);

P_ID	AMENITY_ID
1	1
2	2
3	3
4	4

Insert values into the transact table

INSERT INTO transact (t_id, p_id, client_id, agent_id) VALUES (1, 1, 1, 1);
INSERT INTO transact (t_id, p_id, client_id, agent_id) VALUES (2, 2, 2, 2);
INSERT INTO transact (t_id, p_id, client_id, agent_id) VALUES (3, 3, 3, 3);
INSERT INTO transact (t_id, p_id, client_id, agent_id) VALUES (4, 4, 4, 4);

T_ID	P_ID	CLIENT_ID	AGENT_ID
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

Insert values into the transaction_between table

INSERT INTO transaction_between (t_id, t_price, t_date) VALUES (1, 200000, '01-MAY-2024');
INSERT INTO transaction_between (t_id, t_price, t_date) VALUES (2, 500000, '02-MAY-2024');
INSERT INTO transaction_between (t_id, t_price, t_date) VALUES (3, 1000000, '03-MAY-2024');
INSERT INTO transaction_between (t_id, t_price, t_date) VALUES (4, 150000, '04-MAY-2024');

T_ID	T_PRICE	T_DATE
1	200000	01-MAY-24
2	500000	02-MAY-24
3	1000000	03-MAY-24
4	150000	04-MAY-24

Insert values into the feedback_t table

INSERT INTO feedback_t (t_id, reviewer_name) VALUES (1, 'Sophia');
INSERT INTO feedback_t (t_id, reviewer_name) VALUES (2, 'Sophia');
INSERT INTO feedback_t (t_id, reviewer_name) VALUES (3, 'Michael');
INSERT INTO feedback_t (t_id, reviewer_name) VALUES (4, 'David');

T_ID	REVIEWER_NAME
1	Sophia
2	Sophia
3	Michael
4	David

Triggers and Exception Handling

```
CREATE OR REPLACE TRIGGER before_insert_transaction
BEFORE INSERT ON transact
FOR EACH ROW
DECLARE
   v_status property.status%TYPE;
BEGIN
   -- Retrieve property status
   SELECT status INTO v_status
   FROM property
   WHERE p_id = :NEW.p_id;

   -- Ensure that the property being transacted is available
   IF v_status <> 'available' THEN
      dbms_output.put_line('Property is not available for transaction');
   END IF;
END;
```

```
CREATE OR REPLACE TRIGGER before_insert_transaction
BEFORE INSERT ON transact
FOR EACH ROW
DECLARE
    v_status property.status%TYPE;
BEGIN
    -- Retrieve property status
    SELECT status INTO v_status
    FROM property
    WHERE p_id = :NEW.p_id;

    -- Ensure that the property being transacted is available
    IF v_status <> 'available' THEN
        dbms_output.put_line('Property is not available for transaction');
    END IF;
```

```
Trigger created.
```

```
CREATE OR REPLACE TRIGGER before_insert_property
BEFORE INSERT ON property
FOR EACH ROW
BEGIN
   -- Ensure that the property price is not negative
   IF :NEW.no_of_bed < 0 OR :NEW.no_of_bathroom < 0 THEN
      RAISE_APPLICATION_ERROR(-20001, 'No. of Beds or Bathrooms cannot be
negative');
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      -- Handle other exceptions here
      dbms_output.put_line('An error occurred: ' || SQLERRM);
END;
```

```oracle_plsql
CREATE OR REPLACE TRIGGER before_insert_property
BEFORE INSERT ON property
FOR EACH ROW
BEGIN
    -- Ensure that the property price is not negative
    IF :NEW.no_of_bed < 0 OR :NEW.no_of_bathroom < 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'No. of Beds or Bathrooms cannot be negative');
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        -- Handle other exceptions here
        dbms_output.put_line('An error occurred: ' || SQLERRM);
END;
INSERT INTO property (p_id, address, p_type, status, p_size, no_of_bed, no_of_bathroom)
VALUES (15, '40 Elm St', 'commercial', 'available', '1000 sqft',1,-2);
```

```
1 row(s) inserted.
An error occurred: ORA-20001: No. of Beds or Bathrooms cannot be negative
```

```oracle_plsql
CREATE SEQUENCE property_seq
START WITH 1
INCREMENT BY 1
NOCACHE;

CREATE OR REPLACE TRIGGER before_insert_property
BEFORE INSERT ON property
FOR EACH ROW
BEGIN
   SELECT property_seq.NEXTVAL
   INTO :NEW.p_id
   FROM DUAL;
END;
```

```oracle_plsql
CREATE SEQUENCE property_seq
START WITH 1
INCREMENT BY 1
NOCACHE;

CREATE OR REPLACE TRIGGER before_insert_property
BEFORE INSERT ON property
FOR EACH ROW
BEGIN
    SELECT property_seq.NEXTVAL
    INTO :NEW.p_id
    FROM DUAL;
END;
```

```
Sequence created.

Trigger created.
```

```oracle_plsql
CREATE SEQUENCE transaction_seq
START WITH 1
```

INCREMENT BY 1
NOCACHE;

```
--     SELECT property_seq.NEXTVAL
--     INTO :NEW.p_id
--     FROM DUAL;
-- END;


CREATE SEQUENCE transaction_seq
START WITH 1
INCREMENT BY 1
NOCACHE;
```

```
Sequence created.
```

CREATE OR REPLACE TRIGGER before_insert_transaction
BEFORE INSERT ON transact
FOR EACH ROW
BEGIN
   SELECT transaction_seq.NEXTVAL
   INTO :NEW.t_id
   FROM DUAL;
END;

```
-- NOCACHE;

CREATE OR REPLACE TRIGGER before_insert_transaction
BEFORE INSERT ON transact
FOR EACH ROW
BEGIN
    SELECT transaction_seq.NEXTVAL
    INTO :NEW.t_id
    FROM DUAL;
END;
```

```
Trigger created.
```

CREATE OR REPLACE TRIGGER after_insert_transaction
AFTER INSERT ON transact
FOR EACH ROW
BEGIN
   -- Update property status after a successful transaction
   UPDATE property
   SET status = 'sold'
   WHERE p_id = :NEW.p_id;
END;
CREATE OR REPLACE TRIGGER before_insert_contract
BEFORE INSERT ON contract

```
CREATE OR REPLACE TRIGGER after_insert_transaction
AFTER INSERT ON transact
FOR EACH ROW
BEGIN
    -- Update property status after a successful transaction
    UPDATE property
    SET status = 'sold'
    WHERE p_id = :NEW.p_id;
END;
```

```
Trigger created.
```

```
FOR EACH ROW
BEGIN
   -- Set contract start date to current date if not provided
   IF :NEW.start_date IS NULL THEN
      :NEW.start_date := SYSDATE;
   END IF;

   -- Set contract end date to one year after the start date if not provided
   IF :NEW.end_date IS NULL THEN
      :NEW.end_date := ADD_MONTHS(:NEW.start_date, 12);
   END IF;
END;
```

```
CREATE OR REPLACE TRIGGER before_insert_contract
BEFORE INSERT ON contract
FOR EACH ROW
BEGIN
    -- Set contract start date to current date if not provided
    IF :NEW.start_date IS NULL THEN
        :NEW.start_date := SYSDATE;
    END IF;

    -- Set contract end date to one year after the start date if not provided
    IF :NEW.end_date IS NULL THEN
        :NEW.end_date := ADD_MONTHS(:NEW.start_date, 12);
    END IF;
END;
```

```
Trigger created.
```

```
CREATE OR REPLACE TRIGGER before_insert_update_agency_contact
BEFORE INSERT OR UPDATE ON agency
FOR EACH ROW
BEGIN
   -- Check if the agency_contact is in email format
```

```
    IF NOT REGEXP_LIKE(:NEW.agency_contact, '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$') THEN
        -- Raise an exception if the email format is incorrect
        RAISE_APPLICATION_ERROR(-20001, 'Agency contact must be in email format');
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        dbms_output.put_line('Error: ' || SQLERRM);
END;
```

```
CREATE OR REPLACE TRIGGER before_insert_update_agency_contact
BEFORE INSERT OR UPDATE ON agency
FOR EACH ROW
BEGIN
    -- Check if the agency_contact is in email format
    IF NOT REGEXP_LIKE(:NEW.agency_contact, '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$') THEN
        dbms_output.put_line('Agency contact must be in email format');
    END IF;
END;

INSERT INTO agency (agency_id, agency_name, agency_contact)
VALUES (7, '1 Real Estate', '123realestate.com');
```

```
1 row(s) inserted.
Agency contact must be in email format
```

```
CREATE OR REPLACE TRIGGER before_insert_client_contact
BEFORE INSERT OR UPDATE ON client_contact
FOR EACH ROW
BEGIN
    -- Check if the client_contact is in phone number format
    IF NOT REGEXP_LIKE(:NEW.contact, '^\d{10}$') THEN
        dbms_output.put('Client contact must be a 10-digit phone number');
    END IF;
END;
```

```
CREATE OR REPLACE TRIGGER before_insert_client_contact
BEFORE INSERT OR UPDATE ON client_contact
FOR EACH ROW
BEGIN
    -- Check if the client_contact is in phone number format
    IF NOT REGEXP_LIKE(:NEW.contact, '^\d{10}$') THEN
        dbms_output.put('Client contact must be a 10-digit phone number');
    END IF;
END;
```

```
Trigger created.
```

```
CREATE OR REPLACE TRIGGER before_insert_agent_contact
BEFORE INSERT OR UPDATE ON agent
FOR EACH ROW
BEGIN
    -- Check if the client_contact is in phone number format
    IF NOT REGEXP_LIKE(:NEW.agent_contact, '^\d{10}$') THEN
```

```
        dbms_output.put('Agent contact must be a 10-digit phone number');
    END IF;
END;
```

```
CREATE OR REPLACE TRIGGER before_insert_agent_contact
BEFORE INSERT OR UPDATE ON agent
FOR EACH ROW
BEGIN
    -- Check if the client_contact is in phone number format
    IF NOT REGEXP_LIKE(:NEW.agent_contact, '^\d{10}$') THEN
        dbms_output.put('Agent contact must be a 10-digit phone number');
    END IF;
END;
```

```
Trigger created.
```

```
CREATE OR REPLACE TRIGGER set_default_reviews
BEFORE INSERT ON agent
FOR EACH ROW
BEGIN
    :NEW.Experience_review_num := 0;
END;
```

```
-- END;

CREATE OR REPLACE TRIGGER set_default_reviews
BEFORE INSERT ON agent
FOR EACH ROW
BEGIN
    :NEW.Experience_review_num := 0;
END;
```

```
Trigger created.
```

```
CREATE OR REPLACE TRIGGER update_rating
AFTER INSERT OR DELETE ON feedback_t
FOR EACH ROW
DECLARE
    review_count NUMBER;
    v_agent_id NUMBER;
```

```
BEGIN
   SELECT agent_id INTO v_agent_id FROM transact WHERE t_id = :NEW.t_id;

   IF INSERTING THEN
      UPDATE agent SET experience_review_num = experience_review_num + 1 WHERE
agent_id = v_agent_id;
   ELSIF DELETING THEN
      SELECT experience_review_num INTO review_count FROM agent WHERE agent_id
= v_agent_id;
      IF review_count > 0 THEN
         UPDATE agent SET experience_review_num = experience_review_num - 1
WHERE agent_id = v_agent_id;
      END IF;
   END IF;

   SELECT experience_review_num INTO review_count FROM agent WHERE agent_id =
v_agent_id;
   IF review_count <= 10 THEN
      UPDATE agent SET rating = 'Not Good' WHERE agent_id = v_agent_id;
   ELSIF review_count > 10 AND review_count < 50 THEN
      UPDATE agent SET rating = 'ok' WHERE agent_id = v_agent_id;
   ELSIF review_count >= 50 AND review_count < 100 THEN
      UPDATE agent SET rating = 'good' WHERE agent_id = v_agent_id;
   ELSE
      UPDATE agent SET rating = 'very good' WHERE agent_id = v_agent_id;
   END IF;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Error: Agent ID not found');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

```
CREATE OR REPLACE TRIGGER update_rating
AFTER INSERT OR DELETE ON feedback_t
FOR EACH ROW
DECLARE
    review_count NUMBER;
    v_agent_id NUMBER;
BEGIN
    SELECT agent_id INTO v_agent_id FROM transact WHERE t_id = :NEW.t_id;

    IF INSERTING THEN
        UPDATE agent SET experience_review_num = experience_review_num + 1 WHERE agent_id = v_agent_id;
    ELSIF DELETING THEN
        SELECT experience_review_num INTO review_count FROM agent WHERE agent_id = v_agent_id;
        IF review_count > 0 THEN
            UPDATE agent SET experience_review_num = experience_review_num - 1 WHERE agent_id = v_agent_id;
        END IF;
    END IF;

    SELECT experience_review_num INTO review_count FROM agent WHERE agent_id = v_agent_id;
    IF review_count <= 10 THEN
        UPDATE agent SET rating = 'Not Good' WHERE agent_id = v_agent_id;
    ELSIF review_count > 10 AND review_count < 50 THEN
        UPDATE agent SET rating = 'ok' WHERE agent_id = v_agent_id;
    ELSIF review_count >= 50 AND review_count < 100 THEN
        UPDATE agent SET rating = 'good' WHERE agent_id = v_agent_id;
    ELSE
        UPDATE agent SET rating = 'very good' WHERE agent_id = v_agent_id;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Error: Agent ID not found');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

```
Trigger created.
```

Functions and procedures using cursors to retrieve data

```
-- Function to get the latest transaction date for a specific client
CREATE OR REPLACE FUNCTION get_latest_transaction_date(p_client_id IN
NUMBER) RETURN DATE
IS
   v_latest_date DATE;
BEGIN
   SELECT MAX(t_date)
   INTO v_latest_date
   FROM transaction_between tb
   JOIN transact t ON tb.t_id = t.t_id
   WHERE t.client_id = p_client_id;

   RETURN v_latest_date;
END;
-- Get the latest transaction date for client with ID 3
SELECT get_latest_transaction_date(3) AS latest_transaction_date FROM DUAL;
```

LATEST_TRANSACTION_DATE
03-MAY-24

```
-- Function to get the total number of properties by type
CREATE OR REPLACE FUNCTION get_property_count_by_type(p_property_type
IN VARCHAR2) RETURN NUMBER
IS
   v_property_count NUMBER;
BEGIN
   SELECT COUNT(*)
   INTO v_property_count
   FROM property
   WHERE p_type = p_property_type;

   RETURN v_property_count;
END;
-- Get the total number of residential properties
SELECT get_property_count_by_type('residential') AS residential_property_count
FROM DUAL;
```

RESIDENTIAL_PROPERTY_COUNT
2

```
-- Create a procedure to retrieve transaction details for a specific client
CREATE OR REPLACE PROCEDURE get_transactions_by_client(p_client_id IN
NUMBER)
IS
   CURSOR trans_cursor IS
      SELECT *
      FROM transact
      WHERE client_id = p_client_id;
BEGIN
   FOR trans_rec IN trans_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Transaction ID: ' || trans_rec.t_id || ', Property ID:
' || trans_rec.p_id || ', Agent ID: ' || trans_rec.agent_id);
   END LOOP;
END;
-- Retrieve transactions for client with ID 2
EXEC get_transactions_by_client(2);
```

```
Statement processed.
Transaction ID: 2, Property ID: 2, Agent ID: 2
```

```
-- Create a procedure to retrieve agent information based on agency
CREATE OR REPLACE PROCEDURE get_agents_by_agency(p_agency_id IN
NUMBER)
IS
   CURSOR agent_cursor IS
      SELECT *
      FROM agent
      WHERE agency_id = p_agency_id;
BEGIN
   FOR agent_rec IN agent_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Agent ID: ' || agent_rec.agent_id || ', Name: ' ||
agent_rec.agent_name || ', Contact: ' || agent_rec.agent_contact);
   END LOOP;
END;
-- Retrieve agents from agency with ID 1
EXEC get_agents_by_agency(1);
```

```
Statement processed.
Agent ID: 1, Name: John Doe, Contact: 1234567890
Agent ID: 2, Name: Jane Smith, Contact: 9876543210
```

```
-- Create a procedure to retrieve property information based on property type
CREATE OR REPLACE PROCEDURE get_properties_by_type(p_property_type IN
VARCHAR2)
IS
   CURSOR prop_cursor IS
      SELECT *
      FROM property
      WHERE p_type = p_property_type;
BEGIN
   FOR prop_rec IN prop_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('Property ID: ' || prop_rec.p_id || ', Address: ' ||
prop_rec.address || ', Status: ' || prop_rec.status);
   END LOOP;
END;

-- Retrieve properties of type 'residential'
EXEC get_properties_by_type(‘residential');
```

```
Statement processed.
Property ID: 1, Address: 123 Main St, Status: available
Property ID: 4, Address: 101 Pine St, Status: available
```

Conclusion

In conclusion, our DBMS project for Real Estate Management has been successfully developed and implemented to fullfill the diverse requirements of stakeholders involved in property transactions. The system encompasses a comprehensive database structure, including tables for Property, Transaction, Client, Agent, Agency, Contract, Amenities, and Feedback, to facilitate efficient management and tracking of real estate operations.

The core of our Real Estate DBMS is the Property table, which meticulously stores detailed information about various properties, ranging from houses, apartments, commercial buildings to land parcels. Each property entry includes essential attributes such as address, type, size, amenities, and geographic information, enabling streamlined property management and search functionalities.

In summary, our Real Estate Management DBMS project offers a robust platform for efficiently managing and tracking real estate operations, ensuring transparency, accountability, and enhanced client satisfaction. With its comprehensive features and user-friendly interface, we believe our project sets a high standard for real estate management systems and can serve as a valuable template for similar endeavors in the industry.