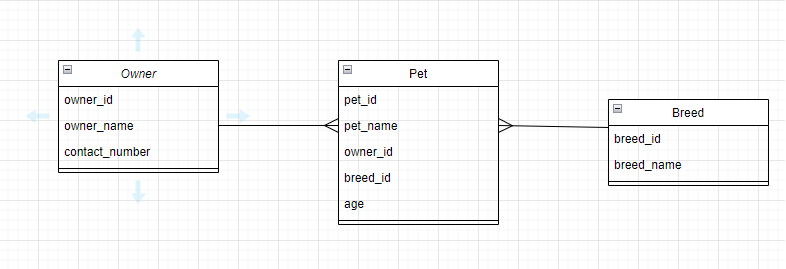
**Introduction**

In this assignment, we'll embark the creation of meticulously crafted tables, procedures, functions, and triggers using PL/SQL.We will exemplify the seamless integration of database design principles with the power and flexibility of PL/SQL programming. By delving into the intricacies of pet ownership data management, we aim to illustrate the importance of structured data handling and the role of procedural logic in ensuring data integrity and efficiency.We will use the layers of database design, testing, and cleanup, providing insights and best practices along the way. Whether you're a seasoned database administrator or an aspiring developer, this demo promises to equip you with valuable knowledge and techniques for building robust database solutions.

**Setup**

**Schema**

****

Before we begin, let's ensure the environment is properly set up by executing the following commands:

COLUMN object\_name FORMAT A30;

COLUMN object\_type FORMAT A12;

PURGE RECYCLEBIN;

SELECT object\_name, object\_type FROM user\_objects;

This will display the current objects in the schema and prepare us for the demonstration. We will also run it in the end

**Skeleton Table**

1. **Owner Table:**

CREATE TABLE owner (

owner\_id INT PRIMARY KEY,

owner\_name VARCHAR2(100),

contact\_number VARCHAR2(20)

);

1. **Breed Table:**

CREATE TABLE breed (

breed\_id INT PRIMARY KEY,

breed\_name VARCHAR2(100)

);

1. **Pet Table:**

CREATE TABLE pet (

pet\_id INT PRIMARY KEY,

pet\_name VARCHAR2(100),

owner\_id INT,

breed\_id INT,

age INT,

CONSTRAINT fk1\_owner FOREIGN KEY (owner\_id) REFERENCES owner(owner\_id),

CONSTRAINT fk1\_breed FOREIGN KEY (breed\_id) REFERENCES breed(breed\_id)

);

**Functions**

**Function:** Get Pet Count By Owner

**Description:** Retrieves the count of pets belonging to a specific owner.

**Parameters:** p\_owner\_id: INT - The ID of the owner for whom the pet count is to be retrieved.

**Returns:** INT - The count of pets owned by the specified owner.

**Code:**

CREATE FUNCTION get\_pet\_count\_by\_owner (

p\_owner\_id IN INT

) RETURN INT AS

pet\_count INT;

BEGIN

SELECT COUNT(\*) INTO pet\_count

FROM pets

WHERE owner\_id = p\_owner\_id;

RETURN pet\_count;

END get\_pet\_count\_by\_owner;

/

**Function:** Get Pet By ID

**Description:** Retrieves pet information based on the provided pet ID.

**Parameters:** p\_pet\_id: INT - The ID of the pet to retrieve.

**Returns:** pet%ROWTYPE - Record containing information about the specified pet.

**Code:**

CREATE OR REPLACE FUNCTION get\_pet\_by\_id (

p\_pet\_id IN INT

) RETURN pet%ROWTYPE AS

pet\_record pet%ROWTYPE;

BEGIN

SELECT \* INTO pet\_record

FROM pet

WHERE pet\_id = p\_pet\_id;

RETURN pet\_record;

END get\_pet\_by\_id;

/

**Function:** Get Breed By ID

**Description:** Retrieves information about a breed based on the provided breed ID.

**Parameters:** p\_breed\_id: INT - The ID of the breed to retrieve.

**Returns:** breed%ROWTYPE - Record containing information about the specified breed.

**Code:**

CREATE OR REPLACE FUNCTION get\_breed\_by\_id (

p\_breed\_id IN INT

) RETURN breed%ROWTYPE AS

breed\_record breed%ROWTYPE;

BEGIN

SELECT \* INTO breed\_record

FROM breed

WHERE breed\_id = p\_breed\_id;

RETURN breed\_record;

END get\_breed\_by\_id;

/

**Procedures**

**Procedure:** Add Pet

**Description:** Adds a new pet to the database.

**Parameters: 1.** p\_pet\_id: INT - The ID of the new pet.

1. p\_pet\_name: VARCHAR2 - The name of the new pet.
2. p\_owner\_id: INT - The ID of the owner of the new pet.
3. p\_breed\_id: INT - The ID of the breed of the new pet.
4. p\_age: INT - The age of the new pet.

**Code:**

CREATE PROCEDURE add\_pet (

p\_pet\_id IN INT,

p\_pet\_name IN VARCHAR2,

p\_owner\_id IN INT,

p\_breed\_id IN INT,

p\_age IN INT

) AS

BEGIN

INSERT INTO pet (pet\_id, pet\_name, owner\_id, breed\_id, age)

VALUES (p\_pet\_id, p\_pet\_name, p\_owner\_id, p\_breed\_id, p\_age);

COMMIT;

END add\_pet;

/

**Procedure:** Test Get Pets By Owner With Pets

**Description:** Tests the retrieval of pets for a specific owner and displays the results.

**Code:**

CREATE PROCEDURE test\_get\_pets\_by\_owner\_with\_pets AS

pet\_list SYS\_REFCURSOR;

pet\_rec pet%ROWTYPE;

owner\_id\_to\_test INT := 3; -- Modify the owner ID as needed

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Testing Retrieval of Pets for Owner...');

pet\_list := get\_pets\_by\_owner(owner\_id\_to\_test);

LOOP

FETCH pet\_list INTO pet\_rec;

EXIT WHEN pet\_list%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Pet ID: ' || pet\_rec.pet\_id || ', Pet Name: ' || pet\_rec.pet\_name || ', Owner ID: ' || pet\_rec.owner\_id || ', Breed ID: ' || pet\_rec.breed\_id || ', Age: ' || pet\_rec.age);

END LOOP;

CLOSE pet\_list;

END;

/

**Procedure:** Update Pet Age

**Description:** Updates the age of a pet in the database.

**Parameters:1.** p\_pet\_id: INT - The ID of the pet whose age is to be updated.

1. p\_new\_age: INT - The new age of the pet.

**Code:**

CREATE PROCEDURE update\_pet\_age (

p\_pet\_id IN INT,

p\_new\_age IN INT

) AS

BEGIN

UPDATE pet

SET age = p\_new\_age

WHERE pet\_id = p\_pet\_id;

COMMIT;

END update\_pet\_age;

/

**Procedure:** Delete Pet

**Description:** Deletes a pet from the database based on the provided pet ID.

**Parameters:** p\_pet\_id: INT - The ID of the pet to be deleted.

**Code:**

CREATE OR REPLACE PROCEDURE delete\_pet (

p\_pet\_id IN INT

) AS

BEGIN

DELETE FROM pet

WHERE pet\_id = p\_pet\_id;

COMMIT;

END delete\_pet;

/

**Procedure:** Update Owner Contact

**Description:** Updates the contact number of an owner in the database.

**Parameters: 1.** p\_owner\_id: INT - The ID of the owner whose contact number is to be updated.

1. p\_new\_contact\_number: VARCHAR2 - The new contact number of the owner.

**Code:**

CREATE OR REPLACE PROCEDURE update\_owner\_contact (

p\_owner\_id IN INT,

p\_new\_contact\_number IN VARCHAR2

) AS

BEGIN

UPDATE owner

SET contact\_number = p\_new\_contact\_number

WHERE owner\_id = p\_owner\_id;

COMMIT;

END update\_owner\_contact;

/

**Triger**

**Trigger to Check Owner Existence** (trg\_check\_owner\_exists):

**Descrition :** This trigger ensures that a pet cannot be inserted without a valid owner by checking the existence of the provided owner\_id in the owner table before insertion.

**Code:**

CREATE OR REPLACE TRIGGER trg\_check\_owner\_exists

BEFORE INSERT ON pet

FOR EACH ROW

DECLARE

owner\_count INT;

BEGIN

SELECT COUNT(\*) INTO owner\_count

FROM owner

WHERE owner\_id = :NEW.owner\_id;

IF owner\_count = 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Owner does not exist');

END IF;

END;

/

**Trigger to Check Breed Existence** (trg\_check\_breed\_exists):

**Descrition :**This trigger ensures that a pet cannot be inserted without a valid breed by checking the existence of the provided breed\_id in the breed table before insertion.

**Code:**

CREATE OR REPLACE TRIGGER trg\_check\_breed\_exists

BEFORE INSERT ON pet

FOR EACH ROW

DECLARE

breed\_count INT;

BEGIN

SELECT COUNT(\*) INTO breed\_count

FROM breed

WHERE breed\_id = :NEW.breed\_id;

IF breed\_count = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Breed does not exist');

END IF;

END;

/

**Testing**

**Test Plan: Test Cases**

**Test 1:** Check Pet Count for an Existing Owner

**Description:** Checks if the function correctly retrieves the count of pets for an existing owner.

**Code:**

DECLARE

owner\_id INT := 1; -- Assuming owner\_id 1 has pets in the database

pet\_count\_result INT;

BEGIN

pet\_count\_result := get\_pet\_count\_by\_owner(owner\_id);

IF pet\_count\_result > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Test 1 Passed: Owner has ' || pet\_count\_result || ' pets.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Test 1 Failed: Owner does not have any pets.');

END IF;

END;

/

**Test 2:** Check Pet Count for a Non-existent Owner

**Description:** Checks if the function correctly handles a non-existent owner ID.

**Code:**

DECLARE

owner\_id INT := 3; -- Assuming owner\_id -1 does not exist in the database

pet\_count\_result INT;

BEGIN

pet\_count\_result := get\_pet\_count\_by\_owner(owner\_id);

IF pet\_count\_result = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Test 2 Passed: Owner does not exist or has no pets.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Test 2 Failed: Unexpected result. Owner may have pets or does not exist.');

END IF;

END;

/

**Test 3:** Retrieve Pets for Owner

**Description:** Tests the retrieval of pets for a specific owner and displays the results.

**Code:**

BEGIN

test\_get\_pets\_by\_owner\_with\_pets;

END;

/

**Test 4:** Retrieve Pet by ID

**Description:** Tests if the function retrieves the correct pet information based on the provided pet ID.

**Code:**

DECLARE

pet\_id\_to\_test INT := 1; -- Assuming pet\_id 1 exists in the database

pet\_record pet%ROWTYPE;

BEGIN

pet\_record := get\_pet\_by\_id(pet\_id\_to\_test);

IF pet\_record.pet\_id IS NOT NULL THEN

DBMS\_OUTPUT.PUT\_LINE('Test 4 Passed: Pet found. Pet ID: ' || pet\_record.pet\_id || ', Pet Name: ' || pet\_record.pet\_name);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Test 4 Failed: Pet not found.');

END IF;

END;

/

**Test 5:** Update Pet Age

**Description:** Tests if the procedure successfully updates the age of a pet in the database.

**Code:**

DECLARE

pet\_id\_to\_update INT := 1; -- Assuming pet\_id 1 exists in the database

new\_age INT := 9; -- New age to update

BEGIN

update\_pet\_age(pet\_id\_to\_update, new\_age);

DBMS\_OUTPUT.PUT\_LINE('Test 5 Passed: Pet age updated successfully.');

END;

/

**Test 6:** Delete Pet

**Description:** Tests if the procedure successfully deletes a pet from the database.

**Code:**

DECLARE

pet\_id\_to\_delete INT := 1; -- Assuming pet\_id 1 exists in the database

BEGIN

delete\_pet(pet\_id\_to\_delete);

DBMS\_OUTPUT.PUT\_LINE('Test 6 Passed: Pet deleted successfully.');

END;

/

**Test 7:** Retrieve Breed by ID

**Description:** Tests if the function retrieves the correct breed information based on the provided breed ID.

**Code:**

DECLARE

breed\_id\_to\_test INT := 101; -- Assuming breed\_id 101 exists in the database

breed\_record breed%ROWTYPE;

BEGIN

breed\_record := get\_breed\_by\_id(breed\_id\_to\_test);

IF breed\_record.breed\_id IS NOT NULL THEN

DBMS\_OUTPUT.PUT\_LINE('Test 7 Passed: Breed found. Breed ID: ' || breed\_record.breed\_id || ', Breed Name: ' || breed\_record.breed\_name);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Test 7 Failed: Breed not found.');

END IF;

END;

/

**Test 8:** Update Owner Contact

**Description:** Tests if the procedure successfully updates the contact number of an owner in the database.

**Code:**

DECLARE

owner\_id\_to\_update INT := 1; -- Assuming owner\_id 1 exists in the database

new\_contact\_number VARCHAR2(20) := '999-888-7777'; -- New contact number to update

BEGIN

update\_owner\_contact(owner\_id\_to\_update, new\_contact\_number);

DBMS\_OUTPUT.PUT\_LINE('Test 8 Passed: Owner contact number updated successfully.');

END;

/

**Cleanup**

Clean up the environment by dropping the tables:

DROP TABLE pet;

DROP TABLE owner;

DROP TABLE breed;

**Packup**

Now again executing the following commands:

COLUMN object\_name FORMAT A30;

COLUMN object\_type FORMAT A12;

PURGE RECYCLEBIN;

SELECT object\_name, object\_type FROM user\_objects;

This will display the current objects in the schema and prepare .

**Conclusion**

We've explored the intricacies of designing and implementing a database schema tailored for pet ownership management. By creating tables for owners, breeds, and pets, alongside procedures, functions, and triggers, we've illustrated the versatility and robustness of PL/SQL in database development. Through testing scenarios, we've ensured the functionality and reliability of our implemented procedures and functions. Additionally, we've emphasized the importance of proper cleanup procedures to maintain data integrity and efficiency within the database environment. As we conclude, it's evident that a well-structured database schema, coupled with meticulously crafted PL/SQL logic, forms the backbone of any efficient data management system. By adhering to best practices and considering scalability and flexibility, we pave the way for future enhancements and adaptations to meet evolving business needs.