CSE 3110: Database Management System

Name of the Project: Pet Shop Management

Fariha Sanzana

Roll: 2007024

Submitted To:

Nazia Jahan Khan Chowdhury

Assistant Professor, Department of Computer Science and Engineering,
Khulna University of Engineering & Technology (KUET)

Md. Shahidul Salim

Lecturer, Department of Computer Science and Engineering,
Khulna University of Engineering & Technology (KUET)

Introduction:

A database is a structured system for collecting, storing, managing, and retrieving information. It provides the means to organize and handle vast amounts of data efficiently and securely. Databases are integral to modern information systems and are used extensively across a myriad of industries to facilitate data manipulation and analysis. They are designed to support the creation, storage, retrieval, update, and management of data in a structured form

Pet shop management involves overseeing the operations of a store that sells pets, pet food, supplies, and sometimes offers services such as grooming or veterinary care. Effective management is crucial to ensure the health and well-being of the animals, meet customer expectations, and achieve business success.

Using a database effectively can significantly enhance the management of animal health, well-being, and ethical sourcing in a pet shop. By integrating a robust database system, pet shop managers can maintain comprehensive records and ensure that all aspects of animal care are met with high standards.

By enhancing the database, I can facilitate features such as real-time data updates, which ensure that the information on animal health, inventory levels, and customer interactions is always current and accessible. Advanced querying capabilities allow for quick retrieval of specific data, aiding in faster decision-making, such as quickly identifying which animals require immediate medical attention or restocking essential supplies. Additionally, integrating automated alerts and reminders into the database can significantly improve the scheduling of health check-ups, vaccinations, and grooming services, ensuring no animal is overlooked.

Objective:

With my project "Pet Shop Management" I tried to demonstrate that by leveraging robust database enhancements, a pet shop can operate more efficiently, maintain high standards of animal care, and offer superior customer service, all of which contribute to a successful and reputable business. If we look through the objectives-

- **Comprehensive Animal Inventory Management:**
  By maintaining a real-time database of all animals, including details such as species, age, health status, and special requirements. This allows for efficient tracking and ensures that the health and welfare needs of each animal are met and communicated effectively to potential customers.
- **Detailed Health and Requirement Tracking:**
  Utilize the database to record detailed health histories, dietary requirements, and care schedules for each animal. Automated alerts can be set up for regular check-ups and treatments, ensuring no animal's health care is overlooked.
- **Accessories and Product Inventory Control:**
  Implement inventory management systems within the database to track stock levels, update prices, and reorder products automatically when inventory falls below predetermined levels. This helps in maintaining adequate stock and avoiding overstock situations.
- **Comprehensive Customer Management:**
  The database can store extensive information on customers, including contact information, purchase history, preferences, and pet details. This data can be used to personalize services, offer targeted promotions, and build lasting customer relationships.
- **Efficient Order List Management:**
  Use the database to manage orders, track processing status, and ensure timely fulfillment. Integration with inventory management ensures that stock levels are automatically updated as orders are processed.
- **Delivery Status and Customer Review Monitoring:**
  The database tracks the delivery status of orders, providing customers and management with real-time updates. Additionally, storing customer reviews in the database helps in assessing customer satisfaction and identifying areas for improvement.
- **Sales and Financial Analysis:**
  Utilize triggers and other database functionalities to automatically update sales data and generate real-time financial reports. This allows for immediate visibility into the financial health of the business and supports informed decision-making.

Database Design:

Here's a brief overview of the database tables I might need for my project.

1. Pets: Stores details of named pets for sale, including ID, name, type, price, availability, and description. Helps manage pet inventory and provides key details to customers.

2. UnnamedPets: Similar to the Pets table but for grouped animals like fish or birds. Includes ID, type, quantity, price, and availability. Manages inventory of unnamed animal groups.

3. Health_Records: Keep records health information for each pet, such as age, health status, diet, last checkup, and notes. Ensures regular monitoring and maintenance of pet health.

4. Customers: Records customer data including ID, name, unique email, phone, and address. Supports customer service and personalized marketing.

5. Products: Catalogs items like pet food and accessories, listing product ID, name, price, stock quantity, and category.

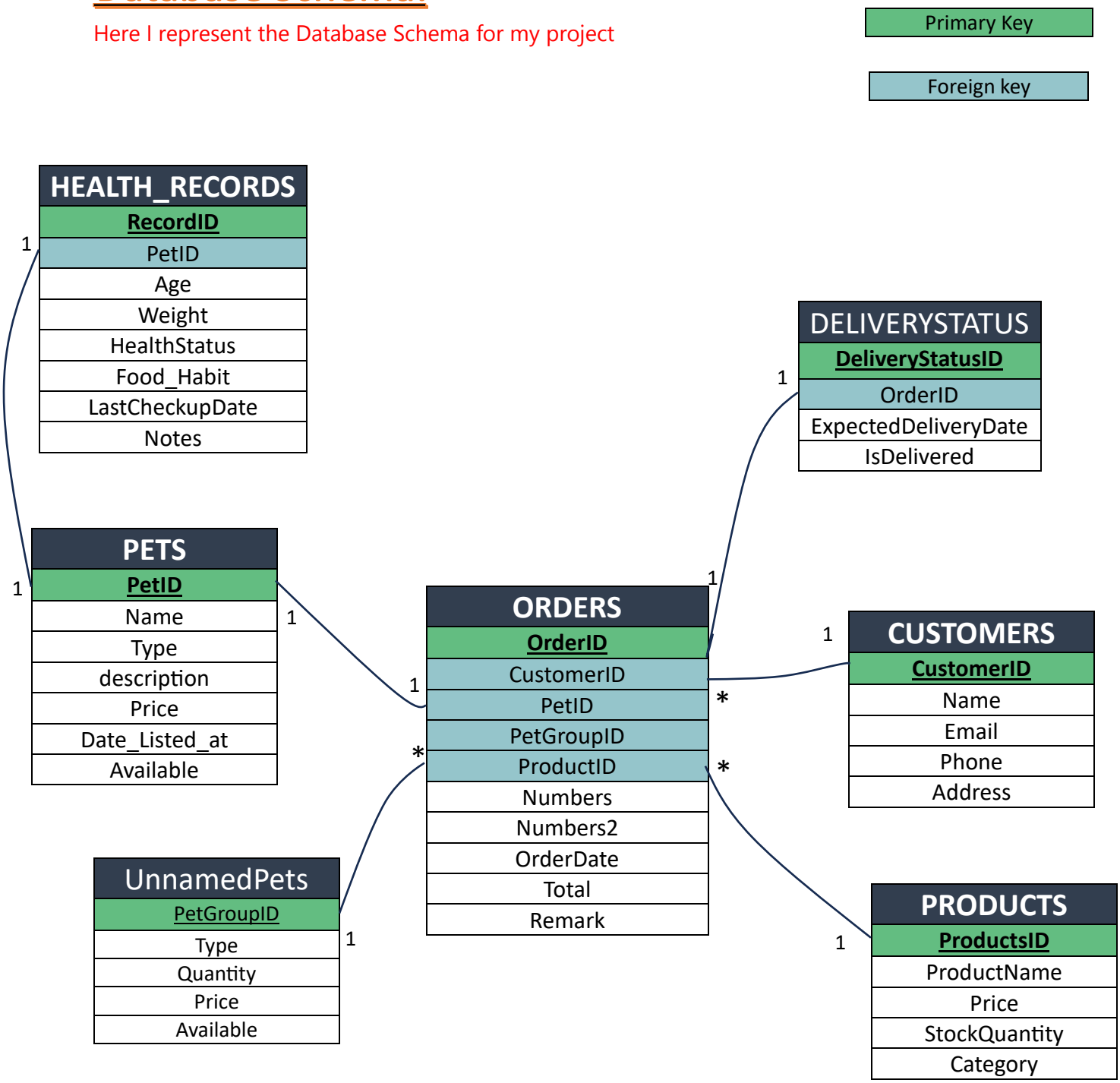
6. Orders: Logs purchased details with links to customers and products or pets bought, including order ID, date, total price, and remarks. Crucial for sales transactions and delivery management. It depends on many other tables

7. Delivery_Status: Tracks delivery details for orders, including status ID, order ID, expected and actual delivery dates, completion status, and customer satisfaction. Ensures timely delivery and tracks customer feedback.

It ensure the management of a dynamic pet shop environment. It presents tables such as **'PETS'** , ' **UnnamedPets**,**'HEALTH_RECORDS'**, '**ORDERS'**, **'CUSTOMES'**, **'PRODUCTS**', **'DELIVERY_STATUS'** providing comprehensive insight into inventory, pet health, and customer interactions. Additionally, specialized tables like **'PETS'** and **'UnnamedPets** cater to unique inventory needs, ensuring efficient order processing and customer satisfaction.

Database Schema:

Here I represent the Database Schema for my project



Database Table Creation:

Here is the command of 'Run SQL Command Line' for Oracle. I have created 7 tables for the need of my project. Also CONSTRAINTs are added such as foreign key, primary key to connect the tables and others as per need.

```
CREATE TABLE Pets (
    PetID NUMBER PRIMARY KEY,
    Name VARCHAR2(255),
    Type VARCHAR2(100),
    Description VARCHAR2(500),
    Price DECIMAL(10, 2),
    Listed_at_Date DATE,
    Available CHAR(1) CHECK (Available IN ('Y', 'N'))
);


CREATE TABLE UnnamedPets (
    PetGroupID NUMBER PRIMARY KEY,
    Type VARCHAR2(100),
    Quantity NUMBER,
    Price DECIMAL(10, 2),
    Available CHAR(1) CHECK (Available IN ('Y', 'N'))
);

CREATE TABLE Health_Records (
    RecordID NUMBER PRIMARY KEY,
    PetID NUMBER,
```

```
  Age NUMBER(10,2),
   Weight NUMBER(10,2),
   HealthStatus VARCHAR2(255),
   Food_Habit VARCHAR2(255),
   LastCheckupDate DATE,
   Notes VARCHAR2(1000),
   FOREIGN KEY (PetID) REFERENCES Pets(PetID)
);

CREATE TABLE Customers (
   CustomerID NUMBER PRIMARY KEY,
   Name VARCHAR2(255),
   Email VARCHAR2(255) UNIQUE,
   Phone VARCHAR2(20),
   Address VARCHAR2(500)
);
CREATE TABLE Products (
   ProductID NUMBER PRIMARY KEY,
   ProductName VARCHAR2(255),
   Price DECIMAL(10, 2),
   StockQuantity NUMBER,
   Category VARCHAR2(100)
);

 CREATE TABLE Orders (
   OrderID NUMBER PRIMARY KEY,
   CustomerID NUMBER,
   PetID NUMBER NULL,
   ProductID NUMBER NULL,
```

```
    PetGroupID NUMBER NULL,
    Numbers NUMBER,
    OrderDate DATE,
    Total DECIMAL(10, 2) NULL,
    Discount_code VARCHAR2(10),
    Remark VARCHAR2(1000),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (PetID) REFERENCES Pets(PetID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID),
    FOREIGN KEY (PetGroupID) REFERENCES UnnamedPets(PetGroupID),
    CONSTRAINT Check_ProductID_Numbers2
CHECK ((ProductID IS NOT NULL AND Numbers2 IS NOT NULL) OR (ProductID IS NULL AND Numbers2 IS
NULL))
    CONSTRAINT Check_PetGroupID_Numbers
CHECK ((PetGroupID IS NOT NULL AND Numbers IS NOT NULL) OR (PetGroupID IS NULL AND Numbers IS
NULL))
);

CREATE TABLE DeliveryStatus (
    DeliveryStatusID NUMBER PRIMARY KEY,
    OrderID NUMBER,
    ExpectedDeliveryDate DATE,
    DeliveryDate DATE,
    IsDelivered CHAR(1) CHECK (IsDelivered IN ('Y', 'N')), -- 'Y' for yes, 'N' for no
    CustomerSatisfaction VARCHAR2(255),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);
```

Tabe Description and Attributes:

1.Pets

- PetID: Unique identifier for each pet.
- Name: The name of the pet.
- Type: Type or species of the pet.
- Description: A brief description about the pet.
- Price: Sale price of the pet.
- Listed_at_Date: Date when the pet was listed as available.
- Available: Indicates whether the pet is currently available ('Y' for yes, 'N' for no).
    - PetID is referenced by the Orders table for orders involving specific pets.
    - PetID is referenced by the Health_Records table for health records involving specific pets.

2. UnnamedPets

- PetGroupID: Unique identifier for a group of unnamed pets.
- Type: Species or type of the unnamed pets.
- Quantity: Number of pets in this group.
- Price: Total price for the entire group of pets.
- Available: Availability status of the pet group ('Y' for yes, 'N' for no).
    - PetGroupID is referenced by the Orders table for orders involving groups of unnamed pets.

3. Health_Records

- RecordID: Unique identifier for each health record.
- PetID: Foreign key linking to the Pets table.
- Age: Age of the pet.
- Weight: Listing the wight measurements.
- HealthStatus: Current health status of the pet.
- Food_Habit: Dietary preferences and habits of the pet.
- LastCheckupDate: Date of the last health checkup.
- Notes: Additional notes regarding the pet's health and care.

4. Customers

- CustomerID: Unique identifier for each customer.
- Name: Name of the customer.

- Email: Email address of the customer, must be unique.
- Phone: Contact phone number.
- Address: Residential address of the customer.
    - CustomerID is referenced by the Orders table, linking customers to their orders.

5. Products

- ProductID: Unique identifier for each product.
- ProductName: Name of the product.
- Price: Selling price of the product.
- StockQuantity: Number of units available in stock.
- Category: Category to which the product belongs.
    - ProductID is referenced by the Orders table for orders involving products.

6. Orders

- OrderID: Unique identifier for each order.
- CustomerID: Foreign key linking to the Customers table.
- PetID: Foreign key linking to the Pets table; can be null if the order does not include a pet.
- ProductID: Foreign key linking to the Products table; can be null if the order does not include a product.
- PetGroupID: Foreign key linking to the UnnamedPets table; can be null if the order does not include a group of pets.
- Numbers: Number of pet groups ordered, must align with presence or absence of PetGroupID.
- Numbers2: Number of products ordered, must align with presence or absence of ProductID.
- OrderDate: Date when the order was placed.
- Total: Total cost of the order; can be null if not yet calculated.
- Remark: Additional remarks or notes related to the order.

7. DeliveryStatus

- DeliveryStatusID: Unique identifier for each delivery status entry.
- OrderID: Foreign key linking to the Orders table.
- ExpectedDeliveryDate: The expected date of delivery.
- DeliveryDate: Actual delivery date, can be null if not yet delivered.
- IsDelivered: Indicates whether the order has been delivered ('Y' for yes, 'N' for no).
- CustomerSatisfaction: Customer feedback or satisfaction rating post-delivery.
    - OrderID is referenced from the Orders table, linking delivery statuses to specific orders.

---------- ---------- :

Relation Among the Tables:

1. Customers and Orders

Relationship: One-to-Many

Explanation: One customer can place multiple orders, but each order is associated with only one customer. This is evident from the CustomerID foreign key in the Orders table.

2. Orders and Pets

Relationship: Many-to-One

Explanation: Each order can include at most one specific pet (though some orders might not include any pet), while a pet can be involved in multiple orders over time. This relationship is enabled by the PetID foreign key in the Orders table.

3. Orders and UnnamedPets

Relationship: Many-to-One

Explanation: Each order can involve one group of unnamed pets at a time, but a group of unnamed pets can be referenced in multiple orders. This is managed by the PetGroupID foreign key in the Orders table.

4. Orders and Products

Relationship: Many-to-One

Explanation: Each order can contain at most one type of product (though some orders might not include any product), while a product can be part of multiple orders. This is reflected by the ProductID foreign key in the Orders table.

5. Orders and DeliveryStatus

Relationship: One-to-One

Explanation: Each order has one delivery status entry. This relationship is specified by the OrderID foreign key in the DeliveryStatus table, linking each delivery status uniquely back to one order.

Basic SQL commands:

- Insert: To Insert into a table as row or for particular attributes in a row
  - **Command:** INSERT INTO TABLE_NAME('attribute-1','attribute-2', ...) VALUES ('value-1','value-2',.....);
  - Example:
    INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available) VALUES (1001, 'Bella', 'Dog', 'breed: Labrador, furtype: Short, color: Black', 12000.00, TO_DATE('2023-05-01', 'YYYY-MM-DD'), 'Y');

- Seletct:For selecting particular information, row or cell with or without any specialize requirements.
  - Example:
    select * from pets where type='Cat';

- Update: Update data from table, modify or add any new column, constraints, modifying data type etc
  - Example: To update a particular cell of a row
    update pets set description='breed-local,color-brown,fur_type-short' where petid=1014;

- Delete: To delete a column we have to use ALTER command, to delete particular cell, we use DELETE command, to delete a table (without constraints such as unique/primary keys in table referenced by foreign keys) we use DROP command
  - **Command:** To delete a column from a table
    ALTER TABLE ExampleTable DROP COLUMN ExampleColumn;
  - Example:
    ALTER TABLE HEALTH_RECORDS DROP COLUMN WEIGHT;
  - **Command:**Deletetion of particular row
    DELETE FROM TABLE_NAME WHERE CONDITION;
  - Example:
    DELETE FROM PETS WHERE NAME='Pitus';

Here is a link where to view multiple commands associated with my project and understand the necessity of each query: https://github.com/sannzana/sql.git

PL/SQL Commands:

In a database where tables are interconnected, updates to one table often necessitate corresponding updates in related tables to maintain an accurate and cohesive view of the data. In such scenarios, the robust capabilities of PL/SQL prove invaluable. Utilizing arrays, loops, while loops, and cursors in PL/SQL can streamline and automate these updates. Additionally, to effectively manage changes across connected tables, employing procedures, functions, and triggers is essential. These tools help ensure that any modifications in one table trigger the necessary adjustments in others, keeping the database synchronized and efficient. Here are some commands that I believe will be particularly useful for managing a pet shop database:

- **Purpose:**
  To insert a new pet into PETS table (for pets table)

  - **Solution:**

```
SET SERVEROUTPUT ON;
DECLARE
   v_petId PETS.PetID%TYPE := 1020;
   v_name PETS.Name%TYPE := 'Puti';
   v_type PETS.Type%TYPE := 'Dog';
   v_description PETS.Description%TYPE := 'Friendly and energetic golden
retriever';
   v_price PETS.Price%TYPE := 23000.00;
   v_listed_at_date PETS.Listed_at_Date%TYPE := SYSDATE;
   v_available PETS.Available%TYPE := 'Y';
BEGIN
   INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date,
Available)
   VALUES (v_petId, v_name, v_type, v_description, v_price, v_listed_at_date,
v_available);
   -- Optionally output a message confirming the insertion
   DBMS_OUTPUT.PUT_LINE('Inserted pet: ' || v_name);
END;
/
```

- **Purpose:**
  To ensure Available in PETS table is checked before ordering a pet, as it is the indication if a pet is available or not (for pets and orders table)

- Solution: Using a trigger

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE TRIGGER Check_Pet_Availability
BEFORE INSERT OR UPDATE OF PetID ON Orders
FOR EACH ROW
DECLARE
   v_available CHAR(1);
BEGIN
   -- Check the availability of the pet if PetID is not NULL
   IF :NEW.PetID IS NOT NULL THEN
      SELECT Available INTO v_available FROM Pets WHERE PetID = :NEW.PetID;
      IF v_available = 'N' THEN
         RAISE_APPLICATION_ERROR(-20001, 'This pet is not currently available for
orders.');
      END IF;
   END IF;
END;
/
```

- Purpose:
  After each insertion of a pet from PETS table to the 'Orders' table, as each pet is listed as only one here, availability must be set to 'N' so that the same pet can't be ordered again (for pets and orders table)

  - Solution: Creating a trigger to update the available status

```
CREATE OR REPLACE TRIGGER Update_Pet_Availability

        AFTER INSERT OR UPDATE ON Orders
        FOR EACH ROW
        BEGIN
           -- Check if a PetID is included in the order
           IF :NEW.PetID IS NOT NULL THEN
              -- Update the pet's availability to 'N'
              UPDATE Pets
              SET Available = 'N'
              WHERE PetID = :NEW.PetID;
           END IF;
        END;
        /
```

- **Purpose:**

  To ensure when quantity of ant unnamed grouped pet is zero, meaning all SOLD OUT , ordering any of those won't be possible (for unnamedpets and orders table)

  - **Solution:** Using a trigger

```
SET SERVEROUTPUT ON;

CREATE OR REPLACE TRIGGER CHECK_QUANTITY

BEFORE INSERT OR UPDATE OF PETGROUPID ON ORDERS

FOR EACH ROW

DECLARE
V_QUAN NUMBER;
BEGIN
   IF :NEW.PETGROUPID IS NOT NULL THEN
      SELECT QUANTITY INTO V_QUAN FROM UNNAMEDPETS WHERE
PetGroupID =  :NEW.PETGROUPID;
```

```
            IF V_QUAN <= 0 THEN
                RAISE_APPLICATION_ERROR(-20002, 'This Pet Group Has been
    Sold Out');
            END IF;
        END IF;
    END;
/
```

- **Same type of trigger for the case of ProductId: (for products and orders table)**

```
SET SERVEROUTPUT ON;

CREATE OR REPLACE TRIGGER CHECK_QUANTITY_product
BEFORE INSERT OR UPDATE OF PRODUCTID ON ORDERS
FOR EACH ROW
DECLARE
V_QUAN NUMBER;
BEGIN
        IF :NEW.PRODUCTID IS NOT NULL THEN
         SELECT STOCKQUANTITY INTO V_QUAN FROM PRODUCTS
WHERE PRODUCTID =   :NEW.PRODUCTID;
            IF V_QUAN <= 0 THEN
                RAISE_APPLICATION_ERROR(-20002, 'This Product is Sold
Out');
             END IF;
        END IF;
END;/
```

- **Purpose:**
  When a product is ordered, update the information of quantity of that product in 'Products' table. (For orders and products table)

  - **Solution:** Writing a trigger command

```
CREATE OR REPLACE TRIGGER count_numbers
BEFORE INSERT OR UPDATE OF numbers ON orders
FOR EACH ROW
DECLARE
  vcon PRODUCTS.StockQuantity%TYPE;
BEGIN
  IF :NEW.productid IS NOT NULL AND :NEW.numbers2 IS NOT NULL
THEN
    SELECT stockquantity INTO vcon FROM products WHERE productid =
:NEW.productid;
       IF vcon >= :NEW.numbers2 THEN
        UPDATE products
         SET stockquantity = stockquantity - :NEW.numbers2
         WHERE productid = :NEW.productid;
      ELSE
          RAISE_APPLICATION_ERROR(-20003, 'Not available');
     END IF;
END IF;
END;
/
```

```
SQL> CREATE OR REPLACE TRIGGER count_numbers
  2  BEFORE INSERT OR UPDATE OF numbers ON orders
  3  FOR EACH ROW
  4  DECLARE
  5      vcon PRODUCTS.StockQuantity%TYPE;
  6  BEGIN
  7      IF :NEW.productid IS NOT NULL AND :NEW.numbers IS NOT NULL THEN
  8          SELECT stockquantity INTO vcon FROM products WHERE productid = :NEW.productid;
  9          IF vcon >= :NEW.numbers THEN
 10              UPDATE products
 11              SET stockquantity = stockquantity - :NEW.numbers
 12              WHERE productid = :NEW.productid;
 13          ELSE
 14              RAISE_APPLICATION_ERROR(-20003, 'Not available');
 15          END IF;
 16      END IF;
 17  END;
 18  /

Trigger created.
```

- **Purpose:**
  When a petGroup is ordered from 'UNNAMEDPETS ', first check if the order is possible depending on whether the quantity neened is available, if available then update the information of quantity of that product in Products table; else showing that purchase is not possible (for unnamedpets and orders table)

  - Solution: Writing a trigger command

```
CREATE OR REPLACE TRIGGER Update_UnnamedPet_Quantity
BEFORE INSERT OR UPDATE ON Orders
FOR EACH ROW
DECLARE
      v_quantity NUMBER;
BEGIN
    IF :NEW.PetGroupID IS NOT NULL AND :NEW.Numbers IS NOT NULL
THEN
        SELECT Quantity INTO v_quantity FROM UnnamedPets WHERE
PetGroupID =      :NEW.PetGroupID;


        IF v_quantity >= :NEW.Numbers THEN
           UPDATE UnnamedPets
           SET Quantity = Quantity - :NEW.Numbers
           WHERE PetGroupID = :NEW.PetGroupID;
        ELSE
           RAISE_APPLICATION_ERROR(-20003, 'Not enough unnamed pets
available for              PetGroupID: ' || TO_CHAR(:NEW.PetGroupID));
       END IF;
   END IF;
EXCEPTION
        WHEN NO_DATA_FOUND THEN
 RAISE_APPLICATION_ERROR(-20004, 'PetGroupID not found in
UnnamedPets table: ' ||   TO_CHAR(:NEW.PetGroupID));
END;
/
```

```
SQL>
SQL> CREATE OR REPLACE TRIGGER Update_UnnamedPet_Quantity
  2  BEFORE INSERT OR UPDATE ON Orders
  3  FOR EACH ROW
  4  DECLARE
  5      v_quantity NUMBER;
  6  BEGIN
  7      -- Check if the order involves a pet group and there is a specified quantity
  8      IF :NEW.PetGroupID IS NOT NULL AND :NEW.Numbers IS NOT NULL THEN
  9          -- Get the current quantity of the unnamed pets group
 10          SELECT Quantity INTO v_quantity FROM UnnamedPets WHERE PetGroupID = :NEW.PetGroupID;
 11
 12          -- Check if there are enough unnamed pets available to fulfill the order
 13          IF v_quantity >= :NEW.Numbers THEN
 14              -- Update the quantity of the unnamed pets
 15              UPDATE UnnamedPets
 16              SET Quantity = Quantity - :NEW.Numbers
 17              WHERE PetGroupID = :NEW.PetGroupID;
 18          ELSE
 19              -- Not enough pets available, raise an error
 20              RAISE_APPLICATION_ERROR(-20003, 'Not enough unnamed pets available for PetGroupID: ' || TO_CHAR(:NEW.PetGroupID));
 21          END IF;
 22      END IF;
 23  EXCEPTION
 24      WHEN NO_DATA_FOUND THEN
 25          -- No pet group found with the given ID, raise an error
 26          RAISE_APPLICATION_ERROR(-20004, 'PetGroupID not found in UnnamedPets table: ' || TO_CHAR(:NEW.PetGroupID));
 27  END;
 28  /

Trigger created.
```

Purpose:

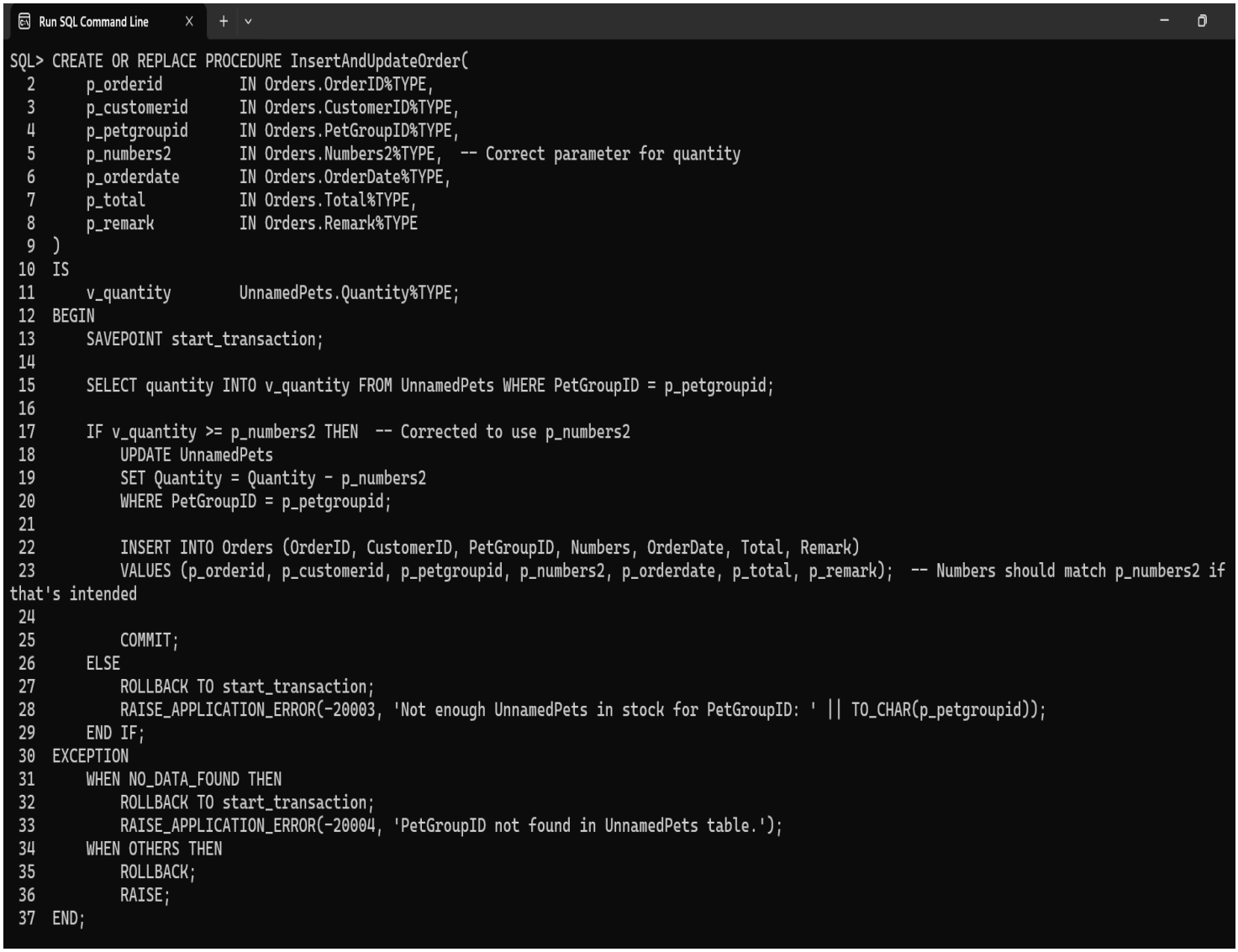
Insert data into 'Orders' table and at the same time update the corresponding information of 'Unnamedpets' table (for unnamedpets and orders table)

- **Solution:** Creating a procedure for insertion and corresponding update

```
CREATE OR REPLACE PROCEDURE InsertAndUpdateOrder(
    p_orderid        IN Orders.OrderID%TYPE,
    p_customerid     IN Orders.CustomerID%TYPE,
    p_petgroupid     IN Orders.PetGroupID%TYPE,
    p_numbers2       IN Orders.Numbers2%TYPE,  -- Correct parameter for quantity
    p_orderdate      IN Orders.OrderDate%TYPE,
    p_total          IN Orders.Total%TYPE,
    p_remark         IN Orders.Remark%TYPE
```

```
)
IS
   v_quantity        UnnamedPets.Quantity%TYPE;
BEGIN
   SAVEPOINT start_transaction;

   SELECT quantity INTO v_quantity FROM UnnamedPets WHERE
PetGroupID = p_petgroupid;

   IF v_quantity >= p_numbers2 THEN  -- Corrected to use p_numbers2
      UPDATE UnnamedPets
      SET Quantity = Quantity - p_numbers2
      WHERE PetGroupID = p_petgroupid;

      INSERT INTO Orders (OrderID, CustomerID, PetGroupID, Numbers,
OrderDate, Total, Remark)
      VALUES (p_orderid, p_customerid, p_petgroupid, p_numbers2,
p_orderdate, p_total, p_remark);  -- Numbers should match p_numbers2 if
that's intended

      COMMIT;
   ELSE
      ROLLBACK TO start_transaction;
      RAISE_APPLICATION_ERROR(-20003, 'Not enough UnnamedPets in
stock for PetGroupID: ' || TO_CHAR(p_petgroupid));
   END IF;
EXCEPTION
```

```
    WHEN NO_DATA_FOUND THEN
        ROLLBACK TO start_transaction;
        RAISE_APPLICATION_ERROR(-20004, 'PetGroupID not found in
UnnamedPets table.');
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END;
/
```



```
SQL> BEGIN
  2      InsertAndUpdateOrder(
  3          p_orderid         => 5016,
  4          p_customerid      => 2,
  5          p_petgroupid      => 3004,
  6          p_numbers2        => 1,        -- Assuming this should be 'Numbers' based on previous
 context
  7          p_orderdate       => SYSDATE,
  8          p_total           => 1450.00,
  9          p_remark          => 'Bulk order for event'
 10      );
 11  END;
 12  /

PL/SQL procedure successfully completed.
```

- **Purpose:**
  It is quite troublesome to calculate the total price in each order where there might be orders for pet, product or grouped pet, also the numbers may vary (for unnamedpets and orders table)

  - **Solution:** Creating a trigger so that the calculation is done automatically for the TOTAL in 'orders' table

```
CREATE OR REPLACE TRIGGER Calculate_Order_Total
BEFORE INSERT OR UPDATE ON Orders
FOR EACH ROW
BEGIN
   -- Initialize total to zero
   :NEW.Total := 0;

   -- Calculate total from Products
   IF :NEW.ProductID IS NOT NULL AND :NEW.Numbers2 IS NOT NULL
THEN
      SELECT Price * :NEW.Numbers2 INTO :NEW.Total
      FROM Products
```

```
        WHERE ProductID = :NEW.ProductID;
    END IF;

    -- Add total from UnnamedPets if applicable
    IF :NEW.PetGroupID IS NOT NULL AND :NEW.Numbers IS NOT NULL
THEN
        SELECT (:NEW.Total + (Price * :NEW.Numbers)) INTO :NEW.Total
        FROM UnnamedPets
        WHERE PetGroupID = :NEW.PetGroupID;
    END IF;

    -- Add total for a specific Pet if applicable
    IF :NEW.PetID IS NOT NULL AND :NEW.Numbers IS NOT NULL THEN
        SELECT (:NEW.Total + (Price * :NEW.Numbers)) INTO :NEW.Total
        FROM Pets
        WHERE PetID = :NEW.PetID;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- If no product, pet group, or individual pet found, do not set total
        :NEW.Total := NULL;
    WHEN OTHERS THEN
        -- Reraise any unexpected error
        RAISE;
END;
```

- **Purpose:**
  Ensuring that expecteddeliverydate is after Date_Listed_at for each pets (for deliverystatus and orders table)

  - **Solution:** Creating a trigger

```
CREATE OR REPLACE TRIGGER Check_Order_Date
BEFORE INSERT OR UPDATE OF OrderDate, PetID ON Orders
FOR EACH ROW
DECLARE
   v_listed_at_date DATE;
BEGIN
   -- Check if PetID is not null and retrieve the listed date
   IF :NEW.PetID IS NOT NULL THEN
      SELECT Listed_at_Date INTO v_listed_at_date FROM Pets WHERE
PetID = :NEW.PetID;

      -- If the order date is before the pet was listed, raise an error
      IF :NEW.OrderDate < v_listed_at_date THEN
         RAISE_APPLICATION_ERROR(-20005, 'Order date must be after the
pet''s listed date.');
      END IF;
   END IF;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR(-20006, 'Pet not found with the given
ID.');
END;
/
```

```
SQL> connect fariha024
Enter password:
Connected.
SQL> CREATE OR REPLACE TRIGGER Check_Delivery_Date
  2  BEFORE INSERT OR UPDATE OF ExpectedDeliveryDate ON DeliveryStatus
  3  FOR EACH ROW
  4  DECLARE
  5      v_order_date DATE;
  6  BEGIN
  7      -- Retrieve the order date from the Orders table using the OrderID
  8      SELECT OrderDate INTO v_order_date FROM Orders WHERE OrderID = :NEW.OrderID;
  9
 10      -- Check if the expected delivery date is set before the order date
 11      IF :NEW.ExpectedDeliveryDate < v_order_date THEN
 12          RAISE_APPLICATION_ERROR(-20007, 'Expected delivery date must be after the order date.');
 13      END IF;
 14  EXCEPTION
 15      WHEN NO_DATA_FOUND THEN
 16          RAISE_APPLICATION_ERROR(-20008, 'Order not found with the given ID.');
 17  END;
 18  /

Trigger created.
```

```
SQL> INSERT INTO DeliveryStatus (DeliveryStatusID, OrderID, ExpectedDeliveryDate, IsDelivered, CustomerSatisfa
ction)
  2  VALUES (8, 5007, TO_DATE('2024-02-15', 'YYYY-MM-DD'), 'N', NULL);
INSERT INTO DeliveryStatus (DeliveryStatusID, OrderID, ExpectedDeliveryDate, IsDelivered, CustomerSatisfaction
)
            *
ERROR at line 1:
ORA-20007: Expected delivery date must be after the order date.
ORA-06512: at "FARIHA024.CHECK_DELIVERY_DATE", line 9
ORA-04088: error during execution of trigger 'FARIHA024.CHECK_DELIVERY_DATE'
```

Purpose:

It is not possible to order any pet before it is listed into the Pets table (for pets and orders table)

- Solution: Creating a trigger

```
CREATE OR REPLACE TRIGGER Check_date
BEFORE INSERT OR UPDATE ON Orders
FOR EACH ROW
```

```
DECLARE
   v_listed_at_date DATE;
BEGIN
   -- Only proceed if a PetID is associated with the order
   IF :NEW.PetID IS NOT NULL THEN
      -- Retrieve the listed date of the pet from the Pets table
      SELECT Listed_at_Date INTO v_listed_at_date FROM Pets WHERE
PetID = :NEW.PetID;

      -- Compare the order date with the pet's listed date
      IF :NEW.OrderDate < v_listed_at_date THEN
         RAISE_APPLICATION_ERROR(-20010, 'Order date must be after the
pet''s listed date.');
      END IF;
   END IF;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      -- This exception handles the case where the PetID does not exist in
the Pets table
      RAISE_APPLICATION_ERROR(-20011, 'Pet not found with the given
ID.');
END;
/
```

```
SQL> CREATE OR REPLACE TRIGGER Check_date
  2  BEFORE INSERT OR UPDATE ON Orders
  3  FOR EACH ROW
  4  DECLARE
  5      v_listed_at_date DATE;
  6  BEGIN
  7      -- Only proceed if a PetID is associated with the order
  8      IF :NEW.PetID IS NOT NULL THEN
  9          -- Retrieve the listed date of the pet from the Pets table
 10          SELECT Listed_at_Date INTO v_listed_at_date FROM Pets WHERE PetID = :NEW.PetID;
 11
 12          -- Compare the order date with the pet's listed date
 13          IF :NEW.OrderDate < v_listed_at_date THEN
 14              RAISE_APPLICATION_ERROR(-20010, 'Order date must be after the pet''s listed date.');
 15          END IF;
 16      END IF;
 17  EXCEPTION
 18      WHEN NO_DATA_FOUND THEN
 19          -- This exception handles the case where the PetID does not exist in the Pets table
 20          RAISE_APPLICATION_ERROR(-20011, 'Pet not found with the given ID.');
 21  END;
 22  /

Trigger created.
```

```
SQL>
SQL> INSERT INTO Orders (OrderID, CustomerID, petID, OrderDate)
  2  VALUES (5017, 3, 1007, TO_DATE('2023-04-11', 'YYYY-MM-DD'));
INSERT INTO Orders (OrderID, CustomerID, petID, OrderDate)
            *
ERROR at line 1:
ORA-20010: Order date must be after the pet's listed date.
ORA-06512: at "FARIHA024.CHECK_DATE", line 11
ORA-04088: error during execution of trigger 'FARIHA024.CHECK_DATE'
```

After presenting the constraints, trigger, procedure and other command, finally I will insert into the tables for my project:

For Pets table:

-- Insert available pets

INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available)

VALUES (1001, 'Bella', 'Dog', 'breed: Labrador, furtype: Short, color: Black', 12000.00, TO_DATE('2023-05-01', 'YYYY-MM-DD'), 'Y');

INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available)

VALUES (1002, 'Charlie', 'Dog', 'breed: Beagle, furtype: Short, color: Brown', 18000.00, TO_DATE('2023-05-02', 'YYYY-MM-DD'), 'Y');

INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available)

VALUES (1003, 'Luna', 'Cat', 'breed: Persian, furtype: Long, color: White', 9000.00, TO_DATE('2023-05-03', 'YYYY-MM-DD'), 'Y');

INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available)

VALUES (1004, 'Maxy', 'Dog', 'breed: Local, furtype: Medium, color: Black and Tan', 1500.00, TO_DATE('2023-05-04', 'YYYY-MM-DD'), 'Y');

INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available)

VALUES (1005, 'Lucy', 'Cat', 'breed: Siamese, furtype: Short, color: Cream', 7000.00, TO_DATE('2023-05-05', 'YYYY-MM-DD'), 'Y');

INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available)

VALUES (1006, 'Daisy', 'Dog', 'breed: Local, furtype: Smooth, color: White', 1300.00, TO_DATE('2023-05-06', 'YYYY-MM-DD'), 'Y');

INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available)

VALUES (1007, 'Milo', 'Cat', 'breed: Maine Coon, furtype: Long, color: Brown', 11000.00, TO_DATE('2023-05-07', 'YYYY-MM-DD'), 'Y');

INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available)

VALUES (1008, 'Bailey', 'Dog', 'breed: Local, furtype: Short, color: Brown', 7550.00, TO_DATE('2023-05-08', 'YYYY-MM-DD'), 'Y');

INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available)

VALUES (1010, 'Sadie', 'Dog', 'breed: Poodle, furtype: Curly, color: White', 15400.00, TO_DATE('2023-05-10', 'YYYY-MM-DD'), 'Y');

INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available)

VALUES (1011, 'Sky', 'Parrot', 'breed: Macaw, furtype: Feathered, color: Blue and Gold', 22200.00, TO_DATE('2023-05-11', 'YYYY-MM-DD'), 'Y');

INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available)

VALUES (1013, 'Echo', 'Parrot', 'breed: African Grey, furtype: Feathered, color: Grey', 15500.00, TO_DATE('2023-05-13', 'YYYY-MM-DD'), 'Y');

INSERT INTO Pets (PetID, Name, Type, Description, Price, Listed_at_Date, Available)

VALUES (1016, 'Majesty2', 'Eagle', 'breed: Bald Eagle, furtype: Feathered, color: White and Brown', 30000.00, TO_DATE('2023-05-15', 'YYYY-MM-DD'), 'N');

For UNNAMEDPETS table:

INSERT INTO UnnamedPets (PetGroupID, Type, Quantity, Price, Available)

VALUES (3001, 'Lovebirds', 30, 800.00, 'Y');

INSERT INTO UnnamedPets (PetGroupID, Type, Quantity, Price, Available)

VALUES (3002, 'Goldfish', 20, 400.00, 'Y');

INSERT INTO UnnamedPets (PetGroupID, Type, Quantity, Price, Available)

VALUES (3003, 'Koi', 15, 1500.00, 'Y');

INSERT INTO UnnamedPets (PetGroupID, Type, Quantity, Price, Available)

VALUES (3004, 'Rabbits', 10, 1450.00, 'Y');

For HEALTH_RECORDS table:

INSERT INTO Health_Records (RecordID, PetID, Age, Weight, HealthStatus, Food_Habit, LastCheckupDate, Notes) **VALUES**

(2001, 1001, 3, 10.0, 'Healthy', 'Dry food', TO_DATE('2023-04-30', 'YYYY-MM-DD'), 'No known issues');

INSERT INTO Health_Records (RecordID, PetID, Age, Weight, HealthStatus, Food_Habit, LastCheckupDate, Notes) **VALUES**

(2002, 1002, 2, 20.0, 'Healthy', 'Mix of dry and wet food', TO_DATE('2024-01-01', 'YYYY-MM-DD'), 'Regular vaccinations needed');

INSERT INTO Health_Records (RecordID, PetID, Age, Weight, HealthStatus, Food_Habit, LastCheckupDate, Notes) **VALUES**

(2003, 1003, 4, 5.0, 'Healthy', 'Premium cat food', TO_DATE('2024-01-01', 'YYYY-MM-DD'), 'Slight allergy to dust');

INSERT INTO Health_Records (RecordID, PetID, Age, Weight, HealthStatus, Food_Habit, LastCheckupDate, Notes) **VALUES**

(2004, 1004, 5, 25.0, 'Healthy', 'Home cooked food', TO_DATE('2023-12-03', 'YYYY-MM-DD'), 'Requires monthly grooming');

INSERT INTO Health_Records (RecordID, PetID, Age, Weight, HealthStatus, Food_Habit, LastCheckupDate, Notes) **VALUES**

(2005, 1005, 6, 4.5, 'Healthy', 'Cat treats and kibble', TO_DATE('2022-02-04', 'YYYY-MM-DD'), 'No special care needed');

INSERT INTO Health_Records (RecordID, PetID, Age, Weight, HealthStatus, Food_Habit, LastCheckupDate, Notes) **VALUES**

(2006, 1006, 1, 15.0, 'Healthy', 'Puppy food', TO_DATE('2023-05-05', 'YYYY-MM-DD'), 'Undergoing training'),

INSERT INTO Health_Records (RecordID, PetID, Age, Weight, HealthStatus, Food_Habit, LastCheckupDate, Notes) **VALUES**

(2007, 1007, 3, 6.0, 'Healthy', 'Fish and wet food', TO_DATE('2024-02-06', 'YYYY-MM-DD'), 'Regular check-ups recommended');

INSERT INTO Health_Records (RecordID, PetID, Age, Weight, HealthStatus, Food_Habit, LastCheckupDate, Notes) **VALUES**

(2008, 1008, 2, 22.0, 'Healthy', 'Commercial dog food', TO_DATE('2024-01-01', 'YYYY-MM-DD'), 'Active and needs daily exercise');

INSERT INTO Health_ Records (RecordID, PetID, Age, Weight, HealthStatus, Food_Habit, LastCheckupDate, Notes) **VALUES**

(2010, 1010, 3, 12.0, 'Healthy', 'Grain-free diet', TO_DATE('2023-12-03', 'YYYY-MM-DD'), 'Sensitive skin condition');

INSERT INTO Health_ Records (RecordID, PetID, Age, Weight, HealthStatus, Food_Habit, LastCheckupDate, Notes) **VALUES**

(2011, 1011, 7, 1.0, 'Healthy', 'Seeds and nuts', TO_DATE('2023-05-10', 'YYYY-MM-DD'), 'Bright and alert');

INSERT INTO Health_Records (RecordID, PetID, Age, Weight, HealthStatus, Food_Habit, LastCheckupDate, Notes) **VALUES**

(2013, 1013, 8, 0.9, 'Healthy', 'Specialized parrot food', TO_DATE('2023-10-12', 'YYYY-MM-DD'), 'Social and enjoys interaction');

INSERT INTO Health_Records (RecordID, PetID, Age, Weight, HealthStatus, Food_Habit, LastCheckupDate, Notes) **VALUES**

(2015, 1015, 10, 4.0, 'Healthy', 'Meat-based diet', TO_DATE('2024-05-1', 'YYYY-MM-DD'), 'Requires large enclosure');

For CUSTOMERS table:

INSERT INTO Customers (CustomerID, Name, Email, Phone, Address) **VALUES**

(1, 'Asa Devi', 'devi@email.com', '01575622811', '123/A New Elephant Road');

INSERT INTO Customers (CustomerID, Name, Email, Phone, Address) VALUES

(2, 'Abir haider', 'abir@gmail.com', '01865555678', '456,Third Floor,Baily Road');

INSERT INTO Customers (CustomerID, Name, Email, Phone, Address) VALUES

(3, 'Raufun Mia', 'rau12@gmail.com', '01722094433', 'Boro Bari, Khan road,Savar');

INSERT INTO Customers (CustomerID, Name, Email, Phone, Address) VALUES

(4, 'Jolly Begum', 'begum@gmail.com', '01923434234', '321 Street Road,Dhaka');

INSERT INTO Customers (CustomerID, Name, Email, Phone, Address) VALUES

(5, '', 'maryb@example.com', '555-6543', '654 Spruce Street, Hilltown')

(6, 'Alex Garcia', 'alexg@example.com', '555-7890', '987 Cedar Street, Lakeside');

For PRODUCTS table:

INSERT INTO Products (ProductID, ProductName, Price, StockQuantity, Category) VALUES

(101, 'Premium Dog Food', 1150, 10, 'Pet Food');

INSERT INTO Products (ProductID, ProductName, Price, StockQuantity, Category) VALUES

(102, 'Cat Litter Box', 800,5 , 'Cat Accessories')

INSERT INTO Products (ProductID, ProductName, Price, StockQuantity, Category) VALUES

(103, 'Bird Cage Large', 1550, 5, 'Bird Accessories');

INSERT INTO Products (ProductID, ProductName, Price, StockQuantity, Category) VALUES

(104, 'Aquarium Filter', 875, 8, 'Aquarium Supplies');

INSERT INTO Products (ProductID, ProductName, Price, StockQuantity, Category) VALUES

(105, 'Rabbit Hutch', 300, 4, 'Small Pet Accessories');

```
INSERT INTO Products (ProductID, ProductName, Price, StockQuantity, Category)
VALUES
(106, 'Rubber Toys', 50, 300, 'Toys');
INSERT INTO Products (ProductID, ProductName, Price, StockQuantity, Category)
VALUES
(107, 'Dog Leash', 200.00, 20, 'Dog Accessories');
INSERT INTO Products (ProductID, ProductName, Price, StockQuantity, Category)
VALUES
(108, 'Catnip', 1110.99, 8, 'For Cats');
INSERT INTO Products (ProductID, ProductName, Price, StockQuantity, Category)
VALUES
(119, 'Tropical Fish Food', 400, 20, 'Fish Food');
```

For ORDERS table:

```
INSERT INTO Orders (OrderID, CustomerID, ProductID, Numbers2, OrderDate)
VALUES (5002, 2, 101, 2, TO_DATE('2024-05-01', 'YYYY-MM-DD'));
INSERT INTO Orders (OrderID, CustomerID, petid, OrderDate,total)
VALUES (5007, 2, 1010, TO_DATE('2024-05-01', 'YYYY-MM-DD'),15400);
INSERT INTO Orders (OrderID, CustomerID, PetID, OrderDate)
VALUES (5003, 4, 1011, SYSDATE);
INSERT INTO Orders (OrderID, CustomerID, PetID, PetGroupID, Numbers, OrderDate)
VALUES (5010,4, 1005, 3001, 4, TO_DATE('2024-05-01', 'YYYY-MM-DD'));
INSERT INTO Orders (OrderID, CustomerID, ProductID, Numbers2, PetID, OrderDate)
VALUES (5012, 3, 106, 2, 1008,  TO_DATE('2024-04-11', 'YYYY-MM-DD'));
INSERT INTO Orders (OrderID, CustomerID, petID,OrderDate )
VALUES (5007, 4, 1016, SYSDATE);
```

INSERT INTO Orders (OrderID, CustomerID, PetID, PetGroupID, Numbers, OrderDate)

VALUES (5012,4, 1013, 3001, 1, TO_DATE('2024-05-01', 'YYYY-MM-DD'));

For DELIVERYSTATUS table:

INSERT INTO DeliveryStatus (DeliveryStatusID, OrderID, ExpectedDeliveryDate, IsDelivered, CustomerSatisfaction)

VALUES (1, 5002, TO_DATE('2024-05-07', 'YYYY-MM-DD'), 'N', NULL);

INSERT INTO DeliveryStatus (DeliveryStatusID, OrderID, ExpectedDeliveryDate, IsDelivered, CustomerSatisfaction)

VALUES (2, 5007, TO_DATE('2024-05-08', 'YYYY-MM-DD'), 'N', NULL);

INSERT INTO DeliveryStatus (DeliveryStatusID, OrderID, ExpectedDeliveryDate, IsDelivered, CustomerSatisfaction)

VALUES (3, 5003, SYSDATE + 7, 'N', NULL);

INSERT INTO DeliveryStatus (DeliveryStatusID, OrderID, ExpectedDeliveryDate, IsDelivered, CustomerSatisfaction)

VALUES (4, 5009, TO_DATE('2024-05-08', 'YYYY-MM-DD'), 'N', NULL);

INSERT INTO DeliveryStatus (DeliveryStatusID, OrderID, ExpectedDeliveryDate, IsDelivered, CustomerSatisfaction)

VALUES (5, 5005, TO_DATE('2024-05-15', 'YYYY-MM-DD'), 'N', NULL);

INSERT INTO DeliveryStatus (DeliveryStatusID, OrderID, ExpectedDeliveryDate, IsDelivered, CustomerSatisfaction)

VALUES (6, 5010, TO_DATE('2024-04-21', 'YYYY-MM-DD'), 'Y', 'Happy to receive the pets, said will order in future too');

INSERT INTO DeliveryStatus (DeliveryStatusID, OrderID, ExpectedDeliveryDate, IsDelivered, CustomerSatisfaction)

VALUES (7, 5005, TO_DATE('2024-04-15', 'YYYY-MM-DD'), 'N', NULL);

Discussion:

During the design of the database for the pet store, I initially struggled to determine the optimal number of tables needed to keep the system organized and efficient. My goal was to create just enough tables to meet the requirements without cluttering the database with unnecessary details.

I did an elimination of a separate table named 'PetInfo' and integrated its functionality directly into the 'Pets' table through a column named 'Description' . This decision helped simplIl had some changes from the before submitted idea of my database project.
To further enhance the usability and integrity of the database, I incorporated several triggers that automate important checks and updates. These triggers are particularly useful in maintaining data accuracy and enforcing business rules. These triggers help prevent common data entry errors and ensure that the business logic is consistently applied across the database

I think this version is better and more useable for many reasons as-

Shop Management: Tracks both individual pets and groups of unnamed pets, along with products, their stock levels, and categories.

Order Management: Processes orders that may include pets, products, or both, and also special remarks. Here it is possible to order either all type or any particular with a specific number. Also added constrains made the database more real life useable.

Customer Management: Stores customer information including contact details and addresses, which is vital for communication and deliveries.

Delivery Tracking: It is actually an optional table for better management of the system.

Handling Null Values: Designing the Orders table to accommodate orders that might not include a pet, product, or unnamed pet group required careful planning to ensure integrity without restricting order flexibility.

Complex Relationships: Establishing the correct foreign key relationships and ensuring that each table properly references others to be updated with each table's change ,modify the database and made it easier to use as there is less numbers of table .

Rationale behind the Design:

The design of the database for the pet store management system has been carefully considered to meet specific operational needs, enhance data integrity, and facilitate efficient data management. Below are the key rationales behind the major design decisions:

- **Entity Segregation:**
  Pets vs. UnnamedPets: By differentiating between individual pets (Pets) and groups of unnamed pets (UnnamedPets), the system can handle sales and inventory differently based on whether the pets are sold individually or in groups. This distinction aids in better inventory tracking and pricing strategies.
- **Normalization:**
  To reduce redundancy and improve data integrity, the database schema is normalized. This ensures that updates in one part of the database do not inadvertently affect other unrelated data, which can be critical in maintaining accurate records for pets, customers, and transactions.
- **Primary and Foreign Keys:**
  Primary keys in each table uniquely identify each record, which is crucial for indexing and quick data retrieval.
  Foreign keys establish relationships between tables, which are essential for enforcing referential integrity. They ensure that relationships between entities like pets and their health records or customers and their orders are consistently maintained.
- **Use of Check Constraints:**
  Constraints such as Available in Pets and UnnamedPets using the CHAR(1) CHECK (Available IN ('Y', 'N')) ensure that only valid values ('Y' or 'N') are entered. This enforces data validity at the database level.
- **Date Handling:**
  The Listed_at_Date in the Pets table ensures that the system can track when each

I hope I have understood the task correctly and have represented a good database.

- pet was made available for sale, which is important for inventory aging and marketing strategies.
- **Scalability Considerations:**
  The design allows for easy scalability. As the business grows, new tables like Health_Records or new attributes can be added without disrupting existing functionalities. The modular nature of the schema supports expansion into new features like online orders or customer loyalty programs.

Conclusion:

This project was for the course CSE 3110: DATABASE MANAGEMENT SYSTEM, taught by respected Nazia Jahan Khan Chowdhury ma'am and Md. Shahidul Salim sir. The purpose was to understand and implement various database queries and commands, applying them to real-life situations. The lectures and slides provided during the labs were truly invaluable.

I tried my best to present a database that not only solves data-related problems but also reflects the learning from the course. It will be up to our respected teachers to judge this and provide necessary advice for future improvements in database implementation.