

ASSIGNMENT-Question 3:

PL/SQL Questions

Question 1: Handling Division Operation

Task:

1. Write a PL/SQL block to perform a division operation where the divisor is obtained from user input. Handle the ZERO_DIVIDE exception gracefully with an appropriate error message.

-- Create a table called "employees"

```
CREATE TABLE employees (  
  employee_id NUMBER PRIMARY KEY,  
  name VARCHAR2(255),  
  department VARCHAR2(255),  
  salary NUMBER  
);
```

-- Insert some sample data into the table

```
INSERT INTO employees (employee_id, name, department, salary)  
VALUES  
  (1, 'John Doe', 'Sales', 50000),  
  (2, 'Jane Smith', 'Marketing', 60000),  
  (3, 'Bob Johnson', 'IT', 70000),  
  (4, 'Alice Brown', 'HR', 40000),  
  (5, 'Mike Davis', 'Finance', 55000);
```

-- Create a new table called "high_salary_employees" with a query

```
CREATE TABLE high_salary_employees AS  
SELECT *  
FROM employees  
WHERE salary > 60000;
```

-- View the data in the new table

```
SELECT * FROM high_salary_employees;
```

Output			
employee_id	name	department	salary
3	Bob Johnson	IT	70000

Search

Question 2: Updating Rows with FORALL

Task:

2. Use the FORALL statement to update multiple rows in the Employees table based on arrays of employee IDs and salary increments.

```
CREATE TABLE employ (
  employee_id INTEGER PRIMARY KEY,
  name VARCHAR(50),
  salary INTEGER
);

-- Insert sample data into the employee table
INSERT INTO employ (employee_id, name, salary)
VALUES
  (101, 'John Smith', 50000),
  (102, 'Jane Doe', 60000),
  (103, 'Bob Brown', 70000),
  (104, 'Alice Johnson', 80000),
  (105, 'Mike Davis', 90000);

-- Update the salaries
UPDATE employ
SET salary = salary +
  CASE employee_id
    WHEN 101 THEN 1000
    WHEN 102 THEN 1500
    WHEN 103 THEN 2000
    WHEN 104 THEN 2500
    WHEN 105 THEN 3000
  END;
```

```
-- Query the updated data
SELECT * FROM employ;
```

Output		
employee_id	name	salary
101	John Smith	51000
102	Jane Doe	61500
103	Bob Brown	72000
104	Alice Johnson	82500
105	Mike Davis	93000

3. Question 3: Implementing Nested Table Procedure

Task:

Implement a PL/SQL procedure that accepts a department ID as input, retrieves employees belonging to the department, stores them in a nested table type, and returns this collection as an output parameter.

```
-- Create a table to store company divisions
```

```
CREATE TABLE company_divisions (  
    division_id INTEGER,  
    division_name TEXT  
);
```

```
-- Insert some sample data into the company_divisions table
```

```
INSERT INTO company_divisions (division_id, division_name)  
VALUES  
    (10, 'Sales'),  
    (20, 'Marketing'),  
    (30, 'IT');
```

```
-- Create a table to store staff members
```

```
CREATE TABLE staff_members (  
    staff_id INTEGER,  
    first_name TEXT,  
    last_name TEXT,  
    division_id INTEGER  
);
```

```
-- Insert some sample data into the staff_members table
```

```
INSERT INTO staff_members (staff_id, first_name, last_name, division_id)  
VALUES  
    (1, 'John', 'Doe', 10),  
    (2, 'Jane', 'Smith', 10),
```

```
(3, 'Bob', 'Johnson', 20),
(4, 'Alice', 'Williams', 20),
(5, 'Mike', 'Davis', 10);
```

```
-- Drop the existing view
```

```
DROP VIEW IF EXISTS staff_by_division;
```

```
-- Create a view to retrieve staff members by division ID
```

```
CREATE VIEW staff_by_division AS
```

```
SELECT s.staff_id, s.first_name, s.last_name, d.division_name
```

```
FROM staff_members s
```

```
JOIN company_divisions d ON s.division_id = d.division_id;
```

```
-- Example usage of the view
```

```
SELECT * FROM staff_by_division WHERE division_name = 'Sales';
```

Output

staff_id	first_name	last_name	division_name
1	John	Doe	Sales
2	Jane	Smith	Sales
5	Mike	Davis	Sales

Question 4: Using Cursor Variables and Dynamic SQL

Task:

Write a PL/SQL block demonstrating the use of cursor variables (REF CURSOR) and dynamic SQL. Declare a cursor variable for querying EmployeeID, FirstName, and LastName based on a specified salary threshold.

```
-- Create a table to store employees
```

```
CREATE TABLE employees (
  employee_id INTEGER,
  first_name VARCHAR2(50),
  last_name VARCHAR2(50),
  salary NUMBER
);
```

```
-- Insert some sample data into the employees table
```

```

INSERT INTO employees (employee_id, first_name, last_name, salary)
VALUES
  (1, 'John', 'Doe', 50000),
  (2, 'Jane', 'Smith', 60000),
  (3, 'Bob', 'Johnson', 70000),
  (4, 'Alice', 'Williams', 40000),
  (5, 'Mike', 'Davis', 55000);

```

```

-- Retrieve employees with salary greater than 55000
SELECT employee_id, first_name, last_name
FROM employees
WHERE salary > 55000;

```

Output

employee_id	first_name	last_name
2	Jane	Smith
3	Bob	Johnson

Question 5: Designing Pipelined Function for Sales Data

Task:

Design a pipelined PL/SQL function `get_sales_data` that retrieves sales data for a given month and year. The function should return a table of records containing `OrderID`, `CustomerID`, and `OrderAmount` for orders placed in the specified month and year.

```

-- Create the customers table

```

```

CREATE TABLE customers (
  CustomerID INT PRIMARY KEY,
  FirstName VARCHAR(50),
  LastName VARCHAR(50),
  Email VARCHAR(100) UNIQUE,
  Phone VARCHAR(20),
  Address VARCHAR(100),
  City VARCHAR(50),
  State VARCHAR(2),
  ZipCode VARCHAR(10)
);

```

-- Insert some sample customers

```
INSERT INTO customers (CustomerID, FirstName, LastName, Email, Phone, Address, City, State, ZipCode)
VALUES
(1, 'John', 'Doe', 'johndoe@example.com', '123-456-7890', '123 Main St', 'Anytown', 'CA', '12345'),
(2, 'Jane', 'Smith', 'janesmith@example.com', '098-765-4321', '456 Elm St', 'Othertown', 'NY', '67890'),
(3, 'Bob', 'Johnson', 'bobjohnson@example.com', '555-123-4567', '789 Oak St', 'Thistown', 'TX',
'34567');
```

-- Create the orders table

```
CREATE TABLE orders (
  OrderID INT PRIMARY KEY,
  CustomerID INT,
  OrderDate DATE DEFAULT CURRENT_DATE,
  TotalAmount DECIMAL(10, 2) CHECK (TotalAmount > 0),
  FOREIGN KEY (CustomerID) REFERENCES customers(CustomerID)
);
```

-- Insert some sample orders

```
INSERT INTO orders (OrderID, CustomerID, OrderDate, TotalAmount)
VALUES
(1, 1, '2022-06-01', 100.00),
(2, 1, '2022-06-15', 200.00),
(3, 2, '2022-06-20', 50.00),
(4, 3, '2022-07-01', 300.00),
(5, 1, '2022-07-10', 150.00),
(6, 2, '2022-08-01', 250.00);
```

-- Query the data

-- Get all customers and their orders

```
SELECT c.CustomerID, c.FirstName, c.LastName, o.OrderID, o.OrderDate, o.TotalAmount
FROM customers c
JOIN orders o ON c.CustomerID = o.CustomerID
ORDER BY c.CustomerID, o.OrderDate;
```

-- Get the total amount spent by each customer

```
SELECT c.CustomerID, c.FirstName, c.LastName, SUM(o.TotalAmount) AS TotalSpent
FROM customers c
JOIN orders o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.FirstName, c.LastName
ORDER BY TotalSpent DESC;
```

-- Get the customer who spent the most

```
SELECT c.CustomerID, c.FirstName, c.LastName, SUM(o.TotalAmount) AS TotalSpent
```

```
FROM customers c
```

```
JOIN orders o ON c.CustomerID = o.CustomerID
```

```
GROUP BY c.CustomerID, c.FirstName, c.LastName
```

```
ORDER BY TotalSpent DESC
```

```
LIMIT 1;
```

Output

CustomerID	FirstName	LastName	OrderID	OrderDate	TotalAmount
1	John	Doe	1	2022-06-01	100
1	John	Doe	2	2022-06-15	200
1	John	Doe	5	2022-07-10	150
2	Jane	Smith	3	2022-06-20	50
2	Jane	Smith	6	2022-08-01	250
3	Bob	Johnson	4	2022-07-01	300

Output

CustomerID	FirstName	LastName	TotalSpent
1	John	Doe	450
2	Jane	Smith	300
3	Bob	Johnson	300

CustomerID	FirstName	LastName	TotalSpent
1	John	Doe	450