**PLSQL\_EXERCISES**

**Exercise 1: Control Structures**

**Scenario – 1:**

The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**Query Coade:**

**For creating database -**

CREATE DATABASE cognizantBank;

USE cognizantBank;

CREATE TABLE customers (

id INT PRIMARY KEY,

name VARCHAR(100),

age INT,

balance DECIMAL(10, 2),

interestRate DECIMAL(4,2),

IsVIP BOOLEAN DEFAULT FALSE

);

CREATE TABLE loans (

loan\_id INT PRIMARY KEY,

customer\_id INT,

due\_date DATE,

FOREIGN KEY (customer\_id) REFERENCES customers(id)

);

INSERT INTO customers (id, name, age, balance, interestRate, IsVIP) VALUES

(1, 'Raj', 65, 12000.00, 8.5, FALSE),

(2, 'Anita', 45, 9500.00, 9.0, FALSE),

(3, 'Prakash', 70, 10500.00, 8.0, FALSE),

(4, 'Meena', 32, 15000.00, 7.5, FALSE),

(5, 'Ali', 61, 8000.00, 9.2, FALSE),

(6, 'Tina', 28, 5000.00, 10.0, FALSE);

INSERT INTO loans (loan\_id, customer\_id, due\_date) VALUES

(101, 1, CURDATE() + INTERVAL 10 DAY), -- due in 10 days

(102, 2, CURDATE() + INTERVAL 29 DAY), -- due in 29 days

(103, 3, CURDATE() + INTERVAL 40 DAY), -- due after 30 days

(104, 4, CURDATE() + INTERVAL 5 DAY), -- due in 5 days

(105, 5, CURDATE() - INTERVAL 2 DAY); -- already overdue

SELECT \* FROM customers;

**PL/SQL Block -**

DELIMITER //

CREATE PROCEDURE ApplySeniorDiscount()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE cust\_id INT;

DECLARE cust\_age INT;

DECLARE cur CURSOR FOR

SELECT id, age FROM customers;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO cust\_id, cust\_age;

IF done THEN

LEAVE read\_loop;

END IF;

IF cust\_age > 60 THEN

UPDATE customers

SET interestRate = interestRate - 1.00

WHERE id = cust\_id;

END IF;

END LOOP;

CLOSE cur;

END //

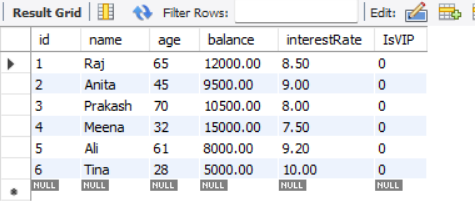
DELIMITER ;

-- To run:

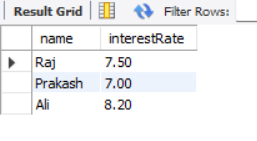
CALL ApplySeniorDiscount();

SELECT name, interestRate FROM customers WHERE age > 60;

**Before Running Query Output Table:**



**After Running Query Output Table:**

****

**Scenario – 2:**

A customer can be promoted to VIP status based on their balance.

**Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**Query Code:**

DELIMITER //

CREATE PROCEDURE PromoteVIPCustomers()

BEGIN

UPDATE customers

SET IsVIP = TRUE

WHERE balance > 10000;

END //

DELIMITER ;

-- To run:

CALL PromoteVIPCustomers();

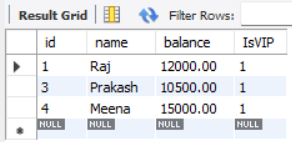
SELECT \* FROM customers;

SELECT id, name, balance, IsVIP

FROM customers

WHERE IsVIP = TRUE;

**After running Query output table:**

****

**Scenario – 3:**

The bank wants to send reminders to customers whose loans are due within the next 30 days.

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**Query Code:**

DELIMITER //

CREATE PROCEDURE SendLoanReminders()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE cust\_id INT;

DECLARE due\_dt DATE;

DECLARE cur CURSOR FOR

SELECT customer\_id, due\_date FROM loans

WHERE due\_date BETWEEN CURDATE() AND DATE\_ADD(CURDATE(), INTERVAL 30 DAY);

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

reminder\_loop: LOOP

FETCH cur INTO cust\_id, due\_dt;

IF done THEN

LEAVE reminder\_loop;

END IF;

SELECT CONCAT('Reminder: Loan for customer ID ', cust\_id, ' is due on ', due\_dt) AS Reminder;

END LOOP;

CLOSE cur;

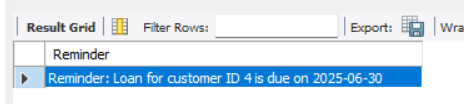
END //

DELIMITER ;

-- To run:

CALL SendLoanReminders();

**After running Query output table:**

****

**PLSQL\_EXERCISES**

**Exercise 2: Stored Procedures**

**Scenario – 1:**

The bank needs to process monthly interest for all savings accounts.

**Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Query Code:**

**For creating database –**

CREATE DATABASE IF NOT EXISTS bank\_cognizant;

USE bank\_cognizant;

CREATE TABLE IF NOT EXISTS SavingsAccount (

account\_id INT PRIMARY KEY,

account\_holder\_name VARCHAR(100),

balance DECIMAL(15,2) CHECK (balance >= 0)

);

CREATE TABLE IF NOT EXISTS Employee (

employee\_id INT PRIMARY KEY,

employee\_name VARCHAR(100),

department VARCHAR(100),

salary DECIMAL(15,2) CHECK (salary >= 0)

);

INSERT INTO SavingsAccount VALUES

(101, 'Riddhi', 6000.00),

(102, 'Arpita', 15000.00),

(103, 'Neelam', 5000.00);

-- Insert sample employees

INSERT INTO Employee VALUES

(1, 'Palak', 'Sales', 30000.00),

(2, 'Mohit', 'IT', 50000.00),

(3, 'Vivek', 'Sales', 40000.00);

**PL/SQL Block -**

DELIMITER //

CREATE PROCEDURE ProcessMonthlyInterest()

BEGIN

UPDATE SavingsAccount

SET balance = balance + (balance \* 0.01);

SELECT 'Monthly interest of 1% has been applied to all accounts.' AS message;

END //

-- Reset delimiter back to normal

DELIMITER ;

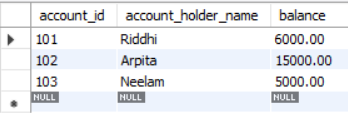
CALL ProcessMonthlyInterest();

SET SQL\_SAFE\_UPDATES = 0;

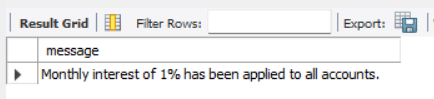
CALL ProcessMonthlyInterest();

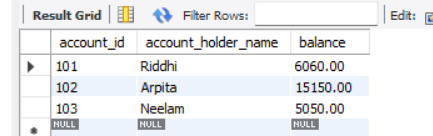
SELECT \* FROM SavingsAccount;

**Before Running Query Output Table:**



**After running Query output table:**





**Scenario – 2:**

The bank wants to implement a bonus scheme for employees based on their performance.

**Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Query Code:**

CREATE PROCEDURE UpdateEmployeeBonus(

IN dept\_name VARCHAR(100),

IN bonus\_percent DECIMAL(5,2)

)

BEGIN

UPDATE Employee

SET salary = salary + (salary \* bonus\_percent / 100)

WHERE department = dept\_name;

SELECT CONCAT('Bonus of ', bonus\_percent, '% applied to department: ', dept\_name) AS message;

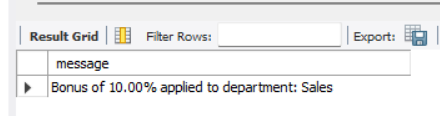
END //

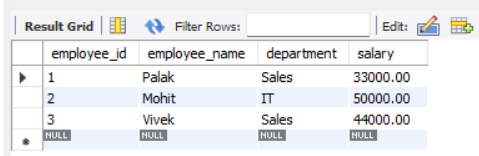
DELIMITER ;

CALL UpdateEmployeeBonus('Sales', 10.0);

SELECT \* FROM Employee;

**After running Query output table:**





**Scenario – 3:**

Customers should be able to transfer funds between their accounts.

**Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

**Query Code:**

DELIMITER //

CREATE PROCEDURE TransferFunds(

IN from\_account\_id INT,

IN to\_account\_id INT,

IN transfer\_amount DECIMAL(15,2)

)

BEGIN

DECLARE sender\_balance DECIMAL(15,2);

-- Get the balance of the source account

SELECT balance INTO sender\_balance

FROM SavingsAccount

WHERE account\_id = from\_account\_id;

-- Check if the source account has enough balance

IF sender\_balance < transfer\_amount THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Insufficient funds in the source account.';

ELSE

-- Deduct amount from sender

UPDATE SavingsAccount

SET balance = balance - transfer\_amount

WHERE account\_id = from\_account\_id;

-- Add amount to receiver

UPDATE SavingsAccount

SET balance = balance + transfer\_amount

WHERE account\_id = to\_account\_id;

-- Confirmation message

SELECT CONCAT('₹', transfer\_amount, ' transferred from Account ', from\_account\_id,

' to Account ', to\_account\_id) AS message;

END IF;

END //

DELIMITER ;

CALL TransferFunds(101, 102, 2000.00);

SELECT \* FROM SavingsAccount;

**After running Query output table:**

