**NAME : TRISHILA SINGHA ROY**

**COLLEGE : UNIVERSITY OF ENGINEERING AND MANAGEMENT, KOLKATA**

**ENROLLMENT NO : 12021002002016**

**SUPERSET ID : 5016812**

**COGNIZANT DIGITAL NURTURE 3.0 PROGRAM**

**WEEK 2 ANSWERS PLSQL**

**Exercise 1: Control Structures**

**Scenario 1:**

BEGIN

FOR customer IN (SELECT CustomerID, LoanInterestRate FROM Customers WHERE Age > 60) LOOP

UPDATE Customers

SET LoanInterestRate = LoanInterestRate - (LoanInterestRate \* 0.01)

WHERE CustomerID = customer.CustomerID;

END LOOP;

COMMIT;

END;

/

**Scenario 2:**

BEGIN

FOR customer IN (SELECT CustomerID, Balance FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = customer.CustomerID;

END LOOP;

COMMIT;

END;

/

**Scenario 3:**

BEGIN

FOR loan IN (SELECT l.CustomerID, l.DueDate, c.CustomerName FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || loan.CustomerName || ', your loan is due on ' || TO\_CHAR(loan.DueDate, 'DD-MON-YYYY') || '.');

END LOOP;

END;

/

**Exercise 2 : Error Handling**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

insufficient\_funds EXCEPTION;

account\_not\_found EXCEPTION;

v\_balance NUMBER;

BEGIN

-- Check if from account exists and has sufficient funds

SELECT balance INTO v\_balance FROM Accounts WHERE account\_id = p\_from\_account\_id FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

-- Check if to account exists

BEGIN

SELECT balance INTO v\_balance FROM Accounts WHERE account\_id = p\_to\_account\_id FOR UPDATE;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE account\_not\_found;

END;

-- Perform the fund transfer

UPDATE Accounts SET balance = balance - p\_amount WHERE account\_id = p\_from\_account\_id;

UPDATE Accounts SET balance = balance + p\_amount WHERE account\_id = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in the source account.');

WHEN account\_not\_found THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Destination account not found.');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred - ' || SQLERRM);

END;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

employee\_not\_found EXCEPTION;

v\_salary NUMBER;

BEGIN

-- Check if employee exists

BEGIN

SELECT salary INTO v\_salary FROM Employees WHERE employee\_id = p\_employee\_id FOR UPDATE;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE employee\_not\_found;

END;

-- Update the salary

UPDATE Employees SET salary = salary + (salary \* (p\_percentage / 100)) WHERE employee\_id = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN employee\_not\_found THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_employee\_id || ' does not exist.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred - ' || SQLERRM);

END;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_customer\_name IN VARCHAR2,

p\_contact IN VARCHAR2

) AS

customer\_exists EXCEPTION;

BEGIN

-- Check if customer already exists

BEGIN

SELECT customer\_id INTO v\_temp FROM Customers WHERE customer\_id = p\_customer\_id;

RAISE customer\_exists;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Proceed with insertion as no such customer exists

INSERT INTO Customers (customer\_id, customer\_name, contact) VALUES (p\_customer\_id, p\_customer\_name, p\_contact);

COMMIT;

END;

EXCEPTION

WHEN customer\_exists THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customer\_id || ' already exists.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred - ' || SQLERRM);

END;

/

**Exercise 3: Stored Procedures**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

v\_interest\_rate CONSTANT NUMBER := 0.01;

BEGIN

-- Update the balance of all savings accounts by applying the interest rate

UPDATE SavingsAccounts

SET balance = balance + (balance \* v\_interest\_rate);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred - ' || SQLERRM);

END;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department\_id IN NUMBER,

p\_bonus\_percentage IN NUMBER

) AS

BEGIN

-- Update the salary of employees in the specified department by adding the bonus percentage

UPDATE Employees

SET salary = salary + (salary \* (p\_bonus\_percentage / 100))

WHERE department\_id = p\_department\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred - ' || SQLERRM);

END;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

insufficient\_funds EXCEPTION;

v\_balance NUMBER;

BEGIN

-- Check if the source account has sufficient balance

SELECT balance INTO v\_balance FROM Accounts WHERE account\_id = p\_from\_account\_id FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

-- Perform the fund transfer

UPDATE Accounts SET balance = balance - p\_amount WHERE account\_id = p\_from\_account\_id;

UPDATE Accounts SET balance = balance + p\_amount WHERE account\_id = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in the source account.');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred - ' || SQLERRM);

END;

/

**Exercise 4: Functions**

**Scenario 1:**

CREATE OR REPLACE FUNCTION CalculateAge (

p\_date\_of\_birth DATE

) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

v\_age := TRUNC((SYSDATE - p\_date\_of\_birth) / 365.25);

RETURN v\_age;

END;

/

**Scenario 2:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_duration\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER;

v\_number\_of\_payments NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

v\_monthly\_rate := p\_interest\_rate / 12 / 100;

v\_number\_of\_payments := p\_duration\_years \* 12;

v\_monthly\_installment := p\_loan\_amount \* v\_monthly\_rate / (1 - POWER(1 + v\_monthly\_rate, -v\_number\_of\_payments));

RETURN v\_monthly\_installment;

END;

/

**Scenario 3:**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance FROM Accounts WHERE account\_id = p\_account\_id;

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

END;

/

**Exercise 5: Triggers**

**Scenario 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Scenario 2:**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, AccountID, TransactionType, Amount, TransactionDate)

VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.TransactionType, :NEW.Amount, :NEW.TransactionDate);

END;

/

**Scenario 3:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

-- Check if the transaction is a withdrawal

IF :NEW.TransactionType = 'WITHDRAWAL' THEN

-- Get the current balance of the account

SELECT balance INTO v\_balance FROM Accounts WHERE account\_id = :NEW.AccountID FOR UPDATE;

-- Ensure withdrawals do not exceed the balance

IF :NEW.Amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance for the withdrawal.');

END IF;

ELSIF :NEW.TransactionType = 'DEPOSIT' THEN

-- Ensure deposits are positive

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be positive.');

END IF;

END IF;

END;

/

**Exercise 6: Cursors**

**Scenario 1:**

DECLARE

CURSOR cur\_transactions IS

SELECT t.CustomerID, t.TransactionID, t.TransactionType, t.Amount, t.TransactionDate

FROM Transactions t

WHERE EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE)

ORDER BY t.CustomerID;

v\_customer\_id Customers.CustomerID%TYPE;

v\_transaction\_id Transactions.TransactionID%TYPE;

v\_transaction\_type Transactions.TransactionType%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_transaction\_date Transactions.TransactionDate%TYPE;

BEGIN

OPEN cur\_transactions;

LOOP

FETCH cur\_transactions INTO v\_customer\_id, v\_transaction\_id, v\_transaction\_type, v\_amount, v\_transaction\_date;

EXIT WHEN cur\_transactions%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id || ', Transaction ID: ' || v\_transaction\_id ||

', Type: ' || v\_transaction\_type || ', Amount: ' || v\_amount ||

', Date: ' || TO\_CHAR(v\_transaction\_date, 'DD-MON-YYYY'));

END LOOP;

CLOSE cur\_transactions;

END;

/

**Scenario 2:**

DECLARE

CURSOR cur\_accounts IS

SELECT AccountID, Balance

FROM Accounts

FOR UPDATE;

v\_account\_id Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_annual\_fee CONSTANT NUMBER := 50; -- Example annual fee

BEGIN

OPEN cur\_accounts;

LOOP

FETCH cur\_accounts INTO v\_account\_id, v\_balance;

EXIT WHEN cur\_accounts%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee

WHERE AccountID = v\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Applied annual fee to Account ID: ' || v\_account\_id || '. New Balance: ' || (v\_balance - v\_annual\_fee));

END LOOP;

CLOSE cur\_accounts;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END;

/

**Scenario 3:**

DECLARE

CURSOR cur\_loans IS

SELECT LoanID, InterestRate

FROM Loans

FOR UPDATE;

v\_loan\_id Loans.LoanID%TYPE;

v\_interest\_rate Loans.InterestRate%TYPE;

v\_new\_interest\_rate NUMBER;

BEGIN

OPEN cur\_loans;

LOOP

FETCH cur\_loans INTO v\_loan\_id, v\_interest\_rate;

EXIT WHEN cur\_loans%NOTFOUND;

-- Example policy: New interest rate is 90% of the current interest rate

v\_new\_interest\_rate := v\_interest\_rate \* 0.9;

UPDATE Loans

SET InterestRate = v\_new\_interest\_rate

WHERE LoanID = v\_loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Updated Loan ID: ' || v\_loan\_id || ' to new interest rate: ' || v\_new\_interest\_rate);

END LOOP;

CLOSE cur\_loans;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END;

/

**Exercise 7: Packages**

**Scenario 1:**

**Package Specification: CustomerManagement**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(

p\_customer\_id IN NUMBER,

p\_customer\_name IN VARCHAR2,

p\_contact IN VARCHAR2

);

PROCEDURE UpdateCustomerDetails(

p\_customer\_id IN NUMBER,

p\_customer\_name IN VARCHAR2,

p\_contact IN VARCHAR2

);

FUNCTION GetCustomerBalance(

p\_customer\_id IN NUMBER

) RETURN NUMBER;

END CustomerManagement;

/

**Package Body: CustomerManagement**

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(

p\_customer\_id IN NUMBER,

p\_customer\_name IN VARCHAR2,

p\_contact IN VARCHAR2

) IS

BEGIN

INSERT INTO Customers (customer\_id, customer\_name, contact)

VALUES (p\_customer\_id, p\_customer\_name, p\_contact);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails(

p\_customer\_id IN NUMBER,

p\_customer\_name IN VARCHAR2,

p\_contact IN VARCHAR2

) IS

BEGIN

UPDATE Customers

SET customer\_name = p\_customer\_name, contact = p\_contact

WHERE customer\_id = p\_customer\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(

p\_customer\_id IN NUMBER

) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT SUM(balance) INTO v\_balance

FROM Accounts

WHERE customer\_id = p\_customer\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2:**

**Package Specification: EmployeeManagement**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireNewEmployee(

p\_employee\_id IN NUMBER,

p\_employee\_name IN VARCHAR2,

p\_department\_id IN NUMBER,

p\_salary IN NUMBER

);

PROCEDURE UpdateEmployeeDetails(

p\_employee\_id IN NUMBER,

p\_employee\_name IN VARCHAR2,

p\_department\_id IN NUMBER,

p\_salary IN NUMBER

);

FUNCTION CalculateAnnualSalary(

p\_employee\_id IN NUMBER

) RETURN NUMBER;

END EmployeeManagement;

/

**Package Body: EmployeeManagement**

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireNewEmployee(

p\_employee\_id IN NUMBER,

p\_employee\_name IN VARCHAR2,

p\_department\_id IN NUMBER,

p\_salary IN NUMBER

) IS

BEGIN

INSERT INTO Employees (employee\_id, employee\_name, department\_id, salary)

VALUES (p\_employee\_id, p\_employee\_name, p\_department\_id, p\_salary);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END HireNewEmployee;

PROCEDURE UpdateEmployeeDetails(

p\_employee\_id IN NUMBER,

p\_employee\_name IN VARCHAR2,

p\_department\_id IN NUMBER,

p\_salary IN NUMBER

) IS

BEGIN

UPDATE Employees

SET employee\_name = p\_employee\_name, department\_id = p\_department\_id, salary = p\_salary

WHERE employee\_id = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(

p\_employee\_id IN NUMBER

) RETURN NUMBER IS

v\_monthly\_salary NUMBER;

v\_annual\_salary NUMBER;

BEGIN

SELECT salary INTO v\_monthly\_salary

FROM Employees

WHERE employee\_id = p\_employee\_id;

v\_annual\_salary := v\_monthly\_salary \* 12;

RETURN v\_annual\_salary;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

RETURN NULL;

END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3:**

**Package Specification: AccountOperations**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount(

p\_account\_id IN NUMBER,

p\_customer\_id IN NUMBER,

p\_initial\_balance IN NUMBER

);

PROCEDURE CloseAccount(

p\_account\_id IN NUMBER

);

FUNCTION GetTotalBalance(

p\_customer\_id IN NUMBER

) RETURN NUMBER;

END AccountOperations;

/

**Package Body: AccountOperations**

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenNewAccount(

p\_account\_id IN NUMBER,

p\_customer\_id IN NUMBER,

p\_initial\_balance IN NUMBER

) IS

BEGIN

INSERT INTO Accounts (account\_id, customer\_id, balance)

VALUES (p\_account\_id, p\_customer\_id, p\_initial\_balance);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END OpenNewAccount;

PROCEDURE CloseAccount(

p\_account\_id IN NUMBER

) IS

BEGIN

DELETE FROM Accounts

WHERE account\_id = p\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END CloseAccount;

FUNCTION GetTotalBalance(

p\_customer\_id IN NUMBER

) RETURN NUMBER IS

v\_total\_balance NUMBER;

BEGIN

SELECT SUM(balance) INTO v\_total\_balance

FROM Accounts

WHERE customer\_id = p\_customer\_id;

RETURN v\_total\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

RETURN NULL;

END GetTotalBalance;

END AccountOperations;

/