NAME- SRIJITA SARKAR

CANDIDATE ID (SUPERSET ID) – 5017615

WEEK-2: PLSQL

**Exercise 1: Control Structures**

**Scenario 1:**

DECLARE

CURSOR customer\_cursor IS

SELECT customer\_id, loan\_interest\_rate

FROM customers

WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM date\_of\_birth) > 60;

v\_customer\_id customers.customer\_id%TYPE;

v\_loan\_interest\_rate customers.loan\_interest\_rate%TYPE;

BEGIN

OPEN customer\_cursor;

LOOP

FETCH customer\_cursor INTO v\_customer\_id, v\_loan\_interest\_rate;

EXIT WHEN customer\_cursor%NOTFOUND;

UPDATE customers

SET loan\_interest\_rate = v\_loan\_interest\_rate \* 0.99

WHERE customer\_id = v\_customer\_id;

END LOOP;

CLOSE customer\_cursor;

COMMIT;

END;

**Scenario 2:**

DECLARE

CURSOR customer\_cursor IS

SELECT customer\_id, balance

FROM customers

WHERE balance > 10000;

v\_customer\_id customers.customer\_id%TYPE;

BEGIN

OPEN customer\_cursor;

LOOP

FETCH customer\_cursor INTO v\_customer\_id;

EXIT WHEN customer\_cursor%NOTFOUND;

UPDATE customers

SET is\_vip = TRUE

WHERE customer\_id = v\_customer\_id;

END LOOP;

CLOSE customer\_cursor;

COMMIT;

END;

**Scenario 3:**

DECLARE

CURSOR loan cursor IS

SELECT customer id, due date

FROM loans

WHERE due date BETWEEN SYSDATE AND SYSDATE + 30;

v\_customer\_id loans.customer\_id%TYPE;

v\_due\_date loans.due\_date%TYPE;

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_customer\_id, v\_due\_date;

EXIT WHEN loan\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear customer ' || v\_customer\_id || ', your loan is due on ' || TO\_CHAR(v\_due\_date, 'YYYY-MM-DD') || '.');

END LOOP;

CLOSE loan\_cursor;

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

)

IS

insufficient\_funds EXCEPTION;

v\_balance\_from NUMBER;

BEGIN

-- Check the balance of the source account

SELECT balance INTO v\_balance\_from

FROM accounts

WHERE account\_id = p\_from\_account\_id

FOR UPDATE;

IF v\_balance\_from < p\_amount THEN

RAISE insufficient\_funds;

END IF;

-- Deduct from the source account

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account\_id;

-- Add to the destination account

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 during the transfer.');

END;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

)

IS

employee\_not\_found EXCEPTION;

v\_current\_salary NUMBER;

BEGIN

-- Check if employee exists and get current salary

SELECT salary INTO v\_current\_salary

FROM employees

WHERE employee\_id = p\_employee\_id;

-- Increase the salary by the given percentage

UPDATE employees

SET salary = salary + (v\_current\_salary \* p\_percentage / 100)

WHERE employee\_id = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred while updating the salary.');

END;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_date\_of\_birth IN DATE

)

IS

customer\_already\_exists EXCEPTION;

BEGIN

-- Insert new customer

INSERT INTO customers (customer\_id, name, date\_of\_birth)

VALUES (p\_customer\_id, p\_name, p\_date\_of\_birth);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred while adding the new customer.');

END;

/

**Exercise 3: Stored Procedures**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

v\_interest\_rate CONSTANT NUMBER := 0.01;

BEGIN

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

UPDATE accounts

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

WHERE account\_type = 'Savings';

COMMIT;

END;

/

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department\_id IN NUMBER,

p\_bonus\_percentage IN NUMBER

)

IS

BEGIN

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

UPDATE employees

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

WHERE department\_id = p\_department\_id;

COMMIT;

END;

/

**Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

insufficient\_funds EXCEPTION;

v\_balance\_from NUMBER;

BEGIN

-- Check the balance of the source account

SELECT balance INTO v\_balance\_from

FROM accounts

WHERE account\_id = p\_from\_account\_id

FOR UPDATE;

IF v\_balance\_from < p\_amount THEN

RAISE insufficient\_funds;

END IF;

-- Deduct from the source account

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account\_id;

-- Add to the destination account

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 during the transfer.');

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(MONTHS\_BETWEEN(SYSDATE, p\_date\_of\_birth) / 12);

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;

IF v\_monthly\_rate = 0 THEN

v\_monthly\_installment := p\_loan\_amount / v\_number\_of\_payments;

ELSE

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

END IF;

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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

**Exercise 5: Triggers**

**Scenario 1:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON customers

FOR EACH ROW

BEGIN

:NEW.last\_modified := SYSDATE;

END;

/

**Scenario 2:**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON transactions

FOR EACH ROW

BEGIN

INSERT INTO audit\_log (transaction\_id, account\_id, transaction\_type, amount, transaction\_date)

VALUES (:NEW.transaction\_id, :NEW.account\_id, :NEW.transaction\_type, :NEW.amount, :NEW.transaction\_date);

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.transaction\_type = 'Withdrawal' THEN

-- Get the current balance of the account

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = :NEW.account\_id

FOR UPDATE;

-- Ensure the withdrawal does not exceed the balance

IF :NEW.amount > v\_balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error: Withdrawal amount exceeds account balance.');

END IF;

END IF;

-- Check if the transaction is a deposit

IF :NEW.transaction\_type = 'Deposit' THEN

-- Ensure the deposit amount is positive

IF :NEW.amount <= 0 THEN

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

END IF;

END IF;

END;

/

**Exercise 6: Cursors**

**Scenario 1:**

DECLARE

CURSOR transaction\_cursor IS

SELECT customer\_id, transaction\_date, transaction\_type, amount

FROM transactions

WHERE TRUNC(transaction\_date, 'MM') = TRUNC(SYSDATE, 'MM');

v\_customer\_id transactions.customer\_id%TYPE;

v\_transaction\_date transactions.transaction\_date%TYPE;

v\_transaction\_type transactions.transaction\_type%TYPE;

v\_amount transactions.amount%TYPE;

BEGIN

OPEN transaction\_cursor;

LOOP

FETCH transaction\_cursor INTO v\_customer\_id, v\_transaction\_date, v\_transaction\_type, v\_amount;

EXIT WHEN transaction\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id);

DBMS\_OUTPUT.PUT\_LINE('Date: ' || TO\_CHAR(v\_transaction\_date, 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Type: ' || v\_transaction\_type);

DBMS\_OUTPUT.PUT\_LINE('Amount: ' || v\_amount);

DBMS\_OUTPUT.PUT\_LINE('-----------------------------------');

END LOOP;

CLOSE transaction\_cursor;

END;

/

**Scenario 2:**

DECLARE

CURSOR account\_cursor IS

SELECT account\_id, balance

FROM accounts;

v\_account\_id accounts.account\_id%TYPE;

v\_balance accounts.balance%TYPE;

v\_annual\_fee CONSTANT NUMBER := 50; -- Define the annual fee amount

BEGIN

OPEN account\_cursor;

LOOP

FETCH account\_cursor INTO v\_account\_id, v\_balance;

EXIT WHEN account\_cursor%NOTFOUND;

-- Deduct the annual fee from the account balance

UPDATE accounts

SET balance = balance - v\_annual\_fee

WHERE account\_id = v\_account\_id;

END LOOP;

CLOSE account\_cursor;

COMMIT;

END;

/

**Scenario 3:**

DECLARE

CURSOR loan\_cursor IS

SELECT loan\_id, interest\_rate

FROM loans;

v\_loan\_id loans.loan\_id%TYPE;

v\_interest\_rate loans.interest\_rate%TYPE;

v\_new\_interest\_rate CONSTANT NUMBER := 0.05; -- Define the new interest rate

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_loan\_id, v\_interest\_rate;

EXIT WHEN loan\_cursor%NOTFOUND;

-- Update the interest rate for the loan based on the new policy

UPDATE loans

SET interest\_rate = v\_new\_interest\_rate

WHERE loan\_id = v\_loan\_id;

END LOOP;

CLOSE loan\_cursor;

COMMIT;

END;

/

**Exercise 7: Packages**

**Scenario 1:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_date\_of\_birth IN DATE);

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_date\_of\_birth IN DATE);

FUNCTION GetCustomerBalance(p\_customer\_id IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_date\_of\_birth IN DATE) IS

BEGIN

INSERT INTO customers (customer\_id, name, date\_of\_birth)

VALUES (p\_customer\_id, p\_name, p\_date\_of\_birth);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX 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 while adding the new customer.');

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_date\_of\_birth IN DATE) IS

BEGIN

UPDATE customers

SET name = p\_name, date\_of\_birth = p\_date\_of\_birth

WHERE customer\_id = p\_customer\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred while updating customer details.');

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: An unexpected error occurred while retrieving customer balance.');

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

/  
**Scenario 2:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_department\_id IN NUMBER, p\_salary IN NUMBER);

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_department\_id IN NUMBER, p\_salary IN NUMBER);

FUNCTION CalculateAnnualSalary(p\_employee\_id IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_department\_id IN NUMBER, p\_salary IN NUMBER) IS

BEGIN

INSERT INTO employees (employee\_id, name, department\_id, salary)

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred while hiring the new employee.');

END HireEmployee;

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_department\_id IN NUMBER, p\_salary IN NUMBER) IS

BEGIN

UPDATE employees

SET name = p\_name, department\_id = p\_department\_id, salary = p\_salary

WHERE employee\_id = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_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 while updating employee details.');

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(p\_employee\_id IN NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT salary INTO v\_salary

FROM employees

WHERE employee\_id = p\_employee\_id;

RETURN v\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred while calculating annual salary.');

RETURN NULL;

END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount(p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER, p\_balance IN NUMBER, p\_account\_type IN VARCHAR2);

PROCEDURE CloseAccount(p\_account\_id IN NUMBER);

FUNCTION GetTotalBalance(p\_customer\_id IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenNewAccount(p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER, p\_balance IN NUMBER, p\_account\_type IN VARCHAR2) IS

BEGIN

INSERT INTO accounts (account\_id, customer\_id, balance, account\_type)

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred while opening the new account.');

END OpenNewAccount;

PROCEDURE CloseAccount(p\_account\_id IN NUMBER) IS

BEGIN

DELETE FROM accounts

WHERE account\_id = p\_account\_id;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: An unexpected error occurred while closing the account.');

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: An unexpected error occurred while retrieving total balance.');

RETURN NULL;

END GetTotalBalance;

END AccountOperations;

/