WEEK 2:

**Exercise 1: Control Structures**

**Scenario 1: Discount for Senior Citizens**

**Task**: Apply a **1% discount** to loan interest rates for customers **above 60 years** old.

sql

CopyEdit

DECLARE

CURSOR cur\_customers IS

SELECT customer\_id, loan\_id, interest\_rate

FROM customers

JOIN loans USING (customer\_id)

WHERE age > 60;

BEGIN

FOR rec IN cur\_customers LOOP

UPDATE loans

SET interest\_rate = interest\_rate - 1

WHERE loan\_id = rec.loan\_id;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Interest rate discount applied for customers above 60.');

END;

**Scenario 2: Promote to VIP Based on Balance**

**Task**: Set IsVIP = TRUE for customers with a balance over **$10,000**.

sql

CopyEdit

DECLARE

CURSOR cur\_balance IS

SELECT customer\_id

FROM customers

WHERE balance > 10000;

BEGIN

FOR rec IN cur\_balance LOOP

UPDATE customers

SET IsVIP = 'TRUE'

WHERE customer\_id = rec.customer\_id;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('VIP status updated for eligible customers.');

END;

**Scenario 3: Reminders for Loan Dues**

**Task**: Fetch all **loans due in the next 30 days** and print reminder messages.

sql

CopyEdit

DECLARE

CURSOR cur\_due\_loans IS

SELECT c.customer\_id, c.customer\_name, l.loan\_id, l.due\_date

FROM customers c

JOIN loans l ON c.customer\_id = l.customer\_id

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR rec IN cur\_due\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.loan\_id ||

' for customer ' || rec.customer\_name ||

' is due on ' || TO\_CHAR(rec.due\_date, 'DD-MON-YYYY'));

END LOOP;

END;

These blocks assume tables: customers(customer\_id, customer\_name, age, balance, IsVIP) and loans(loan\_id, customer\_id, interest\_rate, due\_date).

**Exercise 3: Stored Procedures**

**Scenario 1: Monthly Interest for Savings Accounts**

**Procedure Name**: ProcessMonthlyInterest  
**Action**: Apply **1% interest** to all savings account balances.

sql

CopyEdit

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE accounts

SET balance = balance + (balance \* 0.01)

WHERE account\_type = 'SAVINGS';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all savings accounts.');

END;

/

**Scenario 2: Bonus Scheme for Employees**

**Procedure Name**: UpdateEmployeeBonus  
**Action**: Update salary by a **bonus percentage** for a **given department**.

sql

CopyEdit

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept\_id IN NUMBER,

p\_bonus\_pct IN NUMBER

) AS

BEGIN

UPDATE employees

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

WHERE department\_id = p\_dept\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to employees in department ' || p\_dept\_id);

END;

/

**Example Call**:

sql

CopyEdit

EXEC UpdateEmployeeBonus(101, 10); -- 10% bonus for department 101

**Scenario 3: Fund Transfer Between Accounts**

**Procedure Name**: TransferFunds  
 **Action**: Transfer amount from source account to destination, after balance check.

sql

CopyEdit

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_source\_acc IN NUMBER,

p\_dest\_acc IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

-- Check if source has enough balance

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = p\_source\_acc;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

END IF;

-- Deduct from source

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_source\_acc;

-- Credit to destination

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_dest\_acc;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || p\_amount || ' completed from Account ' || p\_source\_acc || ' to Account ' || p\_dest\_acc);

END;

/

**Example Call**:

sql

CopyEdit

EXEC TransferFunds(1001, 1002, 500);

accounts(account\_id, account\_type, balance)

employees(employee\_id, department\_id, salary)

account\_type = 'SAVINGS'