**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.

**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.

**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

Solution:

Scenario 1: Apply 1% discount for customers above 60:

Code:

BEGIN

FOR cust\_rec IN (SELECT customer\_id, age, interest\_rate FROM customers) LOOP

IF cust\_rec.age > 60 THEN

UPDATE customers

SET interest\_rate = interest\_rate - 1

WHERE customer\_id = cust\_rec.customer\_id;

END IF;

END LOOP;

COMMIT;

END;

Output Example:

| **Customer ID** | **Age** | **Old Interest Rate** | **New Interest Rate** |
| --- | --- | --- | --- |
| 101 | 65 | 8.5% | 7.5% |
| 102 | 59 | 9.0% | 9.0% |
| 103 | 70 | 7.0% | 6.0% |

Scenario 2: Promote to VIP based on balance

PL/SQL Code

BEGIN

FOR cust\_rec IN (SELECT customer\_id, balance FROM customers) LOOP

IF cust\_rec.balance > 10000 THEN

UPDATE customers

SET is\_vip = 'TRUE'

WHERE customer\_id = cust\_rec.customer\_id;

END IF;

END LOOP;

COMMIT;

END;

Output Example:

| **Customer ID** | **Balance ($)** | **IsVIP Before** | **IsVIP After** |
| --- | --- | --- | --- |
| 101 | 12000 | FALSE | TRUE |
| 102 | 8000 | FALSE | FALSE |
| 103 | 15000 | FALSE | TRUE |

Scenario 3: Reminders for due loans in next 30 days

PL/SQL Code:

DECLARE

v\_due\_date DATE;

BEGIN

FOR loan\_rec IN (

SELECT customer\_id, loan\_id, due\_date

FROM loans

WHERE due\_date <= SYSDATE + 30

) LOOP

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

' for Customer ' || loan\_rec.customer\_id ||

' is due on ' || TO\_CHAR(loan\_rec.due\_date, 'DD-Mon-YYYY'));

END LOOP;

END;

Output Example:

Reminder: Loan ID L001 for Customer 101 is due on 05-Jul-2025

Reminder: Loan ID L004 for Customer 103 is due on 15-Jul-2025

Reminder: Loan ID L007 for Customer 109 is due on 28-Jun-2025

**Exercise 3: 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.

**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.

**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.

Solution:

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

**Stored Procedure Code**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (SELECT account\_id, balance FROM accounts WHERE account\_type = 'SAVINGS') LOOP

UPDATE accounts

SET balance = balance + (acc.balance \* 0.01)

WHERE account\_id = acc.account\_id;

END LOOP;

COMMIT;

END;

Output Example:

| **Account ID** | **Account Type** | **Balance (Before)** | **Balance (After 1% Interest)** |
| --- | --- | --- | --- |
| A001 | SAVINGS | 10,000 | 10,100 |
| A002 | SAVINGS | 20,000 | 20,200 |
| A003 | CURRENT | 15,000 | 15,000 (no change) |

**Scenario 2: Add Bonus for Employees in a Department**

**Stored Procedure Code**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_id IN NUMBER,

bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE employees

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

WHERE department\_id = dept\_id;

COMMIT;

END;

Call:

EXEC UpdateEmployeeBonus(10, 10);

Output:

| **Employee ID** | **Department ID** | **Salary (Before)** | **Salary (After 10% Bonus)** |
| --- | --- | --- | --- |
| E001 | 10 | 50,000 | 55,000 |
| E002 | 20 | 60,000 | 60,000 (no change) |
| E003 | 10 | 70,000 | 77,000 |

**Scenario 3: Transfer Funds Between Accounts**

**Stored Procedure Code**

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_acc\_id IN VARCHAR2,

to\_acc\_id IN VARCHAR2,

amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance FROM accounts WHERE account\_id = from\_acc\_id;

IF v\_balance >= amount THEN

-- Deduct from source

UPDATE accounts SET balance = balance - amount WHERE account\_id = from\_acc\_id;

-- Add to target

UPDATE accounts SET balance = balance + amount WHERE account\_id = to\_acc\_id;

COMMIT;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in source account.');

END IF;

END;

Call:

EXEC TransferFunds('A001', 'A002', 5000);

Output:

| **Account ID** | **Balance Before** | **Balance After** |
| --- | --- | --- |
| A001 | 10,000 | 5,000 |
| A002 | 15,000 | 20,000 |