### **EXERCISE 1**

### **Scenario 1: Apply a Discount to Loan Interest Rates for Customers Above 60 Years Old**

DECLARE

CURSOR c\_customers IS

SELECT CustomerID, DOB

FROM Customers;

v\_customer\_id NUMBER;

v\_dob DATE;

v\_age NUMBER;

BEGIN

OPEN c\_customers;

LOOP

FETCH c\_customers INTO v\_customer\_id, v\_dob;

EXIT WHEN c\_customers%NOTFOUND;

-- Calculate age

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, v\_dob) / 12);

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate \* 0.99 -- 1% discount

WHERE CustomerID = v\_customer\_id;

END IF;

END LOOP;

CLOSE c\_customers;

END;

/

**Scenario 2: Promote Customers to VIP Status Based on Their Balance**

DECLARE

CURSOR c\_customers IS

SELECT CustomerID, Balance

FROM Customers;

v\_customer\_id NUMBER;

v\_balance NUMBER;

BEGIN

OPEN c\_customers;

LOOP

FETCH c\_customers INTO v\_customer\_id, v\_balance;

EXIT WHEN c\_customers%NOTFOUND;

IF v\_balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = v\_customer\_id;

END IF;

END LOOP;

CLOSE c\_customers;

END;

/

**Scenario 3: Send Reminders to Customers Whose Loans Are Due Within the Next 30 Days**

DECLARE

CURSOR c\_loans IS

SELECT CustomerID, LoanID

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;

v\_customer\_id NUMBER;

v\_loan\_id NUMBER;

BEGIN

OPEN c\_loans;

LOOP

FETCH c\_loans INTO v\_customer\_id, v\_loan\_id;

EXIT WHEN c\_loans%NOTFOUND;

-- Print reminder message (replace with actual notification logic)

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ' || v\_customer\_id || ' has loan ' || v\_loan\_id || ' due within 30 days.');

END LOOP;

CLOSE c\_loans;

END;

**/**

**EXERCISE 2**

### **Scenario 1: Handle Exceptions During Fund Transfers Between Accounts**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

source\_account\_id NUMBER,

destination\_account\_id NUMBER,

amount NUMBER

)

IS

v\_source\_balance NUMBER;

BEGIN

BEGIN

-- Check source account balance

SELECT balance INTO v\_source\_balance

FROM accounts

WHERE account\_id = source\_account\_id;

IF v\_source\_balance < amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

END IF;

-- Update account balances

UPDATE accounts

SET balance = balance - amount

WHERE account\_id = source\_account\_id;

UPDATE accounts

SET balance = balance + amount

WHERE account\_id = destination\_account\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

-- Log error (replace with your logging mechanism)

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

END;

END;

/

### **Scenario 2: Manage Errors When Updating Employee Salaries**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) IS

employee\_not\_found EXCEPTION;

PRAGMA EXCEPTION\_INIT(employee\_not\_found, -20002);

BEGIN

-- Update the salary

UPDATE employees

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

WHERE employee\_id = p\_employee\_id;

-- Check if the update was successful

IF SQL%ROWCOUNT = 0 THEN

RAISE employee\_not\_found;

END IF;

-- Commit the transaction

COMMIT;

EXCEPTION

WHEN employee\_not\_found THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID not found.');

WHEN OTHERS THEN

ROLLBACK;

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

END UpdateSalary;

/

### **Scenario 3: Ensure Data Integrity When Adding a New Customer**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_customer\_name IN VARCHAR2,

p\_customer\_age IN NUMBER

) IS

duplicate\_customer EXCEPTION;

PRAGMA EXCEPTION\_INIT(duplicate\_customer, -20003);

BEGIN

-- Insert the new customer

INSERT INTO customers (customer\_id, customer\_name, customer\_age)

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

-- Commit the transaction

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: A customer with this ID already exists.');

WHEN OTHERS THEN

ROLLBACK;

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

END AddNewCustomer;

/

**EXERCISE 3**

### **Scenario 1: ProcessMonthlyInterest**

CREATE PROCEDURE ProcessMonthlyInterest

AS

BEGIN

DECLARE @InterestRate DECIMAL(18,2) = 0.01; -- 1% interest rate

UPDATE SavingsAccounts

SET Balance = Balance + (Balance \* @InterestRate);

END;

### **Scenario 2: UpdateEmployeeBonus**

CREATE PROCEDURE UpdateEmployeeBonus

@DepartmentID INT,

@BonusPercentage DECIMAL(18,2)

AS

BEGIN

DECLARE @BonusMultiplier DECIMAL(18,2) = 1 + @BonusPercentage / 100;

UPDATE Employees

SET Salary = Salary \* @BonusMultiplier

WHERE DepartmentID = @DepartmentID;

END;

**Scenario 3: TransferFunds**

CREATE PROCEDURE TransferFunds

@SourceAccountID INT,

@DestinationAccountID INT,

@Amount DECIMAL(18,2)

AS

BEGIN

BEGIN TRANSACTION;

-- Check source account balance

DECLARE @SourceBalance DECIMAL(18,2);

SELECT @SourceBalance = Balance FROM Accounts WHERE AccountID = @SourceAccountID;

IF @SourceBalance < @Amount

BEGIN

RAISERROR('Insufficient funds in source account.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END;

-- Update source account

UPDATE Accounts

SET Balance = Balance - @Amount

WHERE AccountID = @SourceAccountID;

-- Update destination account

UPDATE Accounts

SET Balance = Balance + @Amount

WHERE AccountID = @DestinationAccountID;

COMMIT TRANSACTION;

END;

**EXERCISE 4**

### **Scenario 1: Calculate Age**

CREATE OR REPLACE FUNCTION CalculateAge (p\_dob DATE)

RETURN NUMBER

IS

v\_age NUMBER;

BEGIN

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END;

/

**Scenario 2: Calculate Monthly Instalment**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_loan\_duration\_years NUMBER

)

RETURN NUMBER

IS

v\_monthly\_interest\_rate NUMBER;

v\_number\_of\_payments NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

v\_monthly\_interest\_rate := p\_interest\_rate / 1200;

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

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

(POWER(1 + v\_monthly\_interest\_rate, v\_number\_of\_payments) - 1);

RETURN v\_monthly\_installment;

END;

/

**Scenario 3: Check Sufficient Balance**

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;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE; -- Or handle the exception differently

END;

/

**EXERCISE 5**

**Scenario 1: Update last modified date**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

AFTER UPDATE ON Customers

FOR EACH ROW

BEGIN

UPDATE Customers

SET LastModified = SYSDATE

WHERE CustomerID = :OLD.CustomerID;

END;

/

**Scenario 2: Maintain an audit log**

CREATE TABLE AuditLog (

AuditID NUMBER GENERATED ALWAYS AS IDENTITY,

TableName VARCHAR2(30),

PrimaryKeyValue NUMBER,

Operation VARCHAR2(10),

OldData VARCHAR2(4000),

NewData VARCHAR2(4000),

Timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT OR UPDATE OR DELETE ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TableName, PrimaryKeyValue, Operation, OldData, NewData)

VALUES (

'Transactions',

:OLD.TransactionID, -- Use :NEW.TransactionID for INSERT

CASE WHEN INSERTING THEN 'INSERT'

WHEN UPDATING THEN 'UPDATE'

WHEN DELETING THEN 'DELETE'

END,

TO\_CHAR(:OLD),

TO\_CHAR(:NEW)

);

END;

/

**Scenario 3: Enforce business rules on deposits and withdrawals**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT OR UPDATE ON Transactions

FOR EACH ROW

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

IF :NEW.Amount > (SELECT Balance FROM Accounts WHERE AccountID = :NEW.AccountID) THEN

RAISE\_APPLICATION\_ERROR(-20005, 'Insufficient funds');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20006, 'Invalid deposit amount');

END IF;

END IF;

END;

/

**EXERCISE 6**

**Scenario 1: Generate Monthly Statements**

DECLARE

CURSOR c\_accounts IS

SELECT CustomerID, AccountID

FROM Accounts;

v\_customer\_id NUMBER;

v\_account\_id NUMBER;

v\_transaction\_date DATE;

v\_amount NUMBER;

v\_transaction\_type VARCHAR2(10);

BEGIN

OPEN c\_accounts;

LOOP

FETCH c\_accounts INTO v\_customer\_id, v\_account\_id;

EXIT WHEN c\_accounts%NOTFOUND;

-- Generate statement logic here

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

FOR r\_transaction IN (

SELECT TransactionDate, Amount, TransactionType

FROM Transactions

WHERE AccountID = v\_account\_id

AND TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(TRUNC(SYSDATE, 'MM'))

)

LOOP

DBMS\_OUTPUT.PUT\_LINE(r\_transaction.TransactionDate || ', ' || r\_transaction.Amount || ', ' || r\_transaction.TransactionType);

END LOOP;

-- Calculate balance, fees, etc. and print summary

END LOOP;

CLOSE c\_accounts;

END;

/

**Scenario 2: Apply Annual Fee**

DECLARE

CURSOR c\_accounts IS

SELECT AccountID, Balance

FROM Accounts;

v\_account\_id NUMBER;

v\_balance NUMBER;

v\_annual\_fee NUMBER := 10; -- Replace with actual fee amount

BEGIN

OPEN c\_accounts;

LOOP

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

EXIT WHEN c\_accounts%NOTFOUND;

IF v\_balance >= v\_annual\_fee THEN

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee

WHERE AccountID = v\_account\_id;

END IF;

END LOOP;

CLOSE c\_accounts;

END;

/

**Scenario 3: Update Loan Interest Rates**

DECLARE

CURSOR c\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

v\_loan\_id NUMBER;

v\_interest\_rate NUMBER;

v\_new\_interest\_rate NUMBER := 5; -- Replace with new interest rate

BEGIN

OPEN c\_loans;

LOOP

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

EXIT WHEN c\_loans%NOTFOUND;

UPDATE Loans

SET InterestRate = v\_new\_interest\_rate

WHERE LoanID = v\_loan\_id;

END LOOP;

CLOSE c\_loans;

END;

/

**EXERCISE 7**

**Scenario 1: Customer Management Package**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2(100),

p\_dob DATE,

p\_balance NUMBER

);

PROCEDURE UpdateCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2(100),

p\_dob DATE,

p\_balance NUMBER

);

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

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2(100),

p\_dob DATE,

p\_balance NUMBER

)

IS

BEGIN

-- Insert customer into Customers table

END AddCustomer;

PROCEDURE UpdateCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2(100),

p\_dob DATE,

p\_balance NUMBER

)

IS

BEGIN

-- Update customer information in Customers table

END UpdateCustomer;

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

IS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance

FROM Customers

WHERE CustomerID = p\_customer\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0; -- Or handle the exception appropriately

END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2: Employee Management Package**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee (

p\_employee\_id NUMBER,

p\_name VARCHAR2(100),

p\_position VARCHAR2(50),

p\_salary NUMBER,

p\_department VARCHAR2(50),

p\_hire\_date DATE

);

PROCEDURE UpdateEmployee (

p\_employee\_id NUMBER,

p\_name VARCHAR2(100),

p\_position VARCHAR2(50),

p\_salary NUMBER,

p\_department VARCHAR2(50)

);

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

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee (

-- ...

)

IS

BEGIN

-- Insert employee into Employees table

END HireEmployee;

PROCEDURE UpdateEmployee (

-- ...

)

IS

BEGIN

-- Update employee information in Employees table

END UpdateEmployee;

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

IS

v\_salary NUMBER;

BEGIN

SELECT salary INTO v\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

RETURN v\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0; -- Or handle the exception appropriately

END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3: Account Operations Package**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount (

p\_account\_id NUMBER,

p\_customer\_id NUMBER,

p\_account\_type VARCHAR2(20),

p\_balance NUMBER

);

PROCEDURE CloseAccount (p\_account\_id NUMBER);

FUNCTION GetCustomerTotalBalance (p\_customer\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount (

-- ...

)

IS

BEGIN

-- Insert account into Accounts table

END OpenAccount;

PROCEDURE CloseAccount (

-- ...

)

IS

BEGIN

-- Delete account from Accounts table

END CloseAccount;

FUNCTION GetCustomerTotalBalance (p\_customer\_id NUMBER) RETURN NUMBER

IS

v\_total\_balance NUMBER := 0;

BEGIN

SELECT SUM(balance) INTO v\_total\_balance

FROM Accounts

WHERE CustomerID = p\_customer\_id;

RETURN v\_total\_balance;

END GetCustomerTotalBalance;

END AccountOperations;

**/**