# Exercise 1: Control Structures

Scenario 1: Apply a discount to loan interest rates for customers above 60 years old

DECLARE

CURSOR cust\_cursor IS

SELECT CustomerID, DOB, InterestRate

FROM Loans JOIN Customers ON Loans.CustomerID = Customers.CustomerID;

age NUMBER;

BEGIN

FOR rec IN cust\_cursor LOOP

age := FLOOR(MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12);

IF age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

END;

/

Scenario 2: Promote customers to VIP status based on balance

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = TRUE

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

END;

/

Scenario 3: Send reminders to customers with loans due within the next 30 days

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR rec IN loan\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID || ' for Customer ' || rec.CustomerID || ' is due within 30 days.');

END LOOP;

END;

/

# **Exercise 2: Error Handling**

Scenario 1: Handle exceptions during fund transfers between accounts

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_from\_account\_id NUMBER, p\_to\_account\_id NUMBER, p\_amount NUMBER) IS

insufficient\_funds EXCEPTION;

pragma exception\_init(insufficient\_funds, -20101);

BEGIN

-- Check if the from account has sufficient funds

IF (SELECT Balance FROM Accounts WHERE AccountID = p\_from\_account\_id) < p\_amount THEN

RAISE insufficient\_funds;

END IF;

-- Perform the transfer

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_account\_id;

-- Commit the transaction

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

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

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END SafeTransferFunds;

/

Scenario 2: Manage errors when updating employee salaries

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_employee\_id NUMBER, p\_percentage IN NUMBER) IS

employee\_not\_found EXCEPTION;

pragma exception\_init(employee\_not\_found, -20102);

BEGIN

-- Update the salary

UPDATE Employees

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

WHERE EmployeeID = 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

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

WHEN OTHERS THEN

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 NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

duplicate\_customer EXCEPTION;

pragma exception\_init(duplicate\_customer, -20103);

BEGIN

-- Insert the new customer

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

-- Commit the transaction

COMMIT;

EXCEPTION

WHEN duplicate\_customer THEN

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

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END AddNewCustomer;

/

# Exercise 3: Stored Procedures

Scenario 1: Process monthly interest for all savings accounts

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

END ProcessMonthlyInterest;

/

Scenario 2: Implement a bonus scheme for employees

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_department VARCHAR2, p\_bonus\_percentage NUMBER) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

END UpdateEmployeeBonus;

/

Scenario 3: Transfer funds between accounts

CREATE OR REPLACE PROCEDURE TransferFunds(p\_from\_account\_id NUMBER, p\_to\_account\_id NUMBER, p\_amount NUMBER) IS

insufficient\_balance EXCEPTION;

pragma exception\_init(insufficient\_balance, -20101);

BEGIN

-- Check if the source account has sufficient balance

IF (SELECT Balance FROM Accounts WHERE AccountID = p\_from\_account\_id) < p\_amount THEN

RAISE insufficient\_balance;

END IF;

-- Transfer the funds

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_account\_id;

-- Commit the transaction

COMMIT;

EXCEPTION

WHEN insufficient\_balance THEN

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

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END TransferFunds;

/

# Exercise 4: Functions

Scenario 1: Calculate the age of customers

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

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

RETURN v\_age;

END CalculateAge;

/

Scenario 2: Compute the monthly installment for a loan

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_loan\_amount NUMBER, p\_interest\_rate NUMBER, p\_duration\_years NUMBER) RETURN NUMBER IS

v\_monthly\_installment NUMBER;

BEGIN

v\_monthly\_installment := p\_loan\_amount \* (p\_interest\_rate / 1200) / (1 - POWER(1 + (p\_interest\_rate / 1200), -p\_duration\_years \* 12));

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

/

Scenario 3: Check if a customer has 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 AccountID = p\_account\_id;

RETURN v\_balance >= p\_amount;

END HasSufficientBalance;

/

# Exercise 5: Triggers

Scenario 1: Update the last modified date when a customer's record is updated

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

/

Scenario 2: Maintain an audit log for all transactions

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

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

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

END LogTransaction;

/

Scenario 3: Enforce business rules on deposits and withdrawals

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

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

RAISE\_APPLICATION\_ERROR(-20001, 'Error: Insufficient balance.');

ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

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

END IF;

END CheckTransactionRules;

/

# Exercise 6: Cursors

Scenario 1: Generate monthly statements for all customers

DECLARE

CURSOR statement\_cursor IS

SELECT c.CustomerID, c.Name, t.TransactionDate, t.Amount, t.TransactionType

FROM Customers c JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

WHERE t.TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE);

BEGIN

FOR rec IN statement\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || rec.CustomerID || ', Name: ' || rec.Name ||