**Exercise-1: Control Structures**

**Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.**

DECLARE

CURSOR cust\_cursor IS

SELECT c.CustomerID,

c.Name,

(TRUNC(MONTHS\_BETWEEN(SYSDATE, c.DOB) / 12)) AS Age,

l.LoanID,

l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID;

v\_customer\_id Customers.CustomerID%TYPE;

v\_name Customers.Name%TYPE;

v\_age NUMBER;

v\_loan\_id Loans.LoanID%TYPE;

v\_interest\_rate Loans.InterestRate%TYPE;

BEGIN

FOR cust\_rec IN cust\_cursor LOOP

v\_customer\_id := cust\_rec.CustomerID;

v\_name := cust\_rec.Name;

v\_age := cust\_rec.Age;

v\_loan\_id := cust\_rec.LoanID;

v\_interest\_rate := cust\_rec.InterestRate;

IF v\_age > 60 THEN

v\_interest\_rate := v\_interest\_rate \* 0.99;

DBMS\_OUTPUT.PUT\_LINE('Applied discount to customer ' || v\_name || ' (ID: ' || v\_customer\_id || ': New interest rate = ' || v\_interest\_rate);

END IF;

END LOOP;

END;

**Scenario 2: A customer can be promoted to VIP status based on their balance.**

DECLARE

CURSOR cus IS

select balance,customerid,name from customers;

v\_b customers.balance%TYPE;

v\_ci customers.customerid%type;

v\_n customers.name%type;

v\_isVIP boolean;

begin

for i IN cus LOOP

v\_b:=i.balance;

v\_ci:=i.customerid;

v\_n := i.name;

v\_isVIP := false;

if v\_b >= 10000 then

v\_isVIP := true;

dbms\_output.put\_line('Customer ID ' || v\_ci || ' Customer name ' || v\_n || 'is VIP');

END if;

END LOOP;

END;

**Scenario 3: The bank wants to send reminders to customers whose loans are due within the next 30 days.**

DECLARE

CURSOR LON IS

SELECT customerid, startdate, enddate

FROM loans;

v\_cusid loans.customerid%TYPE;

v\_start loans.startdate%TYPE;

v\_end loans.enddate%TYPE; -- Corrected this line from enddata to enddate

v\_duration NUMBER;

BEGIN

FOR i IN LON LOOP

v\_cusid := i.customerid;

v\_start := i.startdate;

v\_end := i.enddate;

v\_duration := v\_end-v\_start;

--dbms\_output.put\_line(v\_duration);

IF v\_duration <= 30 THEN

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || v\_cusid || ' has to pay the loan');

END IF;

END LOOP;

END;

**Exercise 2: Error Handling**

**Scenario 1: Handle exceptions during fund transfers between accounts.**

create or REPLACE SafeTransferFunds(

p\_source\_account\_id IN Accounts.AccountID%TYPE,

p\_dest\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

)

IS

v\_source\_balance Accounts.Balance%TYPE;

v\_dest\_balance Accounts.Balance%TYPE;

v\_insufficient\_funds EXCEPTION;

BEGIN

select balance into v\_source\_balance from accounts where accountid = p\_source\_account\_id for update;

if v\_source\_balance < p\_amount then

raise v\_insufficient\_funds;

end if;

update accounts

set balance = balance - p\_amount, lastmodified=sysdate where accountid = p\_source\_account\_id;

SELECT Balance INTO v\_dest\_balance

FROM Accounts

WHERE AccountID = p\_dest\_account\_id

FOR UPDATE;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_dest\_account\_id;

INSERT INTO Transactions (

TransactionID, AccountID, TransactionDate, Amount, TransactionType

) VALUES (

Transactions\_SEQ.NEXTVAL, -- Assuming you have a sequence for TransactionID

p\_source\_account\_id, SYSDATE, -p\_amount, 'Deposit'

);

INSERT INTO Transactions (

TransactionID, AccountID, TransactionDate, Amount, TransactionType

) VALUES (

Transactions\_SEQ.NEXTVAL, -- Assuming you have a sequence for TransactionID

p\_dest\_account\_id, SYSDATE, p\_amount, 'withdrawal'

);

DBMS\_OUTPUT.PUT\_LINE('Transfer successful from Account ' || p\_source\_account\_id || ' to Account ' || p\_dest\_account\_id);

EXCEPTION

WHEN v\_insufficient\_funds THEN

-- Rollback in case of insufficient funds

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in Account ' || p\_source\_account\_id);

WHEN OTHERS THEN

-- Rollback for any other errors

ROLLBACK;

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

END SafeTransferFunds;

**Scenario 2: Manage errors when updating employee salaries.**

CREATE OR REPLACE PROCEDURE UpdateSalary(

em\_id IN employees.employeeid%TYPE,

em\_percent NUMBER

) IS

v\_salary employees.salary%TYPE;

BEGIN

SELECT salary INTO v\_salary FROM employees WHERE employeeid = em\_id FOR UPDATE;

UPDATE employees

SET salary = salary + (v\_salary \* (em\_percent / 100))

WHERE employeeid = em\_id;

dbms\_output.put\_line('Employee ID ' || em\_id || ' salary = ' || v\_salary);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

dbms\_output.put\_line('Employee ID not found');

WHEN OTHERS THEN

ROLLBACK;

dbms\_output.put\_line('An unexpected error occurred');

END UpdateSalary;

begin

UpdateSalary(2,10);

end;

**Scenario 3: Ensure data integrity when adding a new customer.**

create or replace procedure AddNewCustomer(

n\_id in customers.customerid%type,

n\_name in customers.name%type,

n\_dob in customers.dob%type,

n\_b in customers.balance%type,

n\_mod in customers.lastmodified%type

) is

cursor cus is

select customerid from customers;

v\_emid customers.customerid%type;

same exception;

begin

for i in cus loop

v\_emid := i.customerid;

if v\_emid = n\_id then

raise same;

end if;

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

VALUES (n\_id,n\_name, TO\_DATE(n\_dob, 'YYYY-MM-DD'),n\_b , n\_mod);

dbms\_output.put\_line('added new customers');

end loop;

exception

when same then

rollback;

dbms\_output.put\_line('they have same id');

when others then

rollback;

dbms\_output.put\_line('some error has occured');

end AddNewCustomer;

begin

AddNewCustomer(7,'Batman','2003-02-05',10000,SYSDATE);

END;

**Exercise 3: Stored Procedures**

**Scenario 1: The bank needs to process monthly interest for all savings accounts.**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

AS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

UPDATE Accounts

SET LastModified = SYSDATE

WHERE AccountType = 'Savings';

END;

**scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_Department IN VARCHAR2,

p\_BonusPercentage IN NUMBER

)

AS

BEGIN

UPDATE Employees

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

WHERE Department = p\_Department;

END;

BEGIN

UpdateEmployeeBonus('IT', 30);

END;

**scenario 3: Customers should be able to transfer funds between their accounts.**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_SourceAccountID IN NUMBER,

p\_TargetAccountID IN NUMBER,

p\_Amount IN NUMBER

)

AS

v\_SourceBalance NUMBER;

BEGIN

-- Retrieve the balance of the source account

SELECT Balance INTO v\_SourceBalance

FROM Accounts

WHERE AccountID = p\_SourceAccountID

FOR UPDATE;

-- Check if the source account has sufficient balance

IF v\_SourceBalance < p\_Amount THEN

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

ELSE

-- Deduct the amount from the source account

UPDATE Accounts

SET Balance = Balance - p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_SourceAccountID;

-- Add the amount to the target account

UPDATE Accounts

SET Balance = Balance + p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_TargetAccountID;

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END;

BEGIN

TransferFunds(1, 2, 10);

END;

**Exercise 4: Functions**

**Scenario 1: Calculate the age of customers for eligibility checks.**

CREATE OR REPLACE FUNCTION CalculateAge(

p\_DOB DATE

)

RETURN NUMBER

IS

v\_Age NUMBER;

BEGIN

-- Calculate the age in years

v\_Age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

RETURN v\_Age;

END;

SELECT CalculateAge(DOB) AS Age

FROM Customers

WHERE CustomerID = 1;

**Scenario 2: The bank needs to compute the monthly installment for a loan.**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_LoanAmount NUMBER,

p\_InterestRate NUMBER,

p\_LoanDurationYears NUMBER

)

RETURN NUMBER

IS

v\_MonthlyInstallment NUMBER;

v\_MonthlyInterestRate NUMBER;

v\_TotalMonths NUMBER;

BEGIN

v\_MonthlyInterestRate := p\_InterestRate / 12 / 100;

v\_TotalMonths := p\_LoanDurationYears \* 12;

IF v\_MonthlyInterestRate > 0 THEN

v\_MonthlyInstallment := p\_LoanAmount \* v\_MonthlyInterestRate /

(1 - POWER(1 + v\_MonthlyInterestRate, -v\_TotalMonths));

ELSE

v\_MonthlyInstallment := p\_LoanAmount / v\_TotalMonths;

END IF;

RETURN v\_MonthlyInstallment;

END;

SELECT CalculateMonthlyInstallment(10000, 5, 5) AS MonthlyInstallment

FROM DUAL;

**scenario 3: Check if a customer has sufficient balance before making a transaction.**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_AccountID IN NUMBER,

p\_Amount IN NUMBER

)

RETURN BOOLEAN

IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = p\_AccountID;

IF v\_Balance >= p\_Amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

SELECT HasSufficientBalance(2, 100) AS Sufficient

FROM DUAL;

**Exercise 5: Triggers**

**Scenario 1: Automatically 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;

UPDATE Customers

SET Name = 'Josna'

WHERE CustomerID = 1;

**Scenario 2: Maintain an audit log for all transactions.**

CREATE TABLE AuditLog (

LogID NUMBER PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

LogTimestamp DATE DEFAULT SYSDATE

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

-- Insert a new record into the AuditLog table

INSERT INTO AuditLog (

LogID, TransactionID, AccountID, TransactionDate, Amount, TransactionType, LogTimestamp

) VALUES (

AuditLog\_SEQ.NEXTVAL, -- Assuming a sequence is used for LogID

:NEW.TransactionID,

:NEW.AccountID,

:NEW.TransactionDate,

:NEW.Amount,

:NEW.TransactionType,

SYSDATE

);

END;

CREATE SEQUENCE AuditLog\_SEQ START WITH 1 INCREMENT BY 1;

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

VALUES (1, 1, SYSDATE, 500, 'Deposit');

**scenario 3: Enforce business rules on deposits and withdrawals.**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_Balance NUMBER;

BEGIN

-- Retrieve the current balance of the account

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = :NEW.AccountID

FOR UPDATE;

-- Check if the transaction is a withdrawal

IF :NEW.TransactionType = 'Withdrawal' THEN

-- Ensure the withdrawal does not exceed the balance

IF :NEW.Amount > v\_Balance THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance for withdrawal.');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

-- Ensure the deposit amount is positive

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Deposit amount must be positive.');

END IF;

ELSE

-- Handle any other transaction types (optional)

RAISE\_APPLICATION\_ERROR(-20004, 'Unknown transaction type.');

END IF;

END;

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

VALUES (1, 1, SYSDATE, 1000, 'Withdrawal');

**Exercise 6: Cursors**

**Scenario 1: Generate monthly statements for all customers.**

DECLARE

-- Declare the cursor to retrieve transactions for the current month

CURSOR cur\_Transactions IS

SELECT

t.TransactionID,

t.AccountID,

t.TransactionDate,

t.Amount,

t.TransactionType,

c.CustomerID,

c.Name

FROM

Transactions t

JOIN

Accounts a ON t.AccountID = a.AccountID

JOIN

Customers c ON a.CustomerID = c.CustomerID

WHERE

TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM'); -- Filter for current month transactions

v\_TransactionID Transactions.TransactionID%TYPE;

v\_AccountID Accounts.AccountID%TYPE;

v\_TransactionDate Transactions.TransactionDate%TYPE;

v\_Amount Transactions.Amount%TYPE;

v\_TransactionType Transactions.TransactionType%TYPE;

v\_CustomerID Customers.CustomerID%TYPE;

v\_CustomerName Customers.Name%TYPE;

BEGIN

-- Open the cursor

OPEN cur\_Transactions;

LOOP

-- Fetch the next record into the variables

FETCH cur\_Transactions INTO

v\_TransactionID,

v\_AccountID,

v\_TransactionDate,

v\_Amount,

v\_TransactionType,

v\_CustomerID,

v\_CustomerName;

EXIT WHEN cur\_Transactions%NOTFOUND;

-- Print the statement for each customer

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

DBMS\_OUTPUT.PUT\_LINE('Customer Name: ' || v\_CustomerName);

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || v\_AccountID);

DBMS\_OUTPUT.PUT\_LINE('Transaction ID: ' || v\_TransactionID);

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

DBMS\_OUTPUT.PUT\_LINE('Transaction Type: ' || v\_TransactionType);

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

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

END LOOP;

CLOSE cur\_Transactions;

END;

**Scenario 2: Apply annual fee to all accounts.**

DECLARE

CURSOR ApplyAnnualFee IS

SELECT AccountID, Balance

FROM Accounts;

v\_account\_id NUMBER;

v\_balance NUMBER;

v\_annual\_fee CONSTANT NUMBER := 100; -- Replace with actual annual fee

BEGIN

OPEN ApplyAnnualFee;

LOOP

FETCH ApplyAnnualFee INTO v\_account\_id, v\_balance;

EXIT WHEN ApplyAnnualFee%NOTFOUND;

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

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee,

LastModified = SYSDATE

WHERE AccountID = v\_account\_id;

ELSE

-- Handle insufficient balance, e.g., log an error or send a notification

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance for account ' || v\_account\_id);

END IF;

END LOOP;

CLOSE ApplyAnnualFee;

END;

**Scenario 3: Update the interest rate for all loans based on a new policy.**

DECLARE

CURSOR UpdateLoanInterestRates IS

SELECT LoanID, InterestRate

FROM Loans;

v\_loan\_id Loans.LoanID%TYPE;

v\_interest Loans.InterestRate%TYPE;

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

BEGIN

OPEN UpdateLoanInterestRates;

LOOP

FETCH UpdateLoanInterestRates INTO v\_loan\_id, v\_interest;

EXIT WHEN UpdateLoanInterestRates%NOTFOUND;

UPDATE Loans

SET InterestRate = v\_new\_rate

WHERE LoanID = v\_loan\_id;

END LOOP;

CLOSE UpdateLoanInterestRates;

END;

**Exercise 7: Packages**

**scenario 1: Group all customer-related procedures and functions into a package.**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer (

p\_name VARCHAR2,

p\_dob DATE

);

PROCEDURE UpdateCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE

);

FUNCTION GetCustomerBalance (

p\_customer\_id NUMBER

) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer (

p\_name VARCHAR2,

p\_dob DATE

) IS

v\_customer\_id NUMBER;

BEGIN

SELECT CustomerID\_SEQ.NEXTVAL INTO v\_customer\_id FROM DUAL; -- Assuming a sequence for CustomerID

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

VALUES (v\_customer\_id, p\_name, p\_dob, 0, SYSDATE);

END AddCustomer;

PROCEDURE UpdateCustomer (

p\_customer\_id NUMBER,

p\_name VARCHAR2,

p\_dob DATE

) IS

BEGIN

UPDATE Customers

SET Name = p\_name,

DOB = p\_dob,

LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

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 NULL; -- Or handle the exception differently

END GetCustomerBalance;

END CustomerManagement;

**Scenario 2: Create a package to manage employee data.**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee (

p\_name VARCHAR2,

p\_position VARCHAR2,

p\_salary NUMBER,

p\_department VARCHAR2

);

PROCEDURE UpdateEmployee (

p\_employee\_id NUMBER,

p\_name VARCHAR2 DEFAULT NULL,

p\_position VARCHAR2 DEFAULT NULL,

p\_salary NUMBER DEFAULT NULL,

p\_department VARCHAR2 DEFAULT NULL

);

FUNCTION CalculateAnnualSalary (

p\_employee\_id NUMBER

) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee (

p\_name VARCHAR2,

p\_position VARCHAR2,

p\_salary NUMBER,

p\_department VARCHAR2

) IS

v\_employee\_id NUMBER;

BEGIN

SELECT EmployeeID\_SEQ.NEXTVAL INTO v\_employee\_id FROM DUAL; -- Assuming a sequence for EmployeeID

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

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

END HireEmployee;

PROCEDURE UpdateEmployee (

p\_employee\_id NUMBER,

p\_name VARCHAR2 DEFAULT NULL,

p\_position VARCHAR2 DEFAULT NULL,

p\_salary NUMBER DEFAULT NULL,

p\_department VARCHAR2 DEFAULT NULL

) IS

BEGIN

UPDATE Employees

SET Name = NVL(p\_name, Name),

Position = NVL(p\_position, Position),

Salary = NVL(p\_salary, Salary),

Department = NVL(p\_department, Department),

LastModified = SYSDATE

WHERE EmployeeID = p\_employee\_id;

END UpdateEmployee;

FUNCTION CalculateAnnualSalary (

p\_employee\_id NUMBER

) RETURN NUMBER IS

v\_annual\_salary NUMBER;

BEGIN

SELECT Salary \* 12 INTO v\_annual\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

RETURN v\_annual\_salary;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL; -- Or handle the exception differently

END CalculateAnnualSalary;

END EmployeeManagement;

**Scenario 3: Group all account-related operations into a package.**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount (

p\_customer\_id NUMBER,

p\_initial\_balance NUMBER

);

PROCEDURE CloseAccount (

p\_account\_id NUMBER

);

FUNCTION GetTotalBalance (

p\_customer\_id NUMBER

) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount (

p\_customer\_id NUMBER,

p\_initial\_balance NUMBER

) IS

v\_account\_id NUMBER;

BEGIN

SELECT AccountID\_SEQ.NEXTVAL INTO v\_account\_id FROM DUAL; -- Assuming a sequence for AccountID

INSERT INTO Accounts (AccountID, CustomerID, Balance, OpenDate)

VALUES (v\_account\_id, p\_customer\_id, p\_initial\_balance, SYSDATE);

END OpenAccount;

PROCEDURE CloseAccount (

p\_account\_id NUMBER

) IS

BEGIN

UPDATE Accounts

SET CloseDate = SYSDATE

WHERE AccountID = p\_account\_id;

END CloseAccount;

FUNCTION GetTotalBalance (

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 AND CloseDate IS NULL;

RETURN v\_total\_balance;

END GetTotalBalance;

END AccountOperations;