# ****Exercise 1: Control Structures****

## ****Scenario 1: Apply discount to loan interest rates****

sql

CopyEdit

BEGIN

FOR r IN (SELECT LoanID, InterestRate FROM Loans) LOOP

DECLARE

v\_age NUMBER;

BEGIN

SELECT FLOOR(MONTHS\_BETWEEN(SYSDATE, DOB)/12)

INTO v\_age

FROM Customers

WHERE CustomerID = (SELECT CustomerID FROM Loans WHERE LoanID = r.LoanID);

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = r.LoanID;

END IF;

END;

END LOOP;

COMMIT;END;/

## ****Scenario 2: Promote customers to VIP status****

You’ll need to **add the column first**:

sql

CopyEdit

ALTER TABLE Customers ADD (IsVIP VARCHAR2(5));

sql

CopyEdit

BEGIN

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

IF r.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = r.CustomerID;

END IF;

END LOOP;

COMMIT;END;/

## ****Scenario 3: Send reminders for loans due in 30 days****

sql

CopyEdit

BEGIN

FOR r IN (

SELECT LoanID, CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ' || r.CustomerID ||

', Loan ' || r.LoanID ||

' is due on ' || TO\_CHAR(r.EndDate, 'YYYY-MM-DD'));

END LOOP;END;/

# 🟢 ****Exercise 2: Error Handling****

## ****Scenario 1: Safe transfer funds****

sql

CopyEdit

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_FromAccountID NUMBER,

p\_ToAccountID NUMBER,

p\_Amount NUMBER

) AS

v\_Balance NUMBER;BEGIN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_FromAccountID FOR UPDATE;

IF v\_Balance < p\_Amount THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - p\_Amount WHERE AccountID = p\_FromAccountID;

UPDATE Accounts SET Balance = Balance + p\_Amount WHERE AccountID = p\_ToAccountID;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

## ****Scenario 2: Update salary with error handling****

sql

CopyEdit

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_EmployeeID NUMBER,

p\_Percent NUMBER

) ASBEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_EmployeeID;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee not found');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error updating salary: ' || SQLERRM);END UpdateSalary;/

## ****Scenario 3: Add new customer with duplicate check****

sql

CopyEdit

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_CustomerID NUMBER,

p\_Name VARCHAR2,

p\_DOB DATE,

p\_Balance NUMBER

) ASBEGIN

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

VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID already exists.');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error adding customer: ' || SQLERRM);END AddNewCustomer;/

# 🟢 ****Exercise 3: Stored Procedures****

## ****Scenario 1: Process monthly interest****

sql

CopyEdit

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;END ProcessMonthlyInterest;/

## ****Scenario 2: Update employee bonus****

sql

CopyEdit

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department VARCHAR2,

p\_BonusPercent NUMBER

) ASBEGIN

UPDATE Employees

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

WHERE Department = p\_Department;

COMMIT;END UpdateEmployeeBonus;/

## ****Scenario 3: Transfer funds****

sql

CopyEdit

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAccountID NUMBER,

p\_ToAccountID NUMBER,

p\_Amount NUMBER

) AS

v\_Balance NUMBER;BEGIN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_FromAccountID FOR UPDATE;

IF v\_Balance < p\_Amount THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient balance.');

END IF;

UPDATE Accounts SET Balance = Balance - p\_Amount WHERE AccountID = p\_FromAccountID;

UPDATE Accounts SET Balance = Balance + p\_Amount WHERE AccountID = p\_ToAccountID;

COMMIT;END TransferFunds;/

# 🟢 ****Exercise 4: Functions****

## ****Scenario 1: Calculate age****

sql

CopyEdit

CREATE OR REPLACE FUNCTION CalculateAge (

p\_DOB DATE

) RETURN NUMBER ASBEGIN

RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);END CalculateAge;/

## ****Scenario 2: Compute monthly installment****

sql

CopyEdit

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_LoanAmount NUMBER,

p\_InterestRate NUMBER,

p\_DurationYears NUMBER

) RETURN NUMBER AS

v\_MonthlyRate NUMBER := p\_InterestRate / (12 \* 100);

v\_NumPayments NUMBER := p\_DurationYears \* 12;BEGIN

RETURN p\_LoanAmount \* v\_MonthlyRate / (1 - POWER(1 + v\_MonthlyRate, -v\_NumPayments));END CalculateMonthlyInstallment;/

## ****Scenario 3: Check sufficient balance****

sql

CopyEdit

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_AccountID NUMBER,

p\_Amount NUMBER

) RETURN BOOLEAN AS

v\_Balance NUMBER;BEGIN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_AccountID;

RETURN v\_Balance >= p\_Amount;END HasSufficientBalance;/

# 🟢 ****Exercise 5: Triggers****

## ****Scenario 1: Update last modified date****

sql

CopyEdit

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON CustomersFOR EACH ROWBEGIN

:NEW.LastModified := SYSDATE;END;/

## ****Scenario 2: Log transactions****

sql

CopyEdit

CREATE TABLE AuditLog (

AuditID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

LogDate DATE,

Message VARCHAR2(200)

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON TransactionsFOR EACH ROWBEGIN

INSERT INTO AuditLog (TransactionID, LogDate, Message)

VALUES (:NEW.TransactionID, SYSDATE, 'Transaction recorded.');END;/

## ****Scenario 3: Enforce transaction rules****

sql

CopyEdit

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON TransactionsFOR EACH ROWDECLARE

v\_Balance NUMBER;BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = :NEW.AccountID;

IF :NEW.Amount > v\_Balance THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Withdrawal exceeds balance.');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20005, 'Deposit amount must be positive.');

END IF;

END IF;END;/

# 🟢 ****Exercise 6: Cursors****

## ****Scenario 1: Generate monthly statements****

sql

CopyEdit

DECLARE

CURSOR cur\_transactions IS

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

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');

BEGIN

FOR r IN cur\_transactions LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ' || r.CustomerID ||

' - ' || r.TransactionType ||

' of $' || r.Amount ||

' on ' || TO\_CHAR(r.TransactionDate, 'YYYY-MM-DD'));

END LOOP;END;/

## ****Scenario 2: Apply annual fee****

sql

CopyEdit

DECLARE

CURSOR cur\_accounts IS

SELECT AccountID FROM Accounts;BEGIN

FOR r IN cur\_accounts LOOP

UPDATE Accounts

SET Balance = Balance - 50

WHERE AccountID = r.AccountID;

END LOOP;

COMMIT;END;/

## ****Scenario 3: Update loan interest rates****

sql

CopyEdit

DECLARE

CURSOR cur\_loans IS

SELECT LoanID, InterestRate FROM Loans;BEGIN

FOR r IN cur\_loans LOOP

UPDATE Loans

SET InterestRate = InterestRate + 0.5

WHERE LoanID = r.LoanID;

END LOOP;

COMMIT;END;/

# 🟢 ****Exercise 7: Packages****

## ****Scenario 1: CustomerManagement Package****

sql

CopyEdit

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

PROCEDURE UpdateCustomer(p\_CustomerID NUMBER, p\_Balance NUMBER);

FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER;END CustomerManagement;/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) IS

BEGIN

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

VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

END;

PROCEDURE UpdateCustomer(p\_CustomerID NUMBER, p\_Balance NUMBER) IS

BEGIN

UPDATE Customers

SET Balance = p\_Balance, LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

END;

FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Customers WHERE CustomerID = p\_CustomerID;

RETURN v\_Balance;

END;END CustomerManagement;/

## ****Scenario 2: EmployeeManagement Package****

sql

CopyEdit

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2);

PROCEDURE UpdateEmployee(p\_EmployeeID NUMBER, p\_Salary NUMBER);

FUNCTION GetAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER;END EmployeeManagement;/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2) IS

BEGIN

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

VALUES (p\_EmployeeID, p\_Name, p\_Position, p\_Salary, p\_Department, SYSDATE);

END;

PROCEDURE UpdateEmployee(p\_EmployeeID NUMBER, p\_Salary NUMBER) IS

BEGIN

UPDATE Employees

SET Salary = p\_Salary

WHERE EmployeeID = p\_EmployeeID;

END;

FUNCTION GetAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

SELECT Salary INTO v\_Salary FROM Employees WHERE EmployeeID = p\_EmployeeID;

RETURN v\_Salary \* 12;

END;END EmployeeManagement;/

## ****Scenario 3: AccountOperations Package****

sql

CopyEdit

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_Type VARCHAR2, p\_Balance NUMBER);

PROCEDURE CloseAccount(p\_AccountID NUMBER);

FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER;END AccountOperations;/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_Type VARCHAR2, p\_Balance NUMBER) IS

BEGIN

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (p\_AccountID, p\_CustomerID, p\_Type, p\_Balance, SYSDATE);

END;

PROCEDURE CloseAccount(p\_AccountID NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_AccountID;

END;

FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_Total NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_Total FROM Accounts WHERE CustomerID = p\_CustomerID;

RETURN NVL(v\_Total, 0);

END;END AccountOperations;/