PL/SQL Exercise Solutions

# Exercise 1: Control Structures

## Scenario 1: Apply Discount for Senior Citizens

BEGIN  
 FOR rec IN (SELECT CustomerID FROM Customers WHERE MONTHS\_BETWEEN(SYSDATE, DOB)/12 > 60) LOOP  
 UPDATE Loans  
 SET InterestRate = InterestRate - 1  
 WHERE CustomerID = rec.CustomerID;  
 END LOOP;  
 COMMIT;  
END;

## Scenario 2: Promote to VIP

BEGIN  
 UPDATE Customers  
 SET Balance = Balance, -- retain balance  
 IsVIP = 'TRUE'  
 WHERE Balance > 10000;  
 COMMIT;  
END;

## Scenario 3: Loan Due Reminder

BEGIN  
 FOR rec IN (SELECT CustomerID, LoanID FROM Loans WHERE EndDate <= SYSDATE + 30) LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID || ' is due soon for Customer ' || rec.CustomerID);  
 END LOOP;  
END;

# Exercise 2: Error Handling

## Scenario 1: SafeTransferFunds

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amount NUMBER) AS  
BEGIN  
 UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from;  
 IF SQL%ROWCOUNT = 0 THEN RAISE\_APPLICATION\_ERROR(-20001, 'Source account not found'); END IF;  
  
 UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to;  
 IF SQL%ROWCOUNT = 0 THEN RAISE\_APPLICATION\_ERROR(-20002, 'Destination account not found'); END IF;  
  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
END;

## Scenario 2: UpdateSalary

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_emp\_id NUMBER, p\_percent NUMBER) AS  
BEGIN  
 UPDATE Employees  
 SET Salary = Salary + (Salary \* p\_percent / 100)  
 WHERE EmployeeID = p\_emp\_id;  
  
 IF SQL%ROWCOUNT = 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20003, 'Employee not found');  
 END IF;  
  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
END;

## Scenario 3: AddNewCustomer

CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) AS  
BEGIN  
 INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)  
 VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE);  
 COMMIT;  
EXCEPTION  
 WHEN DUP\_VAL\_ON\_INDEX THEN  
 DBMS\_OUTPUT.PUT\_LINE('Customer already exists');  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
END;

# Exercise 3: Stored Procedures

## Scenario 1: ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

END;

## Scenario 2: UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department VARCHAR2,

p\_bonus\_percent NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary \* (1 + p\_bonus\_percent/100)

WHERE Department = p\_department;

COMMIT;

END;

## Scenario 3: TransferFunds

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_source\_acc NUMBER,

p\_target\_acc NUMBER,

p\_amount NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_source\_acc FOR UPDATE;

IF v\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_source\_acc;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_target\_acc;

COMMIT;

END;

# Exercise 4: Functions

## Scenario 1: CalculateAge

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

BEGIN

RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob)/12);

END;

## Scenario 2: CalculateMonthlyInstallment

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_amount NUMBER, p\_rate NUMBER, p\_years NUMBER

) RETURN NUMBER IS

v\_r NUMBER := p\_rate / (12 \* 100); -- monthly rate

v\_n NUMBER := p\_years \* 12;

BEGIN

RETURN p\_amount \* (v\_r \* POWER(1+v\_r, v\_n)) / (POWER(1+v\_r, v\_n)-1);

END;

## Scenario 3: HasSufficientBalance

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;

# Exercise 5: Triggers

## Scenario 1: UpdateCustomerLastModified

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

## Scenario 2: LogTransaction

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, ActionDate)

VALUES (:NEW.TransactionID, SYSDATE);

END;

## Scenario 3: CheckTransactionRules

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

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

IF v\_balance < :NEW.Amount THEN

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

END IF;

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

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

END IF;

END;

# Exercise 6: Cursors

## Scenario 1: GenerateMonthlyStatements

DECLARE

CURSOR c IS SELECT CustomerID, AccountID, Amount, TransactionDate

FROM Transactions

WHERE TransactionDate >= TRUNC(SYSDATE, 'MONTH')

AND TransactionDate < ADD\_MONTHS(TRUNC(SYSDATE, 'MONTH'),1);

BEGIN

FOR t IN c LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: '||t.CustomerID||', Transaction: '||t.Amount||' on '||t.TransactionDate);

END LOOP;

END;

## Scenario 2: ApplyAnnualFee

DECLARE

CURSOR acc\_c IS SELECT AccountID, Balance FROM Accounts;

BEGIN

FOR a IN acc\_c LOOP

UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = a.AccountID;

END LOOP;

COMMIT;

END;

## Scenario 3: UpdateLoanInterestRates

DECLARE

CURSOR l\_c IS SELECT LoanID, InterestRate FROM Loans;

BEGIN

FOR l IN l\_c LOOP

UPDATE Loans SET InterestRate = InterestRate - 0.5 WHERE LoanID = l.LoanID;

END LOOP;

COMMIT;

END;

# Exercise 7: Packages

## Scenario 1: CustomerManagement package

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2);

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

BEGIN

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

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

END;

PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2) IS

BEGIN

UPDATE Customers SET Name=p\_name, LastModified=SYSDATE WHERE CustomerID=p\_id; COMMIT;

END;

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER IS v\_bal NUMBER;

BEGIN

SELECT Balance INTO v\_bal FROM Customers WHERE CustomerID=p\_id;

RETURN v\_bal;

END;

END;

## Scenario 2: EmployeeManagement package

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_name VARCHAR2, p\_pos VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2);

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER);

FUNCTION CalculateAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(p\_name VARCHAR2, p\_pos VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2) IS

BEGIN

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

VALUES(EMP\_SEQ.NEXTVAL, p\_name, p\_pos, p\_salary, p\_dept, SYSDATE); COMMIT;

END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER) IS

BEGIN

UPDATE Employees SET Salary=p\_salary WHERE EmployeeID=p\_id; COMMIT;

END;

FUNCTION CalculateAnnualSalary(p\_id NUMBER) RETURN NUMBER IS v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID=p\_id;

RETURN v\_salary\*12;

END;

END;

## Scenario 3: AccountOperations package

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_acc\_id NUMBER);

FUNCTION TotalCustomerBalance(p\_cust\_id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS

BEGIN

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

VALUES(ACC\_SEQ.NEXTVAL, p\_cust\_id, p\_type, p\_balance, SYSDATE); COMMIT;

END;

PROCEDURE CloseAccount(p\_acc\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID=p\_acc\_id; COMMIT;

END;

FUNCTION TotalCustomerBalance(p\_cust\_id NUMBER) RETURN NUMBER IS v\_total NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID=p\_cust\_id;

RETURN NVL(v\_total,0);

END;

END;