Exercise-01:

Discount for Senior Customers

BEGIN

FOR cust IN (SELECT CustomerID FROM Customers WHERE FLOOR(MONTHS\_BETWEEN(SYSDATE, DOB)/12) > 60) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

END LOOP;

END;

Set VIP Flag

BEGIN

FOR cust IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

END LOOP;

END;

Loan Reminder

BEGIN

FOR rec IN (SELECT LoanID, CustomerID, EndDate FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID ||

' for Customer ' || rec.CustomerID ||

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

END LOOP;

END;

Exercise-02:

Safe Fund Transfer

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

fromAcc NUMBER,

toAcc NUMBER,

amount NUMBER

) AS

fromBalance NUMBER;

BEGIN

SELECT Balance INTO fromBalance FROM Accounts WHERE AccountID = fromAcc;

IF fromBalance < amount THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = fromAcc;

UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = toAcc;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

Update Salary

CREATE OR REPLACE PROCEDURE UpdateSalary(empId NUMBER, percentage NUMBER) AS

BEGIN

UPDATE Employees

SET Salary = Salary + Salary \* (percentage / 100)

WHERE EmployeeID = empId;

IF SQL%ROWCOUNT = 0 THEN

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

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

Add New Customer

CREATE OR REPLACE PROCEDURE AddNewCustomer(

c\_id NUMBER, c\_name VARCHAR2, c\_dob DATE, c\_balance NUMBER

) AS

BEGIN

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

VALUES (c\_id, c\_name, c\_dob, c\_balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

ROLLBACK;

END;

Exercise-03:

Monthly Interest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

END;

Employee Bonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept IN VARCHAR2,

bonusPct IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + Salary \* (bonusPct / 100)

WHERE Department = dept;

COMMIT;

END;

Transfer Funds

CREATE OR REPLACE PROCEDURE TransferFunds(

fromAcc NUMBER, toAcc NUMBER, amt NUMBER

) AS

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Accounts WHERE AccountID = fromAcc;

IF bal < amt THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Not enough balance');

END IF;

UPDATE Accounts SET Balance = Balance - amt WHERE AccountID = fromAcc;

UPDATE Accounts SET Balance = Balance + amt WHERE AccountID = toAcc;

COMMIT;

END;

Exercise-04:

Calculate Age

CREATE OR REPLACE FUNCTION CalculateAge(dob DATE) RETURN NUMBER IS

BEGIN

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

END;

Monthly instalment

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

loanAmount NUMBER, annualRate NUMBER, years NUMBER

) RETURN NUMBER IS

r NUMBER := annualRate / 12 / 100;

n NUMBER := years \* 12;

BEGIN

RETURN loanAmount \* r / (1 - POWER(1 + r, -n));

END;

Has sufficient balance

CREATE OR REPLACE FUNCTION HasSufficientBalance(

accId NUMBER, amt NUMBER

) RETURN BOOLEAN IS

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM Accounts WHERE AccountID = accId;

RETURN bal >= amt;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

Exercise-05:

Update LastModified

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

Audit Log

CREATE TABLE AuditLog (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10)

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

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

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

END;

Check rules

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

acc\_balance NUMBER;

BEGIN

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

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > acc\_balance THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Insufficient balance');

ELSIF :NEW.Amount <= 0 THEN

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

END IF;

END;

Exercise-06:

Generate Monthly statements

DECLARE

CURSOR txn\_cur IS

SELECT a.CustomerID, t.\*

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

BEGIN

FOR txn IN txn\_cur LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || txn.CustomerID || ', TxnID: ' || txn.TransactionID || ', Amount: ' || txn.Amount);

END LOOP;

END;

Apply annual fee

DECLARE

CURSOR acc\_cur IS SELECT AccountID FROM Accounts;

BEGIN

FOR acc IN acc\_cur LOOP

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

END LOOP;

COMMIT;

END;

Update loan rates

DECLARE

CURSOR loan\_cur IS SELECT LoanID, InterestRate FROM Loans;

BEGIN

FOR l IN loan\_cur LOOP

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

END LOOP;

COMMIT;

END;

Exercise -07:

Customer Management

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, balance NUMBER);

PROCEDURE UpdateCustomer(id NUMBER, name VARCHAR2);

FUNCTION GetBalance(id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, balance NUMBER) IS

BEGIN

INSERT INTO Customers VALUES (id, name, dob, balance, SYSDATE);

END;

PROCEDURE UpdateCustomer(id NUMBER, name VARCHAR2) IS

BEGIN

UPDATE Customers SET Name = name WHERE CustomerID = id;

END;

FUNCTION GetBalance(id NUMBER) RETURN NUMBER IS

b NUMBER;

BEGIN

SELECT Balance INTO b FROM Customers WHERE CustomerID = id;

RETURN b;

END;

END;

Employee Management

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(id NUMBER, name VARCHAR2, pos VARCHAR2, sal NUMBER, dept VARCHAR2, hdate DATE);

PROCEDURE UpdateEmployee(id NUMBER, name VARCHAR2);

FUNCTION GetAnnualSalary(id NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(...) IS

BEGIN

INSERT INTO Employees VALUES (...);

END;

PROCEDURE UpdateEmployee(...) IS

BEGIN

UPDATE Employees SET Name = name WHERE EmployeeID = id;

END;

FUNCTION GetAnnualSalary(id NUMBER) RETURN NUMBER IS

s NUMBER;

BEGIN

SELECT Salary INTO s FROM Employees WHERE EmployeeID = id;

RETURN s \* 12;

END;

END;

Account Operations

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(accId NUMBER, custId NUMBER, type VARCHAR2, balance NUMBER);

PROCEDURE CloseAccount(accId NUMBER);

FUNCTION GetTotalBalance(custId NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(...) IS

BEGIN

INSERT INTO Accounts VALUES (...);

END;

PROCEDURE CloseAccount(accId NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = accId;

END;

FUNCTION GetTotalBalance(custId NUMBER) RETURN NUMBER IS

total NUMBER;

BEGIN

SELECT SUM(Balance) INTO total FROM Accounts WHERE CustomerID = custId;

RETURN total;

END;

END;