Exercise 1: Control Structures

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

o Question: Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

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

o Question: Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

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

o Question: Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

CODE:-

SCENARIO 1:-

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1955-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Loans VALUES (1, 1, 5000, 7, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans VALUES (2, 2, 3000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 36));

BEGIN

FOR rec IN (

SELECT l.LoanID, l.InterestRate, c.DOB

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

)

LOOP

IF MONTHS\_BETWEEN(SYSDATE, rec.DOB)/12 > 60 THEN

UPDATE Loans

SET InterestRate = rec.InterestRate - 1

WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to LoanID: ' || rec.LoanID);

ELSE

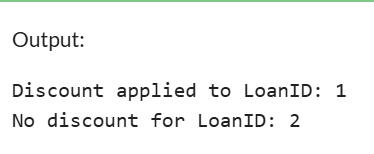
DBMS\_OUTPUT.PUT\_LINE('No discount for LoanID: ' || rec.LoanID);

END IF;

END LOOP;

COMMIT;

END;



SCENARIO 2:-

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE,

IsVIP VARCHAR2(5)

);

INSERT INTO Customers

VALUES (1, 'John Doe', TO\_DATE('1955-05-15', 'YYYY-MM-DD'), 1000, SYSDATE, 'FALSE');

INSERT INTO Customers

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 15000, SYSDATE, 'FALSE');

INSERT INTO Customers

VALUES (3, 'Mark Taylor', TO\_DATE('1982-03-10', 'YYYY-MM-DD'), 25000, SYSDATE, 'FALSE');

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers)

LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('VIP status granted to CustomerID: ' || rec.CustomerID);

ELSE

UPDATE Customers

SET IsVIP = 'FALSE'

WHERE CustomerID = rec.CustomerID;

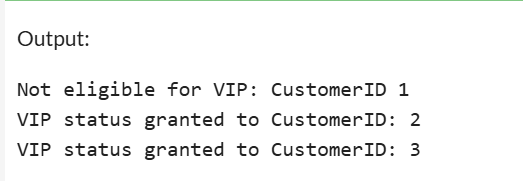
DBMS\_OUTPUT.PUT\_LINE('Not eligible for VIP: CustomerID ' || rec.CustomerID);

END IF;

END LOOP;

COMMIT;

END;



SCENARIO 3:-

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1955-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Customers VALUES (3, 'Ravi Kumar', TO\_DATE('1985-01-12', 'YYYY-MM-DD'), 8000, SYSDATE);

INSERT INTO Loans VALUES (1, 1, 5000, 7, SYSDATE, SYSDATE + 10);

INSERT INTO Loans VALUES (2, 2, 3000, 6, SYSDATE, SYSDATE + 45);

INSERT INTO Loans VALUES (3, 3, 7000, 5.5, SYSDATE, SYSDATE);

SET SERVEROUTPUT ON;

DECLARE

v\_name Customers.Name%TYPE;

BEGIN

FOR rec IN (

SELECT l.LoanID, l.EndDate, l.CustomerID

FROM Loans l

WHERE l.EndDate <= SYSDATE + 30

)

LOOP

SELECT Name INTO v\_name

FROM Customers

WHERE CustomerID = rec.CustomerID;

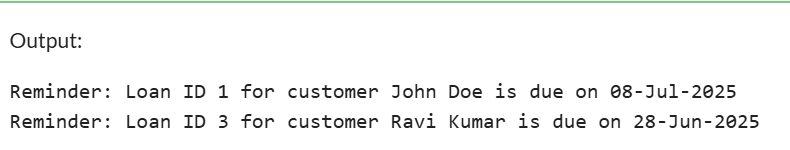
DBMS\_OUTPUT.PUT\_LINE(' Reminder: Loan ID ' || rec.LoanID ||

' for customer ' || v\_name ||

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

END LOOP;

END;



Exercise 2: Error Handling

Scenario 1: Handle exceptions during fund transfers between accounts.

o Question: Write a stored procedure SafeTransferFunds that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

Scenario 2: Manage errors when updating employee salaries.

o Question: Write a stored procedure UpdateSalary that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

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

o Question: Write a stored procedure AddNewCustomer that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

CODE:-

SCENARIO 1:-

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

CREATE TABLE TransferLogs (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

FromAccount NUMBER,

ToAccount NUMBER,

Amount NUMBER,

Status VARCHAR2(50),

Message VARCHAR2(200),

LogTime DATE DEFAULT SYSDATE

);

INSERT INTO Accounts VALUES (1, 101, 5000);

INSERT INTO Accounts VALUES (2, 102, 3000);

INSERT INTO Accounts VALUES (3, 103, 100);

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account;

IF v\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

COMMIT;

END;

EXEC SafeTransferFunds(1, 2, 1000);

EXEC SafeTransferFunds(3, 2, 1000);

EXEC SafeTransferFunds(999, 2, 500);

SCENARIO 2:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE SalaryUpdateLogs';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

CREATE TABLE SalaryUpdateLogs (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

EmployeeID NUMBER,

OldSalary NUMBER,

NewSalary NUMBER,

Status VARCHAR2(20),

Message VARCHAR2(200),

LogTime DATE DEFAULT SYSDATE

);

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

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

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

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percent IN NUMBER

)

IS

v\_old\_salary NUMBER;

v\_new\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_old\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

v\_new\_salary := v\_old\_salary + (v\_old\_salary \* p\_percent / 100);

UPDATE Employees

SET Salary = v\_new\_salary

WHERE EmployeeID = p\_employee\_id;

INSERT INTO SalaryUpdateLogs (

EmployeeID, OldSalary, NewSalary, Status, Message

)

VALUES (

p\_employee\_id, v\_old\_salary, v\_new\_salary, 'SUCCESS', 'Salary updated successfully'

);

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

INSERT INTO SalaryUpdateLogs (

EmployeeID, OldSalary, NewSalary, Status, Message

)

VALUES (

p\_employee\_id, NULL, NULL, 'FAILED', 'Employee not found');

WHEN OTHERS THEN

BEGIN

ROLLBACK;

INSERT INTO SalaryUpdateLogs (

EmployeeID, OldSalary, NewSalary, Status, Message

)

VALUES (

p\_employee\_id, NULL, NULL, 'FAILED', SQLERRM);

END;

END;

/

BEGIN

UpdateSalary(1, 10);

END;

/

BEGIN

UpdateSalary(99, 15);

END;

/

SCENARIO 3:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE DeleteAccountLogs';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE DeleteAccountLogs (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

AccountID NUMBER,

Status VARCHAR2(20),

Message VARCHAR2(200),

LogTime DATE DEFAULT SYSDATE

);

INSERT INTO Accounts VALUES (1, 101, 'Savings', 1000);

INSERT INTO Accounts VALUES (2, 102, 'Checking', 1500);

INSERT INTO Transactions VALUES (1, 2, SYSDATE, 300, 'Deposit');

CREATE OR REPLACE PROCEDURE SafeDeleteAccount (

p\_account\_id IN NUMBER

)

IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_account\_id;

INSERT INTO DeleteAccountLogs (

AccountID, Status, Message

) VALUES (

p\_account\_id, 'SUCCESS', 'Account deleted successfully'

);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

INSERT INTO DeleteAccountLogs (

AccountID, Status, Message

) VALUES (

p\_account\_id, 'FAILED', SQLERRM

);

ROLLBACK;

END;

/

BEGIN

SafeDeleteAccount(1);

END;

/

BEGIN

SafeDeleteAccount(2);

END;

/

Exercise 3: Stored Procedures

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

o Question: Write a stored procedure ProcessMonthlyInterest that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

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

o Question: Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

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

o Question: Write a stored procedure TransferFunds that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

CODE:-

SCENARIO 1:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE InterestLogs';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER

);

CREATE TABLE InterestLogs (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

AccountID NUMBER,

OldBalance NUMBER,

NewBalance NUMBER,

Status VARCHAR2(20),

Message VARCHAR2(200),

LogTime DATE DEFAULT SYSDATE

);

INSERT INTO Accounts VALUES (1, 101, 'Savings', 1000);

INSERT INTO Accounts VALUES (2, 102, 'Savings', 2000);

INSERT INTO Accounts VALUES (3, 103, 'Checking', 1500);

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

CURSOR savings\_cursor IS

SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings';

v\_id Accounts.AccountID%TYPE;

v\_old\_balance Accounts.Balance%TYPE;

v\_new\_balance Accounts.Balance%TYPE;

BEGIN

OPEN savings\_cursor;

LOOP

FETCH savings\_cursor INTO v\_id, v\_old\_balance;

EXIT WHEN savings\_cursor%NOTFOUND;

v\_new\_balance := v\_old\_balance + (v\_old\_balance \* 0.01);

UPDATE Accounts SET Balance = v\_new\_balance WHERE AccountID = v\_id;

INSERT INTO InterestLogs (AccountID, OldBalance, NewBalance, Status, Message)

VALUES (v\_id, v\_old\_balance, v\_new\_balance, 'SUCCESS', 'Interest applied');

END LOOP;

CLOSE savings\_cursor;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

END;

/

BEGIN

ProcessMonthlyInterest;

END;

/

SELECT \* FROM InterestLogs;

SCENARIO 2:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE BonusLogs';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

CREATE TABLE BonusLogs (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

EmployeeID NUMBER,

OldSalary NUMBER,

NewSalary NUMBER,

Department VARCHAR2(50),

Status VARCHAR2(20),

Message VARCHAR2(200),

LogTime DATE DEFAULT SYSDATE

);

INSERT INTO Employees VALUES (1, 'Alice', 'Manager', 70000, 'HR', SYSDATE);

INSERT INTO Employees VALUES (2, 'Bob', 'Developer', 60000, 'IT', SYSDATE);

INSERT INTO Employees VALUES (3, 'Carol', 'Analyst', 50000, 'IT', SYSDATE);

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

)

IS

CURSOR emp\_cursor IS

SELECT EmployeeID, Salary FROM Employees WHERE Department = p\_department;

v\_emp\_id Employees.EmployeeID%TYPE;

v\_old\_salary Employees.Salary%TYPE;

v\_new\_salary Employees.Salary%TYPE;

BEGIN

OPEN emp\_cursor;

LOOP

FETCH emp\_cursor INTO v\_emp\_id, v\_old\_salary;

EXIT WHEN emp\_cursor%NOTFOUND;

v\_new\_salary := v\_old\_salary + (v\_old\_salary \* p\_bonus\_percent / 100);

UPDATE Employees SET Salary = v\_new\_salary WHERE EmployeeID = v\_emp\_id;

INSERT INTO BonusLogs (

EmployeeID, OldSalary, NewSalary, Department, Status, Message

)

VALUES (

v\_emp\_id, v\_old\_salary, v\_new\_salary, p\_department, 'SUCCESS', 'Bonus applied'

);

END LOOP;

CLOSE emp\_cursor;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

END;

/

BEGIN

UpdateEmployeeBonus('IT', 10);

END;

/

SELECT \* FROM BonusLogs;

SCENARIO 3:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE FundTransferLogs';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER

);

CREATE TABLE FundTransferLogs (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

FromAccount NUMBER,

ToAccount NUMBER,

Amount NUMBER,

Status VARCHAR2(20),

Message VARCHAR2(200),

LogTime DATE DEFAULT SYSDATE

);

INSERT INTO Accounts VALUES (1, 101, 'Savings', 5000);

INSERT INTO Accounts VALUES (2, 102, 'Checking', 3000);

INSERT INTO Accounts VALUES (3, 103, 'Savings', 100);

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account;

IF v\_balance < p\_amount THEN

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

END IF;

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

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

INSERT INTO FundTransferLogs (

FromAccount, ToAccount, Amount, Status, Message

) VALUES (

p\_from\_account, p\_to\_account, p\_amount, 'SUCCESS', 'Transfer completed'

);

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

INSERT INTO FundTransferLogs (

FromAccount, ToAccount, Amount, Status, Message

) VALUES (

p\_from\_account, p\_to\_account, p\_amount, 'FAILED', 'Account not found'

);

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO FundTransferLogs (

FromAccount, ToAccount, Amount, Status, Message

) VALUES (

p\_from\_account, p\_to\_account, p\_amount, 'FAILED', SQLERRM

);

END;

/

BEGIN

TransferFunds(1, 2, 1000);

END;

/

BEGIN

TransferFunds(3, 2, 2000);

END;

/

SELECT \* FROM FundTransferLogs;

Exercise 4: Functions

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

o Question: Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

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

o Question: Write a function CalculateMonthlyInstallment that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

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

o Question: Write a function HasSufficientBalance that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

CODE:-

SCENARIO 1:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1995-06-28', 'YYYY-MM-DD'));

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('2001-01-15', 'YYYY-MM-DD'));

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob DATE

) RETURN NUMBER

IS

v\_age NUMBER;

BEGIN

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

RETURN v\_age;

END;

/

SELECT CustomerID, Name, CalculateAge(DOB) AS Age FROM Customers;

SCENARIO 2:-

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_annual\_rate NUMBER,

p\_years NUMBER

) RETURN NUMBER

IS

v\_monthly\_rate NUMBER := p\_annual\_rate / 12 / 100;

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

v\_installment NUMBER;

BEGIN

v\_installment := (p\_loan\_amount \* v\_monthly\_rate) / (1 - POWER(1 + v\_monthly\_rate, -v\_months));

RETURN v\_installment;

END;

/

SELECT CalculateMonthlyInstallment(100000, 8.5, 5) AS EMI FROM DUAL;

SCENARIO 3:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

INSERT INTO Accounts VALUES (1, 101, 5000);

INSERT INTO Accounts VALUES (2, 102, 1000);

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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

SELECT HasSufficientBalance(1, 3000) AS Check1 FROM DUAL;

SELECT HasSufficientBalance(2, 2000) AS Check2 FROM DUAL;

Exercise 5: Triggers

Scenario 1: Automatically update the last modified date when a customer's record is updated.

o Question: Write a trigger UpdateCustomerLastModified that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

Scenario 2: Maintain an audit log for all transactions.

o Question: Write a trigger LogTransaction that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

Scenario 3: Enforce business rules on deposits and withdrawals.

o Question: Write a trigger CheckTransactionRules that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

CODE:-

SCENARIO 1:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1995-01-01','YYYY-MM-DD'), 1000, SYSDATE);

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

UPDATE Customers SET Balance = Balance + 500 WHERE CustomerID = 1;

SELECT \* FROM Customers;

SCENARIO 2:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE AuditLog';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10)

);

CREATE TABLE AuditLog (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

LoggedAt DATE DEFAULT SYSDATE

);

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;

/

INSERT INTO Transactions VALUES (1, 101, SYSDATE, 500, 'Deposit');

SELECT \* FROM AuditLog;

SCENARIO 3:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10)

);

INSERT INTO Accounts VALUES (1, 101, 5000);

INSERT INTO Accounts VALUES (2, 102, 1000);

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

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

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

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

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

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

END IF;

END;

/

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 2000, 'Withdrawal');

INSERT INTO Transactions VALUES (2, 2, SYSDATE, 2000, 'Withdrawal');

INSERT INTO Transactions VALUES (3, 1, SYSDATE, -100, 'Deposit');

Exercise 6: Cursors

Scenario 1: Generate monthly statements for all customers.

o Question: Write a PL/SQL block using an explicit cursor GenerateMonthlyStatements that retrieves all transactions for the current month and prints a statement for each customer.

Scenario 2: Apply annual fee to all accounts.

o Question: Write a PL/SQL block using an explicit cursor ApplyAnnualFee that deducts an annual maintenance fee from the balance of all accounts.

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

o Question: Write a PL/SQL block using an explicit cursor UpdateLoanInterestRates that fetches all loans and updates their interest rates based on the new policy.

CODE:-

SCENARIO 1:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

CustomerID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10)

);

INSERT INTO Customers VALUES (1, 'John Doe');

INSERT INTO Customers VALUES (2, 'Jane Smith');

INSERT INTO Transactions VALUES (1, 101, 1, SYSDATE, 500, 'Deposit');

INSERT INTO Transactions VALUES (2, 102, 1, SYSDATE, 200, 'Withdrawal');

INSERT INTO Transactions VALUES (3, 103, 2, SYSDATE, 1000, 'Deposit');

DECLARE

CURSOR txn\_cursor IS

SELECT CustomerID, TransactionDate, Amount, TransactionType

FROM Transactions

WHERE TO\_CHAR(TransactionDate, 'MM-YYYY') = TO\_CHAR(SYSDATE, 'MM-YYYY');

v\_customer\_id NUMBER;

v\_date DATE;

v\_amount NUMBER;

v\_type VARCHAR2(10);

BEGIN

OPEN txn\_cursor;

LOOP

FETCH txn\_cursor INTO v\_customer\_id, v\_date, v\_amount, v\_type;

EXIT WHEN txn\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id || ' | Date: ' || v\_date || ' | Type: ' || v\_type || ' | Amount: ' || v\_amount);

END LOOP;

CLOSE txn\_cursor;

END;

/

SCENARIO 2:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

INSERT INTO Accounts VALUES (1, 101, 5000);

INSERT INTO Accounts VALUES (2, 102, 3000);

DECLARE

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

v\_id Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_fee NUMBER := 500;

BEGIN

OPEN acc\_cursor;

LOOP

FETCH acc\_cursor INTO v\_id, v\_balance;

EXIT WHEN acc\_cursor%NOTFOUND;

UPDATE Accounts SET Balance = Balance - v\_fee WHERE AccountID = v\_id;

END LOOP;

CLOSE acc\_cursor;

COMMIT;

END;

/

SELECT \* FROM Accounts;

SCENARIO 3:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE

);

INSERT INTO Loans VALUES (1, 1, 10000, 5.0, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans VALUES (2, 2, 20000, 6.0, SYSDATE, ADD\_MONTHS(SYSDATE, 36));

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate FROM Loans;

v\_id Loans.LoanID%TYPE;

v\_rate Loans.InterestRate%TYPE;

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_id, v\_rate;

EXIT WHEN loan\_cursor%NOTFOUND;

UPDATE Loans SET InterestRate = v\_rate + 0.5 WHERE LoanID = v\_id;

END LOOP;

CLOSE loan\_cursor;

COMMIT;

END;

/

SELECT \* FROM Loans;

Exercise 7: Packages

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

o Question: Create a package CustomerManagement with procedures for adding a new customer, updating customer details, and a function to get customer balance.

Scenario 2: Create a package to manage employee data.

o Question: Write a package EmployeeManagement with procedures to hire new employees, update employee details, and a function to calculate annual salary.

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

o Question: Create a package AccountOperations with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

CODE:-

SCENARIO 1:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER

);

/

CREATE OR REPLACE PACKAGE CustomerManagement AS

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

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

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

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

BEGIN

INSERT INTO Customers VALUES (p\_id, p\_name, p\_dob, p\_balance);

END;

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

BEGIN

UPDATE Customers SET Name = p\_name, DOB = p\_dob WHERE CustomerID = p\_id;

END;

FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

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

RETURN v\_balance;

END;

END CustomerManagement;

/

BEGIN

CustomerManagement.AddCustomer(1, 'John Doe', TO\_DATE('1995-01-01', 'YYYY-MM-DD'), 1000);

DBMS\_OUTPUT.PUT\_LINE(CustomerManagement.GetCustomerBalance(1));

END;

/

SCENARIO 2:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50)

);

/

CREATE OR REPLACE PACKAGE EmployeeManagement AS

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

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER, p\_dept VARCHAR2);

FUNCTION AnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

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

BEGIN

INSERT INTO Employees VALUES (p\_id, p\_name, p\_position, p\_salary, p\_dept);

END;

PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER, p\_dept VARCHAR2) IS

BEGIN

UPDATE Employees SET Salary = p\_salary, Department = p\_dept WHERE EmployeeID = p\_id;

END;

FUNCTION AnnualSalary(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 EmployeeManagement;

/

BEGIN

EmployeeManagement.HireEmployee(1, 'Alice', 'Manager', 60000, 'HR');

DBMS\_OUTPUT.PUT\_LINE(EmployeeManagement.AnnualSalary(1));

END;

/

SCENARIO 3:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

/

CREATE OR REPLACE PACKAGE AccountOperations AS

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

PROCEDURE CloseAccount(p\_acc\_id NUMBER);

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

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

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

BEGIN

INSERT INTO Accounts VALUES (p\_acc\_id, p\_cust\_id, p\_balance);

END;

PROCEDURE CloseAccount(p\_acc\_id NUMBER) IS

BEGIN

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

END;

FUNCTION TotalBalance(p\_cust\_id NUMBER) RETURN NUMBER IS

v\_total NUMBER;

BEGIN

SELECT NVL(SUM(Balance), 0) INTO v\_total FROM Accounts WHERE CustomerID = p\_cust\_id;

RETURN v\_total;

END;

END AccountOperations;

/

BEGIN

AccountOperations.OpenAccount(1, 101, 5000);

AccountOperations.OpenAccount(2, 101, 3000);

DBMS\_OUTPUT.PUT\_LINE(AccountOperations.TotalBalance(101));

END;

/