**Exercise 1: Control Structures**

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

BEGIN

FOR customer IN (SELECT \* FROM Customers) LOOP

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

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = customer.CustomerID;

END IF;

END LOOP;

END;

/

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

BEGIN

FOR customer IN (SELECT \* FROM Customers) LOOP

IF customer.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = TRUE

WHERE CustomerID = customer.CustomerID;

END IF;

END LOOP;

END;

/

Scenario 3: Send Reminders to Customers Whose Loans are Due Within the Next 30 Days

BEGIN

FOR loan IN (SELECT \* FROM Loans WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ' || loan.CustomerID || ', your loan is due on ' || loan.EndDate);

END LOOP;

END;

/

**Exercise 2: Error Handling**

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

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

insufficient\_funds EXCEPTION;

PRAGMA EXCEPTION\_INIT(insufficient\_funds, -20001);

BEGIN

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

IF SQL%NOTFOUND OR SQL%ROWCOUNT = 0 THEN

RAISE insufficient\_funds;

END IF;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in the source account.');

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END SafeTransferFunds;

/

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

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

no\_employee\_found EXCEPTION;

PRAGMA EXCEPTION\_INIT(no\_employee\_found, -20001);

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

IF SQL%NOTFOUND THEN

RAISE no\_employee\_found;

END IF;

COMMIT;

EXCEPTION

WHEN no\_employee\_found THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID not found.');

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END UpdateSalary;

/

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

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) AS

customer\_exists EXCEPTION;

PRAGMA EXCEPTION\_INIT(customer\_exists, -20001);

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END AddNewCustomer;

/

**Exercise 3: Stored Procedures**

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

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR account IN (SELECT \* FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = account.AccountID;

END LOOP;

COMMIT;

END ProcessMonthlyInterest;

/

**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\_bonus\_percentage IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

END UpdateEmployeeBonus;

/

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

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

insufficient\_funds EXCEPTION;

BEGIN

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE insufficient\_funds;

END IF;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in the source account.');

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END TransferFunds;

/

**Exercise 4: Functions**

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

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob IN DATE

) RETURN NUMBER AS

BEGIN

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

END CalculateAge;

/

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

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount IN NUMBER,

p\_interest\_rate IN NUMBER,

p\_duration\_years IN NUMBER

) RETURN NUMBER AS

v\_monthly\_rate NUMBER;

v\_number\_of\_payments NUMBER;

BEGIN

v\_monthly\_rate := p\_interest\_rate / 12 / 100;

v\_number\_of\_payments := p\_duration\_years \* 12;

RETURN p\_loan\_amount \* v\_monthly\_rate / (1 - POWER(1 + v\_monthly\_rate, -v\_number\_of\_payments));

END CalculateMonthlyInstallment;

/

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

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id IN NUMBER,

p\_amount IN NUMBER

) RETURN BOOLEAN AS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

RETURN v\_balance >= p\_amount;

END HasSufficientBalance;

/

**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 UpdateCustomerLastModified;

/

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

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 LogTransaction;

/

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

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

DECLARE

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

IF v\_balance < :NEW.Amount THEN

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

END IF;

END;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

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

END IF;

END IF;

END CheckTransactionRules;

/

**Exercise 6: Cursors**

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

DECLARE

CURSOR cur\_transactions IS

SELECT CustomerID, TransactionDate, Amount, TransactionType

FROM Transactions

WHERE TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE);

BEGIN

FOR transaction IN cur\_transactions LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ' || transaction.CustomerID || ': ' || transaction.TransactionDate || ' ' || transaction.TransactionType || ' ' || transaction.Amount);

END LOOP;

END;

/

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

DECLARE

CURSOR cur\_accounts IS

SELECT AccountID, Balance

FROM Accounts;

BEGIN

FOR account IN cur\_accounts LOOP

UPDATE Accounts

SET Balance = Balance - 50

WHERE AccountID = account.AccountID;

END LOOP;

COMMIT;

END;

/

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

DECLARE

CURSOR cur\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

BEGIN

FOR loan IN cur\_loans LOOP

UPDATE Loans

SET InterestRate = loan.InterestRate + 0.5

WHERE LoanID = loan.LoanID;

END LOOP;

COMMIT;

END;

/

**Exercise 7: Packages**

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

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer (p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_dob IN DATE, p\_balance IN NUMBER);

PROCEDURE UpdateCustomerDetails (p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_dob IN DATE, p\_balance IN NUMBER);

FUNCTION GetCustomerBalance (p\_customer\_id IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) IS

BEGIN

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

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) IS

BEGIN

UPDATE Customers

SET Name = p\_name, DOB = p\_dob, Balance = p\_balance, LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance (

p\_customer\_id IN NUMBER

) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Customers

WHERE CustomerID = p\_customer\_id;

RETURN v\_balance;

END GetCustomerBalance;

END CustomerManagement;

/

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

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee (p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2, p\_hire\_date IN DATE);

PROCEDURE UpdateEmployeeDetails (p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2);

FUNCTION CalculateAnnualSalary (p\_employee\_id IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee (

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_position IN VARCHAR2,

p\_salary IN NUMBER,

p\_department IN VARCHAR2,

p\_hire\_date IN DATE

) IS

BEGIN

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

VALUES (p\_employee\_id, p\_name, p\_position, p\_salary, p\_department, p\_hire\_date);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee with this ID already exists.');

WHEN OTHERS THEN

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

END HireEmployee;

PROCEDURE UpdateEmployeeDetails (

p\_employee\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_position IN VARCHAR2,

p\_salary IN NUMBER,

p\_department IN VARCHAR2

) IS

BEGIN

UPDATE Employees

SET Name = p\_name, Position = p\_position, Salary = p\_salary, Department = p\_department

WHERE EmployeeID = p\_employee\_id;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary (

p\_employee\_id IN NUMBER

) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

RETURN v\_salary \* 12;

END CalculateAnnualSalary;

END EmployeeManagement;

/

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

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount (p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER, p\_account\_type IN VARCHAR2, p\_balance IN NUMBER);

PROCEDURE CloseAccount (p\_account\_id IN NUMBER);

FUNCTION GetTotalBalance (p\_customer\_id IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount (

p\_account\_id IN NUMBER,

p\_customer\_id IN NUMBER,

p\_account\_type IN VARCHAR2,

p\_balance IN NUMBER

) IS

BEGIN

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

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account with this ID already exists.');

WHEN OTHERS THEN

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

END OpenAccount;

PROCEDURE CloseAccount (

p\_account\_id IN NUMBER

) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_account\_id;

EXCEPTION

WHEN OTHERS THEN

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

END CloseAccount;

FUNCTION GetTotalBalance (

p\_customer\_id IN NUMBER

) RETURN NUMBER IS

v\_total\_balance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_total\_balance

FROM Accounts

WHERE CustomerID = p\_customer\_id;

RETURN v\_total\_balance;

END GetTotalBalance;

END AccountOperations;

/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*---------------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Done by Mohit Raj on (05-08-2024)