Exercise 1: Control Structures

Scenario 1:

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.

Answer:

BEGIN

FOR rec IN (SELECT CustomerID, Age, LoanInterestRate FROM Customers) LOOP

IF rec.Age > 60 THEN

UPDATE Customers

SET LoanInterestRate = rec.LoanInterestRate - 1

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

END;

Scenario 2:

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.

Answer:

BEGIN

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

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = TRUE

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

END;

Scenario 3:

Question:

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

Answer:

BEGIN

FOR rec IN (SELECT LoanID, CustomerID, LoanDueDate FROM Loans WHERE LoanDueDate <= SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID || ' is due within 30 days for customer ' || rec.CustomerID);

END LOOP;

END;

Exercise 2: Error Handling

Scenario 1:

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.

Answer:

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

InsufficientFunds EXCEPTION;

BEGIN

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

IF SQL%ROWCOUNT = 0 THEN

RAISE InsufficientFunds;

END IF;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

EXCEPTION

WHEN InsufficientFunds THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error occurred during fund transfer.');

END SafeTransferFunds;

Scenario 2:

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.

Answer:

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_EmployeeID IN NUMBER,

p\_Percentage IN NUMBER

) IS

EmployeeNotFound EXCEPTION;

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_EmployeeID;

IF SQL%ROWCOUNT = 0 THEN

RAISE EmployeeNotFound;

END IF;

COMMIT;

EXCEPTION

WHEN EmployeeNotFound THEN

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

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred while updating salary.');

END UpdateSalary;

Scenario 3:

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.

Answer:

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

) IS

CustomerExists EXCEPTION;

BEGIN

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

RAISE CustomerExists;

WHEN CustomerExists THEN

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

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred while adding new customer.');

END AddNewCustomer;

Exercise 3: Stored Procedures

Scenario 1:

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.

Answer:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR rec IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

SET Balance = rec.Balance + (rec.Balance \* 0.01)

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

END ProcessMonthlyInterest;

Scenario 2:

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.

Answer:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPercentage IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_Department;

COMMIT;

END UpdateEmployeeBonus;

Scenario 3:

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.

Answer:

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

BEGIN

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID AND Balance >= p\_Amount;

IF SQL%ROWCOUNT = 0 THEN

ROLLBACK;

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

RETURN;

END IF;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

END TransferFunds;

Exercise 4: Functions

Scenario 1:

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

Answer:

CREATE OR REPLACE FUNCTION CalculateAge (

p\_DOB IN DATE

) RETURN NUMBER IS

v\_Age NUMBER;

BEGIN

SELECT FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12) INTO v\_Age FROM DUAL;

RETURN v\_Age;

END CalculateAge;

Scenario 2:

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.

Answer:

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_LoanAmount IN NUMBER,

p\_InterestRate IN NUMBER,

p\_DurationYears IN NUMBER

) RETURN NUMBER IS

v\_MonthlyInstallment NUMBER;

BEGIN

v\_MonthlyInstallment := (p\_LoanAmount \* (1 + p\_InterestRate/100)) / (p\_DurationYears \* 12);

RETURN v\_MonthlyInstallment;

END CalculateMonthlyInstallment;

Scenario 3:

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.

Answer:

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_AccountID IN NUMBER,

p\_Amount IN NUMBER

) RETURN BOOLEAN IS

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:

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.

Answer:

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

AFTER UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

Scenario 2:

Question:

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

Answer:

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

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

VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.Amount, :NEW.Transaction

Type, :NEW.TransactionDate);

END LogTransaction;

Scenario 3:

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.

Answer:

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 :NEW.Amount > v\_Balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error: Withdrawal amount exceeds the balance.');

END IF;

END;

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

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

END IF;

END CheckTransactionRules;

Exercise 6: Cursors

Scenario 1:

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.

Answer:

DECLARE

CURSOR cur\_Transactions IS

SELECT t.TransactionID, t.AccountID, t.Amount, t.TransactionDate, c.Name

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

WHERE t.TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE);

rec\_Transaction cur\_Transactions%ROWTYPE;

BEGIN

OPEN cur\_Transactions;

LOOP

FETCH cur\_Transactions INTO rec\_Transaction;

EXIT WHEN cur\_Transactions%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec\_Transaction.Name || ' | Transaction ID: ' || rec\_Transaction.TransactionID || ' | Amount: ' || rec\_Transaction.Amount || ' | Date: ' || rec\_Transaction.TransactionDate);

END LOOP;

CLOSE cur\_Transactions;

END;

Scenario 2:

Question:

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

Answer:

DECLARE

CURSOR cur\_Accounts IS

SELECT AccountID, Balance FROM Accounts;

rec\_Account cur\_Accounts%ROWTYPE;

v\_AnnualFee CONSTANT NUMBER := 100; -- Example fee

BEGIN

OPEN cur\_Accounts;

LOOP

FETCH cur\_Accounts INTO rec\_Account;

EXIT WHEN cur\_Accounts%NOTFOUND;

UPDATE Accounts

SET Balance = rec\_Account.Balance - v\_AnnualFee

WHERE AccountID = rec\_Account.AccountID;

END LOOP;

CLOSE cur\_Accounts;

COMMIT;

END;

Scenario 3:

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.

Answer:

DECLARE

CURSOR cur\_Loans IS

SELECT LoanID, InterestRate FROM Loans;

rec\_Loan cur\_Loans%ROWTYPE;

v\_NewInterestRate CONSTANT NUMBER := 3; -- Example new interest rate

BEGIN

OPEN cur\_Loans;

LOOP

FETCH cur\_Loans INTO rec\_Loan;

EXIT WHEN cur\_Loans%NOTFOUND;

UPDATE Loans

SET InterestRate = v\_NewInterestRate

WHERE LoanID = rec\_Loan.LoanID;

END LOOP;

CLOSE cur\_Loans;

COMMIT;

END;

Exercise 7: Packages

Scenario 1:

Question:

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

Answer:

CREATE OR REPLACE PACKAGE CustomerManagement AS

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

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

FUNCTION GetCustomerBalance(p\_CustomerID IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_CustomerID 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\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

END AddCustomer;

PROCEDURE UpdateCustomer(p\_CustomerID 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\_CustomerID;

END UpdateCustomer;

FUNCTION GetCustomerBalance(p\_CustomerID IN NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

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

RETURN v\_Balance;

END GetCustomerBalance;

END CustomerManagement;

Scenario 2:

Question:

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

Answer:

CREATE OR REPLACE PACKAGE EmployeeManagement AS

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

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

FUNCTION CalculateAnnualSalary(p\_EmployeeID IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

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

BEGIN

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

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

END HireEmployee;

PROCEDURE UpdateEmployee(p\_EmployeeID 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\_EmployeeID;

END UpdateEmployee;

FUNCTION CalculateAnnualSalary(p\_EmployeeID IN NUMBER) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

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

RETURN v\_Salary \* 12;

END CalculateAnnualSalary;

END EmployeeManagement;

Scenario 3:

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.

Answer:

CREATE OR REPLACE PACKAGE AccountOperations AS

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

PROCEDURE CloseAccount(p\_AccountID IN NUMBER);

FUNCTION GetTotalBalance(p\_CustomerID IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

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

BEGIN

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

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

END OpenAccount;

PROCEDURE CloseAccount(p\_AccountID IN NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_AccountID;

END CloseAccount;

FUNCTION GetTotalBalance(p\_CustomerID IN NUMBER) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

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

RETURN v\_TotalBalance;

END GetTotalBalance;

END AccountOperations;