**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, applies a 1% discount to their current loan interest rates.

**ANSWER:**

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

age NUMBER;

BEGIN

age := TRUNC(MONTHS\_BETWEEN(SYSDATE, dob) / 12);

RETURN age;

END CalculateAge;

/

BEGIN

FOR rec IN (

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

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

) LOOP

IF CalculateAge(rec.DOB) > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate \* 0.99

WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Updated LoanID: ' || rec.LoanID || ' for CustomerID: ' || rec.CustomerID);

END IF;

END LOOP;

COMMIT;

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 $10000.

**ANSWER:**

ALTER TABLE Customers ADD (IsVIP VARCHAR2(3));

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;

COMMIT;

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 c.CustomerID, c.Name, l.LoanID, l.EndDate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: LoanID ' || rec.LoanID || ' for Customer ' || rec.Name || ' is due on ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY'));

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 NUMBER,

p\_toAccountID NUMBER,

p\_amount NUMBER

) IS

insufficient\_funds EXCEPTION;

BEGIN

-- Check for sufficient funds

FOR rec IN (SELECT Balance FROM Accounts WHERE AccountID = p\_fromAccountID) LOOP

IF rec.Balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

END LOOP;

-- Transfer funds

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_fromAccountID;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_toAccountID;

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**

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 NUMBER,

p\_percentage NUMBER

) IS

employee\_not\_found EXCEPTION;

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employeeID;

IF SQL%NOTFOUND THEN

RAISE employee\_not\_found;

END IF;

COMMIT;

EXCEPTION

WHEN employee\_not\_found THEN

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

WHEN OTHERS THEN

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

ROLLBACK;

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 NUMBER,

p\_name VARCHAR2,

p\_dob DATE,

p\_balance NUMBER

) IS

customer\_exists 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

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

RAISE customer\_exists;

WHEN OTHERS THEN

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

ROLLBACK;

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 = Balance \* 1.01

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 VARCHAR2,

p\_bonus\_percentage NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percentage / 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 NUMBER,

p\_toAccountID NUMBER,

p\_amount NUMBER

) IS

insufficient\_funds EXCEPTION;

BEGIN

FOR rec IN (SELECT Balance FROM Accounts WHERE AccountID = p\_fromAccountID) LOOP

IF rec.Balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

END LOOP;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_fromAccountID;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_toAccountID;

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**

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(dob DATE) RETURN NUMBER IS

age NUMBER;

BEGIN

age := TRUNC(MONTHS\_BETWEEN(SYSDATE, dob) / 12);

RETURN 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 NUMBER,

p\_interestRate NUMBER,

p\_durationYears NUMBER

) RETURN NUMBER IS

monthly\_installment NUMBER;

rate\_per\_month NUMBER;

total\_months NUMBER;

BEGIN

rate\_per\_month := p\_interestRate / 1200;

total\_months := p\_durationYears \* 12;

monthly\_installment := p\_loanAmount \* rate\_per\_month / (1 - POWER(1 + rate\_per\_month, -total\_months));

RETURN monthly\_installment;

END CalculateMonthlyInstallment;

/

**Scenario-3**

Write a function `HasSufficientBalance` that takes an account ID and an amount as input and returns a boolean indicating whether the account

**ANSWER:**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id IN NUMBER,

p\_amount IN NUMBER

) RETURN BOOLEAN

IS

v\_balance NUMBER;

BEGIN

-- Retrieve the balance for the given account ID

SELECT balance

INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

-- Check if the balance is sufficient

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Account ID not found

RETURN FALSE;

WHEN OTHERS THEN

-- Handle other exceptions

RETURN FALSE;

END HasSufficientBalance;

/

**Exercise 5: Triggers**

**Scenario-1**:

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

**ANSWER:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Scenario-2:**

Question: Maintain an audit log for all transactions.

**ANSWER:**

CREATE TABLE AuditLog (

AuditID NUMBER PRIMARY KEY,

TransactionID NUMBER,

ChangeDate DATE,

Operation VARCHAR2(10)

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT OR UPDATE OR DELETE ON Transactions

FOR EACH ROW

BEGIN

IF INSERTING THEN

INSERT INTO AuditLog (AuditID, TransactionID, ChangeDate, Operation)

VALUES (AuditLog\_seq.NEXTVAL, :NEW.TransactionID, SYSDATE, 'INSERT');

ELSIF UPDATING THEN

INSERT INTO AuditLog (AuditID, TransactionID, ChangeDate, Operation)

VALUES (AuditLog\_seq.NEXTVAL, :NEW.TransactionID, SYSDATE, 'UPDATE');

ELSIF DELETING THEN

INSERT INTO AuditLog (AuditID, TransactionID, ChangeDate, Operation)

VALUES (AuditLog\_seq.NEXTVAL, :OLD.TransactionID, SYSDATE, 'DELETE');

END IF;

END;

/

**Scenario-3:**

Question: Enforce business rules on deposits and withdrawals.

**ANSWER:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > (SELECT Balance FROM Accounts WHERE AccountID = :NEW.AccountID) THEN

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

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:** Generate monthly statements for all customers.

**ANSWER:**

DECLARE

CURSOR cur\_statements IS

SELECT CustomerID, TransactionDate, Amount, TransactionType

FROM Transactions

WHERE TransactionDate BETWEEN ADD\_MONTHS(TRUNC(SYSDATE, 'MM'), -1) AND TRUNC(SYSDATE, 'MM') - 1;

BEGIN

FOR rec IN cur\_statements LOOP

DBMS\_OUTPUT.PUT\_LINE('CustomerID: ' || rec.CustomerID || ', Date: ' || rec.TransactionDate || ', Amount: ' || rec.Amount || ', Type: ' || rec.TransactionType);

END LOOP;

END;

/

**Scenario-2:**

Question: Apply annual fee to all accounts.

**ANSWER:**

DECLARE

CURSOR cur\_fees IS

SELECT AccountID, Balance

FROM Accounts;

v\_annual\_fee NUMBER := 50;

BEGIN

FOR rec IN cur\_fees LOOP

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee

WHERE AccountID = rec.AccountID;

END LOOP;

END;

/

**Scenario-3:**

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

**ANSWER:**

DECLARE

CURSOR cur\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

v\_new\_interest\_rate NUMBER := 7; -- Example new interest rate

BEGIN

FOR rec IN cur\_loans LOOP

UPDATE Loans

SET InterestRate = v\_new\_interest\_rate

WHERE LoanID = rec.LoanID;

END LOOP;

END;

/

**Exercise 7: Packages**

**Scenario-1:**

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

**ANSWER:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

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

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

FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

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

BEGIN

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

VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

END;

PROCEDURE UpdateCustomerDetails(p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) IS

BEGIN

UPDATE Customers

SET Name = p\_Name, DOB = p\_DOB, Balance = p\_Balance, LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

END;

FUNCTION GetCustomerBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

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

RETURN v\_Balance;

END;

END CustomerManagement;

/

**Scenario-2:**

Question: Create a package to manage employee data.

**ANSWER:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

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

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

FUNCTION CalculateAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

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

PROCEDURE UpdateEmployeeDetails(p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2) IS

BEGIN

UPDATE Employees

SET Name = p\_Name, Position = p\_Position, Salary = p\_Salary, Department = p\_Department

WHERE EmployeeID = p\_EmployeeID;

END;

FUNCTION CalculateAnnualSalary(p\_EmployeeID NUMBER) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

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

RETURN v\_Salary \* 12;

END;

END EmployeeManagement;

/

**Scenario-3:**

Question: Group all account-related operations into a package.

**ANSWER:**

CREATE OR REPLACE PACKAGE AccountOperations AS

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

PROCEDURE CloseAccount(p\_AccountID NUMBER);

FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

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

BEGIN

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

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

END;

PROCEDURE CloseAccount(p\_AccountID NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_AccountID;

END;

FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

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

RETURN v\_TotalBalance;

END;

END AccountOperations;

/