**WEEK 2 : PL/SQL**

**Exercise 1: Control Structures**

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

* 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.

**CODE :**

DECLARE  
 CURSOR cur\_customers IS SELECT CustomerID, DOB FROM Customers;  
 v\_age NUMBER;  
BEGIN  
 FOR cust IN cur\_customers LOOP  
 v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, cust.DOB) / 12);  
 IF v\_age > 60 THEN  
 UPDATE Loans SET InterestRate = InterestRate - 1  
 WHERE CustomerID = cust.CustomerID;  
 END IF;  
 END LOOP;  
 COMMIT;  
END;

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

* 1. 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.

**CODE :**

BEGIN  
 UPDATE Customers SET IsVIP = TRUE WHERE Balance > 10000;  
 COMMIT;  
END;

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

* 1. 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 :**

BEGIN  
 FOR rec IN (SELECT LoanID, CustomerID FROM Loans   
 WHERE EndDate <= SYSDATE + 30) LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID ||   
 ' for Customer ' || rec.CustomerID ||   
 ' is due within 30 days.');  
 END LOOP;  
END;

**Exercise 2: Error Handling**

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

* 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.

**CODE :**

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');  
 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;  
EXCEPTION  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE(SQLERRM);  
END;

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

* 1. 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.

**CODE :**

CREATE OR REPLACE PROCEDURE UpdateSalary(  
 p\_emp\_id IN NUMBER,  
 p\_percent IN NUMBER  
) IS  
BEGIN  
 UPDATE Employees SET Salary = Salary + (Salary \* p\_percent / 100)  
 WHERE EmployeeID = p\_emp\_id;  
 IF SQL%ROWCOUNT = 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID not found');  
 END IF;  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE(SQLERRM);  
 ROLLBACK;  
END;

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

* 1. 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 :**

CREATE OR REPLACE PROCEDURE AddNewCustomer(  
 p\_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\_id, p\_name, p\_dob, p\_balance, SYSDATE);  
 COMMIT;  
EXCEPTION  
 WHEN DUP\_VAL\_ON\_INDEX THEN  
 DBMS\_OUTPUT.PUT\_LINE('Customer already exists');  
 ROLLBACK;  
END;

**Exercise 3: Stored Procedures**

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

* 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.

**CODE :**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  
BEGIN  
 UPDATE Accounts SET Balance = Balance + (Balance \* 0.01)  
 WHERE AccountType = 'Savings';  
 COMMIT;  
END;

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

* 1. 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.

**CODE :**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(  
 p\_department IN VARCHAR2,  
 p\_bonus\_percent IN NUMBER  
) IS  
BEGIN  
 UPDATE Employees SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)  
 WHERE Department = p\_department;  
 COMMIT;  
END;

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

* 1. 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 :**

CREATE OR REPLACE PROCEDURE TransferFunds(  
 p\_from IN NUMBER,  
 p\_to IN NUMBER,  
 p\_amount IN NUMBER  
) IS  
 v\_balance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from;  
 IF v\_balance < p\_amount THEN  
 RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient balance');  
 END IF;  
 UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from;  
 UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to;  
 COMMIT;  
END;

**Exercise 4: Functions**

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

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

**CODE :**

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE) RETURN NUMBER IS  
BEGIN  
 RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob)/12);  
END;

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

* 1. 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**.**

**CODE :**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(  
 p\_amount IN NUMBER,  
 p\_rate IN NUMBER,  
 p\_years IN NUMBER  
) RETURN NUMBER IS  
 v\_monthly\_rate NUMBER := p\_rate / 12 / 100;  
 v\_months NUMBER := p\_years \* 12;  
BEGIN  
 RETURN (p\_amount \* v\_monthly\_rate) / (1 - POWER(1 + v\_monthly\_rate, -v\_months));  
END;

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

* 1. 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 :**

CREATE OR REPLACE FUNCTION HasSufficientBalance(  
 p\_account\_id IN NUMBER,  
 p\_amount IN 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;  
END;

**Exercise 5: Triggers**

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

* 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.

**CODE :**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified  
BEFORE UPDATE ON Customers  
FOR EACH ROW  
BEGIN  
 :NEW.LastModified := SYSDATE;  
END;

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

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

**CODE :**

CREATE OR REPLACE TRIGGER LogTransaction  
AFTER INSERT ON Transactions  
FOR EACH ROW  
BEGIN  
 INSERT INTO AuditLog (TransactionID, LogDate)  
 VALUES (:NEW.TransactionID, SYSDATE);  
END;

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

* 1. 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 :**

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(-20004, 'Withdrawal exceeds balance');  
 ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20005, 'Deposit amount must be positive');  
 END IF;  
END;

**Exercise 6: Cursors**

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

* 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.

**CODE :**

DECLARE  
 CURSOR cur\_statements IS  
 SELECT CustomerID, TransactionDate, Amount, TransactionType FROM Transactions  
 WHERE TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE);  
BEGIN  
 FOR rec IN cur\_statements LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec.CustomerID || ' | ' ||   
 rec.TransactionType || ': ' || rec.Amount || ' on ' || rec.TransactionDate);  
 END LOOP;  
END;

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

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

**CODE :**

DECLARE  
 CURSOR cur\_accounts IS SELECT AccountID FROM Accounts;  
BEGIN  
 FOR acc IN cur\_accounts LOOP  
 UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = acc.AccountID;  
 END LOOP;  
 COMMIT;  
END;

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

* 1. 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 :**

DECLARE  
 CURSOR cur\_loans IS SELECT LoanID FROM Loans;  
BEGIN  
 FOR l IN cur\_loans LOOP  
 UPDATE Loans SET InterestRate = InterestRate + 0.5 WHERE LoanID = l.LoanID;  
 END LOOP;  
 COMMIT;  
END;

**Exercise 7: Packages**

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

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

**CODE :**

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\_balance NUMBER);  
 FUNCTION GetBalance(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, SYSDATE);  
 END;  
   
 PROCEDURE UpdateCustomer(p\_id NUMBER, p\_balance NUMBER) IS  
 BEGIN  
 UPDATE Customers SET Balance = p\_balance WHERE CustomerID = p\_id;  
 END;  
   
 FUNCTION GetBalance(p\_id NUMBER) RETURN NUMBER IS  
 v\_bal NUMBER;  
 BEGIN  
 SELECT Balance INTO v\_bal FROM Customers WHERE CustomerID = p\_id;  
 RETURN v\_bal;  
 END;  
END CustomerManagement;

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

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

**CODE :**

CREATE OR REPLACE PACKAGE EmployeeManagement AS  
 PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2, p\_date DATE);  
 PROCEDURE UpdateEmployee(p\_id NUMBER, p\_sal NUMBER);  
 FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;  
END EmployeeManagement;  
/  
  
CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS  
 PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2, p\_date DATE) IS  
 BEGIN  
 INSERT INTO Employees VALUES (p\_id, p\_name, p\_pos, p\_sal, p\_dept, p\_date);  
 END;  
   
 PROCEDURE UpdateEmployee(p\_id NUMBER, p\_sal NUMBER) IS  
 BEGIN  
 UPDATE Employees SET Salary = p\_sal WHERE EmployeeID = p\_id;  
 END;  
   
 FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER IS  
 v\_sal NUMBER;  
 BEGIN  
 SELECT Salary INTO v\_sal FROM Employees WHERE EmployeeID = p\_id;  
 RETURN v\_sal \* 12;  
 END;  
END EmployeeManagement;

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

* 1. 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 :**

CREATE OR REPLACE PACKAGE AccountOperations AS  
 PROCEDURE OpenAccount(p\_acc\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_bal NUMBER);  
 PROCEDURE CloseAccount(p\_acc\_id NUMBER);  
 FUNCTION GetTotalBalance(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\_type VARCHAR2, p\_bal NUMBER) IS  
 BEGIN  
 INSERT INTO Accounts VALUES (p\_acc\_id, p\_cust\_id, p\_type, p\_bal, SYSDATE);  
 END;  
  
 PROCEDURE CloseAccount(p\_acc\_id NUMBER) IS  
 BEGIN  
 DELETE FROM Accounts WHERE AccountID = p\_acc\_id;  
 END;  
  
 FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER IS  
 v\_total NUMBER;  
 BEGIN  
 SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_cust\_id;  
 RETURN v\_total;  
 END;  
END AccountOperations;