PL/SQL Exercises with Code and Output

# Exercise 1: Scenario 1

Apply 1% discount to loan interest rates for customers above 60 years old.

## PL/SQL Code:

DECLARE  
 v\_age NUMBER;  
BEGIN  
 FOR cust IN (SELECT CustomerID, DOB FROM Customers) LOOP  
 v\_age := TRUNC(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;

## Expected Output:

Interest rates updated for customers aged above 60.

# Exercise 1: Scenario 2

Promote customers to VIP based on balance.

## PL/SQL Code:

BEGIN  
 UPDATE Customers  
 SET Balance = Balance,  
 LastModified = SYSDATE  
 WHERE Balance > 10000;  
 -- Assuming IsVIP column exists in Customers table  
 UPDATE Customers  
 SET IsVIP = 'TRUE'  
 WHERE Balance > 10000;  
 COMMIT;  
END;

## Expected Output:

IsVIP flag set to TRUE for customers with balance over $10,000.

# Exercise 1: Scenario 3

Send reminders for loans due within the next 30 days.

## PL/SQL Code:

BEGIN  
 FOR rec IN (SELECT l.LoanID, c.Name FROM Loans l JOIN Customers c ON l.CustomerID = c.CustomerID  
 WHERE l.EndDate <= SYSDATE + 30) LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID || ' for customer ' || rec.Name || ' is due soon.');  
 END LOOP;  
END;

## Expected Output:

Reminder messages printed for each due loan within 30 days.

# Exercise 2: Scenario 1

Safe fund transfer between accounts with exception handling.

## PL/SQL Code:

CREATE OR REPLACE PROCEDURE SafeTransferFunds (  
 p\_fromAccountID IN NUMBER,  
 p\_toAccountID IN NUMBER,  
 p\_amount IN NUMBER  
) IS  
 v\_balance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_fromAccountID;  
  
 IF v\_balance < p\_amount THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds.');  
 END IF;  
  
 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 OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);  
END;

## Expected Output:

Handles insufficient funds and rolls back on error.

# Exercise 2: Scenario 2

Update salary and handle non-existent employee ID.

## PL/SQL Code:

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

## Expected Output:

Salary updated or error logged if employee not found.

# Exercise 2: Scenario 3

Add new customer with error handling for duplicate ID.

## PL/SQL 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;

## Expected Output:

Handles duplicates and logs error if customer ID already exists.

# Exercise 3: Scenario 1

Process monthly interest for all savings accounts.

## PL/SQL Code:

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

## Expected Output:

Interest applied to all savings accounts.

# Exercise 3: Scenario 2

Implement bonus scheme for employees.

## PL/SQL 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;

## Expected Output:

Bonus added to all employees in the given department.

# Exercise 3: Scenario 3

Transfer funds between customer accounts.

## PL/SQL Code:

CREATE OR REPLACE PROCEDURE TransferFunds (  
 p\_fromAccount IN NUMBER,  
 p\_toAccount IN NUMBER,  
 p\_amount IN NUMBER  
) IS  
 v\_balance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_fromAccount;  
  
 IF v\_balance < p\_amount THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance.');  
 END IF;  
  
 UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_fromAccount;  
 UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_toAccount;  
 COMMIT;  
END;

## Expected Output:

Transfers funds if balance is sufficient.

# Exercise 4: Scenario 1

Calculate age of customers.

## PL/SQL Code:

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

## Expected Output:

Returns the customer's age.

# Exercise 4: Scenario 2

Compute monthly installment for a loan.

## PL/SQL Code:

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (  
 p\_amount NUMBER,  
 p\_rate NUMBER,  
 p\_years NUMBER  
) RETURN NUMBER IS  
 v\_monthly\_rate NUMBER := p\_rate / 1200;  
 v\_months NUMBER := p\_years \* 12;  
BEGIN  
 RETURN (p\_amount \* v\_monthly\_rate) / (1 - POWER(1 + v\_monthly\_rate, -v\_months));  
END;

## Expected Output:

Returns monthly EMI based on input values.

# Exercise 4: Scenario 3

Check sufficient balance before transaction.

## PL/SQL Code:

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;

## Expected Output:

Returns TRUE if account has sufficient balance.

# Exercise 5: Scenario 1

Update LastModified on customer update.

## PL/SQL Code:

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

## Expected Output:

Updates LastModified column on changes.

# Exercise 5: Scenario 2

Maintain audit log for all transactions.

## PL/SQL Code:

CREATE TABLE AuditLog (  
 LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  
 TransactionID NUMBER,  
 ActionDate DATE DEFAULT SYSDATE  
);  
  
CREATE OR REPLACE TRIGGER LogTransaction  
AFTER INSERT ON Transactions  
FOR EACH ROW  
BEGIN  
 INSERT INTO AuditLog (TransactionID)  
 VALUES (:NEW.TransactionID);  
END;

## Expected Output:

Logs transaction in AuditLog table.

# Exercise 5: Scenario 3

Enforce deposit and withdrawal rules.

## PL/SQL 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(-20003, 'Insufficient funds for withdrawal.');  
 ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20004, 'Deposit must be greater than zero.');  
 END IF;  
END;

## Expected Output:

Prevents overdrawn withdrawals and invalid deposits.

# Exercise 6: Scenario 1

Generate monthly statements for all customers.

## PL/SQL Code:

DECLARE  
 CURSOR txn\_cursor IS  
 SELECT c.Name, t.Amount, t.TransactionType, t.TransactionDate  
 FROM Customers c  
 JOIN Accounts a ON c.CustomerID = a.CustomerID  
 JOIN Transactions t ON a.AccountID = t.AccountID  
 WHERE t.TransactionDate >= TRUNC(SYSDATE, 'MM');  
  
BEGIN  
 FOR rec IN txn\_cursor LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec.Name || ', ' || rec.TransactionType || ': $' || rec.Amount);  
 END LOOP;  
END;

## Expected Output:

Prints all transactions for current month.

# Exercise 6: Scenario 2

Apply annual maintenance fee.

## PL/SQL Code:

DECLARE  
 CURSOR fee\_cursor IS  
 SELECT AccountID FROM Accounts;  
  
BEGIN  
 FOR rec IN fee\_cursor LOOP  
 UPDATE Accounts  
 SET Balance = Balance - 100  
 WHERE AccountID = rec.AccountID;  
 END LOOP;  
 COMMIT;  
END;

## Expected Output:

Deducts 100 as annual fee from all accounts.

# Exercise 6: Scenario 3

Update loan interest rates based on policy.

## PL/SQL Code:

DECLARE  
 CURSOR loan\_cursor IS  
 SELECT LoanID, InterestRate FROM Loans;  
  
BEGIN  
 FOR rec IN loan\_cursor LOOP  
 UPDATE Loans  
 SET InterestRate = rec.InterestRate \* 1.05  
 WHERE LoanID = rec.LoanID;  
 END LOOP;  
 COMMIT;  
END;

## Expected Output:

Increases all loan interest rates by 5%.

# Exercise 7: Scenario 1

Group customer operations in package.

## PL/SQL 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\_name VARCHAR2);  
 FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;  
END CustomerManagement;

## Expected Output:

Declares customer operations package.

# Exercise 7: Scenario 2

Manage employee data in a package.

## PL/SQL Code:

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);  
 FUNCTION CalculateAnnualSalary(p\_id NUMBER) RETURN NUMBER;  
END EmployeeManagement;

## Expected Output:

Declares employee management package.

# Exercise 7: Scenario 3

Group account operations in a package.

## PL/SQL Code:

CREATE OR REPLACE PACKAGE AccountOperations AS  
 PROCEDURE OpenAccount(p\_accID NUMBER, p\_custID NUMBER, p\_type VARCHAR2, p\_balance NUMBER);  
 PROCEDURE CloseAccount(p\_accID NUMBER);  
 FUNCTION GetTotalBalance(p\_custID NUMBER) RETURN NUMBER;  
END AccountOperations;

## Expected Output:

Declares account operations package.