PL/SQL Exercise Answers

# Exercise 1 - Scenario 1: Apply Discount for Customers above 60

BEGIN  
 FOR rec IN (SELECT CustomerID FROM Customers WHERE MONTHS\_BETWEEN(SYSDATE, DOB)/12 > 60) LOOP  
 UPDATE Loans  
 SET InterestRate = InterestRate - 1  
 WHERE CustomerID = rec.CustomerID;  
 END LOOP;  
END;  
/

# Exercise 1 - Scenario 2: Promote Customers to VIP Based on Balance

BEGIN  
 FOR rec IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP  
 UPDATE Customers  
 SET IsVIP = TRUE  
 WHERE CustomerID = rec.CustomerID;  
 END LOOP;  
END;  
/

# Exercise 1 - Scenario 3: Loan Reminders

BEGIN  
 FOR rec IN (SELECT CustomerID, LoanID FROM Loans   
 WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID || ' for Customer ID ' || rec.CustomerID || ' is due soon.');  
 END LOOP;  
END;  
/

# Exercise 2 - Scenario 1: SafeTransferFunds Procedure

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_from\_acc NUMBER, p\_to\_acc NUMBER, p\_amount NUMBER) IS  
BEGIN  
 UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_acc AND Balance >= p\_amount;  
 IF SQL%ROWCOUNT = 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds or invalid source account.');  
 END IF;  
  
 UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_acc;  
  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
END;  
/

# Exercise 2 - Scenario 2: UpdateSalary Procedure

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_emp\_id NUMBER, p\_percent NUMBER) IS  
BEGIN  
 UPDATE Employees SET Salary = Salary + (Salary \* p\_percent / 100) WHERE EmployeeID = p\_emp\_id;  
 IF SQL%ROWCOUNT = 0 THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: Employee not found.');  
 END IF;  
EXCEPTION  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);  
END;  
/

# Exercise 2 - Scenario 3: AddNewCustomer Procedure

CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS  
BEGIN  
 INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)  
 VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE);  
EXCEPTION  
 WHEN DUP\_VAL\_ON\_INDEX THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID already exists.');  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);  
END;  
/

# Exercise 3 - Scenario 1: ProcessMonthlyInterest Procedure

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

# Exercise 3 - Scenario 2: UpdateEmployeeBonus Procedure

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

# Exercise 3 - Scenario 3: TransferFunds Procedure

CREATE OR REPLACE PROCEDURE TransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amount NUMBER) IS  
BEGIN  
 UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from AND Balance >= p\_amount;  
 IF SQL%ROWCOUNT = 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance or invalid source account.');  
 END IF;  
  
 UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to;  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);  
END;  
/

# Exercise 4 - Scenario 1: CalculateAge Function

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

# Exercise 4 - Scenario 2: CalculateMonthlyInstallment Function

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_amount NUMBER, p\_rate NUMBER, p\_years 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;  
/

# Exercise 4 - Scenario 3: HasSufficientBalance Function

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_acc\_id NUMBER, p\_amt NUMBER) RETURN BOOLEAN IS  
 v\_balance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_acc\_id;  
 RETURN v\_balance >= p\_amt;  
EXCEPTION  
 WHEN NO\_DATA\_FOUND THEN  
 RETURN FALSE;  
END;  
/

# Exercise 5 - Scenario 1: UpdateCustomerLastModified Trigger

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

# Exercise 5 - Scenario 2: LogTransaction Trigger

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

# Exercise 5 - Scenario 3: CheckTransactionRules Trigger

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 amount must be positive.');  
 END IF;  
END;  
/

# Exercise 6 - Scenario 1: GenerateMonthlyStatements Cursor

DECLARE  
 CURSOR txn\_cursor IS  
 SELECT AccountID, Amount, TransactionType FROM Transactions  
 WHERE TO\_CHAR(TransactionDate, 'MM-YYYY') = TO\_CHAR(SYSDATE, 'MM-YYYY');  
BEGIN  
 FOR txn IN txn\_cursor LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || txn.AccountID || ', ' ||  
 txn.TransactionType || ': ' || txn.Amount);  
 END LOOP;  
END;  
/

# Exercise 6 - Scenario 2: ApplyAnnualFee Cursor

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

# Exercise 6 - Scenario 3: UpdateLoanInterestRates Cursor

DECLARE  
 CURSOR loan\_cursor IS SELECT LoanID, InterestRate FROM Loans;  
BEGIN  
 FOR loan IN loan\_cursor LOOP  
 UPDATE Loans  
 SET InterestRate = loan.InterestRate + 0.5  
 WHERE LoanID = loan.LoanID;  
 END LOOP;  
 COMMIT;  
END;  
/

# Exercise 7 - Scenario 1: CustomerManagement Package

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;  
/  
  
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\_name VARCHAR2) IS  
 BEGIN  
 UPDATE Customers SET Name = p\_name WHERE CustomerID = p\_id;  
 END;  
  
 FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER IS  
 v\_balance NUMBER;  
 BEGIN  
 SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_id;  
 RETURN v\_balance;  
 END;  
END CustomerManagement;  
/

# Exercise 7 - Scenario 2: EmployeeManagement Package

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

# Exercise 7 - Scenario 3: AccountOperations Package

CREATE OR REPLACE PACKAGE AccountOperations AS  
 PROCEDURE OpenAccount(p\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER);  
 PROCEDURE CloseAccount(p\_id NUMBER);  
 FUNCTION TotalBalance(p\_cust\_id NUMBER) RETURN NUMBER;  
END AccountOperations;  
/  
  
CREATE OR REPLACE PACKAGE BODY AccountOperations AS  
 PROCEDURE OpenAccount(p\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS  
 BEGIN  
 INSERT INTO Accounts VALUES (p\_id, p\_cust\_id, p\_type, p\_balance, SYSDATE);  
 END;  
  
 PROCEDURE CloseAccount(p\_id NUMBER) IS  
 BEGIN  
 DELETE FROM Accounts WHERE AccountID = p\_id;  
 END;  
  
 FUNCTION TotalBalance(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 NVL(v\_total, 0);  
 END;  
END AccountOperations;  
/