PL/SQL Exercise Answers

## Exercise 1 - Scenario 1

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

## Exercise 1 - Scenario 2

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

## Exercise 1 - Scenario 3

DECLARE  
 CURSOR cur\_loans IS SELECT \* FROM Loans WHERE EndDate <= SYSDATE + 30;  
BEGIN  
 FOR rec IN cur\_loans LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan for Customer ID ' || rec.CustomerID || ' is due on ' || rec.EndDate);  
 END LOOP;  
END;

## Exercise 2 - Scenario 1

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amount NUMBER) AS  
BEGIN  
 UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from;  
 UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to;  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Error in transferring funds: ' || SQLERRM);  
END;

## Exercise 2 - Scenario 2

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_emp\_id NUMBER, p\_percent NUMBER) AS  
BEGIN  
 UPDATE Employees SET Salary = Salary + (Salary \* p\_percent / 100) WHERE EmployeeID = p\_emp\_id;  
 IF SQL%ROWCOUNT = 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Employee not found');  
 END IF;  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
END;

## Exercise 2 - Scenario 3

CREATE OR REPLACE PROCEDURE AddNewCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) AS  
BEGIN  
 INSERT INTO Customers VALUES(p\_id, p\_name, p\_dob, p\_balance, SYSDATE);  
 COMMIT;  
EXCEPTION  
 WHEN DUP\_VAL\_ON\_INDEX THEN  
 DBMS\_OUTPUT.PUT\_LINE('Customer ID already exists.');  
END;

## Exercise 3 - Scenario 1

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

## Exercise 3 - Scenario 2

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

## Exercise 3 - Scenario 3

CREATE OR REPLACE PROCEDURE TransferFunds(p\_from NUMBER, p\_to NUMBER, p\_amount NUMBER) AS  
 v\_balance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from;  
 IF v\_balance >= p\_amount THEN  
 UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from;  
 UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to;  
 COMMIT;  
 ELSE  
 DBMS\_OUTPUT.PUT\_LINE('Insufficient balance');  
 END IF;  
END;

## Exercise 4 - Scenario 1

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

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

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_acc\_id NUMBER, p\_amount NUMBER) RETURN BOOLEAN IS  
 v\_balance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_acc\_id;  
 RETURN v\_balance >= p\_amount;  
END;

## Exercise 5 - Scenario 1

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

## Exercise 5 - Scenario 2

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

## Exercise 5 - Scenario 3

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(-20001, 'Insufficient funds');  
 ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20002, 'Invalid deposit amount');  
 END IF;  
END;

## Exercise 6 - Scenario 1

DECLARE  
 CURSOR cur\_transactions IS  
 SELECT \* FROM Transactions WHERE TRUNC(TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');  
BEGIN  
 FOR rec IN cur\_transactions LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Customer Transaction: ID = ' || rec.AccountID || ', Amount = ' || rec.Amount);  
 END LOOP;  
END;

## Exercise 6 - Scenario 2

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

## Exercise 6 - Scenario 3

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

## Exercise 7 - Scenario 1

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

CREATE OR REPLACE PACKAGE EmployeeManagement AS  
 PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_sal NUMBER, p\_dept VARCHAR2);  
 PROCEDURE UpdateEmployee(p\_id NUMBER, p\_name VARCHAR2);  
 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) IS  
 BEGIN  
 INSERT INTO Employees VALUES(p\_id, p\_name, p\_pos, p\_sal, p\_dept, SYSDATE);  
 END;  
  
 PROCEDURE UpdateEmployee(p\_id NUMBER, p\_name VARCHAR2) IS  
 BEGIN  
 UPDATE Employees SET Name = p\_name WHERE EmployeeID = p\_id;  
 END;  
  
 FUNCTION GetAnnualSalary(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

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 GetTotalBalance(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 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;