**Week2**

**PLSQL**

**Q1) Scenario 1**

**DECLARE**

**v\_customer\_id NUMBER;**

**v\_interest\_rate NUMBER;**

**BEGIN**

**FOR rec IN (SELECT CustomerID, EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM DOB) AS Age**

**FROM Customers**

**WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM DOB) > 60) LOOP**

**FOR loan\_rec IN (SELECT LoanID, InterestRate FROM Loans WHERE CustomerID = rec.CustomerID) LOOP**

**v\_interest\_rate := loan\_rec.InterestRate - 1; -- Applying 1% discount**

**UPDATE Loans**

**SET InterestRate = v\_interest\_rate**

**WHERE LoanID = loan\_rec.LoanID;**

**END LOOP;**

**END LOOP;**

**COMMIT;**

**END;**

**/**

**Scenario 2**

**ALTER TABLE Customers ADD (IsVIP CHAR(1) DEFAULT 'N');**

**BEGIN**

**UPDATE Customers**

**SET IsVIP = CASE**

**WHEN Balance > 10000 THEN 'Y'**

**ELSE 'N'**

**END;**

**COMMIT;**

**END;**

**/**

**Scenario 3**

**BEGIN**

**FOR rec IN (SELECT l.LoanID, l.CustomerID, l.EndDate, c.Name**

**FROM Loans l**

**JOIN Customers c ON l.CustomerID = c.CustomerID**

**WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP**

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

**END LOOP;**

**END;**

**/**

**Q2) Scenario 1**

**CREATE OR REPLACE PROCEDURE SafeTransferFunds (**

**p\_from\_account\_id IN NUMBER,**

**p\_to\_account\_id IN NUMBER,**

**p\_amount IN NUMBER**

**) IS**

**insufficient\_funds EXCEPTION;**

**v\_from\_balance NUMBER;**

**BEGIN**

**SELECT Balance INTO v\_from\_balance**

**FROM Accounts**

**WHERE AccountID = p\_from\_account\_id**

**FOR UPDATE;**

**IF v\_from\_balance < p\_amount THEN**

**RAISE insufficient\_funds;**

**END IF;**

**UPDATE Accounts**

**SET Balance = Balance - p\_amount, LastModified = SYSDATE**

**WHERE AccountID = p\_from\_account\_id;**

**UPDATE Accounts**

**SET Balance = Balance + p\_amount, LastModified = SYSDATE**

**WHERE AccountID = p\_to\_account\_id;**

**COMMIT;**

**EXCEPTION**

**WHEN insufficient\_funds THEN**

**ROLLBACK;**

**DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_account\_id);**

**WHEN OTHERS THEN**

**ROLLBACK;**

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

**END SafeTransferFunds;**

**/**

**Scenario 2**

**CREATE OR REPLACE PROCEDURE UpdateSalary (**

**p\_employee\_id IN NUMBER,**

**p\_percentage IN NUMBER**

**) IS**

**employee\_not\_found EXCEPTION;**

**v\_current\_salary NUMBER;**

**BEGIN**

**SELECT Salary INTO v\_current\_salary**

**FROM Employees**

**WHERE EmployeeID = p\_employee\_id**

**FOR UPDATE;**

**UPDATE Employees**

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

**WHERE EmployeeID = p\_employee\_id;**

**COMMIT;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**RAISE employee\_not\_found;**

**WHEN employee\_not\_found THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error: Employee with ID ' || p\_employee\_id || ' not found');**

**WHEN OTHERS THEN**

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

**END UpdateSalary;**

**/**

**Scenario 3**

**CREATE OR REPLACE PROCEDURE AddNewCustomer (**

**p\_customer\_id IN NUMBER,**

**p\_name IN VARCHAR2,**

**p\_dob IN DATE,**

**p\_balance IN NUMBER**

**) IS**

**customer\_exists EXCEPTION;**

**BEGIN**

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

**VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);**

**COMMIT;**

**EXCEPTION**

**WHEN DUP\_VAL\_ON\_INDEX THEN**

**RAISE customer\_exists;**

**WHEN customer\_exists THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customer\_id || ' already exists');**

**WHEN OTHERS THEN**

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

**END AddNewCustomer;**

**/**

**Q3)**

**Scenario 1**

**CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS**

**BEGIN**

**UPDATE Accounts**

**SET Balance = Balance \* 1.01, LastModified = SYSDATE**

**WHERE AccountType = 'Savings';**

**COMMIT;**

**END ProcessMonthlyInterest;**

**/**

**Scenario 2**

**CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (**

**p\_department IN VARCHAR2,**

**p\_bonus\_percentage IN NUMBER**

**) IS**

**BEGIN**

**UPDATE Employees**

**SET Salary = Salary + (Salary \* p\_bonus\_percentage / 100), LastModified = SYSDATE**

**WHERE Department = p\_department;**

**COMMIT;**

**END UpdateEmployeeBonus;**

**/**

**Scenario 3**

**CREATE OR REPLACE PROCEDURE TransferFunds (**

**p\_from\_account\_id IN NUMBER,**

**p\_to\_account\_id IN NUMBER,**

**p\_amount IN NUMBER**

**) IS**

**v\_from\_balance NUMBER;**

**BEGIN**

**SELECT Balance INTO v\_from\_balance**

**FROM Accounts**

**WHERE AccountID = p\_from\_account\_id**

**FOR UPDATE;**

**IF v\_from\_balance < p\_amount THEN**

**RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in the source account.');**

**END IF;**

**UPDATE Accounts**

**SET Balance = Balance - p\_amount, LastModified = SYSDATE**

**WHERE AccountID = p\_from\_account\_id;**

**UPDATE Accounts**

**SET Balance = Balance + p\_amount, LastModified = SYSDATE**

**WHERE AccountID = p\_to\_account\_id;**

**COMMIT;**

**EXCEPTION**

**WHEN OTHERS THEN**

**ROLLBACK;**

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

**END TransferFunds;**

**/**

**Q4) Scenario 1**

**CREATE OR REPLACE FUNCTION CalculateAge (**

**p\_dob IN DATE**

**) RETURN NUMBER IS**

**v\_age NUMBER;**

**BEGIN**

**SELECT TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12)**

**INTO v\_age**

**FROM DUAL;**

**RETURN v\_age;**

**END CalculateAge;**

**/**

**Scenario 2**

**CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (**

**p\_loan\_amount IN NUMBER,**

**p\_interest\_rate IN NUMBER,**

**p\_duration\_years IN NUMBER**

**) RETURN NUMBER IS**

**v\_monthly\_interest\_rate NUMBER;**

**v\_number\_of\_payments NUMBER;**

**v\_monthly\_installment NUMBER;**

**BEGIN**

**v\_monthly\_interest\_rate := p\_interest\_rate / 1200;**

**v\_number\_of\_payments := p\_duration\_years \* 12;**

**v\_monthly\_installment := p\_loan\_amount \* v\_monthly\_interest\_rate /**

**(1 - POWER(1 + v\_monthly\_interest\_rate, -v\_number\_of\_payments));**

**RETURN v\_monthly\_installment;**

**END CalculateMonthlyInstallment;**

**/**

**Scenario 3**

**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;**

**IF v\_balance >= p\_amount THEN**

**RETURN TRUE;**

**ELSE**

**RETURN FALSE;**

**END IF;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**RETURN FALSE;**

**WHEN OTHERS THEN**

**RETURN FALSE;**

**END HasSufficientBalance;**

**/**

**Q5) Scenario 1**

**CREATE OR REPLACE TRIGGER UpdateCustomerLastModified**

**BEFORE UPDATE ON Customers**

**FOR EACH ROW**

**BEGIN**

**:NEW.LastModified := SYSDATE;**

**END UpdateCustomerLastModified;**

**/**

**Scenario 2**

**CREATE TABLE AuditLog (**

**AuditID NUMBER PRIMARY KEY,**

**TransactionID NUMBER,**

**AccountID NUMBER,**

**TransactionDate DATE,**

**Amount NUMBER,**

**TransactionType VARCHAR2(10),**

**LogTimestamp DATE DEFAULT SYSDATE**

**);**

**CREATE OR REPLACE TRIGGER LogTransaction**

**AFTER INSERT ON Transactions**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO AuditLog (**

**AuditID, TransactionID, AccountID, TransactionDate, Amount, TransactionType**

**) VALUES (**

**AuditLog\_seq.NEXTVAL, :NEW.TransactionID, :NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType**

**);**

**END LogTransaction;**

**/**

**CREATE SEQUENCE AuditLog\_seq START WITH 1 INCREMENT BY 1;**

**Scenario 3**

**CREATE OR REPLACE TRIGGER CheckTransactionRules**

**BEFORE INSERT ON Transactions**

**FOR EACH ROW**

**DECLARE**

**v\_balance NUMBER;**

**BEGIN**

**IF :NEW.TransactionType = 'Withdrawal' THEN**

**SELECT Balance INTO v\_balance**

**FROM Accounts**

**WHERE AccountID = :NEW.AccountID**

**FOR UPDATE;**

**IF v\_balance < :NEW.Amount THEN**

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

**END IF;**

**ELSIF :NEW.TransactionType = 'Deposit' THEN**

**IF :NEW.Amount <= 0 THEN**

**RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be positive.');**

**END IF;**

**END IF;**

**END CheckTransactionRules;**

**/**

**Q5) Scenario 1**

**DECLARE**

**CURSOR c\_monthly\_transactions IS**

**SELECT t.CustomerID, c.Name, t.TransactionID, t.TransactionDate, t.Amount, t.TransactionType**

**FROM Transactions t**

**JOIN Accounts a ON t.AccountID = a.AccountID**

**JOIN Customers c ON a.CustomerID = c.CustomerID**

**WHERE t.TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST\_DAY(SYSDATE)**

**ORDER BY t.CustomerID;**

**v\_customer\_id Customers.CustomerID%TYPE;**

**v\_name Customers.Name%TYPE;**

**v\_transaction\_id Transactions.TransactionID%TYPE;**

**v\_transaction\_date Transactions.TransactionDate%TYPE;**

**v\_amount Transactions.Amount%TYPE;**

**v\_transaction\_type Transactions.TransactionType%TYPE;**

**BEGIN**

**OPEN c\_monthly\_transactions;**

**LOOP**

**FETCH c\_monthly\_transactions INTO v\_customer\_id, v\_name, v\_transaction\_id, v\_transaction\_date, v\_amount, v\_transaction\_type;**

**EXIT WHEN c\_monthly\_transactions%NOTFOUND;**

**DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id);**

**DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_name);**

**DBMS\_OUTPUT.PUT\_LINE('Transaction ID: ' || v\_transaction\_id);**

**DBMS\_OUTPUT.PUT\_LINE('Date: ' || TO\_CHAR(v\_transaction\_date, 'YYYY-MM-DD'));**

**DBMS\_OUTPUT.PUT\_LINE('Amount: ' || v\_amount);**

**DBMS\_OUTPUT.PUT\_LINE('Type: ' || v\_transaction\_type);**

**DBMS\_OUTPUT.PUT\_LINE('----------------------------------------');**

**END LOOP;**

**CLOSE c\_monthly\_transactions;**

**END;**

**/**

**Scenario 2**

**DECLARE**

**CURSOR c\_apply\_annual\_fee IS**

**SELECT AccountID, Balance**

**FROM Accounts;**

**v\_account\_id Accounts.AccountID%TYPE;**

**v\_balance Accounts.Balance%TYPE;**

**v\_annual\_fee CONSTANT NUMBER := 50; -- Annual fee amount**

**BEGIN**

**OPEN c\_apply\_annual\_fee;**

**LOOP**

**FETCH c\_apply\_annual\_fee INTO v\_account\_id, v\_balance;**

**EXIT WHEN c\_apply\_annual\_fee%NOTFOUND;**

**UPDATE Accounts**

**SET Balance = Balance - v\_annual\_fee, LastModified = SYSDATE**

**WHERE AccountID = v\_account\_id;**

**DBMS\_OUTPUT.PUT\_LINE('Applied annual fee to Account ID: ' || v\_account\_id || ', New Balance: ' || (v\_balance - v\_annual\_fee));**

**END LOOP;**

**CLOSE c\_apply\_annual\_fee;**

**COMMIT;**

**END;**

**/**

**Scenario 3**

**DECLARE**

**CURSOR c\_update\_loan\_interest IS**

**SELECT LoanID, InterestRate**

**FROM Loans;**

**v\_loan\_id Loans.LoanID%TYPE;**

**v\_interest\_rate Loans.InterestRate%TYPE;**

**v\_new\_interest\_rate NUMBER;**

**BEGIN**

**OPEN c\_update\_loan\_interest;**

**LOOP**

**FETCH c\_update\_loan\_interest INTO v\_loan\_id, v\_interest\_rate;**

**EXIT WHEN c\_update\_loan\_interest%NOTFOUND;**

**v\_new\_interest\_rate := v\_interest\_rate \* 1.05; -- Applying new policy: increase by 5%**

**UPDATE Loans**

**SET InterestRate = v\_new\_interest\_rate, LastModified = SYSDATE**

**WHERE LoanID = v\_loan\_id;**

**DBMS\_OUTPUT.PUT\_LINE('Updated Loan ID: ' || v\_loan\_id || ', New Interest Rate: ' || v\_new\_interest\_rate);**

**END LOOP;**

**CLOSE c\_update\_loan\_interest;**

**COMMIT;**

**END;**

**/**

**Q7 ) Scenario 1**

**CREATE OR REPLACE PACKAGE CustomerManagement AS**

**PROCEDURE AddNewCustomer(**

**p\_customer\_id IN NUMBER,**

**p\_name IN VARCHAR2,**

**p\_dob IN DATE,**

**p\_balance IN NUMBER**

**);**

**PROCEDURE UpdateCustomerDetails(**

**p\_customer\_id IN NUMBER,**

**p\_name IN VARCHAR2,**

**p\_dob IN DATE,**

**p\_balance IN NUMBER**

**);**

**FUNCTION GetCustomerBalance(**

**p\_customer\_id IN NUMBER**

**) RETURN NUMBER;**

**END CustomerManagement;**

**/**

**CREATE OR REPLACE PACKAGE BODY CustomerManagement AS**

**PROCEDURE AddNewCustomer(**

**p\_customer\_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\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);**

**COMMIT;**

**EXCEPTION**

**WHEN DUP\_VAL\_ON\_INDEX THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customer\_id || ' already exists');**

**WHEN OTHERS THEN**

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

**END AddNewCustomer;**

**PROCEDURE UpdateCustomerDetails(**

**p\_customer\_id IN NUMBER,**

**p\_name IN VARCHAR2,**

**p\_dob IN DATE,**

**p\_balance IN NUMBER**

**) IS**

**BEGIN**

**UPDATE Customers**

**SET Name = p\_name,**

**DOB = p\_dob,**

**Balance = p\_balance,**

**LastModified = SYSDATE**

**WHERE CustomerID = p\_customer\_id;**

**COMMIT;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customer\_id || ' not found');**

**WHEN OTHERS THEN**

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

**END UpdateCustomerDetails;**

**FUNCTION GetCustomerBalance(**

**p\_customer\_id IN NUMBER**

**) RETURN NUMBER IS**

**v\_balance NUMBER;**

**BEGIN**

**SELECT Balance INTO v\_balance**

**FROM Customers**

**WHERE CustomerID = p\_customer\_id;**

**RETURN v\_balance;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**RETURN NULL;**

**WHEN OTHERS THEN**

**RETURN NULL;**

**END GetCustomerBalance;**

**END CustomerManagement;**

**/**

**Scenario 2**

**CREATE OR REPLACE PACKAGE EmployeeManagement AS**

**PROCEDURE HireEmployee(**

**p\_employee\_id IN NUMBER,**

**p\_name IN VARCHAR2,**

**p\_position IN VARCHAR2,**

**p\_salary IN NUMBER,**

**p\_department IN VARCHAR2,**

**p\_hire\_date IN DATE**

**);**

**PROCEDURE UpdateEmployeeDetails(**

**p\_employee\_id IN NUMBER,**

**p\_name IN VARCHAR2,**

**p\_position IN VARCHAR2,**

**p\_salary IN NUMBER,**

**p\_department IN VARCHAR2,**

**p\_hire\_date IN DATE**

**);**

**FUNCTION CalculateAnnualSalary(**

**p\_employee\_id IN NUMBER**

**) RETURN NUMBER;**

**END EmployeeManagement;**

**/**

**CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS**

**PROCEDURE HireEmployee(**

**p\_employee\_id IN NUMBER,**

**p\_name IN VARCHAR2,**

**p\_position IN VARCHAR2,**

**p\_salary IN NUMBER,**

**p\_department IN VARCHAR2,**

**p\_hire\_date IN DATE**

**) IS**

**BEGIN**

**INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)**

**VALUES (p\_employee\_id, p\_name, p\_position, p\_salary, p\_department, p\_hire\_date);**

**COMMIT;**

**EXCEPTION**

**WHEN DUP\_VAL\_ON\_INDEX THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error: Employee with ID ' || p\_employee\_id || ' already exists');**

**WHEN OTHERS THEN**

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

**END HireEmployee;**

**PROCEDURE UpdateEmployeeDetails(**

**p\_employee\_id IN NUMBER,**

**p\_name IN VARCHAR2,**

**p\_position IN VARCHAR2,**

**p\_salary IN NUMBER,**

**p\_department IN VARCHAR2,**

**p\_hire\_date IN DATE**

**) IS**

**BEGIN**

**UPDATE Employees**

**SET Name = p\_name,**

**Position = p\_position,**

**Salary = p\_salary,**

**Department = p\_department,**

**HireDate = p\_hire\_date,**

**LastModified = SYSDATE**

**WHERE EmployeeID = p\_employee\_id;**

**COMMIT;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error: Employee with ID ' || p\_employee\_id || ' not found');**

**WHEN OTHERS THEN**

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

**END UpdateEmployeeDetails;**

**FUNCTION CalculateAnnualSalary(**

**p\_employee\_id IN NUMBER**

**) RETURN NUMBER IS**

**v\_salary NUMBER;**

**BEGIN**

**SELECT Salary \* 12 INTO v\_salary**

**FROM Employees**

**WHERE EmployeeID = p\_employee\_id;**

**RETURN v\_salary;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**RETURN NULL;**

**WHEN OTHERS THEN**

**RETURN NULL;**

**END CalculateAnnualSalary;**

**END EmployeeManagement;**

**/**

**Scenario 3**

**CREATE OR REPLACE PACKAGE AccountOperations AS**

**PROCEDURE OpenNewAccount(**

**p\_account\_id IN NUMBER,**

**p\_customer\_id IN NUMBER,**

**p\_account\_type IN VARCHAR2,**

**p\_balance IN NUMBER**

**);**

**PROCEDURE CloseAccount(**

**p\_account\_id IN NUMBER**

**);**

**FUNCTION GetTotalBalance(**

**p\_customer\_id IN NUMBER**

**) RETURN NUMBER;**

**END AccountOperations;**

**/CREATE OR REPLACE PACKAGE BODY AccountOperations AS**

**PROCEDURE OpenNewAccount(**

**p\_account\_id IN NUMBER,**

**p\_customer\_id IN NUMBER,**

**p\_account\_type IN VARCHAR2,**

**p\_balance IN NUMBER**

**) IS**

**BEGIN**

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

**VALUES (p\_account\_id, p\_customer\_id, p\_account\_type, p\_balance, SYSDATE);**

**COMMIT;**

**EXCEPTION**

**WHEN DUP\_VAL\_ON\_INDEX THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error: Account with ID ' || p\_account\_id || ' already exists');**

**WHEN OTHERS THEN**

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

**END OpenNewAccount;**

**PROCEDURE CloseAccount(**

**p\_account\_id IN NUMBER**

**) IS**

**BEGIN**

**DELETE FROM Accounts**

**WHERE AccountID = p\_account\_id;**

**COMMIT;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE('Error: Account with ID ' || p\_account\_id || ' not found');**

**WHEN OTHERS THEN**

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

**END CloseAccount;**

**FUNCTION GetTotalBalance(**

**p\_customer\_id IN NUMBER**

**) RETURN NUMBER IS**

**v\_total\_balance NUMBER;**

**BEGIN**

**SELECT SUM(Balance) INTO v\_total\_balance**

**FROM Accounts**

**WHERE CustomerID = p\_customer\_id;**

**RETURN v\_total\_balance;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**RETURN NULL;**

**WHEN OTHERS THEN**

**RETURN NULL;**

**END GetTotalBalance;**

**END AccountOperations;**

**/**