## EXERCISE 1 :-

### Q1.

DECLARE CURSOR customer\_cursor IS SELECT customer\_id, age, loan\_interest\_rate FROM customers WHERE age > 60;

v\_customer\_id customers.customer\_id%TYPE;

v\_age customers.age%TYPE;

v\_loan\_interest\_rate customers.loan\_interest\_rate%TYPE;

BEGIN

OPEN customer\_cursor;

LOOP

FETCH customer\_cursor INTO v\_customer\_id, v\_age, v\_loan\_interest\_rate;

EXIT WHEN customer\_cursor%NOTFOUND;

v\_loan\_interest\_rate := v\_loan\_interest\_rate - 1;

UPDATE customers SET loan\_interest\_rate = v\_loan\_interest\_rate WHERE customer\_id = v\_customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id || ' | New Loan Interest Rate: ' || v\_loan\_interest\_rate);

END LOOP;

CLOSE customer\_cursor;

COMMIT;

END;

Q2.

DECLARE CURSOR customer\_cursor IS SELECT customer\_id, balance FROM customers WHERE balance > 10000;

v\_customer\_id customers.customer\_id%TYPE;

v\_balance customers.balance%TYPE;

BEGIN

OPEN customer\_cursor;

LOOP FETCH customer\_cursor INTO v\_customer\_id, v\_balance;

EXIT WHEN customer\_cursor%NOTFOUND;

UPDATE customers SET IsVIP = 'TRUE' WHERE customer\_id = v\_customer\_id; DBMS\_OUTPUT.PUT\_LINE ('Customer ID: ' || v\_customer\_id || ' | Balance: ' || v\_balance || ' | IsVIP: TRUE');

END LOOP;

CLOSE customer\_cursor;

COMMIT;

END;

### Q3.

DECLARE

CURSOR loan\_cursor IS SELECT l.loan\_id, l.due\_date, c.customer\_id, c.customer\_name FROM loans l JOIN customers c ON l.customer\_id = c.customer\_id WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30;

v\_loan\_id loans.loan\_id%TYPE;

v\_due\_date loans.due\_date%TYPE;

v\_customer\_id customers.customer\_id%TYPE;

v\_customer\_name customers.customer\_name%TYPE;

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_loan\_id, v\_due\_date, v\_customer\_id, v\_customer\_name;

EXIT WHEN loan\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || v\_customer\_name || ' (Customer ID: ' || v\_customer\_id || '), your loan with ID ' || v\_loan\_id || ' is due on ' || TO\_CHAR(v\_due\_date, 'DD-MON-YYYY') || '. Please make sure to settle it before the due date.');

END LOOP;

CLOSE loan\_cursor;

END;

## EXERCISE 2 :-

### Q1.

CREATE OR REPLACE PROCEDURE SafeTransferFunds( p\_from\_account\_id IN NUMBER, p\_to\_account\_id IN NUMBER, p\_amount IN NUMBER)

IS

v\_from\_account\_balance NUMBER;

v\_to\_account\_balance NUMBER;

insufficient\_funds EXCEPTION;

BEGIN

SELECT balance INTO v\_from\_account\_balance FROM accounts WHERE account\_id = p\_from\_account\_id FOR UPDATE;

SELECT balance INTO v\_to\_account\_balance

FROM accounts

WHERE account\_id = p\_to\_account\_id

FOR UPDATE;

IF v\_from\_account\_balance < p\_amount THEN RAISE insufficient\_funds;

END IF;

UPDATE accounts SET balance = balance - p\_amount WHERE account\_id = p\_from\_account\_id;

UPDATE accounts SET balance = balance + p\_amount WHERE account\_id = p\_to\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully from Account ' || p\_from\_account\_id || ' to Account ' || p\_to\_account\_id || ' Amount: ' || p\_amount);

END SafeTransferFunds;

### Q2.

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_employee\_id IN NUMBER, p\_percentage IN NUMBER) IS

v\_current\_salary employees.salary%TYPE;

employee\_not\_found EXCEPTION;

BEGIN

BEGIN

SELECT salary INTO v\_current\_salary FROM employees WHERE employee\_id = p\_employee\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN RAISE employee\_not\_found;

END;

UPDATE employees SET salary = salary + (salary \* p\_percentage / 100) WHERE employee\_id = p\_employee\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary for Employee ID ' || p\_employee\_id || ' updated by ' || p\_percentage || '%. New Salary: ' || (v\_current\_salary + (v\_current\_salary \* p\_percentage / 100)));

END UpdateSalary;

### Q3.

CREATE OR REPLACE PROCEDURE AddNewCustomer( p\_customer\_id IN NUMBER, p\_customer\_name IN VARCHAR2, p\_customer\_address IN VARCHAR2, p\_customer\_phone IN VARCHAR2) IS

v\_count NUMBER;

customer\_exists EXCEPTION;

BEGIN

SELECT COUNT(\*) INTO v\_count FROM customers WHERE customer\_id = p\_customer\_id;

IF v\_count > 0

THEN RAISE customer\_exists;

END IF;

INSERT INTO customers (customer\_id, customer\_name, customer\_address, customer\_phone)

VALUES (p\_customer\_id, p\_customer\_name, p\_customer\_address, p\_customer\_phone);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || p\_customer\_id || ' added successfully.');

END AddNewCustomer;

## EXERCISE 3 :-

### Q1.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS v\_interest\_rate CONSTANT NUMBER := 0.01;

CURSOR savings\_cursor IS SELECT account\_id, balance FROM accounts WHERE account\_type = 'SAVINGS';

v\_account\_id accounts.account\_id%TYPE;

v\_balance accounts.balance%TYPE;

BEGIN

OPEN savings\_cursor;

LOOP

FETCH savings\_cursor INTO v\_account\_id, v\_balance;

EXIT WHEN savings\_cursor%NOTFOUND;

v\_balance := v\_balance + (v\_balance \* v\_interest\_rate);

UPDATE accounts SET balance = v\_balance WHERE account\_id = v\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || v\_account\_id || ' | New Balance: ' || v\_balance);

END LOOP;

CLOSE savings\_cursor;

COMMIT;

END ProcessMonthlyInterest;

### Q2.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_department\_id IN NUMBER, p\_bonus\_percentage IN NUMBER) IS

BEGIN

UPDATE employees SET salary = salary + (salary \* p\_bonus\_percentage / 100) WHERE department\_id = p\_department\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salaries in Department ID ' || p\_department\_id || ' have been updated with a bonus of ' || p\_bonus\_percentage || '%.');

END UpdateEmployeeBonus;

### Q3.

CREATE OR REPLACE PROCEDURE TransferFunds(p\_source\_account\_id IN NUMBER, p\_target\_account\_id IN NUMBER, p\_amount IN NUMBER) IS

v\_source\_balance NUMBER;

insufficient\_funds EXCEPTION;

BEGIN

SELECT balance INTO v\_source\_balance FROM accounts WHERE account\_id = p\_source\_account\_id FOR UPDATE;

IF v\_source\_balance < p\_amount THEN RAISE insufficient\_funds;

END IF;

UPDATE accounts SET balance = balance - p\_amount WHERE account\_id = p\_source\_account\_id;

UPDATE accounts SET balance = balance + p\_amount WHERE account\_id = p\_target\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ' || p\_amount || ' transferred from Account ID ' || p\_source\_account\_id || ' to Account ID ' || p\_target\_account\_id);

EXCEPTION

WHEN insufficient\_funds THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in Account ID ' || p\_source\_account\_id);

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END TransferFunds;

## EXERCISE 4 :-

### Q1.

CREATE OR REPLACE FUNCTION CalculateAge(p\_date\_of\_birth DATE) RETURN NUMBER IS v\_age NUMBER;

BEGIN

SELECT FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_date\_of\_birth) / 12) INTO v\_age FROM dual;

RETURN v\_age;

EXCEPTION

WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

RETURN NULL;

END CalculateAge;

### Q2.

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(p\_loan\_amount IN NUMBER, p\_annual\_interest\_rate IN NUMBER, p\_loan\_duration\_years IN NUMBER) RETURN NUMBER IS

v\_monthly\_interest\_rate NUMBER;

v\_total\_payments NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

v\_monthly\_interest\_rate := p\_annual\_interest\_rate / 100 / 12;

v\_total\_payments := p\_loan\_duration\_years \* 12;

IF v\_monthly\_interest\_rate > 0

THEN v\_monthly\_installment := (p\_loan\_amount \* v\_monthly\_interest\_rate \* POWER(1 + v\_monthly\_interest\_rate, v\_total\_payments)) / (POWER(1 + v\_monthly\_interest\_rate, v\_total\_payments) - 1);

ELSE

v\_monthly\_installment := p\_loan\_amount / v\_total\_payments;

END IF;

RETURN v\_monthly\_installment;

EXCEPTION

WHEN OTHERS THEN

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

RETURN NULL;

END CalculateMonthlyInstallment;

### Q3.

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 account\_id = p\_account\_id;

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND

THEN DBMS\_OUTPUT.PUT\_LINE('Error: Account ID ' || p\_account\_id || ' does not exist.');

RETURN FALSE;

WHEN OTHERS

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

RETURN FALSE;

END HasSufficientBalance;

## EXERCISE5 :-

### Q1.

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified AFTER UPDATE ON Customers

FOR EACH ROW

BEGIN:NEW.LastModified := SYSDATE;

END;

### Q2.

CREATE OR REPLACE TRIGGER LogTransaction AFTER INSERT ON Transactions FOR EACH ROW

BEGIN

INSERT INTO AuditLog ( AuditID, TransactionID, TransactionDate, TransactionAmount, AuditTimestamp,ActionPerformed )

VALUES ( AuditSeq.NEXTVAL, audit IDs :NEW.TransactionID, :NEW.TransactionDate, :NEW.TransactionAmount, SYSDATE, 'INSERT' );

END;

### Q3.

CREATE OR REPLACE TRIGGER CheckTransactionRules BEFORE INSERT ON Transactions FOR EACH ROW

DECLARE

v\_current\_balance NUMBER;

BEGIN

IF :NEW.TransactionType = 'WITHDRAWAL' THEN

SELECT balance INTO v\_current\_balance FROM accounts WHERE account\_id = :NEW.AccountID;

IF :NEW.TransactionAmount > v\_current\_balance THEN RAISE\_APPLICATION\_ERROR(-20001, 'Error: Withdrawal amount exceeds current balance.');

END IF;

ELSIF :NEW.TransactionType = 'DEPOSIT' THEN

IF :NEW.TransactionAmount <= 0 THEN RAISE\_APPLICATION\_ERROR(-20002, 'Error: Deposit amount must be positive.');

END IF;

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'Error: Unknown transaction type.');

END IF;

END;

## EXERCISE 6 :-

### Q1.

DECLARE

CURSOR transaction\_cursor IS SELECT t.account\_id, a.customer\_name, t.transaction\_date, t.transaction\_amount, t.transaction\_type

FROM Transactions t JOIN Accounts a ON t.account\_id = a.account\_id

WHERE

t.transaction\_date >= TRUNC(SYSDATE, 'MONTH') AND t.transaction\_date < TRUNC(SYSDATE, 'MONTH') + INTERVAL '1' MONTH;

-- Variables to hold transaction data

v\_account\_id Transactions.account\_id%TYPE;

v\_customer\_name Accounts.customer\_name%TYPE;

v\_transaction\_date Transactions.transaction\_date%TYPE;

v\_transaction\_amount Transactions.transaction\_amount%TYPE;

v\_transaction\_type Transactions.transaction\_type%TYPE;

BEGIN

OPEN transaction\_cursor;

LOOP

FETCH transaction\_cursor INTO v\_account\_id, v\_customer\_name, v\_transaction\_date, v\_transaction\_amount, v\_transaction\_type;

EXIT WHEN transaction\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Statement for Account ID: ' || v\_account\_id);

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

DBMS\_OUTPUT.PUT\_LINE('Transaction Date: ' || v\_transaction\_date);

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

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

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

END LOOP;

CLOSE transaction\_cursor;

EXCEPTION

WHEN OTHERS THEN

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

END;

### Q2.

DECLARE

v\_annual\_fee NUMBER := 50;

CURSOR account\_cursor IS SELECT account\_id, balance FROM Accounts; v\_account\_id Accounts.account\_id%TYPE;

v\_balance Accounts.balance%TYPE;

BEGIN

OPEN account\_cursor;

LOOP

FETCH account\_cursor INTO v\_account\_id, v\_balance;

EXIT WHEN account\_cursor%NOTFOUND;

IF v\_balance >= v\_annual\_fee THEN

UPDATE Accounts SET balance = v\_balance - v\_annual\_fee WHERE account\_id = v\_account\_id;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Account ID ' || v\_account\_id || ' has insufficient funds for annual fee deduction.');

END IF;

END LOOP;

CLOSE account\_cursor;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

### Q3.

DECLARE

v\_new\_interest\_rate NUMBER := 0.05;

CURSOR loan\_cursor IS SELECT loan\_id, interest\_rate FROM Loans;

v\_loan\_id Loans.loan\_id%TYPE;

v\_interest\_rate Loans.interest\_rate%TYPE;

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_loan\_id, v\_interest\_rate;

EXIT WHEN loan\_cursor%NOTFOUND;

UPDATE Loans SET interest\_rate = v\_new\_interest\_rate WHERE loan\_id = v\_loan\_id; DBMS\_OUTPUT.PUT\_LINE('Updated Loan ID ' || v\_loan\_id || ' to new interest rate: ' || v\_new\_interest\_rate);

END LOOP; CLOSE loan\_cursor;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

## EXERCISE 7 :-

### Q1.

#### **PACKAGE SPECIFICATION :-**

CREATE OR REPLACE PACKAGE CustomerManagement AS PROCEDURE AddNewCustomer(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_balance IN NUMBER);

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_balance IN NUMBER);

FUNCTION GetCustomerBalance(p\_customer\_id IN NUMBER) RETURN NUMBER;

END CustomerManagement;

#### **PACKAGE BODY :-**

CREATE OR REPLACE PACKAGE BODY PROCEDURE AddNewCustomer(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_balance IN NUMBER) IS

BEGIN

INSERT INTO Customers (customer\_id, name, balance)

VALUES (p\_customer\_id, p\_name, p\_balance);

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\_balance IN NUMBER) IS

BEGIN

UPDATE Customers

SET name = p\_name, balance = p\_balance

WHERE customer\_id = p\_customer\_id;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance(p\_customer\_id IN NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance FROM Customers WHERE customer\_id = p\_customer\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

### Q2.

#### **PACKAGE SPECIFICATION :-**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireNewEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER);

PROCEDURE UpdateEmployeeDetails( p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER);

FUNCTION CalculateAnnualSalary( p\_employee\_id IN NUMBER ) RETURN NUMBER;

END EmployeeManagement;

#### **PACKAGE BODY :-**

CREATE OR REPLACE PACKAGE BODY E PROCEDURE HireNewEmployee( p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER ) IS

BEGIN

INSERT INTO Employees (employee\_id, name, position, salary) VALUES (p\_employee\_id, p\_name, p\_position, p\_salary);

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 HireNewEmployee;

PROCEDURE UpdateEmployeeDetails( p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER ) IS

BEGIN

UPDATE Employees SET name = p\_name, position = p\_position, salary = p\_salary WHERE employee\_id = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(p\_employee\_id IN NUMBER ) RETURN NUMBER IS v\_salary NUMBER;

v\_annual\_salary NUMBER;

BEGIN

SELECT salary INTO v\_salary FROM Employees WHERE employee\_id = p\_employee\_id;

v\_annual\_salary := v\_salary \* 12;

RETURN v\_annual\_salary;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END CalculateAnnualSalary;

END EmployeeManagement;

### Q3.

#### **PACKAGE SPECIFICATION :-**

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;

#### **PACKAGE BODY :-**

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 (account\_id, customer\_id, account\_type, balance)VALUES (p\_account\_id, p\_customer\_id, p\_account\_type, p\_balance);

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 account\_id = p\_account\_id

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

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 customer\_id = p\_customer\_id GROUP BY customer\_id;

RETURN v\_total\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

WHEN OTHERS THEN

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

RETURN NULL;

END GetTotalBalance;

END AccountOperations;