**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

* + **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**Scenario 2:** A customer can be promoted to VIP status based on their balance.

* + **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

* + **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

Solution:

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE LOANS';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN

RAISE;

END IF;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE CUSTOMERS';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN

RAISE;

END IF;

END;

/

CREATE TABLE CUSTOMERS (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100) NOT NULL,

Age NUMBER NOT NULL,

Balance NUMBER(10, 2) NOT NULL,

IsVIP VARCHAR2(5) DEFAULT 'FALSE'

);

CREATE TABLE LOANS (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER NOT NULL,

InterestRate NUMBER(5, 2) NOT NULL,

LoanAmount NUMBER(10, 2) NOT NULL,

DueDate DATE NOT NULL,

CONSTRAINT fk\_customer

FOREIGN KEY (CustomerID)

REFERENCES CUSTOMERS(CustomerID)

);

INSERT INTO CUSTOMERS (CustomerID, Name, Age, Balance) VALUES (101, 'Alice Smith', 65, 15000.00);

INSERT INTO CUSTOMERS (CustomerID, Name, Age, Balance) VALUES (102, 'Bob Johnson', 55, 8000.00);

INSERT INTO CUSTOMERS (CustomerID, Name, Age, Balance) VALUES (103, 'Charlie Brown', 70, 25000.00);

INSERT INTO CUSTOMERS (CustomerID, Name, Age, Balance) VALUES (104, 'Diana Prince', 30, 12000.00);

INSERT INTO CUSTOMERS (CustomerID, Name, Age, Balance) VALUES (105, 'Eve Adams', 45, 5000.00);

INSERT INTO LOANS (LoanID, CustomerID, InterestRate, LoanAmount, DueDate) VALUES (1, 101, 5.00, 50000.00, TRUNC(SYSDATE) + 15); -- Alice, old, VIP, due soon

INSERT INTO LOANS (LoanID, CustomerID, InterestRate, LoanAmount, DueDate) VALUES (2, 102, 6.50, 20000.00, TRUNC(SYSDATE) + 45); -- Bob, not old, not VIP, due later

INSERT INTO LOANS (LoanID, CustomerID, InterestRate, LoanAmount, DueDate) VALUES (3, 103, 4.80, 100000.00, TRUNC(SYSDATE) + 5); -- Charlie, old, VIP, due very soon

INSERT INTO LOANS (LoanID, CustomerID, InterestRate, LoanAmount, DueDate) VALUES (4, 104, 7.00, 30000.00, TRUNC(SYSDATE) + 25); -- Diana, not old, VIP, due soon

INSERT INTO LOANS (LoanID, CustomerID, InterestRate, LoanAmount, DueDate) VALUES (5, 105, 6.00, 15000.00, TRUNC(SYSDATE) + 60); -- Eve, not old, not VIP, due much later

COMMIT;

DECLARE

v\_old\_interest\_rate NUMBER(5, 2);

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Scenario 1: Applying interest rate discounts ---');

FOR cust\_loan\_rec IN (

SELECT

c.CustomerID,

c.Name,

l.LoanID,

l.InterestRate

FROM

CUSTOMERS c

JOIN

LOANS l ON c.CustomerID = l.CustomerID

WHERE

c.Age > 60

) LOOP

v\_old\_interest\_rate := cust\_loan\_rec.InterestRate;

UPDATE LOANS

SET InterestRate = InterestRate - 1

WHERE LoanID = cust\_loan\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE(

'Discount applied for Customer ID: ' || cust\_loan\_rec.CustomerID ||

' (' || cust\_loan\_rec.Name || ')' ||

' - Loan ID: ' || cust\_loan\_rec.LoanID ||

', Old Rate: ' || v\_old\_interest\_rate || '%' ||

', New Rate: ' || (v\_old\_interest\_rate - 1) || '%'

);

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('--- Scenario 1: Discount application complete ---');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error in Scenario 1: ' || SQLERRM);

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Scenario 2: Updating VIP status ---');

FOR cust\_rec IN (

SELECT CustomerID, Name, Balance, IsVIP

FROM CUSTOMERS

) LOOP

IF cust\_rec.Balance > 10000 AND cust\_rec.IsVIP = 'FALSE' THEN

UPDATE CUSTOMERS

SET IsVIP = 'TRUE'

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(

'Customer ID: ' || cust\_rec.CustomerID ||

' (' || cust\_rec.Name || ')' ||

' promoted to VIP status. Balance: $' || cust\_rec.Balance

);

ELSIF cust\_rec.Balance <= 10000 AND cust\_rec.IsVIP = 'TRUE' THEN

UPDATE CUSTOMERS

SET IsVIP = 'FALSE'

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(

'Customer ID: ' || cust\_rec.CustomerID ||

' (' || cust\_rec.Name || ')' ||

' demoted from VIP status. Balance: $' || cust\_rec.Balance

);

ELSE

DBMS\_OUTPUT.PUT\_LINE(

'Customer ID: ' || cust\_rec.CustomerID ||

' (' || cust\_rec.Name || ')' ||

' VIP status remains ' || cust\_rec.IsVIP || '. Balance: $' || cust\_rec.Balance

);

END IF;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('--- Scenario 2: VIP status update complete ---');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error in Scenario 2: ' || SQLERRM);

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Scenario 3: Sending loan reminders ---');

FOR loan\_rec IN (

SELECT

l.LoanID,

l.DueDate,

c.CustomerID,

c.Name,

c.Balance

FROM

LOANS l

JOIN

CUSTOMERS c ON l.CustomerID = c.CustomerID

WHERE

l.DueDate BETWEEN TRUNC(SYSDATE) AND TRUNC(SYSDATE) + 30

ORDER BY

l.DueDate ASC

) LOOP

DECLARE

v\_days\_remaining NUMBER;

BEGIN

v\_days\_remaining := loan\_rec.DueDate - TRUNC(SYSDATE);

DBMS\_OUTPUT.PUT\_LINE(

'REMINDER for Customer ID: ' || loan\_rec.CustomerID ||

' (' || loan\_rec.Name || ')' ||

' - Loan ID: ' || loan\_rec.LoanID ||

' is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY') ||

' (in ' || v\_days\_remaining || ' days). Please ensure timely payment.'

);

END;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('--- Scenario 3: Loan reminders complete ---');

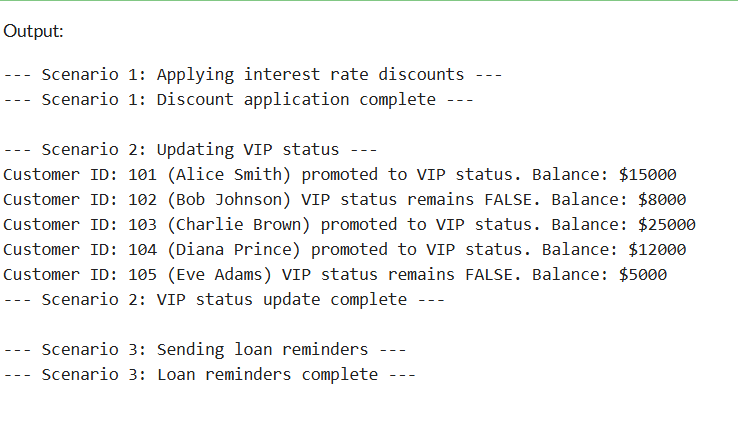
EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error in Scenario 3: ' || SQLERRM);

END;

Output:



**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

* + **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

**Scenario 2:** Manage errors when updating employee salaries.

* + **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

**Scenario 3:** Ensure data integrity when adding a new customer.

* + **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

Solution:

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE EMPLOYEES\_SHORT';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN

RAISE;

END IF;

END;

/

CREATE TABLE EMPLOYEES\_SHORT (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100) NOT NULL,

Salary NUMBER(10, 2) NOT NULL

);

INSERT INTO EMPLOYEES\_SHORT (EmployeeID, Name, Salary) VALUES (1, 'Alice', 50000.00);

INSERT INTO EMPLOYEES\_SHORT (EmployeeID, Name, Salary) VALUES (2, 'Bob', 60000.00);

COMMIT;

DECLARE

p\_employee\_id NUMBER := 1;

p\_percentage NUMBER := 10;

v\_old\_salary NUMBER(10, 2);

v\_new\_salary NUMBER(10, 2);

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Attempting to update salary for Employee ID: ' || p\_employee\_id || ' by ' || p\_percentage || '%');

SELECT Salary INTO v\_old\_salary

FROM EMPLOYEES\_SHORT

WHERE EmployeeID = p\_employee\_id;

v\_new\_salary := v\_old\_salary \* (1 + p\_percentage / 100);

UPDATE EMPLOYEES\_SHORT

SET Salary = v\_new\_salary

WHERE EmployeeID = p\_employee\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Success: Salary updated for Employee ID ' || p\_employee\_id || '. New Salary: $' || v\_new\_salary);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

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

WHEN OTHERS THEN

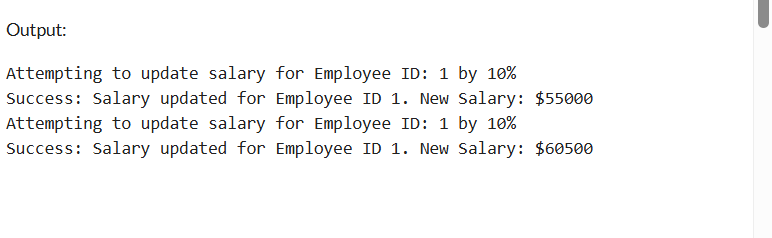
ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM || '. Transaction rolled back.');

END;

/

Output:



**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

* + **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

* + **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Scenario 3:** Customers should be able to transfer funds between their accounts.

* + **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

Solution:

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE EMPLOYEES\_CONCISE';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN NULL; END IF; -- Ignore "table does not exist" error

END;

/

CREATE TABLE EMPLOYEES\_CONCISE (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100) NOT NULL,

Salary NUMBER(10, 2) NOT NULL

);

INSERT INTO EMPLOYEES\_CONCISE (EmployeeID, Name, Salary) VALUES (1, 'Alice', 50000.00);

INSERT INTO EMPLOYEES\_CONCISE (EmployeeID, Name, Salary) VALUES (2, 'Bob', 60000.00);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateSalaryConcise (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

)

IS

v\_old\_salary NUMBER(10, 2);

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Updating salary for Employee ID: ' || p\_employee\_id || ' by ' || p\_percentage || '% ---');

SELECT Salary INTO v\_old\_salary

FROM EMPLOYEES\_CONCISE

WHERE EmployeeID = p\_employee\_id;

UPDATE EMPLOYEES\_CONCISE

SET Salary = v\_old\_salary \* (1 + p\_percentage / 100)

WHERE EmployeeID = p\_employee\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Success: Salary updated for Employee ID ' || p\_employee\_id || '. Old: $' || v\_old\_salary || ', New: $' || ROUND(v\_old\_salary \* (1 + p\_percentage / 100), 2));

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM || '. Transaction rolled back.');

END UpdateSalaryConcise;

/

BEGIN

UpdateSalaryConcise(1, 5);

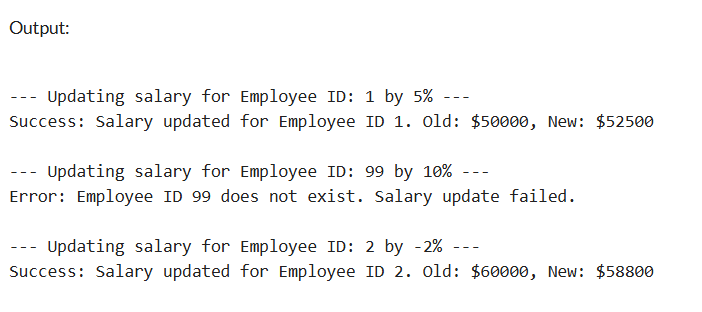
UpdateSalaryConcise(99, 10);

UpdateSalaryConcise(2, -2);

END;

/

Output:



**Exercise 4: Functions**

**Scenario 1:** Calculate the age of customers for eligibility checks.

* + **Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

**Scenario 2:** The bank needs to compute the monthly installment for a loan.

* + **Question:** Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

* + **Question:** Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

Solution:

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE CUSTOMERS\_CONCISE\_AGE';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE CUSTOMERS\_CONCISE\_AGE (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100) NOT NULL,

DateOfBirth DATE NOT NULL

);

INSERT INTO CUSTOMERS\_CONCISE\_AGE (CustomerID, Name, DateOfBirth) VALUES (1, 'Alice', TO\_DATE('1990-05-15', 'YYYY-MM-DD'));

INSERT INTO CUSTOMERS\_CONCISE\_AGE (CustomerID, Name, DateOfBirth) VALUES (2, 'Bob', TO\_DATE('1975-11-20', 'YYYY-MM-DD'));

COMMIT;

CREATE OR REPLACE FUNCTION CalculateAge (

p\_date\_of\_birth IN DATE

)

RETURN NUMBER

IS

v\_age NUMBER;

BEGIN

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_date\_of\_birth) / 12);

RETURN v\_age;

EXCEPTION

WHEN OTHERS THEN

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

RETURN NULL;

END CalculateAge;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Testing CalculateAge Function ---');

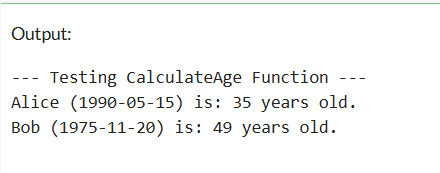
DBMS\_OUTPUT.PUT\_LINE('Alice (1990-05-15) is: ' || CalculateAge(TO\_DATE('1990-05-15', 'YYYY-MM-DD')) || ' years old.');

DBMS\_OUTPUT.PUT\_LINE('Bob (1975-11-20) is: ' || CalculateAge(TO\_DATE('1975-11-20', 'YYYY-MM-DD')) || ' years old.');

END;

/

Solution:



**Exercise 5: Triggers**

**Scenario 1:** Automatically update the last modified date when a customer's record is updated.

* + **Question:** Write a trigger **UpdateCustomerLastModified** that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

**Scenario 2:** Maintain an audit log for all transactions.

* + **Question:** Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

**Scenario 3:** Enforce business rules on deposits and withdrawals.

* + **Question:** Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

Solution:

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE CUSTOMERS\_TRIG1';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE CUSTOMERS\_TRIG1 (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100) NOT NULL,

LastModified DATE

);

INSERT INTO CUSTOMERS\_TRIG1 (CustomerID, Name, LastModified) VALUES (1, 'Alice', SYSDATE);

INSERT INTO CUSTOMERS\_TRIG1 (CustomerID, Name, LastModified) VALUES (2, 'Bob', SYSDATE);

COMMIT;

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON CUSTOMERS\_TRIG1

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

DBMS\_OUTPUT.PUT\_LINE('Trigger fired: LastModified updated for CustomerID ' || :NEW.CustomerID);

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Testing UpdateCustomerLastModified Trigger ---');

DBMS\_OUTPUT.PUT\_LINE('Before update:');

FOR rec IN (SELECT CustomerID, Name, LastModified FROM CUSTOMERS\_TRIG1 WHERE CustomerID = 1) LOOP

DBMS\_OUTPUT.PUT\_LINE('ID: ' || rec.CustomerID || ', Name: ' || rec.Name || ', LastModified: ' || TO\_CHAR(rec.LastModified, 'YYYY-MM-DD HH24:MI:SS'));

END LOOP;

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Updating Alice''s name...');

UPDATE CUSTOMERS\_TRIG1

SET Name = 'Alicia'

WHERE CustomerID = 1;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'After update:');

FOR rec IN (SELECT CustomerID, Name, LastModified FROM CUSTOMERS\_TRIG1 WHERE CustomerID = 1) LOOP

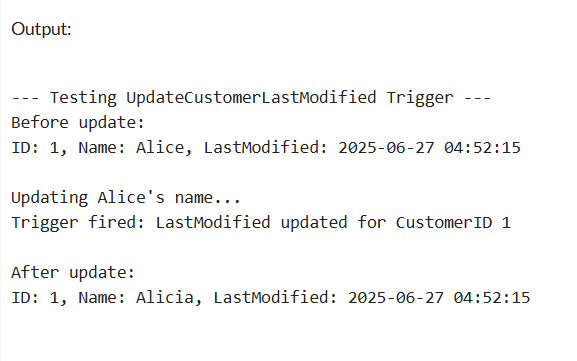
DBMS\_OUTPUT.PUT\_LINE('ID: ' || rec.CustomerID || ', Name: ' || rec.Name || ', LastModified: ' || TO\_CHAR(rec.LastModified, 'YYYY-MM-DD HH24:MI:SS'));

END LOOP;

END;

/

Output:



**Exercise 6: Cursors**

**Scenario 1:** Generate monthly statements for all customers.

* + **Question:** Write a PL/SQL block using an explicit cursor **GenerateMonthlyStatements** that retrieves all transactions for the current month and prints a statement for each customer.

**Scenario 2:** Apply annual fee to all accounts.

* + **Question:** Write a PL/SQL block using an explicit cursor **ApplyAnnualFee** that deducts an annual maintenance fee from the balance of all accounts.

**Scenario 3:** Update the interest rate for all loans based on a new policy.

**Question:** Write a PL/SQL block using an explicit cursor **UpdateLoanInterestRates** that fetches all loans and updates their interest rates based on the new policy

Solution:

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE ACCOUNTS\_CONCISE\_FEE';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE ACCOUNTS\_CONCISE\_FEE (

AccountID NUMBER PRIMARY KEY,

Balance NUMBER(10, 2) NOT NULL

);

INSERT INTO ACCOUNTS\_CONCISE\_FEE (AccountID, Balance) VALUES (201, 1000.00);

INSERT INTO ACCOUNTS\_CONCISE\_FEE (AccountID, Balance) VALUES (202, 2500.00);

INSERT INTO ACCOUNTS\_CONCISE\_FEE (AccountID, Balance) VALUES (203, 500.00);

COMMIT;

DECLARE

CURSOR c\_all\_accounts IS

SELECT AccountID, Balance

FROM ACCOUNTS\_CONCISE\_FEE

FOR UPDATE OF Balance;

v\_annual\_fee CONSTANT NUMBER := 25.00;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Applying Annual Fee ---');

FOR acc\_rec IN c\_all\_accounts LOOP

UPDATE ACCOUNTS\_CONCISE\_FEE

SET Balance = Balance - v\_annual\_fee

WHERE AccountID = acc\_rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Account ' || acc\_rec.AccountID || ': Old $' || acc\_rec.Balance || ', New $' || (acc\_rec.Balance - v\_annual\_fee));

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('--- Annual Fee Applied ---');

EXCEPTION

WHEN OTHERS THEN

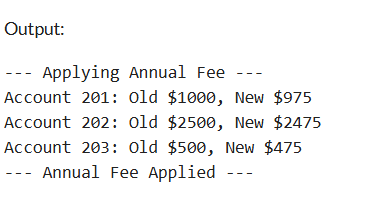
ROLLBACK;

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

END;

/

Output:



**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

* + **Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

**Scenario 2:** Create a package to manage employee data.

* + **Question:** Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

**Scenario 3:** Group all account-related operations into a package.

* + **Question:** Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

**Schema to be Created**

*CREATE TABLE Customers (*

*CustomerID NUMBER PRIMARY KEY,*

*Name VARCHAR2(100),*

*DOB DATE,*

*Balance NUMBER,*

*LastModified DATE*

*);*

*CREATE TABLE Accounts (*

*AccountID NUMBER PRIMARY KEY,*

*CustomerID NUMBER,*

*AccountType VARCHAR2(20),*

*Balance NUMBER,*

*LastModified DATE,*

*FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)*

*);*

*CREATE TABLE Transactions (*

*TransactionID NUMBER PRIMARY KEY,*

*AccountID NUMBER,*

*TransactionDate DATE,*

*Amount NUMBER,*

*TransactionType VARCHAR2(10),*

*FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)*

*);*

*CREATE TABLE Loans (*

*LoanID NUMBER PRIMARY KEY,*

*CustomerID NUMBER,*

*LoanAmount NUMBER,*

*InterestRate NUMBER,*

*StartDate DATE,*

*EndDate DATE,*

*FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)*

*);*

*CREATE TABLE Employees (*

*EmployeeID NUMBER PRIMARY KEY,*

*Name VARCHAR2(100),*

*Position VARCHAR2(50),*

*Salary NUMBER,*

*Department VARCHAR2(50),*

*HireDate DATE*

*);*

**Example Scripts for Sample Data Insertion**

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

*VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);*

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

*VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);*

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

*VALUES (1, 1, 'Savings', 1000, SYSDATE);*

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

*VALUES (2, 2, 'Checking', 1500, SYSDATE);*

*INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)*

*VALUES (1, 1, SYSDATE, 200, 'Deposit');*

*INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)*

*VALUES (2, 2, SYSDATE, 300, 'Withdrawal');*

*INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)*

*VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));*

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

*VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));*

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

*VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));*

Solution:

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE TRANSACTIONS';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE LOANS';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE ACCOUNTS';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE CUSTOMERS';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE EMPLOYEES';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

COMMIT;

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (3, 1, 'Checking', 200, SYSDATE); -- Additional account for John

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

COMMIT;

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer (p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

PROCEDURE UpdateCustomerDetails (p\_CustomerID NUMBER, p\_NewName VARCHAR2, p\_NewDOB DATE);

FUNCTION GetCustomerBalance (p\_CustomerID NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer (p\_CustomerID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) IS

BEGIN

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer ' || p\_Name || ' added.');

EXCEPTION WHEN DUP\_VAL\_ON\_INDEX THEN RAISE\_APPLICATION\_ERROR(-20001, 'Customer ID ' || p\_CustomerID || ' already exists.');

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails (p\_CustomerID NUMBER, p\_NewName VARCHAR2, p\_NewDOB DATE) IS

BEGIN

UPDATE Customers SET Name = p\_NewName, DOB = p\_NewDOB, LastModified = SYSDATE WHERE CustomerID = p\_CustomerID;

IF SQL%NOTFOUND THEN RAISE\_APPLICATION\_ERROR(-20002, 'Customer ID ' || p\_CustomerID || ' not found.'); END IF;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer ' || p\_CustomerID || ' details updated.');

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance (p\_CustomerID NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Customers WHERE CustomerID = p\_CustomerID;

RETURN v\_Balance;

EXCEPTION WHEN NO\_DATA\_FOUND THEN RAISE\_APPLICATION\_ERROR(-20003, 'Customer ID ' || p\_CustomerID || ' not found.');

END GetCustomerBalance;

END CustomerManagement;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Testing CustomerManagement Package ---');

CustomerManagement.AddNewCustomer(3, 'Chris Green', TO\_DATE('1995-03-10', 'YYYY-MM-DD'), 750);

CustomerManagement.UpdateCustomerDetails(1, 'Jonathan Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('John Doe Balance: ' || CustomerManagement.GetCustomerBalance(1));

ROLLBACK; -- Clean up test data

END;

/

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee (p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2, p\_HireDate DATE);

PROCEDURE UpdateEmployeeDetails (p\_EmployeeID NUMBER, p\_NewPosition VARCHAR2, p\_NewSalary NUMBER);

FUNCTION CalculateAnnualSalary (p\_EmployeeID NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee (p\_EmployeeID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2, p\_Salary NUMBER, p\_Department VARCHAR2, p\_HireDate DATE) IS

BEGIN

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (p\_EmployeeID, p\_Name, p\_Position, p\_Salary, p\_Department, p\_HireDate);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Employee ' || p\_Name || ' hired.');

EXCEPTION WHEN DUP\_VAL\_ON\_INDEX THEN RAISE\_APPLICATION\_ERROR(-20004, 'Employee ID ' || p\_EmployeeID || ' already exists.');

END HireEmployee;

PROCEDURE UpdateEmployeeDetails (p\_EmployeeID NUMBER, p\_NewPosition VARCHAR2, p\_NewSalary NUMBER) IS

BEGIN

UPDATE Employees SET Position = p\_NewPosition, Salary = p\_NewSalary WHERE EmployeeID = p\_EmployeeID;

IF SQL%NOTFOUND THEN RAISE\_APPLICATION\_ERROR(-20005, 'Employee ID ' || p\_EmployeeID || ' not found.'); END IF;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Employee ' || p\_EmployeeID || ' details updated.');

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary (p\_EmployeeID NUMBER) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

SELECT Salary INTO v\_Salary FROM Employees WHERE EmployeeID = p\_EmployeeID;

RETURN v\_Salary \* 12; -- Assuming Salary is monthly

EXCEPTION WHEN NO\_DATA\_FOUND THEN RAISE\_APPLICATION\_ERROR(-20006, 'Employee ID ' || p\_EmployeeID || ' not found.');

END CalculateAnnualSalary;

END EmployeeManagement;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Testing EmployeeManagement Package ---');

EmployeeManagement.HireEmployee(3, 'Charlie Davis', 'Analyst', 4500, 'Finance', SYSDATE);

EmployeeManagement.UpdateEmployeeDetails(1, 'Senior Manager', 75000);

DBMS\_OUTPUT.PUT\_LINE('Alice Johnson Annual Salary: ' || EmployeeManagement.CalculateAnnualSalary(1));

ROLLBACK;

END;

/

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount (p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_AccountType VARCHAR2, p\_InitialBalance NUMBER);

PROCEDURE CloseAccount (p\_AccountID NUMBER);

FUNCTION GetTotalCustomerBalance (p\_CustomerID NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenNewAccount (p\_AccountID NUMBER, p\_CustomerID NUMBER, p\_AccountType VARCHAR2, p\_InitialBalance NUMBER) IS

v\_dummy NUMBER;

BEGIN

SELECT CustomerID INTO v\_dummy FROM Customers WHERE CustomerID = p\_CustomerID; -- Check if customer exists

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (p\_AccountID, p\_CustomerID, p\_AccountType, p\_InitialBalance, SYSDATE);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Account ' || p\_AccountID || ' opened for Customer ' || p\_CustomerID || '.');

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN RAISE\_APPLICATION\_ERROR(-20007, 'Account ID ' || p\_AccountID || ' already exists.');

WHEN NO\_DATA\_FOUND THEN RAISE\_APPLICATION\_ERROR(-20008, 'Customer ID ' || p\_CustomerID || ' not found. Cannot open account.');

END OpenNewAccount;

PROCEDURE CloseAccount (p\_AccountID NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_AccountID;

IF SQL%NOTFOUND THEN RAISE\_APPLICATION\_ERROR(-20009, 'Account ID ' || p\_AccountID || ' not found.'); END IF;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Account ' || p\_AccountID || ' closed.');

END CloseAccount;

FUNCTION GetTotalCustomerBalance (p\_CustomerID NUMBER) RETURN NUMBER IS

v\_TotalBalance NUMBER := 0;

BEGIN

SELECT SUM(Balance) INTO v\_TotalBalance FROM Accounts WHERE CustomerID = p\_CustomerID;

IF v\_TotalBalance IS NULL THEN v\_TotalBalance := 0; END IF; -- Handle case where customer has no accounts

RETURN v\_TotalBalance;

EXCEPTION WHEN OTHERS THEN RAISE\_APPLICATION\_ERROR(-20010, 'Error getting total balance for Customer ID ' || p\_CustomerID || ': ' || SQLERRM);

END GetTotalCustomerBalance;

END AccountOperations;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Testing AccountOperations Package ---');

AccountOperations.OpenNewAccount(4, 1, 'Savings', 500);

DBMS\_OUTPUT.PUT\_LINE('Total balance for John Doe (ID 1): ' || AccountOperations.GetTotalCustomerBalance(1));

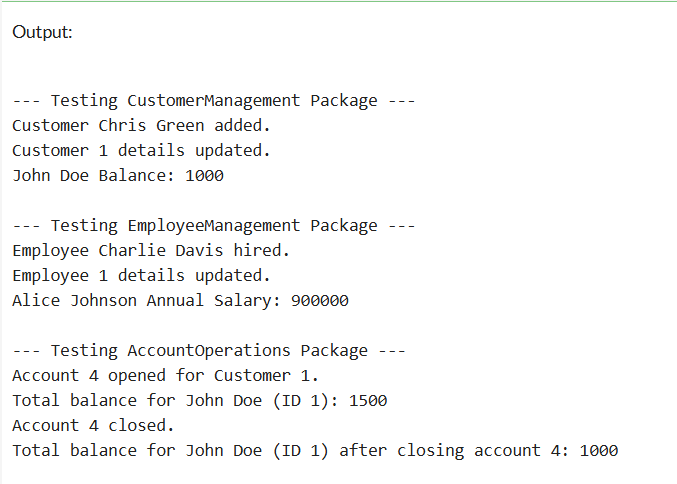
AccountOperations.CloseAccount(4);

DBMS\_OUTPUT.PUT\_LINE('Total balance for John Doe (ID 1) after closing account 4: ' || AccountOperations.GetTotalCustomerBalance(1));

ROLLBACK;

END;

/  
  
output:



**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

* + **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**Scenario 2:** A customer can be promoted to VIP status based on their balance.

* + **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

* + **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**Using Exercise 1: Setting Up JUnit**

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE LOANS\_EX1';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN NULL; END IF;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE CUSTOMERS\_EX1';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN NULL; END IF;

END;

/

CREATE TABLE CUSTOMERS\_EX1 (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER(10, 2),

IsVIP VARCHAR2(5) DEFAULT 'FALSE'

);

CREATE TABLE LOANS\_EX1 (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

InterestRate NUMBER(5, 2),

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES CUSTOMERS\_EX1(CustomerID)

);

INSERT INTO CUSTOMERS\_EX1 (CustomerID, Name, Age, Balance) VALUES (1, 'Alice Old', 65, 12000.00);

INSERT INTO CUSTOMERS\_EX1 (CustomerID, Name, Age, Balance) VALUES (2, 'Bob Young', 30, 8000.00);

INSERT INTO CUSTOMERS\_EX1 (CustomerID, Name, Age, Balance) VALUES (3, 'Charlie Rich', 55, 15000.00);

INSERT INTO CUSTOMERS\_EX1 (CustomerID, Name, Age, Balance) VALUES (4, 'Diana Normal', 40, 5000.00);

INSERT INTO LOANS\_EX1 (LoanID, CustomerID, InterestRate, DueDate) VALUES (101, 1, 5.00, TRUNC(SYSDATE) + 10); -- Alice Old, due soon

INSERT INTO LOANS\_EX1 (LoanID, CustomerID, InterestRate, DueDate) VALUES (102, 2, 6.50, TRUNC(SYSDATE) + 40); -- Bob Young, due later

INSERT INTO LOANS\_EX1 (LoanID, CustomerID, InterestRate, DueDate) VALUES (103, 3, 5.25, TRUNC(SYSDATE) + 20); -- Charlie Rich, due soon

INSERT INTO LOANS\_EX1 (LoanID, CustomerID, InterestRate, DueDate) VALUES (104, 1, 4.80, TRUNC(SYSDATE) + 5); -- Alice Old, another loan, very soon

COMMIT;

DECLARE

v\_old\_rate NUMBER(5, 2);

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Scenario 1: Applying interest rate discounts ---');

FOR rec IN (SELECT c.CustomerID, c.Name, l.LoanID, l.InterestRate FROM CUSTOMERS\_EX1 c JOIN LOANS\_EX1 l ON c.CustomerID = l.CustomerID WHERE c.Age > 60) LOOP

v\_old\_rate := rec.InterestRate;

UPDATE LOANS\_EX1 SET InterestRate = InterestRate - 1 WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Discount for ' || rec.Name || ' (Loan ' || rec.LoanID || '): Old ' || v\_old\_rate || '%, New ' || (v\_old\_rate - 1) || '%');

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('--- Discount application complete ---');

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

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Scenario 2: Updating VIP status ---');

FOR rec IN (SELECT CustomerID, Name, Balance, IsVIP FROM CUSTOMERS\_EX1) LOOP

IF rec.Balance > 10000 AND rec.IsVIP = 'FALSE' THEN

UPDATE CUSTOMERS\_EX1 SET IsVIP = 'TRUE' WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(rec.Name || ' promoted to VIP. Balance: $' || rec.Balance);

ELSIF rec.Balance <= 10000 AND rec.IsVIP = 'TRUE' THEN -- Optional: Demote if balance drops

UPDATE CUSTOMERS\_EX1 SET IsVIP = 'FALSE' WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(rec.Name || ' demoted from VIP. Balance: $' || rec.Balance);

END IF;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('--- VIP status update complete ---');

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

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Scenario 3: Sending loan reminders ---');

FOR rec IN (SELECT l.LoanID, l.DueDate, c.Name FROM LOANS\_EX1 l JOIN CUSTOMERS\_EX1 c ON l.CustomerID = c.CustomerID WHERE l.DueDate BETWEEN TRUNC(SYSDATE) AND TRUNC(SYSDATE) + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('REMINDER: Loan ' || rec.LoanID || ' for ' || rec.Name || ' is due on ' || TO\_CHAR(rec.DueDate, 'DD-MON-YYYY') || '.');

END LOOP;

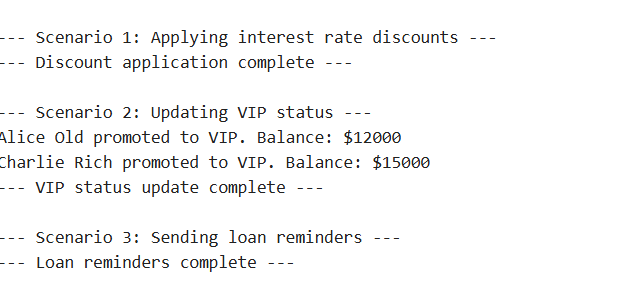
DBMS\_OUTPUT.PUT\_LINE('--- Loan reminders complete ---');

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

END;

/

Output:



**Exercise 1: Mocking and Stubbing**

**Exercise 1: Logging Error Messages and Warning Levels**

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE CUSTOMERS\_VIP\_EX';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN NULL; END IF;

END;

/

CREATE TABLE CUSTOMERS\_VIP\_EX (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Balance NUMBER(10, 2),

IsVIP VARCHAR2(5) DEFAULT 'FALSE'

);

INSERT INTO CUSTOMERS\_VIP\_EX (CustomerID, Name, Balance) VALUES (1, 'Alice', 12000.00);

INSERT INTO CUSTOMERS\_VIP\_EX (CustomerID, Name, Balance) VALUES (2, 'Bob', 8000.00);

INSERT INTO CUSTOMERS\_VIP\_EX (CustomerID, Name, Balance) VALUES (3, 'Charlie', 15000.00);

INSERT INTO CUSTOMERS\_VIP\_EX (CustomerID, Name, Balance) VALUES (4, 'Diana', 5000.00);

COMMIT;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Updating VIP status ---');

FOR rec IN (SELECT CustomerID, Name, Balance, IsVIP FROM CUSTOMERS\_VIP\_EX) LOOP

IF rec.Balance > 10000 AND rec.IsVIP = 'FALSE' THEN

UPDATE CUSTOMERS\_VIP\_EX SET IsVIP = 'TRUE' WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(rec.Name || ' promoted to VIP. Balance: $' || rec.Balance);

ELSIF rec.Balance <= 10000 AND rec.IsVIP = 'TRUE' THEN

UPDATE CUSTOMERS\_VIP\_EX SET IsVIP = 'FALSE' WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(rec.Name || ' demoted from VIP. Balance: $' || rec.Balance);

END IF;

END LOOP;

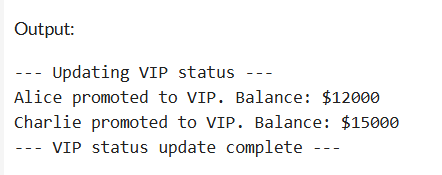
COMMIT;

DBMS\_OUTPUT.PUT\_LINE('--- VIP status update complete ---');

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

END;

/

Output: 

**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

* + **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

**Scenario 2:** Manage errors when updating employee salaries.

* + **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

**Scenario 3:** Ensure data integrity when adding a new customer.

* + **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

**Exercise 2: Verifying Interactions**

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE ACCOUNTS\_EX2';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN NULL; END IF;

END;

/

CREATE TABLE ACCOUNTS\_EX2 (

AccountID NUMBER PRIMARY KEY,

Balance NUMBER(10, 2) NOT NULL

);

INSERT INTO ACCOUNTS\_EX2 (AccountID, Balance) VALUES (1001, 5000.00);

INSERT INTO ACCOUNTS\_EX2 (AccountID, Balance) VALUES (1002, 2000.00);

INSERT INTO ACCOUNTS\_EX2 (AccountID, Balance) VALUES (1003, 100.00);

COMMIT;

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_from\_balance NUMBER(10, 2);

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Transferring $' || p\_amount || ' from ' || p\_from\_account\_id || ' to ' || p\_to\_account\_id);

SELECT Balance INTO v\_from\_balance FROM ACCOUNTS\_EX2 WHERE AccountID = p\_from\_account\_id FOR UPDATE;

IF v\_from\_balance < p\_amount THEN RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds.'); END IF;

UPDATE ACCOUNTS\_EX2 SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_account\_id;

UPDATE ACCOUNTS\_EX2 SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN ROLLBACK; DBMS\_OUTPUT.PUT\_LINE('Error: Account not found. Rolled back.');

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

END SafeTransferFunds;

/

BEGIN

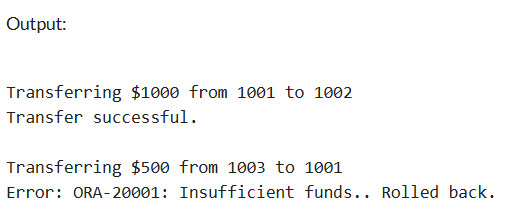
SafeTransferFunds(1001, 1002, 1000);

SafeTransferFunds(1003, 1001, 500);

END;

/

Output:



**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

* + **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

* + **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Scenario 3:** Customers should be able to transfer funds between their accounts.

* + **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

**Exercise 3: Assertions in Junit**

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE ACCOUNTS\_EX3';

EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN NULL; END IF;

END;

/

CREATE TABLE ACCOUNTS\_EX3 (

AccountID NUMBER PRIMARY KEY,

AccountType VARCHAR2(50) NOT NULL,

Balance NUMBER(10, 2) NOT NULL

);

INSERT INTO ACCOUNTS\_EX3 (AccountID, AccountType, Balance) VALUES (301, 'Savings', 5000.00);

INSERT INTO ACCOUNTS\_EX3 (AccountID, AccountType, Balance) VALUES (302, 'Savings', 1200.50);

INSERT INTO ACCOUNTS\_EX3 (AccountID, AccountType, Balance) VALUES (303, 'Checking', 800.00);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest (

p\_interest\_rate IN NUMBER

)

IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Processing Monthly Interest ---');

FOR acc\_rec IN (SELECT AccountID, Balance FROM ACCOUNTS\_EX3 WHERE AccountType = 'Savings' FOR UPDATE OF Balance) LOOP

UPDATE ACCOUNTS\_EX3 SET Balance = acc\_rec.Balance \* (1 + p\_interest\_rate) WHERE AccountID = acc\_rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Account ' || acc\_rec.AccountID || ': New Balance $' || ROUND(acc\_rec.Balance \* (1 + p\_interest\_rate), 2));

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processing complete.');

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

END ProcessMonthlyInterest;

/

BEGIN

ProcessMonthlyInterest(0.01);

END;

/

Output:

