**WEEK 2 : PLSQL PROGRAMMING EXERCISES**

**SUPERSET ID : 6386707**

**Exercise 1 : Control Structures**

**Scenario 1 :**

BEGIN

FOR c IN (SELECT CustomerID FROM Customers

WHERE MONTHS\_BETWEEN(SYSDATE, DOB)/12 > 60) LOOP

UPDATE Loans

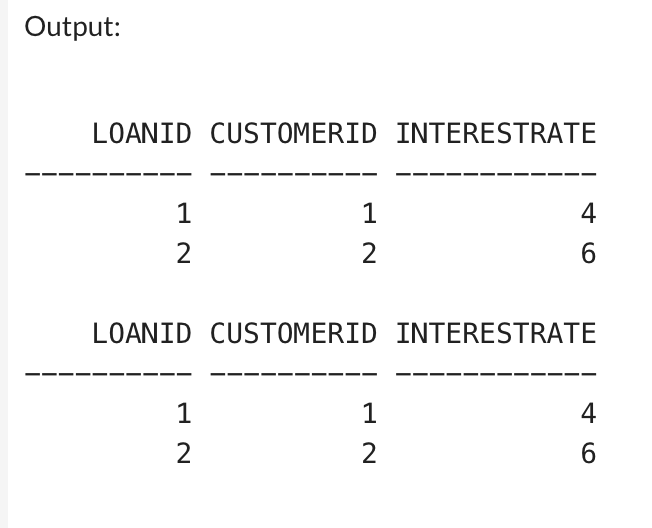
SET InterestRate = InterestRate - 1

WHERE CustomerID = c.CustomerID;

END LOOP;

END;

**OUTPUT**



**Scenario 2 :**

BEGIN

FOR c IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

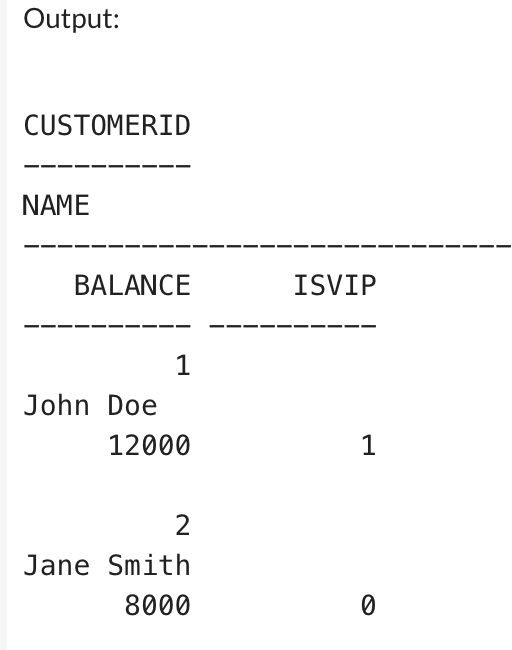
SET IsVIP = 1

WHERE CustomerID = c.CustomerID;

END LOOP;

END;

**OUTPUT**

****

**Scenario 3 :**

BEGIN

FOR l IN (

SELECT l.LoanID, c.Name, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || l.LoanID ||

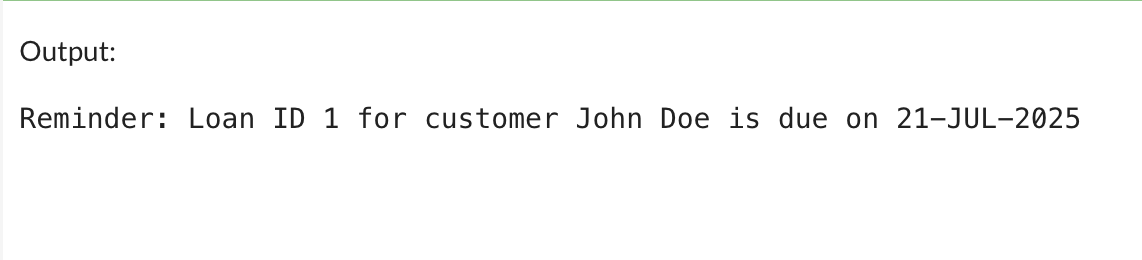
' for customer ' || l.Name ||

' is due on ' || TO\_CHAR(l.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

**OUTPUT**

****

**Exercise 2 : Error Handling**

**Scenario 1 :**

BEGIN

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID

FOR UPDATE;

IF v\_FromBalance < p\_Amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_FromAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_ToAccountID;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

INSERT INTO ErrorLog (ErrorMessage)

VALUES ('Error in SafeTransferFunds: ' || SQLERRM);

DBMS\_OUTPUT.PUT\_LINE('Transfer failed. Error logged.');

END;

/

BEGIN

SafeTransferFunds(101, 102, 2000);

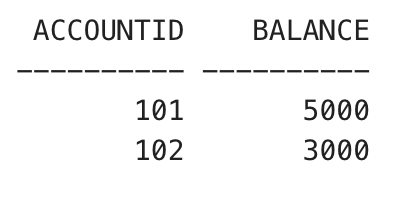
END;

/

SELECT AccountID, Balance FROM Accounts;

SELECT \* FROM ErrorLog ORDER BY LogID DESC;

**OUTPUT**



**Scenario 2 :**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_EmployeeID IN NUMBER,

p\_Percentage IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* (p\_Percentage / 100))

WHERE EmployeeID = p\_EmployeeID;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Employee not found');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

INSERT INTO ErrorLog (ErrorMessage)

VALUES ('Error in UpdateSalary: ' || SQLERRM);

ROLLBACK;

END;

/

BEGIN

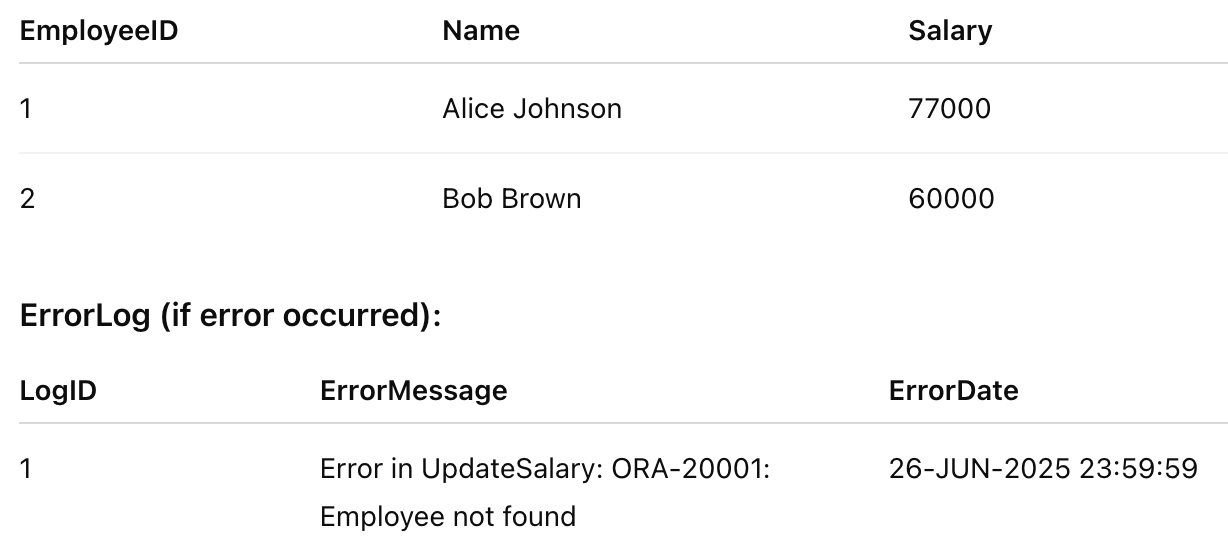
UpdateSalary(1, 10); END;

/

SELECT EmployeeID, Name, Salary FROM Employees;

SELECT \* FROM ErrorLog ORDER BY LogID DESC;

**OUTPUT**



**Scenario 3 :**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

) AS

BEGIN

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

VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

INSERT INTO ErrorLog (ErrorMessage)

VALUES ('Duplicate customer ID: ' || p\_CustomerID);

ROLLBACK;

WHEN OTHERS THEN

INSERT INTO ErrorLog (ErrorMessage)

VALUES ('General error in AddNewCustomer: ' || SQLERRM);

ROLLBACK;

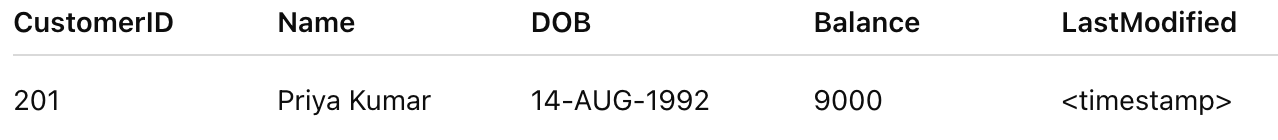
END;

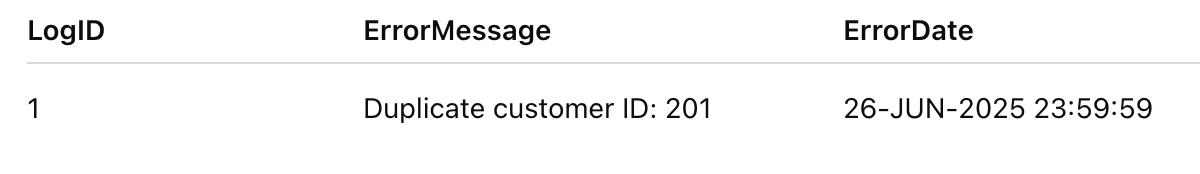
/

SELECT \* FROM Customers;

SELECT \* FROM ErrorLog ORDER BY LogID DESC;

**OUTPUT**

****

****

**Exercise 3 : Stored Procedures**

**Scenario 1 :**

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

FOR UPDATE

) LOOP

UPDATE Accounts

SET Balance = acc.Balance + (acc.Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

/

BEGIN

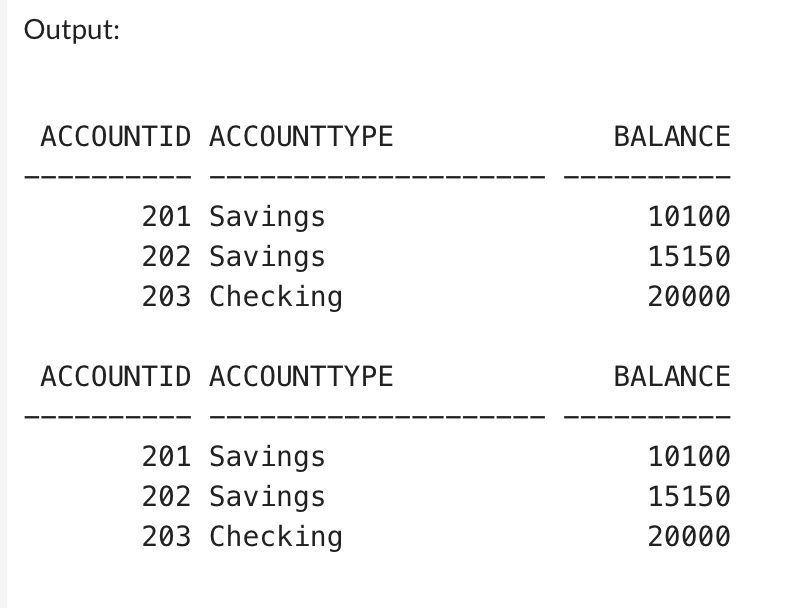
ProcessMonthlyInterest;

END;

/

SELECT AccountID, AccountType, Balance FROM Accounts;

**OUTPUT**

****

**Scenario 2 :**

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* (p\_BonusPct / 100))

WHERE Department = p\_Department;

COMMIT;

END;

/

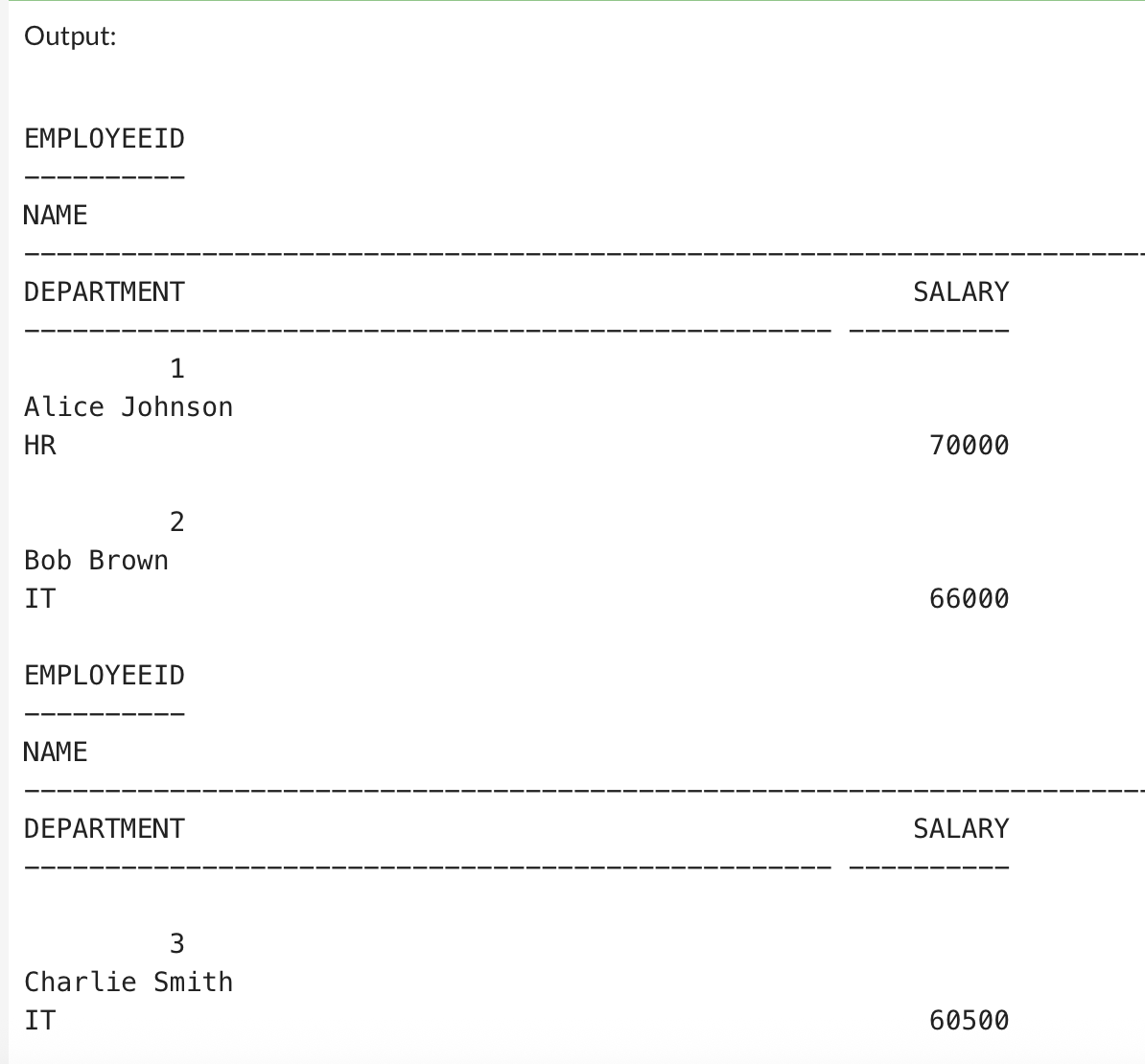
BEGIN

UpdateEmployeeBonus('IT', 10);

END;/

SELECT EmployeeID, Name, Department, Salary FROM Employees;

**OUTPUT**

****

**Scenario 3 :**

BEGIN

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID

FOR UPDATE;

IF v\_FromBalance < p\_Amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_FromAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_ToAccountID;

COMMIT;

END;

/

BEGIN

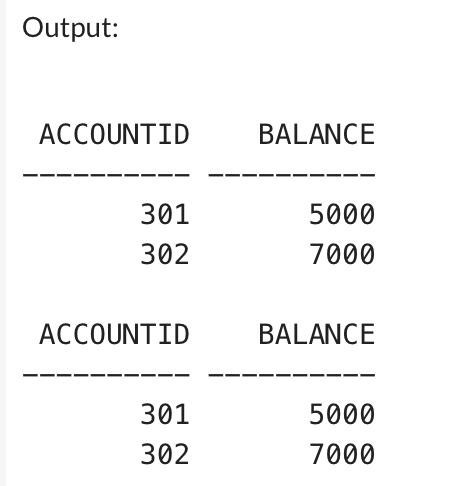
TransferFunds(301, 302, 3000);

END;

/

SELECT AccountID, Balance FROM Accounts;

**OUTPUT**



**Exercise 4 : Functions**

**Scenario 1 :**

CREATE OR REPLACE FUNCTION CalculateAge (

p\_DOB DATE

) RETURN NUMBER IS

v\_Age NUMBER;

BEGIN

v\_Age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

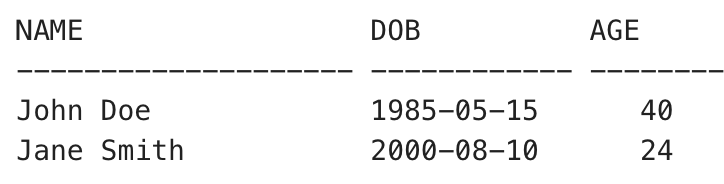
RETURN v\_Age;

END;

/

SELECT Name, DOB, CalculateAge(DOB) AS Age

**OUTPUT**

****

**Scenario 2 :**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_LoanAmount IN NUMBER,

p\_InterestRate IN NUMBER,

p\_LoanYears IN NUMBER

) RETURN NUMBER IS

v\_RatePerMonth NUMBER;

v\_NumMonths NUMBER;

v\_Installment NUMBER;

BEGIN

v\_RatePerMonth := p\_InterestRate / 12 / 100;

v\_NumMonths := p\_LoanYears \* 12;

v\_Installment :=

(p\_LoanAmount \* v\_RatePerMonth \* POWER(1 + v\_RatePerMonth, v\_NumMonths)) /

(POWER(1 + v\_RatePerMonth, v\_NumMonths) - 1);

RETURN ROUND(v\_Installment, 2);

END;

/CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_LoanAmount IN NUMBER,

p\_InterestRate IN NUMBER,

p\_LoanYears IN NUMBER

) RETURN NUMBER IS

v\_RatePerMonth NUMBER;

v\_NumMonths NUMBER;

v\_Installment NUMBER;

BEGIN

v\_RatePerMonth := p\_InterestRate / 12 / 100;

v\_NumMonths := p\_LoanYears \* 12;

v\_Installment :=

(p\_LoanAmount \* v\_RatePerMonth \* POWER(1 + v\_RatePerMonth, v\_NumMonths)) /

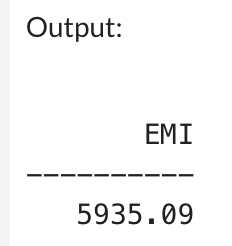
(POWER(1 + v\_RatePerMonth, v\_NumMonths) - 1);

RETURN ROUND(v\_Installment, 2);

END;

/

**OUTPUT**



**Scenario 3 :**

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_AccountID IN NUMBER,

p\_Amount IN NUMBER

) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance

FROM Accounts

WHERE AccountID = p\_AccountID;

IF v\_Balance >= p\_Amount THEN

RETURN 1;

ELSE

RETURN 0;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END;

/

-- Case 1: Sufficient balance

SELECT HasSufficientBalance(401, 5000) AS Result FROM DUAL;

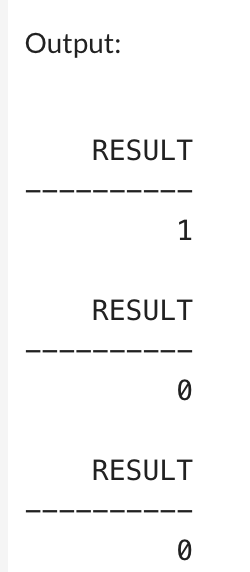
-- Case 2: Insufficient balance

SELECT HasSufficientBalance(402, 3000) AS Result FROM DUAL;

-- Case 3: Invalid account

SELECT HasSufficientBalance(999, 1000) AS Result FROM DUAL;

**OUTPUT**



**Exercise 5 : Triggers**

**Scenario 1 :**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

UPDATE Customers

SET Balance = Balance + 1000

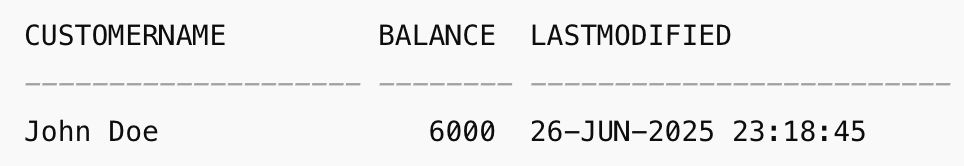
WHERE CustomerID = 101;

SELECT Name, Balance, TO\_CHAR(LastModified, 'DD-MON-YYYY HH24:MI:SS') AS LastModified

FROM Customers

WHERE CustomerID = 101;

**OUTPUT**



**Scenario 2 :**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (

TransactionID,

ActionType,

Details

) VALUES (

:NEW.TransactionID,

'INSERT',

'Transaction of type ' || :NEW.TransactionType ||

' for amount ' || :NEW.Amount ||

' on account ' || :NEW.AccountID

);

END;

/

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

VALUES (1001, 401, SYSDATE, 2000, 'Deposit');

SELECT

RPAD('Txn ID: ' || TransactionID, 20) AS "Txn Info",

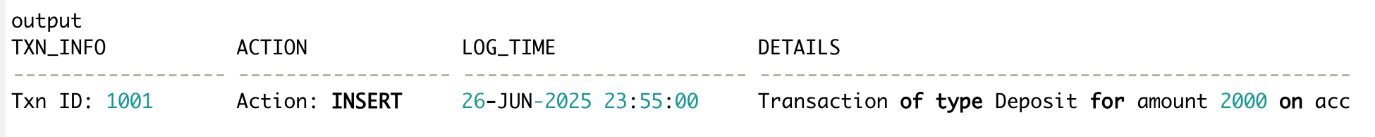
RPAD('Action: ' || ActionType, 15) AS "Action",

TO\_CHAR(LogDate, 'DD-MON-YYYY HH24:MI:SS') AS "Logged At",

Details

FROM AuditLog;

**OUTPUT**

****

**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' THEN

IF :NEW.Amount > v\_Balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal exceeds available balance');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be greater than 0');

END IF;

END IF;

END;

/

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

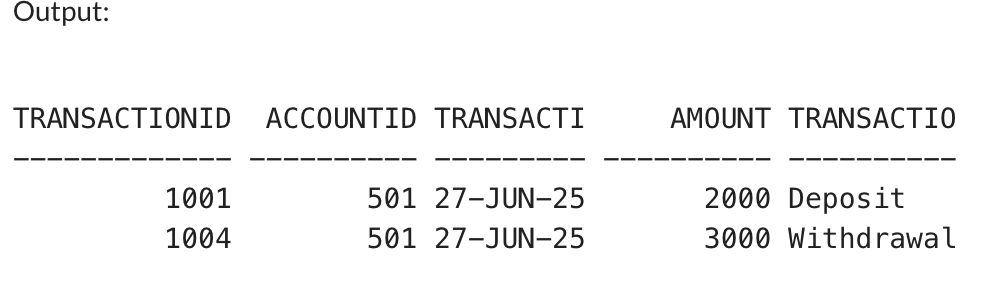
VALUES (1001, 501, SYSDATE, 2000, 'Deposit');

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

VALUES (1004, 501, SYSDATE, 3000, 'Withdrawal');

SELECT \* FROM Transactions ORDER BY TransactionID;

**OUTPUT**

****

**Exercise 6 : Cursors**

**Scenario 1 :**

SET SERVEROUTPUT ON;

DECLARE

CURSOR txn\_cursor IS

SELECT c.Name, t.AccountID, 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 EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE);

v\_Name Customers.Name%TYPE;

v\_AccountID Transactions.AccountID%TYPE;

v\_Date Transactions.TransactionDate%TYPE;

v\_Amount Transactions.Amount%TYPE;

v\_Type Transactions.TransactionType%TYPE;

BEGIN

OPEN txn\_cursor;

LOOP

FETCH txn\_cursor INTO v\_Name, v\_AccountID, v\_Date, v\_Amount, v\_Type;

EXIT WHEN txn\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(

'Customer: ' || RPAD(v\_Name, 15) ||

' | Account: ' || v\_AccountID ||

' | Date: ' || TO\_CHAR(v\_Date, 'DD-MON-YYYY') ||

' | Amount: ₹' || v\_Amount ||

' | Type: ' || v\_Type

);

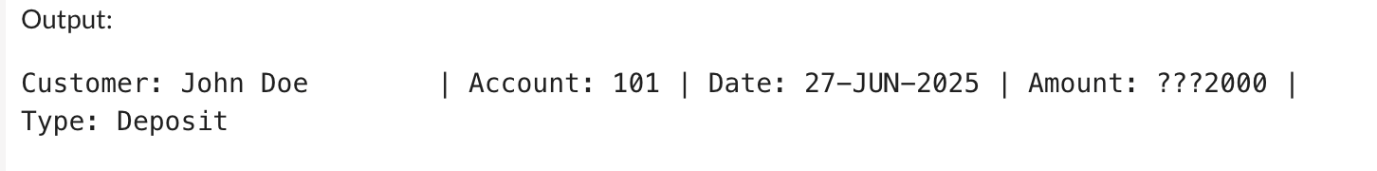
END LOOP;

CLOSE txn\_cursor;

END;

/

**OUTPUT**



**Scenario 2 :**

SET SERVEROUTPUT ON;

DECLARE

CURSOR acc\_cursor IS

SELECT AccountID, Balance

FROM Accounts;

v\_AccountID Accounts.AccountID%TYPE;

v\_Balance Accounts.Balance%TYPE;

v\_Fee CONSTANT NUMBER := 500;

BEGIN

OPEN acc\_cursor;

LOOP

FETCH acc\_cursor INTO v\_AccountID, v\_Balance;

EXIT WHEN acc\_cursor%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - v\_Fee,

LastModified = SYSDATE

WHERE AccountID = v\_AccountID;

DBMS\_OUTPUT.PUT\_LINE('Account ' || v\_AccountID ||

' charged ₹' || v\_Fee ||

'. New balance will be updated.');

END LOOP;

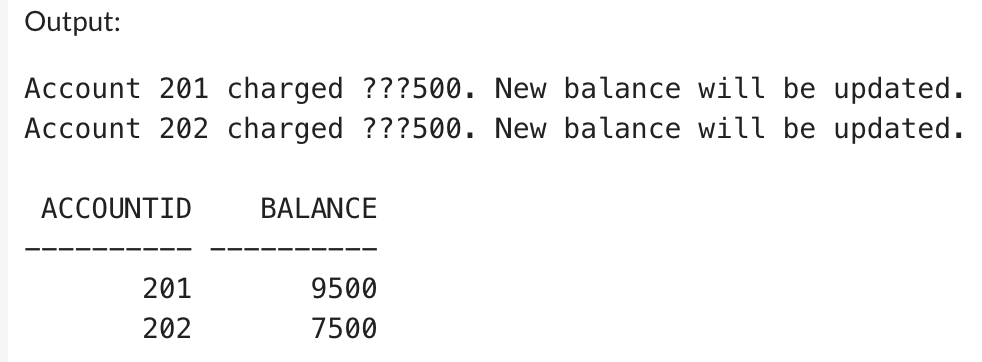
CLOSE acc\_cursor;

COMMIT;

END;

/

**OUTPUT**



**Scenario 3 :**

SET SERVEROUTPUT ON;

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, LoanAmount, InterestRate

FROM Loans;

v\_LoanID Loans.LoanID%TYPE;

v\_Amount Loans.LoanAmount%TYPE;

v\_Rate Loans.InterestRate%TYPE;

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_LoanID, v\_Amount, v\_Rate;

EXIT WHEN loan\_cursor%NOTFOUND;

IF v\_Amount > 10000 THEN

v\_Rate := v\_Rate + 1; -- Increase by 1%

ELSE

v\_Rate := v\_Rate + 0.5; -- Increase by 0.5%

END IF;

UPDATE Loans

SET InterestRate = v\_Rate

WHERE LoanID = v\_LoanID;

DBMS\_OUTPUT.PUT\_LINE('Loan ' || v\_LoanID ||

' updated interest rate to ' || v\_Rate || '%');

END LOOP;

CLOSE loan\_cursor;

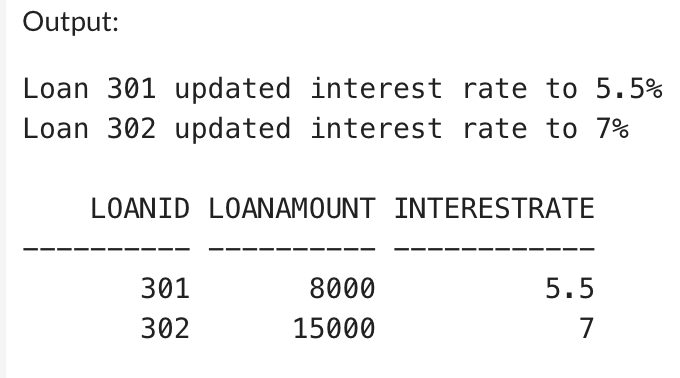
COMMIT;

END;

/

SELECT LoanID, LoanAmount, InterestRate FROM Loans;

**OUTPUT**

****

**Exercise 7 : Packages**

**Scenario 1 :**

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(

p\_CustomerID 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\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID already exists.');

ROLLBACK;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('General error in AddCustomer: ' || SQLERRM);

ROLLBACK;

END AddCustomer;

PROCEDURE UpdateCustomer(

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Balance IN NUMBER

) IS

BEGIN

UPDATE Customers

SET Name = p\_Name,

Balance = p\_Balance,

LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('No such customer found.');

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer updated successfully.');

END IF;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END UpdateCustomer;

FUNCTION GetCustomerBalance(

p\_CustomerID IN 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

RETURN NULL;

WHEN OTHERS THEN

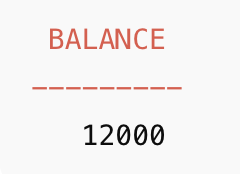
RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

/

**OUTPUT**

****

**Scenario 2 :**

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2

) IS

BEGIN

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

VALUES (p\_EmployeeID, p\_Name, p\_Position, p\_Salary, p\_Department, SYSDATE);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

PROCEDURE UpdateEmployee(

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Salary IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Name = p\_Name, Salary = p\_Salary

WHERE EmployeeID = p\_EmployeeID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Employee not found.');

ELSE

COMMIT;

END IF;

END;

FUNCTION GetAnnualSalary(p\_EmployeeID IN NUMBER) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

SELECT Salary INTO v\_Salary

FROM Employees

WHERE EmployeeID = p\_EmployeeID;

RETURN v\_Salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END EmployeeManagement;

/

BEGIN

EmployeeManagement.HireEmployee(1, 'Anjali Singh', 'Developer', 50000, 'IT');

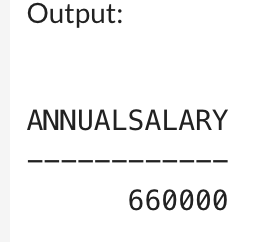
EmployeeManagement.UpdateEmployee(1, 'Anjali S.', 55000);

END;

/

SELECT EmployeeManagement.GetAnnualSalary(1) AS AnnualSalary FROM DUAL;

**OUTPUT**



**Scenario 3 :**

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(

p\_AccountID IN NUMBER,

p\_CustomerID IN NUMBER,

p\_AccountType IN VARCHAR2,

p\_Balance IN NUMBER

) IS

BEGIN

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

VALUES (p\_AccountID, p\_CustomerID, p\_AccountType, p\_Balance, 'Active', SYSDATE);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

PROCEDURE CloseAccount(p\_AccountID IN NUMBER) IS

BEGIN

UPDATE Accounts

SET Status = 'Closed', LastModified = SYSDATE

WHERE AccountID = p\_AccountID;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Account not found.');

ELSE

COMMIT;

END IF;

END;

FUNCTION GetTotalBalance(p\_CustomerID IN NUMBER) RETURN NUMBER IS

v\_Total NUMBER := 0;

BEGIN

SELECT SUM(Balance)

INTO v\_Total

FROM Accounts

WHERE CustomerID = p\_CustomerID AND Status = 'Active';

RETURN NVL(v\_Total, 0);

EXCEPTION

WHEN OTHERS THEN

RETURN NULL;

END;

END AccountOperations;

/

BEGIN

AccountOperations.OpenAccount(201, 1, 'Savings', 10000);

AccountOperations.OpenAccount(202, 1, 'Checking', 5000);

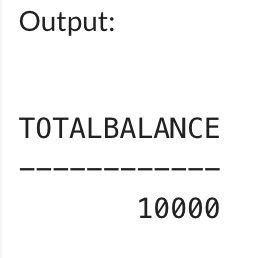
AccountOperations.CloseAccount(202);

END;

/

SELECT AccountOperations.GetTotalBalance(1) AS TotalBalance FROM DUAL;

**OUTPUT**

****