PL-SQL PROGRAMMING- EXERCISES SOLUTIONS

**Exercise 1: Control Structures**

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

Query:-

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Insert sample data

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1960-05-15', 'YYYY-MM-DD'));

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'));

-- Loans: One due in 20 days, one due in 60 days

INSERT INTO Loans VALUES (101, 1, 5000, 6, SYSDATE, SYSDATE + 20);

INSERT INTO Loans VALUES (102, 2, 7000, 7, SYSDATE, SYSDATE + 60);

COMMIT;

-- Enable DBMS\_OUTPUT

BEGIN

DBMS\_OUTPUT.ENABLE;

END;

/

BEGIN

FOR cust IN (

SELECT CustomerID

FROM Customers

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

) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

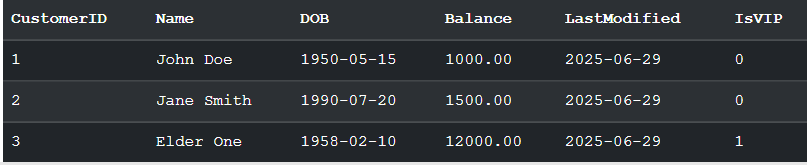
DBMS\_OUTPUT.PUT\_LINE('Discount applied for Customer ID ' || cust.CustomerID);

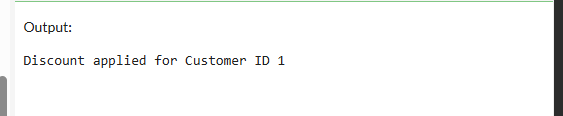
END LOOP;

END;

/

Output:-





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

Query:-

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Insert sample data

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1960-05-15', 'YYYY-MM-DD'));

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'));

-- Loans: One due in 20 days, one due in 60 days

INSERT INTO Loans VALUES (101, 1, 5000, 6, SYSDATE, SYSDATE + 20);

INSERT INTO Loans VALUES (102, 2, 7000, 7, SYSDATE, SYSDATE + 60);

COMMIT;

-- Enable DBMS\_OUTPUT

BEGIN

DBMS\_OUTPUT.ENABLE;

END;

/

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

BEGIN

FOR cust IN (

SELECT CustomerID

FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

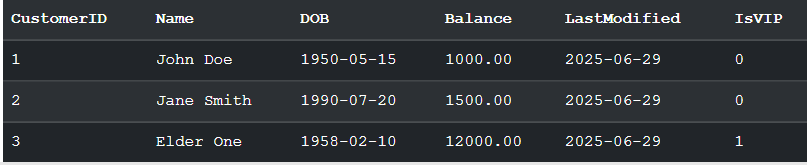
DBMS\_OUTPUT.PUT\_LINE('VIP status granted to Customer ID: ' || cust.CustomerID);

END LOOP;

END;

/

Outputs:-





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

Query:-

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Insert sample data

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1960-05-15', 'YYYY-MM-DD'));

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'));

-- Loans: One due in 20 days, one due in 60 days

INSERT INTO Loans VALUES (101, 1, 5000, 6, SYSDATE, SYSDATE + 20);

INSERT INTO Loans VALUES (102, 2, 7000, 7, SYSDATE, SYSDATE + 60);

COMMIT;

-- Enable DBMS\_OUTPUT

BEGIN

DBMS\_OUTPUT.ENABLE;

END;

/

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

BEGIN

FOR loan\_rec IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate <= SYSDATE + 30

) LOOP

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

' for ' || loan\_rec.Name ||

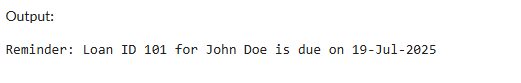
' is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-Mon-YYYY'));

END LOOP;

END;

/

Output:-



**Exercise 2: Error Handling**

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

Query:-

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;

BEGIN

-- Get sender balance

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id;

-- Check for sufficient funds

IF v\_from\_balance < p\_amount THEN

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

END IF;

-- Withdraw from sender

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

-- Deposit to receiver

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

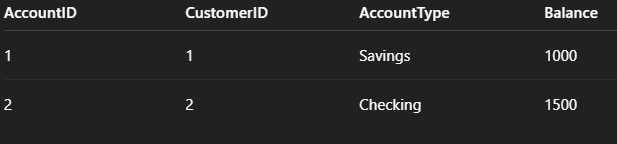
ROLLBACK;

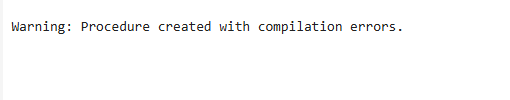
DBMS\_OUTPUT.PUT\_LINE('Error during transfer: ' || SQLERRM);

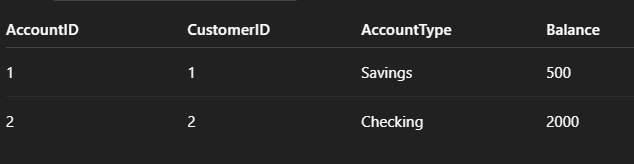
END;

/

Output:-





****

Shows that 500 was transferred from Account 1 → 2.

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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

WHEN OTHERS THEN NULL;

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 Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

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

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1950-07-20', 'YYYY-MM-DD'), 15000, SYSDATE);

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE); -- John Doe

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE); -- Jane Smith

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

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

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_emp\_id IN NUMBER,

p\_percent IN NUMBER

)

IS

v\_old\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_old\_salary

FROM Employees

WHERE EmployeeID = p\_emp\_id;

UPDATE Employees

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

WHERE EmployeeID = p\_emp\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated. New salary: ' || (v\_old\_salary + (v\_old\_salary \* p\_percent / 100)));

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

Output:-





Salary of Employee 1 updated successfully.

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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

WHEN OTHERS THEN NULL; -- Ignore errors if tables don't exist

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 Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

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

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1950-07-20', 'YYYY-MM-DD'), 15000, SYSDATE);

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE); -- John Doe

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE); -- Jane Smith

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

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

COMMIT;

CREATE OR REPLACE 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;

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

ROLLBACK;

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

BEGIN

AddNewCustomer(3, 'David Miller', TO\_DATE('1992-01-10', 'YYYY-MM-DD'), 2000);

END;

/

BEGIN

AddNewCustomer(3, 'Someone Else', TO\_DATE('1990-01-01', 'YYYY-MM-DD'), 1000);

END;

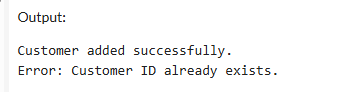
/

Output:-





New customer added.



**Exercise 3: Stored Procedure:-**

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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

INSERT INTO Accounts VALUES (1, 101, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 102, 'Checking', 1500, SYSDATE);

INSERT INTO Accounts VALUES (3, 103, 'Savings', 2000, SYSDATE);

INSERT INTO Accounts VALUES (4, 104, 'Current', 3000, SYSDATE);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

)

LOOP

UPDATE Accounts

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

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE(

'Account ID ' || acc.AccountID || ' updated. New Balance: ' ||

TO\_CHAR(acc.Balance \* 1.01, '9999.99')

);

END LOOP;

COMMIT;

END;

/

BEGIN

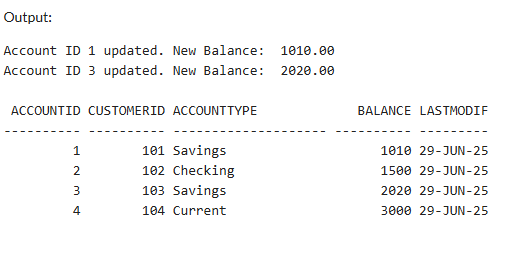
ProcessMonthlyInterest;

END;

/

SELECT \* FROM Accounts;

Output:-



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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

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

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

INSERT INTO Employees VALUES (3, 'Carol Singh', 'Analyst', 50000, 'IT', TO\_DATE('2019-01-10', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (4, 'Dan Paul', 'HR Exec', 45000, 'HR', TO\_DATE('2020-10-05', 'YYYY-MM-DD'));

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

)

IS

BEGIN

FOR emp IN (

SELECT EmployeeID, Name, Salary

FROM Employees

WHERE Department = p\_department

)

LOOP

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)

WHERE EmployeeID = emp.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE(

'Updated ' || emp.Name ||

' (ID: ' || emp.EmployeeID || ') with bonus. New Salary: ' ||

TO\_CHAR(emp.Salary + (emp.Salary \* p\_bonus\_percent / 100), '99999.99')

);

END LOOP;

COMMIT;

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Bonus for IT Employees (10%) ---');

UpdateEmployeeBonus('IT', 10);

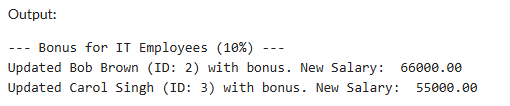
END;

/

SELECT \* FROM Employees;

Output:-





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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

INSERT INTO Accounts VALUES (1, 101, 'Savings', 5000, SYSDATE); -- Source account

INSERT INTO Accounts VALUES (2, 101, 'Checking', 2000, SYSDATE); -- Target account

INSERT INTO Accounts VALUES (3, 102, 'Savings', 3000, SYSDATE); -- Different customer

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

)

IS

v\_from\_balance NUMBER;

v\_from\_cust NUMBER;

v\_to\_cust NUMBER;

BEGIN

-- Check both accounts exist and belong to same customer

SELECT Balance, CustomerID INTO v\_from\_balance, v\_from\_cust

FROM Accounts WHERE AccountID = p\_from\_account;

SELECT CustomerID INTO v\_to\_cust

FROM Accounts WHERE AccountID = p\_to\_account;

IF v\_from\_cust != v\_to\_cust THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Accounts must belong to the same customer.');

END IF;

IF v\_from\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance in source account.');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

BEGIN

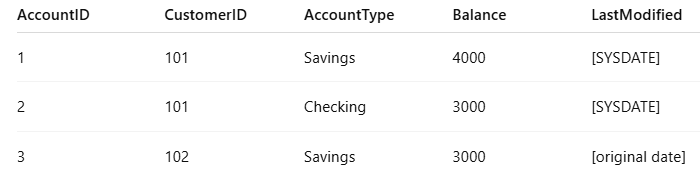
TransferFunds(1, 2, 1000);

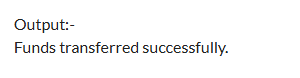
END;

/

SELECT \* FROM Accounts;

Output:-





**Exercise 4: Functions**

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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

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

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('2002-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Customers VALUES (3, 'Young Person', TO\_DATE('2010-12-01', 'YYYY-MM-DD'), 500, SYSDATE);

COMMIT;

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

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

RETURN v\_age;

END;

/

SELECT

CustomerID,

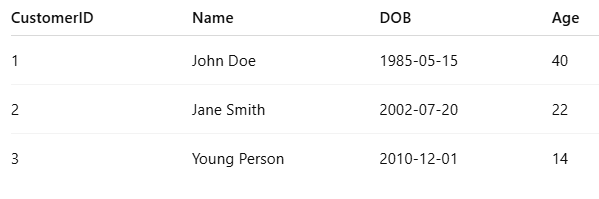
Name,

DOB,

CalculateAge(DOB) AS Age

FROM Customers;

**Output:-**

****

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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER, -- annual rate in %

StartDate DATE,

EndDate DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

INSERT INTO Loans VALUES (1, 101, 120000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 60)); -- 5 yrs

INSERT INTO Loans VALUES (2, 102, 100000, 8, SYSDATE, ADD\_MONTHS(SYSDATE, 36)); -- 3 yrs

INSERT INTO Accounts VALUES (1, 101, 'Savings', 10000, SYSDATE);

INSERT INTO Accounts VALUES (2, 102, 'Checking', 500, SYSDATE);

COMMIT;

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_amount NUMBER,

p\_rate NUMBER,

p\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER := p\_rate / (12 \* 100);

v\_months NUMBER := p\_years \* 12;

v\_emi NUMBER;

BEGIN

-- EMI = P \* r \* (1 + r)^n / ((1 + r)^n - 1)

v\_emi := p\_amount \* v\_monthly\_rate \* POWER(1 + v\_monthly\_rate, v\_months) /

(POWER(1 + v\_monthly\_rate, v\_months) - 1);

RETURN ROUND(v\_emi, 2);

END;

/

SELECT

LoanID,

LoanAmount,

InterestRate,

MONTHS\_BETWEEN(EndDate, StartDate)/12 AS Years,

CalculateMonthlyInstallment(LoanAmount, InterestRate, MONTHS\_BETWEEN(EndDate, StartDate)/12) AS Monthly\_Installment

FROM Loans;

/

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

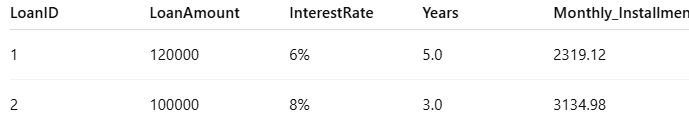
WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

Output:-



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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER, -- annual rate in %

StartDate DATE,

EndDate DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

INSERT INTO Loans VALUES (1, 101, 120000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 60)); -- 5 yrs

INSERT INTO Loans VALUES (2, 102, 100000, 8, SYSDATE, ADD\_MONTHS(SYSDATE, 36)); -- 3 yrs

INSERT INTO Accounts VALUES (1, 101, 'Savings', 10000, SYSDATE);

INSERT INTO Accounts VALUES (2, 102, 'Checking', 500, SYSDATE);

COMMIT;

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_amount NUMBER,

p\_rate NUMBER,

p\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER := p\_rate / (12 \* 100);

v\_months NUMBER := p\_years \* 12;

v\_emi NUMBER;

BEGIN

-- EMI = P \* r \* (1 + r)^n / ((1 + r)^n - 1)

v\_emi := p\_amount \* v\_monthly\_rate \* POWER(1 + v\_monthly\_rate, v\_months) /

(POWER(1 + v\_monthly\_rate, v\_months) - 1);

RETURN ROUND(v\_emi, 2);

END;

/

SELECT

LoanID,

LoanAmount,

InterestRate,

MONTHS\_BETWEEN(EndDate, StartDate)/12 AS Years,

CalculateMonthlyInstallment(LoanAmount, InterestRate, MONTHS\_BETWEEN(EndDate, StartDate)/12) AS Monthly\_Installment

FROM Loans;

/

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

SELECT

AccountID,

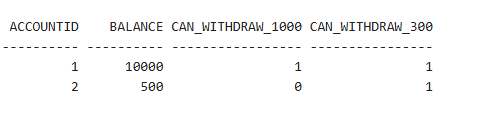
Balance,

HasSufficientBalance(AccountID, 1000) AS Can\_Withdraw\_1000,

HasSufficientBalance(AccountID, 300) AS Can\_Withdraw\_300

FROM Accounts;

Output:-



**Exercise 5: Triggers**

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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

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

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 2000, SYSDATE);

COMMIT;

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

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

FOR rec IN (SELECT \* FROM Customers) LOOP

DBMS\_OUTPUT.PUT\_LINE(rec.CustomerID || ' | ' || rec.Name || ' | ' || rec.Balance || ' | ' || rec.LastModified);

END LOOP;

-- Simulate an update

UPDATE Customers

SET Balance = Balance + 500

WHERE CustomerID = 1;

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

FOR rec IN (SELECT \* FROM Customers) LOOP

DBMS\_OUTPUT.PUT\_LINE(rec.CustomerID || ' | ' || rec.Name || ' | ' || rec.Balance || ' | ' || rec.LastModified);

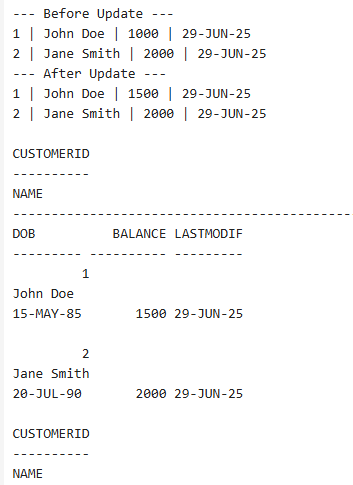
END LOOP;

END;

/

SELECT \* FROM Customers;

Output:-



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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE AuditLog';

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

Action VARCHAR2(50),

Timestamp DATE

);

INSERT INTO Accounts VALUES (1, 101, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 102, 'Checking', 1500, SYSDATE);

COMMIT;

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, AccountID, Action, Timestamp)

VALUES (:NEW.TransactionID, :NEW.AccountID, 'INSERTED TRANSACTION', SYSDATE);

END;

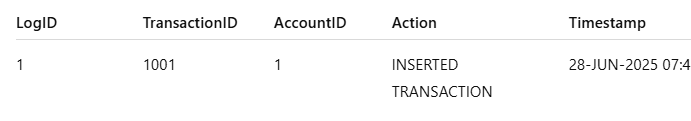
/

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

VALUES (1001, 1, SYSDATE, 500, 'Deposit');

SELECT \* FROM AuditLog;

Output:-



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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

INSERT INTO Accounts VALUES (1, 101, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 102, 'Checking', 500, SYSDATE);

COMMIT;

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 account balance.');

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;

/

BEGIN

INSERT INTO Transactions VALUES (1001, 1, SYSDATE, 500, 'Withdrawal');

INSERT INTO Transactions VALUES (1002, 2, SYSDATE, 0, 'Deposit');

EXCEPTION

WHEN OTHERS THEN

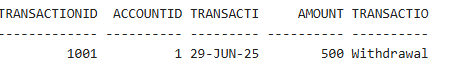
DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

SELECT \* FROM Transactions;

Output:-



**Exercise 6: Cursors**

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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

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)

);

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

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

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions VALUES (101, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (102, 1, SYSDATE, 100, 'Withdrawal');

INSERT INTO Transactions VALUES (103, 2, ADD\_MONTHS(SYSDATE, -1), 150, 'Deposit');

INSERT INTO Transactions VALUES (104, 2, SYSDATE, 250, 'Withdrawal');

COMMIT;

DECLARE

CURSOR txn\_cursor IS

SELECT

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

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

JOIN Customers c ON a.CustomerID = c.CustomerID

WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM')

ORDER BY c.CustomerID, t.TransactionDate;

txn\_record txn\_cursor%ROWTYPE;

current\_customer\_id NUMBER := 0;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Monthly Statements for ' || TO\_CHAR(SYSDATE, 'Month YYYY') || ' ---');

OPEN txn\_cursor;

LOOP

FETCH txn\_cursor INTO txn\_record;

EXIT WHEN txn\_cursor%NOTFOUND;

IF txn\_record.CustomerID != current\_customer\_id THEN

current\_customer\_id := txn\_record.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Customer: ' || txn\_record.Name || ' (ID: ' || txn\_record.CustomerID || ')');

END IF;

DBMS\_OUTPUT.PUT\_LINE(' - ' || txn\_record.TransactionDate || ' | ' || txn\_record.TransactionType || ' | Amount: $' || txn\_record.Amount);

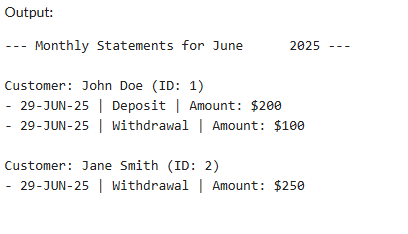
END LOOP;

CLOSE txn\_cursor;

END;

/

Output:-



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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

INSERT INTO Accounts VALUES (1, 101, 'Savings', 1200, SYSDATE);

INSERT INTO Accounts VALUES (2, 102, 'Checking', 800, SYSDATE);

INSERT INTO Accounts VALUES (3, 103, 'Savings', 300, SYSDATE);

COMMIT;

DECLARE

CURSOR acc\_cursor IS

SELECT AccountID, Balance FROM Accounts;

v\_rec acc\_cursor%ROWTYPE;

v\_fee CONSTANT NUMBER := 200;

BEGIN

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

OPEN acc\_cursor;

LOOP

FETCH acc\_cursor INTO v\_rec;

EXIT WHEN acc\_cursor%NOTFOUND;

IF v\_rec.Balance >= v\_fee THEN

UPDATE Accounts

SET Balance = Balance - v\_fee,

LastModified = SYSDATE

WHERE AccountID = v\_rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Fee deducted from Account ID ' || v\_rec.AccountID ||

'. New Balance: ₹' || TO\_CHAR(v\_rec.Balance - v\_fee));

ELSE

DBMS\_OUTPUT.PUT\_LINE('Skipped Account ID ' || v\_rec.AccountID ||

' due to insufficient balance (₹' || TO\_CHAR(v\_rec.Balance) || ')');

END IF;

END LOOP;

CLOSE acc\_cursor;

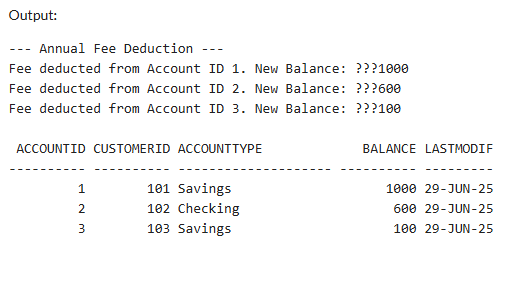
COMMIT;

END;

/

SELECT \* FROM Accounts;

Output:-



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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE

);

INSERT INTO Loans VALUES (1, 101, 15000, 6.5, TO\_DATE('2022-01-01', 'YYYY-MM-DD'), TO\_DATE('2027-01-01', 'YYYY-MM-DD'));

INSERT INTO Loans VALUES (2, 102, 8000, 5.0, TO\_DATE('2023-06-01', 'YYYY-MM-DD'), TO\_DATE('2026-06-01', 'YYYY-MM-DD'));

INSERT INTO Loans VALUES (3, 103, 25000, 7.0, TO\_DATE('2021-08-15', 'YYYY-MM-DD'), TO\_DATE('2026-08-15', 'YYYY-MM-DD'));

COMMIT;

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, LoanAmount, InterestRate FROM Loans;

loan\_rec loan\_cursor%ROWTYPE;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Updating Loan Interest Rates ---');

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO loan\_rec;

EXIT WHEN loan\_cursor%NOTFOUND;

IF loan\_rec.LoanAmount >= 10000 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = loan\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Loan ID ' || loan\_rec.LoanID || ' reduced by 1%. New rate: ' || TO\_CHAR(loan\_rec.InterestRate - 1));

ELSE

UPDATE Loans

SET InterestRate = InterestRate + 0.5

WHERE LoanID = loan\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Loan ID ' || loan\_rec.LoanID || ' increased by 0.5%. New rate: ' || TO\_CHAR(loan\_rec.InterestRate + 0.5));

END IF;

END LOOP;

CLOSE loan\_cursor;

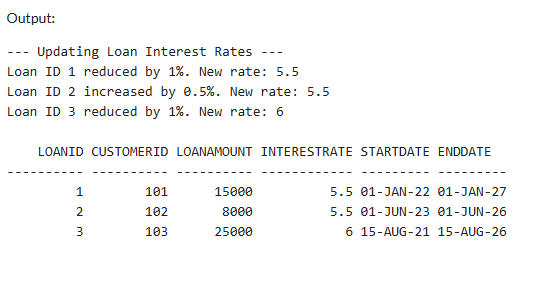
COMMIT;

END;

/

SELECT \* FROM Loans;

Output:-



**Exercise 7: Packages**

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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 5000, SYSDATE);

COMMIT;

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

PROCEDURE UpdateCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_Balance NUMBER);

FUNCTION GetCustomerBalance(p\_ID NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

PROCEDURE AddCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER) IS

BEGIN

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

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

END;

PROCEDURE UpdateCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_Balance NUMBER) IS

BEGIN

UPDATE Customers

SET Name = p\_Name, Balance = p\_Balance, LastModified = SYSDATE

WHERE CustomerID = p\_ID;

IF SQL%ROWCOUNT = 0 THEN

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

END IF;

END;

FUNCTION GetCustomerBalance(p\_ID NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_ID;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END CustomerManagement;

/

BEGIN

CustomerManagement.AddCustomer(2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 6000);

CustomerManagement.UpdateCustomer(1, 'Johnathan Doe', 5500);

DBMS\_OUTPUT.PUT\_LINE('Balance of Customer 1: ₹' || CustomerManagement.GetCustomerBalance(1));

END;

/

SELECT \* FROM Customers;

Output:-



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

Query:-

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- Insert initial data

INSERT INTO Employees VALUES (1, 'Alice Johnson', 'Manager', 80000, 'HR', TO\_DATE('2019-01-15', 'YYYY-MM-DD'));

COMMIT;

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(

p\_ID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2,

p\_Salary NUMBER, p\_Department VARCHAR2, p\_HireDate DATE

);

PROCEDURE UpdateEmployeeDetails(

p\_ID NUMBER, p\_Salary NUMBER, p\_Position VARCHAR2

);

FUNCTION CalculateAnnualSalary(p\_ID NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(

p\_ID NUMBER, p\_Name VARCHAR2, p\_Position VARCHAR2,

p\_Salary NUMBER, p\_Department VARCHAR2, p\_HireDate DATE

) IS

BEGIN

INSERT INTO Employees

VALUES (p\_ID, p\_Name, p\_Position, p\_Salary, p\_Department, p\_HireDate);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Employee already exists.');

END;

PROCEDURE UpdateEmployeeDetails(

p\_ID NUMBER, p\_Salary NUMBER, p\_Position VARCHAR2

) IS

BEGIN

UPDATE Employees

SET Salary = p\_Salary,

Position = p\_Position

WHERE EmployeeID = p\_ID;

IF SQL%ROWCOUNT = 0 THEN

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

END IF;

END;

FUNCTION CalculateAnnualSalary(p\_ID NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_ID;

RETURN v\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END EmployeeManagement;

/

BEGIN

EmployeeManagement.HireEmployee(2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2021-06-10', 'YYYY-MM-DD'));

EmployeeManagement.UpdateEmployeeDetails(1, 85000, 'Senior Manager');

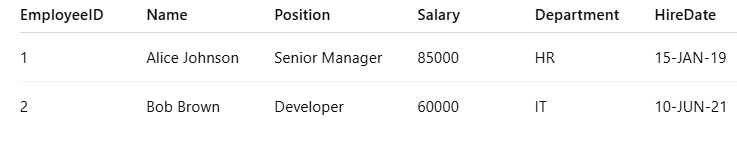
DBMS\_OUTPUT.PUT\_LINE('Annual Salary of Employee 1: ₹' || EmployeeManagement.CalculateAnnualSalary(1));

END;

/

SELECT \* FROM Employees;

Output:-





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

Query:-

BEGIN

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE

);

INSERT INTO Accounts VALUES (1, 101, 'Savings', 5000, SYSDATE);

INSERT INTO Accounts VALUES (2, 101, 'Checking', 2500, SYSDATE);

INSERT INTO Accounts VALUES (3, 102, 'Savings', 8000, SYSDATE);

COMMIT;

CREATE OR REPLACE PACKAGE AccountOperations IS

PROCEDURE OpenAccount(

p\_AccountID NUMBER, p\_CustomerID NUMBER,

p\_Type VARCHAR2, p\_Balance NUMBER

);

PROCEDURE CloseAccount(p\_AccountID NUMBER);

FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations IS

PROCEDURE OpenAccount(

p\_AccountID NUMBER, p\_CustomerID NUMBER,

p\_Type VARCHAR2, p\_Balance NUMBER

) IS

BEGIN

INSERT INTO Accounts

VALUES (p\_AccountID, p\_CustomerID, p\_Type, p\_Balance, SYSDATE);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Account already exists.');

END;

PROCEDURE CloseAccount(p\_AccountID NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_AccountID;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

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

END IF;

END;

FUNCTION GetTotalBalance(p\_CustomerID NUMBER) RETURN NUMBER IS

v\_total NUMBER := 0;

BEGIN

SELECT NVL(SUM(Balance), 0)

INTO v\_total

FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN v\_total;

END;

END AccountOperations;

/

BEGIN

AccountOperations.OpenAccount(4, 101, 'Savings', 1000);

AccountOperations.CloseAccount(3);

DBMS\_OUTPUT.PUT\_LINE('Total Balance of Customer 101: ₹' || AccountOperations.GetTotalBalance(101));

END;

/

SELECT \* FROM Accounts;

Output:-

