**Week-2: PL SQL Programming**

**Code**

Table creation

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),

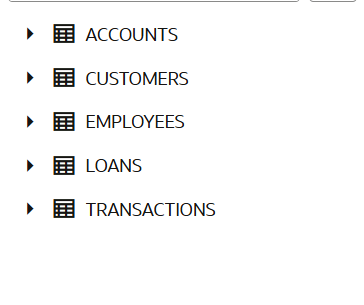
Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);



Inserting columns- Customers Table

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

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000.50, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 15000.75, SYSDATE);

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

VALUES (3, 'Peter Jones', TO\_DATE('1962-03-01', 'YYYY-MM-DD'), 5000.00, SYSDATE);

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

VALUES (4, 'Mary Brown', TO\_DATE('1995-11-25', 'YYYY-MM-DD'), 250.25, SYSDATE);

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

VALUES (5, 'Robert Davis', TO\_DATE('1970-01-10', 'YYYY-MM-DD'), 8000.99, SYSDATE);

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

VALUES (6, 'Linda Miller', TO\_DATE('1958-09-30', 'YYYY-MM-DD'), 22000.40, SYSDATE);

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

VALUES (7, 'William Garcia', TO\_DATE('1980-04-05', 'YYYY-MM-DD'), 300.00, SYSDATE);

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

VALUES (8, 'Patricia Rodriguez', TO\_DATE('1975-02-14', 'YYYY-MM-DD'), 1200.60, SYSDATE);

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

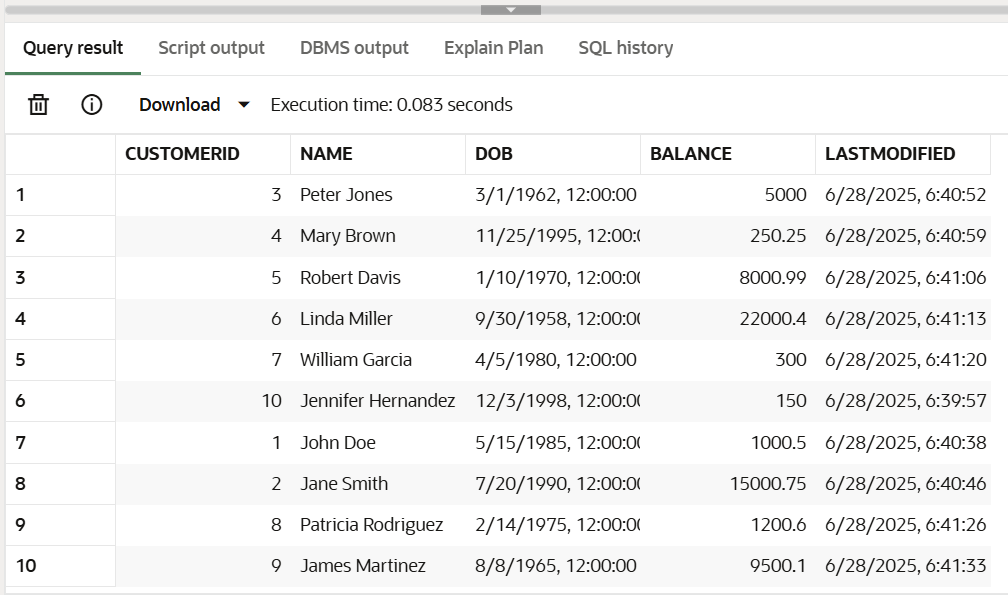
VALUES (9, 'James Martinez', TO\_DATE('1965-08-08', 'YYYY-MM-DD'), 9500.10, SYSDATE);

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

VALUES (10, 'Jennifer Hernandez', TO\_DATE('1998-12-03', 'YYYY-MM-DD'), 150.00, SYSDATE);

SELECT \*

FROM CUSTOMERS;



Accounts Table

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

VALUES (101, 1, 'Savings', 5000.00, SYSDATE);

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

VALUES (102, 2, 'Checking', 12000.50, SYSDATE);

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

VALUES (103, 3, 'Savings', 800.75, SYSDATE);

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

VALUES (104, 4, 'Checking', 300.00, SYSDATE);

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

VALUES (105, 5, 'Savings', 1500.20, SYSDATE);

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

VALUES (106, 6, 'Checking', 25000.00, SYSDATE);

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

VALUES (107, 7, 'Savings', 750.00, SYSDATE);

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

VALUES (108, 8, 'Checking', 4200.50, SYSDATE);

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

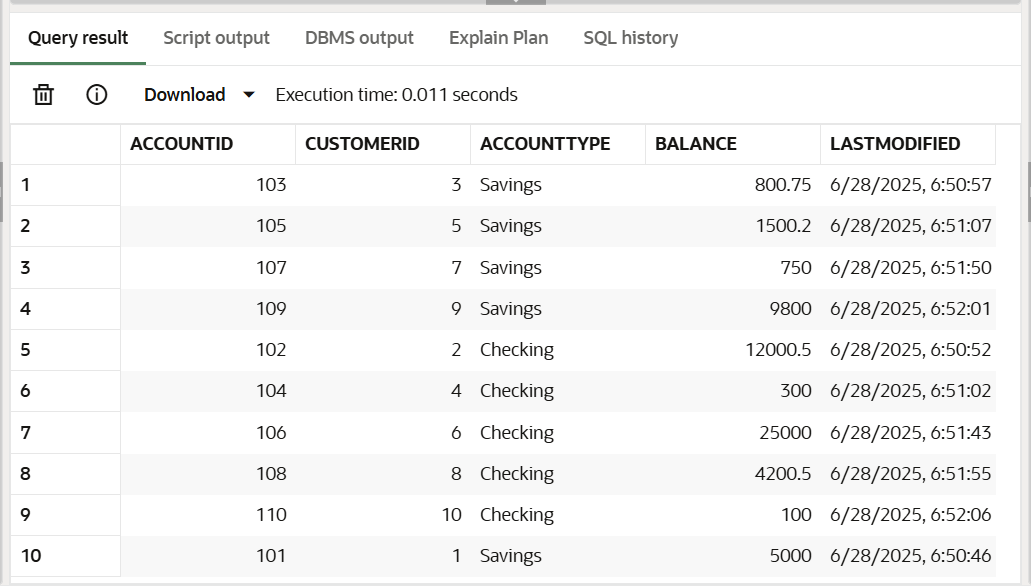
VALUES (109, 9, 'Savings', 9800.00, SYSDATE);

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

VALUES (110, 10, 'Checking', 100.00, SYSDATE);

SELECT \*

FROM Accounts;



Transactions Table

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

VALUES (1001, 101, SYSDATE - 10, 500, 'Deposit');

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

VALUES (1002, 102, SYSDATE - 8, 200, 'Withdrawal');

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

VALUES (1003, 103, SYSDATE - 5, 100, 'Deposit');

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

VALUES (1004, 104, SYSDATE - 3, 50, 'Withdrawal');

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

VALUES (1005, 105, SYSDATE - 1, 150, 'Deposit');

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

VALUES (1006, 106, SYSDATE, 1000, 'Deposit');

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

VALUES (1007, 107, SYSDATE, 20, 'Withdrawal');

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

VALUES (1008, 108, SYSDATE, 75, 'Deposit');

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

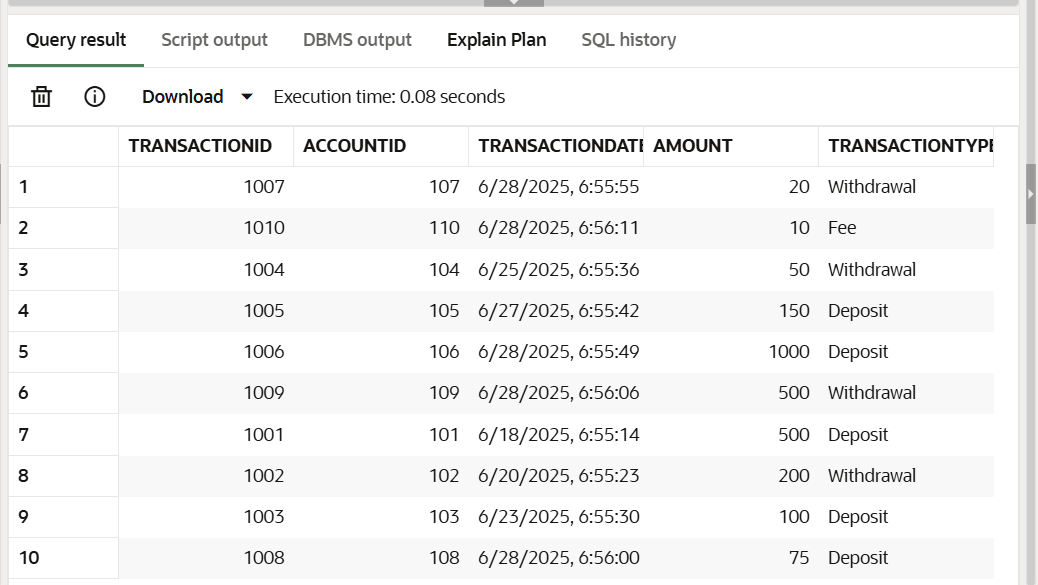
VALUES (1009, 109, SYSDATE, 500, 'Withdrawal');

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

VALUES (1010, 110, SYSDATE, 10, 'Fee');

SELECT \*

FROM TRANSACTIONS;



Loans Table

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

VALUES (201, 1, 10000, 4.5, TO\_DATE('2024-01-01', 'YYYY-MM-DD'), ADD\_MONTHS(TO\_DATE('2024-01-01', 'YYYY-MM-DD'), 60));

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

VALUES (202, 2, 25000, 3.8, TO\_DATE('2023-06-15', 'YYYY-MM-DD'), ADD\_MONTHS(TO\_DATE('2023-06-15', 'YYYY-MM-DD'), 120));

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

VALUES (203, 3, 5000, 6.0, TO\_DATE('2025-03-01', 'YYYY-MM-DD'), ADD\_MONTHS(TO\_DATE('2025-03-01', 'YYYY-MM-DD'), 36));

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

VALUES (204, 4, 1500, 7.2, TO\_DATE('2024-11-20', 'YYYY-MM-DD'), ADD\_MONTHS(TO\_DATE('2024-11-20', 'YYYY-MM-DD'), 24));

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

VALUES (205, 5, 30000, 4.0, TO\_DATE('2022-09-01', 'YYYY-MM-DD'), ADD\_MONTHS(TO\_DATE('2022-09-01', 'YYYY-MM-DD'), 180));

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

VALUES (206, 6, 7500, 5.5, TO\_DATE('2025-01-05', 'YYYY-MM-DD'), ADD\_MONTHS(TO\_DATE('2025-01-05', 'YYYY-MM-DD'), 48));

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

VALUES (207, 7, 2000, 6.8, TO\_DATE('2024-07-10', 'YYYY-MM-DD'), ADD\_MONTHS(TO\_DATE('2024-07-10', 'YYYY-MM-DD'), 30));

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

VALUES (208, 8, 12000, 4.2, TO\_DATE('2023-02-28', 'YYYY-MM-DD'), ADD\_MONTHS(TO\_DATE('2023-02-28', 'YYYY-MM-DD'), 96));

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

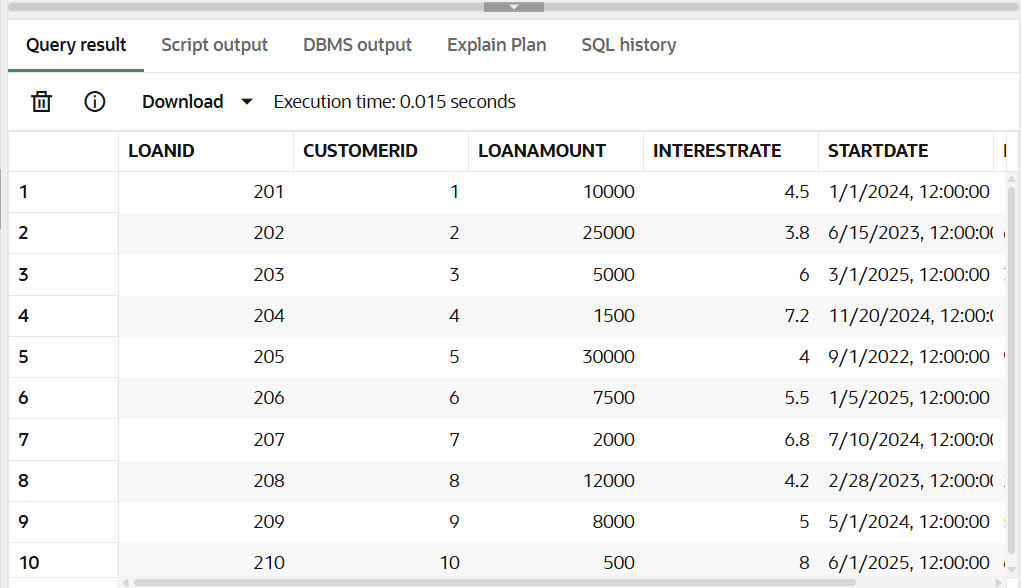
VALUES (209, 9, 8000, 5.0, TO\_DATE('2024-05-01', 'YYYY-MM-DD'), ADD\_MONTHS(TO\_DATE('2024-05-01', 'YYYY-MM-DD'), 72));

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

VALUES (210, 10, 500, 8.0, TO\_DATE('2025-06-01', 'YYYY-MM-DD'), ADD\_MONTHS(TO\_DATE('2025-06-01', 'YYYY-MM-DD'), 12));

SELECT \* FROM

LOANS;



Employees Table

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

VALUES (101, 'Michael Green', 'Senior Manager', 85000, 'Sales', TO\_DATE('2010-08-01', 'YYYY-MM-DD'));

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

VALUES (102, 'Emily White', 'Data Analyst', 62000, 'IT', TO\_DATE('2019-01-20', 'YYYY-MM-DD'));

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

VALUES (103, 'David King', 'Customer Service Rep', 45000, 'Operations', TO\_DATE('2021-04-10', 'YYYY-MM-DD'));

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

VALUES (104, 'Sarah Lee', 'HR Specialist', 58000, 'HR', TO\_DATE('2018-11-05', 'YYYY-MM-DD'));

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

VALUES (105, 'Chris Taylor', 'Software Engineer', 78000, 'IT', TO\_DATE('2020-07-01', 'YYYY-MM-DD'));

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

VALUES (106, 'Jessica Clark', 'Marketing Coordinator', 50000, 'Marketing', TO\_DATE('2022-03-15', 'YYYY-MM-DD'));

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

VALUES (107, 'Daniel Hall', 'Financial Advisor', 72000, 'Finance', TO\_DATE('2017-09-01', 'YYYY-MM-DD'));

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

VALUES (108, 'Laura Young', 'Product Manager', 90000, 'Product Development', TO\_DATE('2015-02-20', 'YYYY-MM-DD'));

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

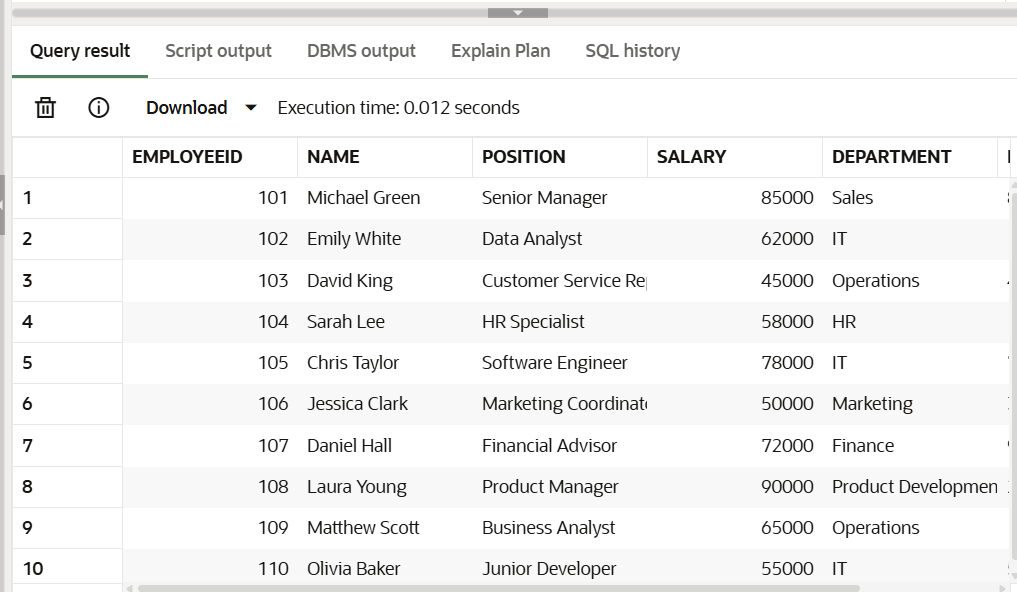
VALUES (109, 'Matthew Scott', 'Business Analyst', 65000, 'Operations', TO\_DATE('2020-10-01', 'YYYY-MM-DD'));

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

VALUES (110, 'Olivia Baker', 'Junior Developer', 55000, 'IT', TO\_DATE('2023-05-10', 'YYYY-MM-DD'));

SELECT \*

FROM EMPLOYEES;



## **Exercise 1: Control Structures**

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

o 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.

DECLARE

CURSOR c\_customers IS

SELECT CustomerID, DOB

FROM Customers;

v\_customer\_id Customers.CustomerID%TYPE;

v\_dob Customers.DOB%TYPE;

v\_age NUMBER;

BEGIN

DBMS\_OUTPUT.ENABLE;

FOR customer\_rec IN c\_customers LOOP

v\_customer\_id := customer\_rec.CustomerID;

v\_dob := customer\_rec.DOB;

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

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate \* 0.99

WHERE CustomerID = v\_customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount for Customer ID: ' || v\_customer\_id || ' (Age: ' || v\_age || ')');

END IF;

END LOOP;

COMMIT;

EXCEPTION

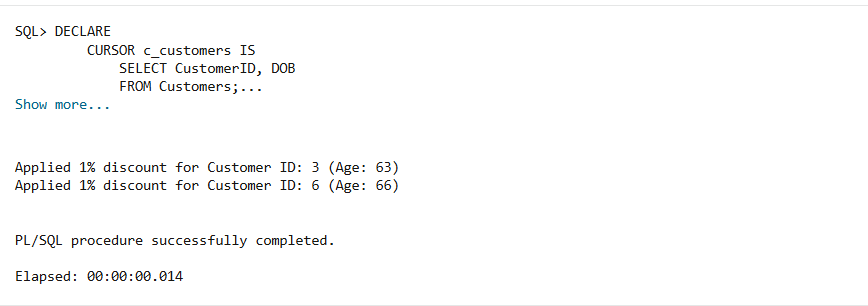
WHEN OTHERS THEN

ROLLBACK;

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

END;

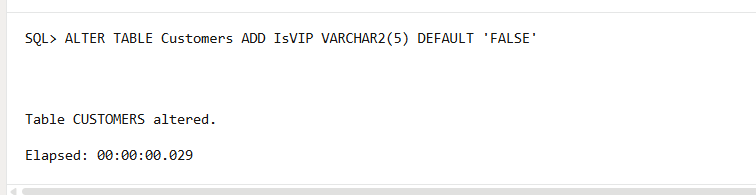
/



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

o 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.

ALTER TABLE Customers ADD IsVIP VARCHAR2(5) DEFAULT 'FALSE';



DECLARE

CURSOR c\_customers IS

SELECT CustomerID, Balance

FROM Customers;

v\_customer\_id Customers.CustomerID%TYPE;

v\_balance Customers.Balance%TYPE;

BEGIN

DBMS\_OUTPUT.ENABLE;

FOR customer\_rec IN c\_customers LOOP

v\_customer\_id := customer\_rec.CustomerID;

v\_balance := customer\_rec.Balance;

IF v\_balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = v\_customer\_id;

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

ELSE

UPDATE Customers

SET IsVIP = 'FALSE'

WHERE CustomerID = v\_customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id || ' is not VIP.');

END IF;

END LOOP;

COMMIT;

EXCEPTION

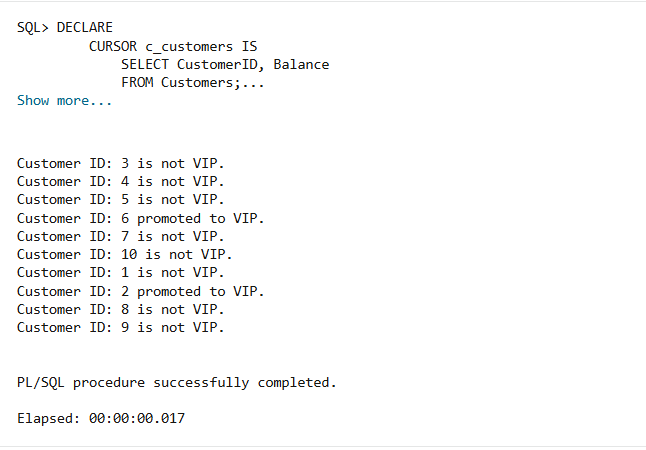
WHEN OTHERS THEN

ROLLBACK;

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

END;

/



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

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

DECLARE

CURSOR c\_due\_loans IS

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND (SYSDATE + 30);

BEGIN

DBMS\_OUTPUT.ENABLE;

DBMS\_OUTPUT.PUT\_LINE('--- Loan Reminders (Next 30 Days) ---');

FOR loan\_rec IN c\_due\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder for ' || loan\_rec.CustomerName || ': Your loan (ID: ' || loan\_rec.LoanID || ') is due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD') || '.');

END LOOP;

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

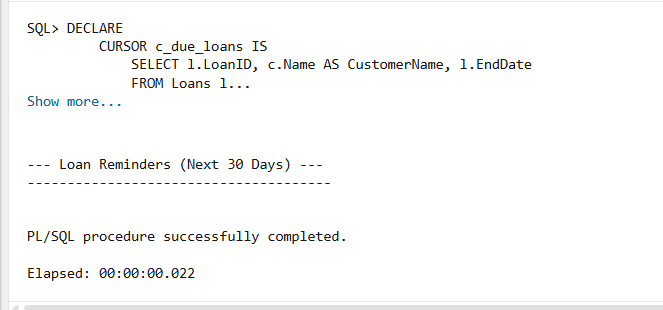
EXCEPTION

WHEN OTHERS THEN

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

END;

/

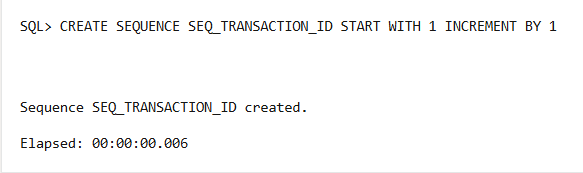


## **Exercise 2: Error Handling**

Scenario 1: Handle exceptions during fund transfers between accounts.

o 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.

CREATE SEQUENCE SEQ\_TRANSACTION\_ID START WITH 1 INCREMENT BY 1;



CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_source\_account\_id IN Accounts.AccountID%TYPE,

p\_destination\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

)

IS

v\_source\_balance Accounts.Balance%TYPE;

e\_insufficient\_funds EXCEPTION;

BEGIN

DBMS\_OUTPUT.ENABLE;

SELECT Balance INTO v\_source\_balance

FROM Accounts

WHERE AccountID = p\_source\_account\_id

FOR UPDATE OF Balance;

IF v\_source\_balance < p\_amount THEN

RAISE e\_insufficient\_funds;

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_source\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_destination\_account\_id;

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

VALUES (SEQ\_TRANSACTION\_ID.NEXTVAL, p\_source\_account\_id, SYSDATE, p\_amount, 'Transfer\_Out');

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

VALUES (SEQ\_TRANSACTION\_ID.NEXTVAL, p\_destination\_account\_id, SYSDATE, p\_amount, 'Transfer\_In');

COMMIT;

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

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: One or both account IDs do not exist or source account is locked.');

WHEN e\_insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in source account ' || p\_source\_account\_id || '. Available: ' || v\_source\_balance || ', Attempted: ' || p\_amount);

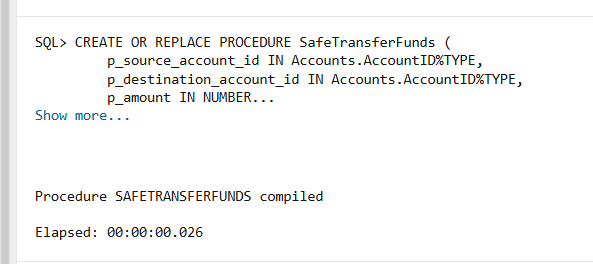
WHEN OTHERS THEN

ROLLBACK;

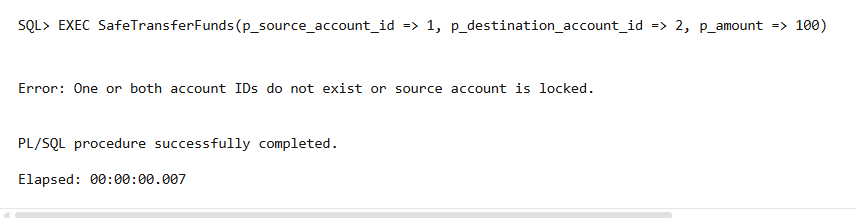
DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred during transfer: ' || SQLERRM);

END SafeTransferFunds;

/



EXEC SafeTransferFunds(p\_source\_account\_id => 1, p\_destination\_account\_id => 2, p\_amount => 100);



Scenario 2: Manage errors when updating employee salaries.

o 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.

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN Employees.EmployeeID%TYPE,

p\_percentage\_increase IN NUMBER

)

IS

v\_rows\_updated NUMBER;

BEGIN

DBMS\_OUTPUT.ENABLE;

UPDATE Employees

SET Salary = Salary \* (1 + p\_percentage\_increase / 100)

WHERE EmployeeID = p\_employee\_id;

v\_rows\_updated := SQL%ROWCOUNT;

IF v\_rows\_updated = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID ' || p\_employee\_id || ' does not exist. Salary update failed.');

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully for Employee ID: ' || p\_employee\_id || '. Increased by ' || p\_percentage\_increase || '%.');

END IF;

EXCEPTION

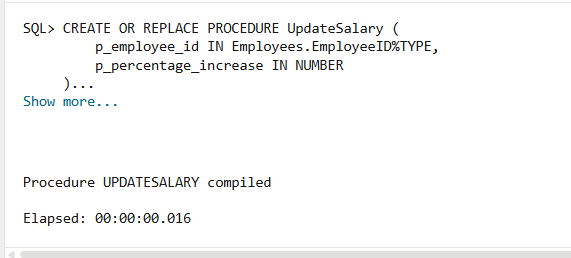
WHEN OTHERS THEN

ROLLBACK;

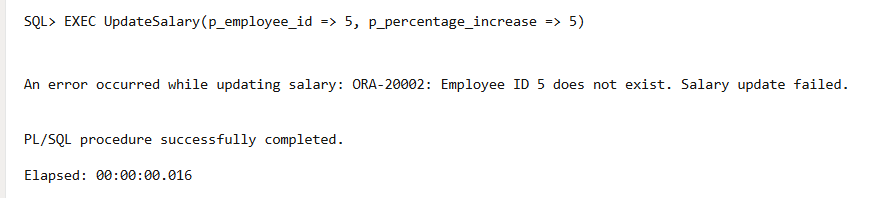
DBMS\_OUTPUT.PUT\_LINE('An error occurred while updating salary: ' || SQLERRM);

END UpdateSalary;

/



EXEC UpdateSalary(p\_employee\_id => 5, p\_percentage\_increase => 5);



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

o 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.

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN Customers.CustomerID%TYPE,

p\_name IN Customers.Name%TYPE,

p\_dob IN Customers.DOB%TYPE,

p\_balance IN Customers.Balance%TYPE

)

IS

e\_duplicate\_customer EXCEPTION;

PRAGMA EXCEPTION\_INIT(e\_duplicate\_customer, -00001);

BEGIN

DBMS\_OUTPUT.ENABLE;

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('New customer added successfully with ID: ' || p\_customer\_id || ' (' || p\_name || ').');

EXCEPTION

WHEN e\_duplicate\_customer THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_customer\_id || ' already exists. Insertion prevented due to duplicate ID.');

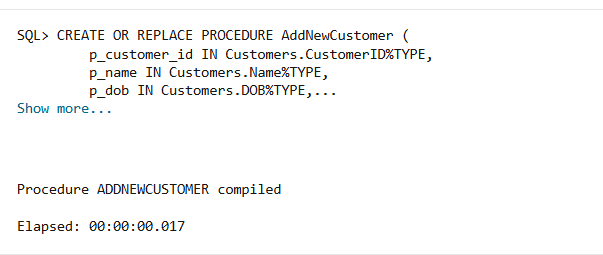
WHEN OTHERS THEN

ROLLBACK;

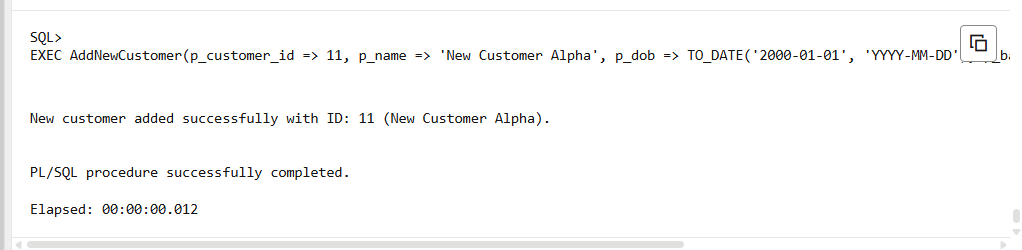
DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred while adding new customer: ' || SQLERRM);

END AddNewCustomer;

/



EXEC AddNewCustomer(p\_customer\_id => 11, p\_name => 'New Customer Alpha', p\_dob => TO\_DATE('2000-01-01', 'YYYY-MM-DD'), p\_balance => 5000);



## **Exercise 3: Stored Procedures**

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

o 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.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

DBMS\_OUTPUT.ENABLE;

UPDATE Accounts

SET Balance = Balance \* 1.01,

LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all savings accounts successfully.');

EXCEPTION

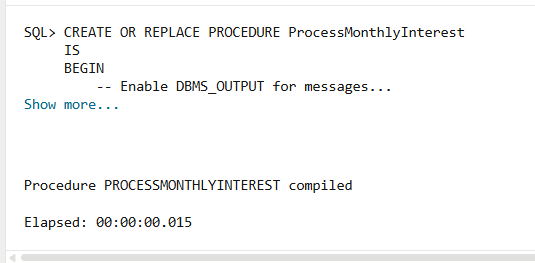
WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('An error occurred while processing monthly interest: ' || SQLERRM);

END ProcessMonthlyInterest;

/

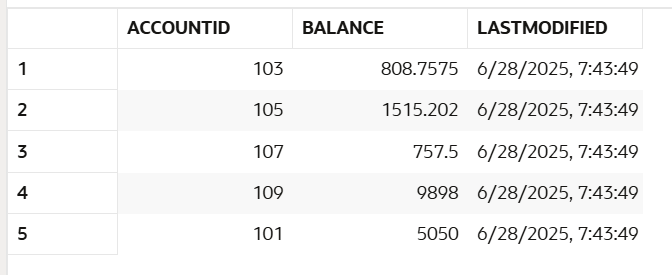


EXEC ProcessMonthlyInterest;

SELECT AccountID, Balance, LastModified

FROM Accounts

WHERE AccountType = 'Savings';



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

o 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.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN Employees.Department%TYPE,

p\_bonus\_percentage IN NUMBER

)

IS

v\_rows\_updated NUMBER;

BEGIN DBMS\_OUTPUT.ENABLE;

UPDATE Employees

SET Salary = Salary \* (1 + p\_bonus\_percentage / 100)

WHERE Department = p\_department;

v\_rows\_updated := SQL%ROWCOUNT;

IF v\_rows\_updated > 0 THEN

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(v\_rows\_updated || ' employees in department ' || p\_department || ' received a ' || p\_bonus\_percentage || '% bonus.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('No employees found in department: ' || p\_department || '. No bonus applied.');

END IF;

EXCEPTION

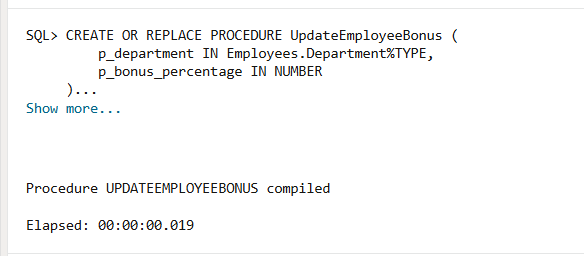
WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('An error occurred while updating employee bonus: ' || SQLERRM);

END UpdateEmployeeBonus;

/

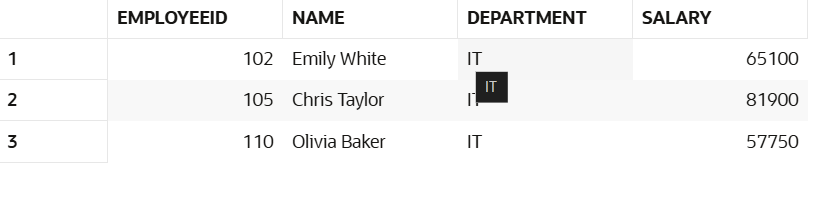


EXEC UpdateEmployeeBonus(p\_department => 'IT', p\_bonus\_percentage => 5);

SELECT EmployeeID, Name, Department, Salary

FROM Employees

WHERE Department = 'IT';



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

o 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.

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_source\_account\_id IN Accounts.AccountID%TYPE,

p\_destination\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

)

IS

v\_source\_balance Accounts.Balance%TYPE;

BEGIN

DBMS\_OUTPUT.ENABLE;

IF p\_amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Transfer amount must be positive.');

END IF;

SELECT Balance INTO v\_source\_balance

FROM Accounts

WHERE AccountID = p\_source\_account\_id

FOR UPDATE OF Balance;

IF v\_source\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Insufficient funds in source account ' || p\_source\_account\_id || '. Available: ' || v\_source\_balance || ', Requested: ' || p\_amount);

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_source\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_destination\_account\_id;

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

VALUES (SEQ\_TRANSACTION\_ID.NEXTVAL, p\_source\_account\_id, SYSDATE, p\_amount, 'Transfer\_Out');

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

VALUES (SEQ\_TRANSACTION\_ID.NEXTVAL, p\_destination\_account\_id, SYSDATE, p\_amount, 'Transfer\_In');

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || p\_amount || ' from Account ' || p\_source\_account\_id || ' to Account ' || p\_destination\_account\_id || ' completed successfully.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: One or both account IDs provided do not exist.');

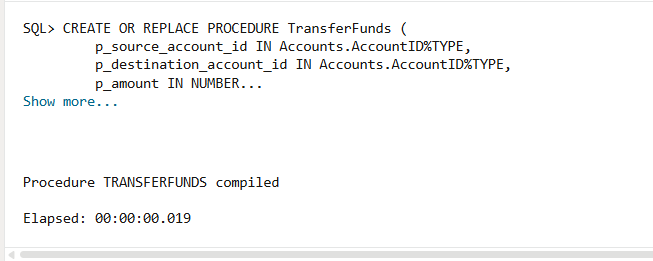
WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred during fund transfer: ' || SQLERRM);

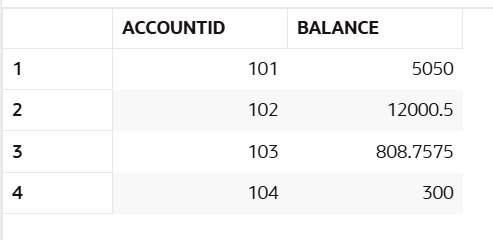
END TransferFunds;

/



EXEC TransferFunds(p\_source\_account\_id => 101, p\_destination\_account\_id => 102, p\_amount => 100);

SELECT AccountID, Balance FROM Accounts WHERE AccountID IN (101, 102, 103, 104);



## **Exercise 4: Functions**

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

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

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob IN DATE

)

RETURN NUMBER

IS

v\_age NUMBER;

BEGIN

IF p\_dob IS NULL THEN

RETURN NULL;

END IF;

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

RETURN v\_age;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20005, 'Error calculating age: ' || SQLERRM);

END CalculateAge;

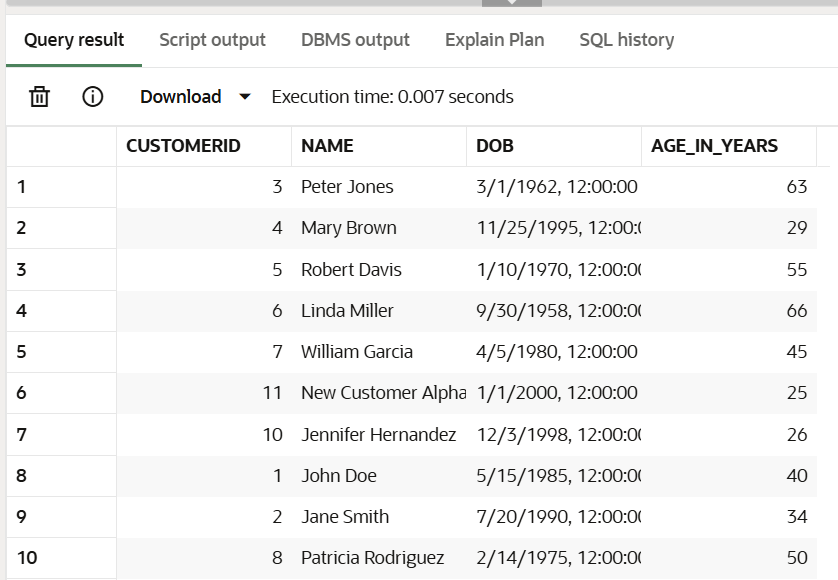
/

SELECT CalculateAge(TO\_DATE('1985-05-15', 'YYYY-MM-DD')) AS Customer\_Age FROM DUAL;

SELECT CustomerID, Name, DOB, CalculateAge(DOB) AS Age\_In\_Years

FROM Customers;





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

o 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.

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount IN NUMBER,

p\_annual\_interest\_rate IN NUMBER,

p\_loan\_duration\_years IN NUMBER

)

RETURN NUMBER

IS

v\_monthly\_interest\_rate NUMBER;

v\_number\_of\_payments NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

IF p\_loan\_amount <= 0 OR p\_loan\_duration\_years <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20006, 'Loan amount and duration must be positive.');

END IF;

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

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

IF v\_monthly\_interest\_rate = 0 THEN

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

ELSE

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

POWER((1 + v\_monthly\_interest\_rate), v\_number\_of\_payments) /

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

END IF;

RETURN ROUND(v\_monthly\_installment, 2);

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20007, 'Error calculating monthly installment: ' || SQLERRM);

END CalculateMonthlyInstallment;

/

SELECT CalculateMonthlyInstallment(

p\_loan\_amount => 10000,

p\_annual\_interest\_rate => 5,

p\_loan\_duration\_years => 5

) AS Monthly\_Installment FROM DUAL;



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

o 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.

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

)

RETURN NUMBER

IS

v\_current\_balance Accounts.Balance%TYPE;

BEGIN

IF p\_amount < 0 THEN

RAISE\_APPLICATION\_ERROR(-20008, 'Amount to check must be non-negative.');

END IF;

SELECT Balance INTO v\_current\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

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

RETURN 1;

ELSE

RETURN 0;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20009, 'Account ID ' || p\_account\_id || ' not found.');

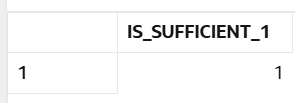
WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20010, 'Error checking sufficient balance: ' || SQLERRM);

END HasSufficientBalance;

/

SELECT HasSufficientBalance(p\_account\_id => 101, p\_amount => 500) AS Is\_Sufficient\_1 FROM DUAL;



## **Exercise 5: Triggers**

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

o 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.

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

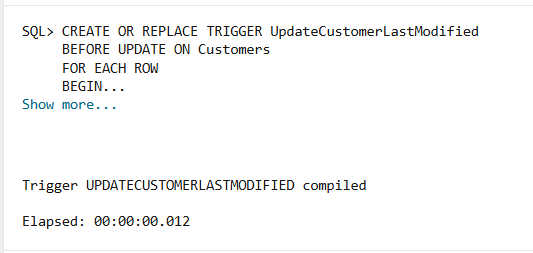
FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END UpdateCustomerLastModified;

/

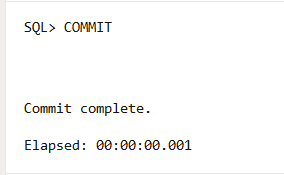


UPDATE Customers

SET Balance = Balance + 100

WHERE CustomerID = 1;

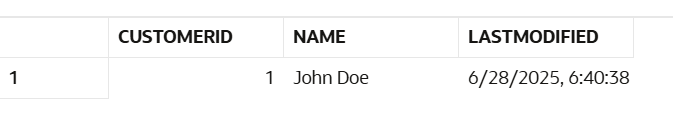
COMMIT;



SELECT CustomerID, Name, LastModified

FROM Customers

WHERE CustomerID = 1;



Scenario 2: Maintain an audit log for all transactions.

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

CREATE TABLE AuditLog (

AuditID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,

TableName VARCHAR2(50) NOT NULL,

OperationType VARCHAR2(20) NOT NULL,

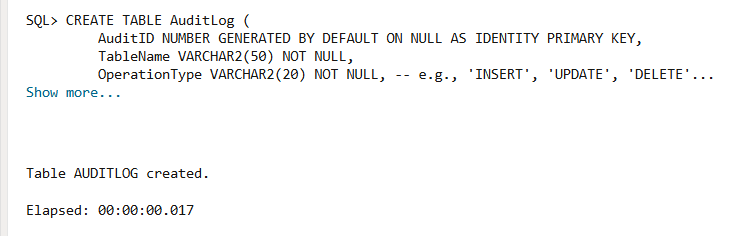
RecordID NUMBER,

ChangeTimestamp TIMESTAMP DEFAULT SYSTIMESTAMP,

ChangedBy VARCHAR2(100) DEFAULT USER,

Detail VARCHAR2(4000)

);



CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TableName, OperationType, RecordID, Detail)

VALUES (

'Transactions',

'INSERT',

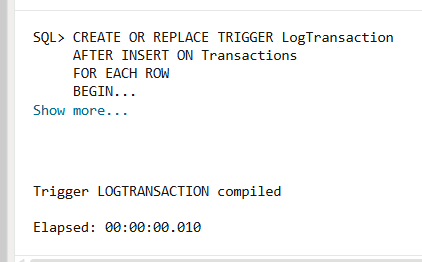
:NEW.TransactionID,

'AccountID: ' || :NEW.AccountID || ', Amount: ' || :NEW.Amount || ', Type: ' || :NEW.TransactionType

);

END LogTransaction;

/



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

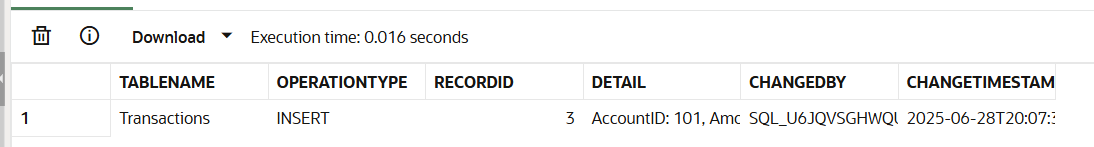
VALUES (SEQ\_TRANSACTION\_ID.NEXTVAL, 101, SYSDATE, 50, 'Deposit');

COMMIT;

SELECT TableName, OperationType, RecordID, Detail, ChangedBy, ChangeTimestamp

FROM AuditLog

ORDER BY ChangeTimestamp DESC;



Scenario 3: Enforce business rules on deposits and withdrawals.

o 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.

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_current\_balance Accounts.Balance%TYPE;

BEGIN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20011, 'Transaction amount must be positive.');

END IF;

SELECT Balance INTO v\_current\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' THEN

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

RAISE\_APPLICATION\_ERROR(-20012, 'Insufficient funds for withdrawal from Account ID ' || :NEW.AccountID || '. Current balance: ' || v\_current\_balance || ', Attempted withdrawal: ' || :NEW.Amount);

END IF;

END IF;

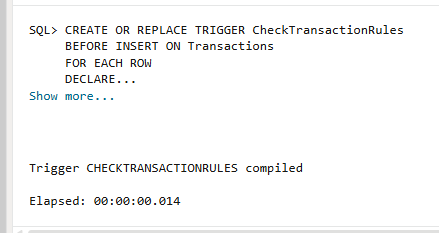
EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20013, 'Account ID ' || :NEW.AccountID || ' not found for transaction.');

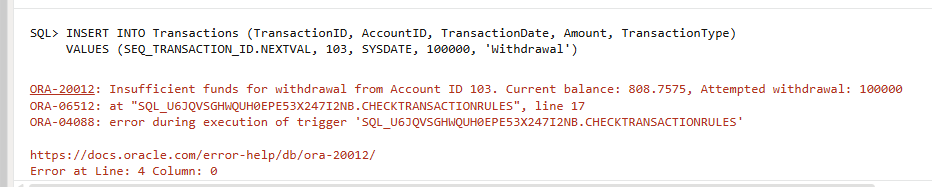
END CheckTransactionRules;

/



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

VALUES (SEQ\_TRANSACTION\_ID.NEXTVAL, 103, SYSDATE, 100000, 'Withdrawal');



## **Exercise 6: Cursors**

Scenario 1: Generate monthly statements for all customers.

o 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.

DECLARE

CURSOR c\_customers\_with\_transactions IS

SELECT DISTINCT c.CustomerID, c.Name

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

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

CURSOR c\_customer\_transactions (p\_customer\_id Customers.CustomerID%TYPE) IS

SELECT t.TransactionID, t.AccountID, t.TransactionDate, t.Amount, t.TransactionType

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

WHERE a.CustomerID = p\_customer\_id

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

ORDER BY t.TransactionDate;

v\_total\_transactions NUMBER := 0;

BEGIN

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

DBMS\_OUTPUT.PUT\_LINE(RPAD('-', 70, '-'));

FOR cust\_rec IN c\_customers\_with\_transactions LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || cust\_rec.Name || ' (ID: ' || cust\_rec.CustomerID || ')');

DBMS\_OUTPUT.PUT\_LINE('Account ID | Date | Type | Amount');

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

v\_total\_transactions := 0;

FOR trans\_rec IN c\_customer\_transactions(cust\_rec.CustomerID) LOOP

DBMS\_OUTPUT.PUT\_LINE(RPAD(trans\_rec.AccountID, 10) || ' | ' ||

TO\_CHAR(trans\_rec.TransactionDate, 'DD-MON-YY') || ' | ' ||

RPAD(trans\_rec.TransactionType, 11) || ' | ' ||

TO\_CHAR(trans\_rec.Amount, '999,999.00'));

v\_total\_transactions := v\_total\_transactions + 1;

END LOOP;

IF v\_total\_transactions = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('No transactions for this customer in the current month.');

END IF;

DBMS\_OUTPUT.PUT\_LINE(RPAD('-', 70, '-'));

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

END LOOP;

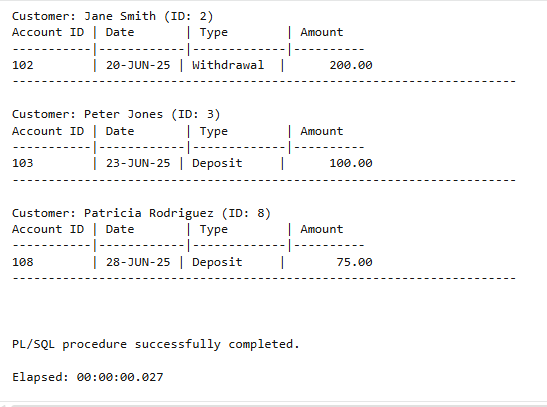
EXCEPTION

WHEN OTHERS THEN

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

END;

/



Scenario 2: Apply annual fee to all accounts.

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

DECLARE

CURSOR c\_all\_accounts IS

SELECT AccountID, Balance

FROM Accounts

FOR UPDATE OF Balance;

v\_annual\_fee CONSTANT NUMBER := 25.00;

BEGIN

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

DBMS\_OUTPUT.PUT\_LINE(RPAD('-', 40, '-'));

FOR account\_rec IN c\_all\_accounts LOOP

UPDATE Accounts

SET Balance = Balance - v\_annual\_fee,

LastModified = SYSDATE

WHERE AccountID = account\_rec.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Deducted ' || TO\_CHAR(v\_annual\_fee, '999.00') ||

' from Account ID: ' || account\_rec.AccountID ||

'. New balance: ' || TO\_CHAR(account\_rec.Balance - v\_annual\_fee, '999,999.00'));

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(RPAD('-', 40, '-'));

DBMS\_OUTPUT.PUT\_LINE('Annual fees applied successfully.');

EXCEPTION

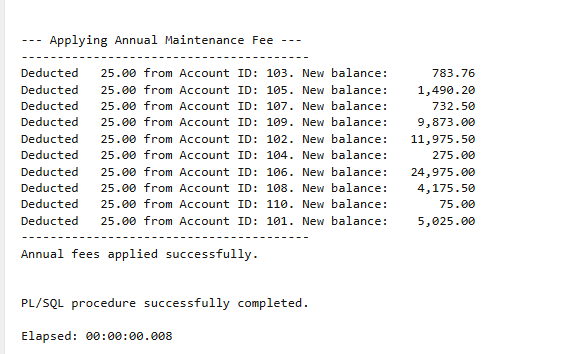
WHEN OTHERS THEN

ROLLBACK;

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

END;

/



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

o 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.

DECLARE

CURSOR c\_all\_loans IS

SELECT LoanID, InterestRate

FROM Loans

FOR UPDATE OF InterestRate;

v\_interest\_rate\_increase CONSTANT NUMBER := 0.2;

BEGIN

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

DBMS\_OUTPUT.PUT\_LINE(RPAD('-', 50, '-'));

FOR loan\_rec IN c\_all\_loans LOOP

UPDATE Loans

SET InterestRate = loan\_rec.InterestRate + v\_interest\_rate\_increase

WHERE LoanID = loan\_rec.LoanID;

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

' - Old Rate: ' || TO\_CHAR(loan\_rec.InterestRate, '99.99') || '%' ||

' - New Rate: ' || TO\_CHAR(loan\_rec.InterestRate + v\_interest\_rate\_increase, '99.99') || '%');

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(RPAD('-', 50, '-'));

DBMS\_OUTPUT.PUT\_LINE('Loan interest rates updated successfully.');

EXCEPTION

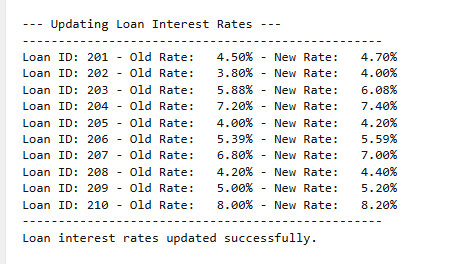
WHEN OTHERS THEN

ROLLBACK;

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

END;

/



## **Exercise 7: Packages**

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

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

Create Package

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer (

p\_customer\_id IN Customers.CustomerID%TYPE,

p\_name IN Customers.Name%TYPE,

p\_dob IN Customers.DOB%TYPE,

p\_balance IN Customers.Balance%TYPE

);

PROCEDURE UpdateCustomerDetails (

p\_customer\_id IN Customers.CustomerID%TYPE,

p\_new\_name IN Customers.Name%TYPE DEFAULT NULL,

p\_new\_dob IN Customers.DOB%TYPE DEFAULT NULL,

p\_new\_balance IN Customers.Balance%TYPE DEFAULT NULL

);

FUNCTION GetCustomerBalance (

p\_customer\_id IN Customers.CustomerID%TYPE

) RETURN Customers.Balance%TYPE;

END CustomerManagement;

/

Create Package Body

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddNewCustomer (

p\_customer\_id IN Customers.CustomerID%TYPE,

p\_name IN Customers.Name%TYPE,

p\_dob IN Customers.DOB%TYPE,

p\_balance IN Customers.Balance%TYPE

)

IS

e\_duplicate\_customer EXCEPTION;

PRAGMA EXCEPTION\_INIT(e\_duplicate\_customer, -00001); -- Unique constraint violated

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('CustomerManagement.AddNewCustomer: New customer ' || p\_name || ' added with ID: ' || p\_customer\_id);

EXCEPTION

WHEN e\_duplicate\_customer THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20014, 'CustomerManagement.AddNewCustomer: Error - Customer with ID ' || p\_customer\_id || ' already exists.');

WHEN OTHERS THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20015, 'CustomerManagement.AddNewCustomer: An unexpected error occurred: ' || SQLERRM);

END AddNewCustomer;

PROCEDURE UpdateCustomerDetails (

p\_customer\_id IN Customers.CustomerID%TYPE,

p\_new\_name IN Customers.Name%TYPE DEFAULT NULL,

p\_new\_dob IN Customers.DOB%TYPE DEFAULT NULL,

p\_new\_balance IN Customers.Balance%TYPE DEFAULT NULL

)

IS

v\_rows\_updated NUMBER;

BEGIN

UPDATE Customers

SET

Name = NVL(p\_new\_name, Name),

DOB = NVL(p\_new\_dob, DOB),

Balance = NVL(p\_new\_balance, Balance),

LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

v\_rows\_updated := SQL%ROWCOUNT;

IF v\_rows\_updated = 0 THEN

RAISE\_APPLICATION\_ERROR(-20016, 'CustomerManagement.UpdateCustomerDetails: Error - Customer ID ' || p\_customer\_id || ' not found.');

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('CustomerManagement.UpdateCustomerDetails: Customer ID ' || p\_customer\_id || ' updated successfully.');

END IF;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20017, 'CustomerManagement.UpdateCustomerDetails: An unexpected error occurred: ' || SQLERRM);

END UpdateCustomerDetails;

FUNCTION GetCustomerBalance (

p\_customer\_id IN Customers.CustomerID%TYPE

) RETURN Customers.Balance%TYPE

IS

v\_balance Customers.Balance%TYPE;

BEGIN

SELECT Balance INTO v\_balance

FROM Customers

WHERE CustomerID = p\_customer\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20018, 'CustomerManagement.GetCustomerBalance: Error - Customer ID ' || p\_customer\_id || ' not found.');

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20019, 'CustomerManagement.GetCustomerBalance: An unexpected error occurred: ' || SQLERRM);

END GetCustomerBalance;

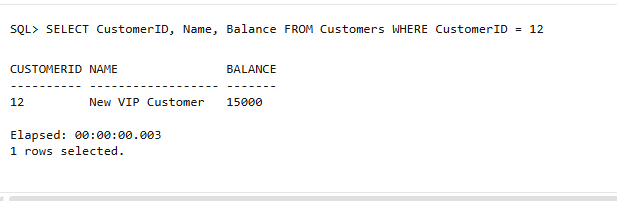
END CustomerManagement;

/

-- Add a new customer

EXEC CustomerManagement.AddNewCustomer(p\_customer\_id => 12, p\_name => 'New VIP Customer', p\_dob => TO\_DATE('1999-04-01', 'YYYY-MM-DD'), p\_balance => 15000);

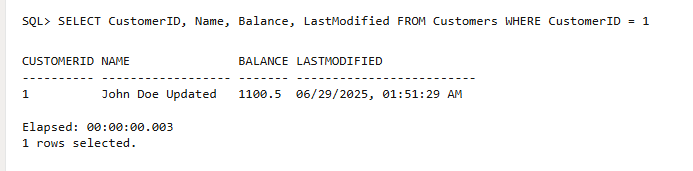
SELECT CustomerID, Name, Balance FROM Customers WHERE CustomerID = 12;



-- Update name and balance for an existing customer

EXEC CustomerManagement.UpdateCustomerDetails(p\_customer\_id => 1, p\_new\_name => 'John Doe Updated', p\_new\_balance => 1100.50);

SELECT CustomerID, Name, Balance, LastModified FROM Customers WHERE CustomerID = 1;



Scenario 2: Create a package to manage employee data.

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

Create Package

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireNewEmployee (

p\_employee\_id IN Employees.EmployeeID%TYPE,

p\_name IN Employees.Name%TYPE,

p\_position IN Employees.Position%TYPE,

p\_salary IN Employees.Salary%TYPE,

p\_department IN Employees.Department%TYPE,

p\_hire\_date IN Employees.HireDate%TYPE

);

PROCEDURE UpdateEmployeeDetails (

p\_employee\_id IN Employees.EmployeeID%TYPE,

p\_new\_position IN Employees.Position%TYPE DEFAULT NULL,

p\_new\_salary IN Employees.Salary%TYPE DEFAULT NULL,

p\_new\_department IN Employees.Department%TYPE DEFAULT NULL

);

FUNCTION CalculateAnnualSalary (

p\_employee\_id IN Employees.EmployeeID%TYPE

) RETURN NUMBER;

END EmployeeManagement;

/

Create Package Body

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireNewEmployee (

p\_employee\_id IN Employees.EmployeeID%TYPE,

p\_name IN Employees.Name%TYPE,

p\_position IN Employees.Position%TYPE,

p\_salary IN Employees.Salary%TYPE,

p\_department IN Employees.Department%TYPE,

p\_hire\_date IN Employees.HireDate%TYPE

)

IS

e\_duplicate\_employee EXCEPTION;

PRAGMA EXCEPTION\_INIT(e\_duplicate\_employee, -00001);

BEGIN

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('EmployeeManagement.HireNewEmployee: New employee ' || p\_name || ' hired with ID: ' || p\_employee\_id);

EXCEPTION

WHEN e\_duplicate\_employee THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20020, 'EmployeeManagement.HireNewEmployee: Error - Employee with ID ' || p\_employee\_id || ' already exists.');

WHEN OTHERS THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20021, 'EmployeeManagement.HireNewEmployee: An unexpected error occurred: ' || SQLERRM);

END HireNewEmployee;

PROCEDURE UpdateEmployeeDetails (

p\_employee\_id IN Employees.EmployeeID%TYPE,

p\_new\_position IN Employees.Position%TYPE DEFAULT NULL,

p\_new\_salary IN Employees.Salary%TYPE DEFAULT NULL,

p\_new\_department IN Employees.Department%TYPE DEFAULT NULL

)

IS

v\_rows\_updated NUMBER;

BEGIN

UPDATE Employees

SET

Position = NVL(p\_new\_position, Position),

Salary = NVL(p\_new\_salary, Salary),

Department = NVL(p\_new\_department, Department)

WHERE EmployeeID = p\_employee\_id;

v\_rows\_updated := SQL%ROWCOUNT;

IF v\_rows\_updated = 0 THEN

RAISE\_APPLICATION\_ERROR(-20022, 'EmployeeManagement.UpdateEmployeeDetails: Error - Employee ID ' || p\_employee\_id || ' not found.');

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('EmployeeManagement.UpdateEmployeeDetails: Employee ID ' || p\_employee\_id || ' updated successfully.');

END IF;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20023, 'EmployeeManagement.UpdateEmployeeDetails: An unexpected error occurred: ' || SQLERRM);

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary (

p\_employee\_id IN Employees.EmployeeID%TYPE

) RETURN NUMBER

IS

v\_salary Employees.Salary%TYPE;

BEGIN

SELECT Salary INTO v\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

RETURN v\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20024, 'EmployeeManagement.CalculateAnnualSalary: Error - Employee ID ' || p\_employee\_id || ' not found.');

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20025, 'EmployeeManagement.CalculateAnnualSalary: An unexpected error occurred: ' || SQLERRM);

END CalculateAnnualSalary;

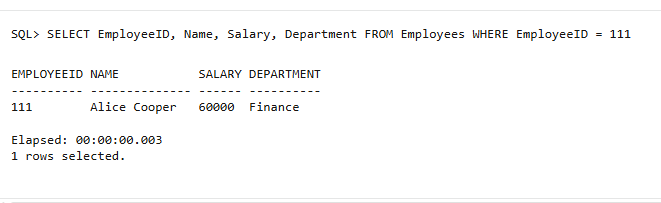
END EmployeeManagement;

/

-- Hire a new employee

EXEC EmployeeManagement.HireNewEmployee(p\_employee\_id => 111, p\_name => 'Alice Cooper', p\_position => 'Analyst', p\_salary => 60000, p\_department => 'Finance', p\_hire\_date => TO\_DATE('2024-01-15', 'YYYY-MM-DD'));

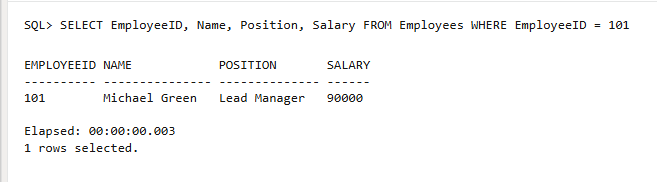
SELECT EmployeeID, Name, Salary, Department FROM Employees WHERE EmployeeID = 111;



-- Update position and salary for an existing employee

EXEC EmployeeManagement.UpdateEmployeeDetails(p\_employee\_id => 101, p\_new\_position => 'Lead Manager', p\_new\_salary => 90000);

SELECT EmployeeID, Name, Position, Salary FROM Employees WHERE EmployeeID = 101;



-- Get annual salary for an existing employee

DECLARE

v\_annual\_salary NUMBER;

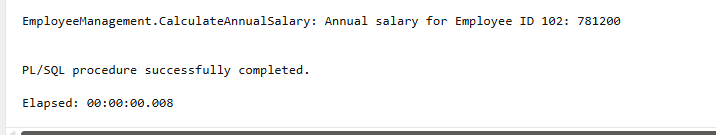
BEGIN

v\_annual\_salary := EmployeeManagement.CalculateAnnualSalary(p\_employee\_id => 102);

DBMS\_OUTPUT.PUT\_LINE('EmployeeManagement.CalculateAnnualSalary: Annual salary for Employee ID 102: ' || v\_annual\_salary);

END;

/



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

o 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.

Create Package

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenNewAccount (

p\_account\_id IN Accounts.AccountID%TYPE,

p\_customer\_id IN Accounts.CustomerID%TYPE,

p\_account\_type IN Accounts.AccountType%TYPE,

p\_initial\_balance IN Accounts.Balance%TYPE

);

PROCEDURE CloseAccount (

p\_account\_id IN Accounts.AccountID%TYPE

);

FUNCTION GetTotalCustomerBalance (

p\_customer\_id IN Customers.CustomerID%TYPE

) RETURN NUMBER;

END AccountOperations;

/

Create Package Body

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

-- Procedure to open a new account

PROCEDURE OpenNewAccount (

p\_account\_id IN Accounts.AccountID%TYPE,

p\_customer\_id IN Accounts.CustomerID%TYPE,

p\_account\_type IN Accounts.AccountType%TYPE,

p\_initial\_balance IN Accounts.Balance%TYPE

)

IS

e\_duplicate\_account EXCEPTION;

PRAGMA EXCEPTION\_INIT(e\_duplicate\_account, -00001);

v\_customer\_exists NUMBER;

BEGIN

-- Check if CustomerID exists

SELECT COUNT(\*) INTO v\_customer\_exists FROM Customers WHERE CustomerID = p\_customer\_id;

IF v\_customer\_exists = 0 THEN

RAISE\_APPLICATION\_ERROR(-20026, 'AccountOperations.OpenNewAccount: Error - Customer ID ' || p\_customer\_id || ' does not exist.');

END IF;

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

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('AccountOperations.OpenNewAccount: New ' || p\_account\_type || ' account ' || p\_account\_id || ' opened for Customer ID: ' || p\_customer\_id);

EXCEPTION

WHEN e\_duplicate\_account THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20027, 'AccountOperations.OpenNewAccount: Error - Account ID ' || p\_account\_id || ' already exists.');

WHEN OTHERS THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20028, 'AccountOperations.OpenNewAccount: An unexpected error occurred: ' || SQLERRM);

END OpenNewAccount;

PROCEDURE CloseAccount (

p\_account\_id IN Accounts.AccountID%TYPE

)

IS

v\_rows\_updated NUMBER;

BEGIN

UPDATE Accounts

SET Balance = 0,

AccountType = 'Closed',

LastModified = SYSDATE

WHERE AccountID = p\_account\_id;

v\_rows\_updated := SQL%ROWCOUNT;

IF v\_rows\_updated = 0 THEN

RAISE\_APPLICATION\_ERROR(-20029, 'AccountOperations.CloseAccount: Error - Account ID ' || p\_account\_id || ' not found.');

ELSE

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('AccountOperations.CloseAccount: Account ID ' || p\_account\_id || ' closed successfully. Balance set to 0 and type to Closed.');

END IF;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20030, 'AccountOperations.CloseAccount: An unexpected error occurred: ' || SQLERRM);

END CloseAccount;

FUNCTION GetTotalCustomerBalance (

p\_customer\_id IN Customers.CustomerID%TYPE

) RETURN NUMBER

IS

v\_total\_balance NUMBER := 0;

v\_customer\_exists NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_customer\_exists FROM Customers WHERE CustomerID = p\_customer\_id;

IF v\_customer\_exists = 0 THEN

RAISE\_APPLICATION\_ERROR(-20031, 'AccountOperations.GetTotalCustomerBalance: Error - Customer ID ' || p\_customer\_id || ' does not exist.');

END IF;

SELECT NVL(SUM(Balance), 0) INTO v\_total\_balance

FROM Accounts

WHERE CustomerID = p\_customer\_id;

RETURN v\_total\_balance;

EXCEPTION

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20032, 'AccountOperations.GetTotalCustomerBalance: An unexpected error occurred: ' || SQLERRM);

END GetTotalCustomerBalance;

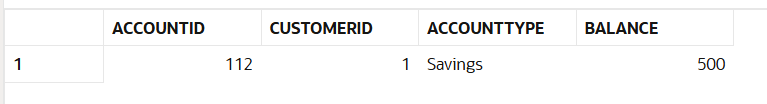
END AccountOperations;

/

-- Open a new savings account for Customer ID 1

EXEC AccountOperations.OpenNewAccount(p\_account\_id => 112, p\_customer\_id => 1, p\_account\_type => 'Savings', p\_initial\_balance => 500);

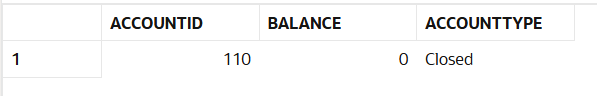
SELECT AccountID, CustomerID, AccountType, Balance FROM Accounts WHERE AccountID = 112;



-- Close an existing account (e.g., Account ID 110 from your inserted data)

EXEC AccountOperations.CloseAccount(p\_account\_id => 110);

SELECT AccountID, Balance, AccountType FROM Accounts WHERE AccountID = 110;



-- Get total balance for Customer ID 1

DECLARE

v\_total\_bal NUMBER;

BEGIN

v\_total\_bal := AccountOperations.GetTotalCustomerBalance(p\_customer\_id => 1);

DBMS\_OUTPUT.PUT\_LINE('AccountOperations.GetTotalCustomerBalance: Total balance for Customer ID 1: ' || v\_total\_bal);

END;

/

