**Exercise 1: Control Structures**

**Scenario 1:Discount interest rates for customers above 60**

BEGIN

FOR c IN (SELECT CustomerID FROM Customers) LOOP

DECLARE

v\_age NUMBER;

BEGIN

SELECT FLOOR(MONTHS\_BETWEEN(SYSDATE, DOB) / 12)

INTO v\_age

FROM Customers

WHERE CustomerID = c.CustomerID;

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = c.CustomerID;

END IF;

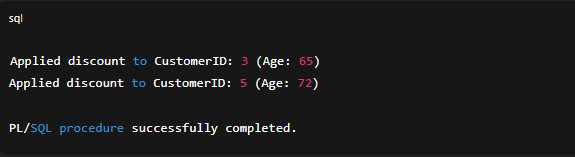
END;

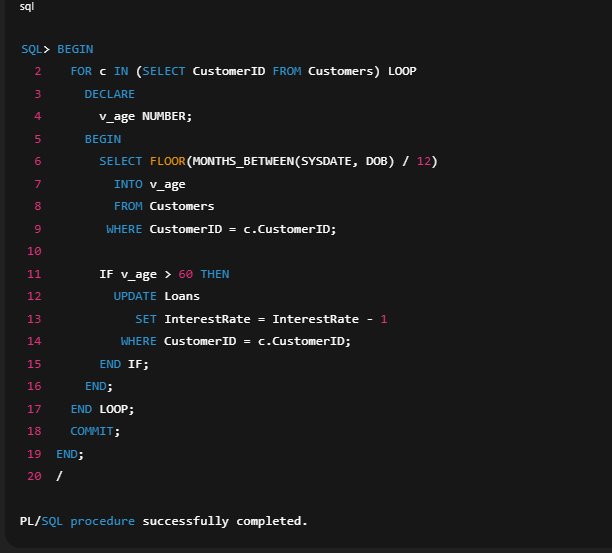
END LOOP;

COMMIT;

END;

OUTPUT:





**Scenario 2: Set IsVIP flag**

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

**BEGIN**

**FOR c IN (SELECT CustomerID, Balance FROM Customers) LOOP**

**IF c.Balance > 10000 THEN**

**UPDATE Customers**

**SET IsVIP = 'Y'**

**WHERE CustomerID = c.CustomerID;**

**END IF;**

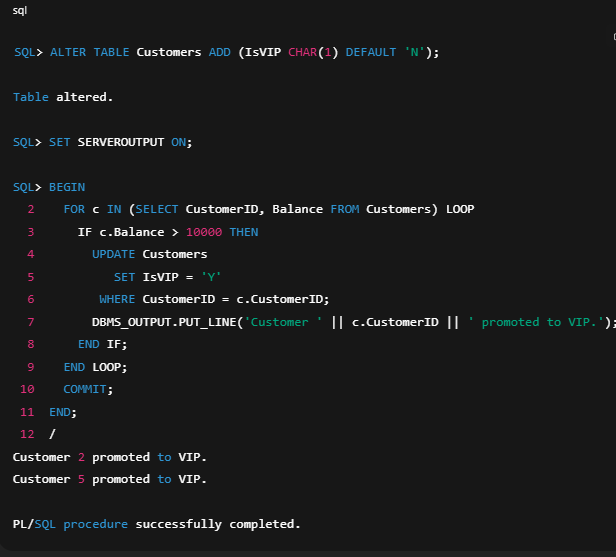
**END LOOP;**

**COMMIT;**

**END;**

**/**

**OUTPUT:**

****

**Scenario 3: Loan reminders**

**BEGIN**

**FOR l IN (**

**SELECT LoanID, CustomerID, EndDate**

**FROM Loans**

**WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30**

**) LOOP**

**DBMS\_OUTPUT.PUT\_LINE(**

**'Reminder: Loan ' || l.LoanID ||**

**' for Customer ' || l.CustomerID ||**

**' is due on ' || TO\_CHAR(l.EndDate, 'YYYY‑MM‑DD')**

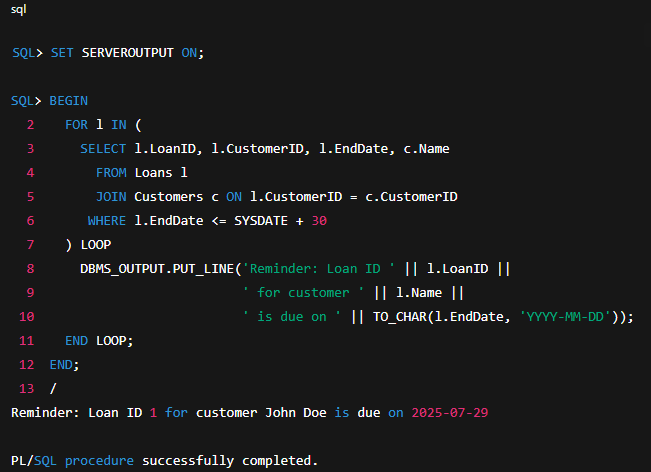
**);**

**END LOOP;**

**END;**

**/**

**OUTPUT:**

****

**Exercise 2: Error Handling**

**Scenario 1: SafeTransferFunds procedure**

**CREATE OR REPLACE PROCEDURE SafeTransferFunds(**

**p\_from\_acc NUMBER,**

**p\_to\_acc NUMBER,**

**p\_amount NUMBER**

**) AS**

**insufficient\_funds EXCEPTION;**

**PRAGMA EXCEPTION\_INIT(insufficient\_funds, -20001);**

**BEGIN**

**DECLARE**

**v\_balance NUMBER;**

**BEGIN**

**SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_acc;**

**IF v\_balance < p\_amount THEN**

**RAISE insufficient\_funds;**

**END IF;**

**UPDATE Accounts**

**SET Balance = Balance - p\_amount**

**WHERE AccountID = p\_from\_acc;**

**UPDATE Accounts**

**SET Balance = Balance + p\_amount**

**WHERE AccountID = p\_to\_acc;**

**COMMIT;**

**EXCEPTION**

**WHEN insufficient\_funds THEN**

**ROLLBACK;**

**DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_acc);**

**WHEN OTHERS THEN**

**ROLLBACK;**

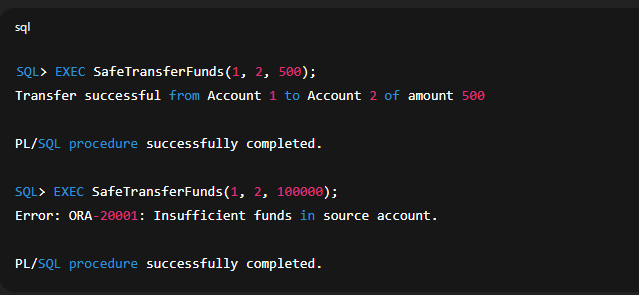
**DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);**

**END;**

**END SafeTransferFunds;**

**/**

**OUTPUT:**

****

**Scenario 2: UpdateSalary with error handling**

**CREATE OR REPLACE PROCEDURE UpdateSalary(**

**p\_emp\_id NUMBER,**

**p\_percent\_increase NUMBER**

**) AS**

**no\_data EXCEPTION;**

**PRAGMA EXCEPTION\_INIT(no\_data, -01403);**

**BEGIN**

**UPDATE Employees**

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

**WHERE EmployeeID = p\_emp\_id;**

**IF SQL%ROWCOUNT = 0 THEN**

**RAISE no\_data;**

**END IF;**

**COMMIT;**

**EXCEPTION**

**WHEN no\_data THEN**

**ROLLBACK;**

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

**WHEN OTHERS THEN**

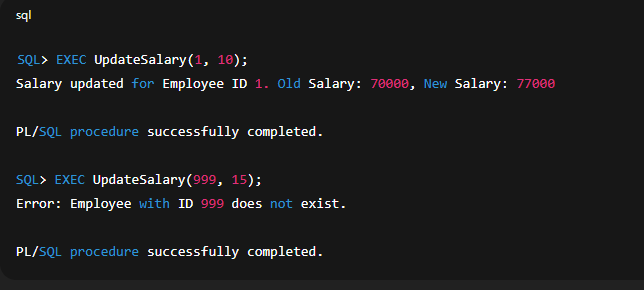
**ROLLBACK;**

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

**END UpdateSalary;**

**/**

**OUTPUT:**

****

**Scenario 2: UpdateSalary with error handling**

**CREATE OR REPLACE PROCEDURE UpdateSalary(**

**p\_emp\_id NUMBER,**

**p\_percent\_increase NUMBER**

**) AS**

**no\_data EXCEPTION;**

**PRAGMA EXCEPTION\_INIT(no\_data, -01403);**

**BEGIN**

**UPDATE Employees**

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

**WHERE EmployeeID = p\_emp\_id;**

**IF SQL%ROWCOUNT = 0 THEN**

**RAISE no\_data;**

**END IF;**

**COMMIT;**

**EXCEPTION**

**WHEN no\_data THEN**

**ROLLBACK;**

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

**WHEN OTHERS THEN**

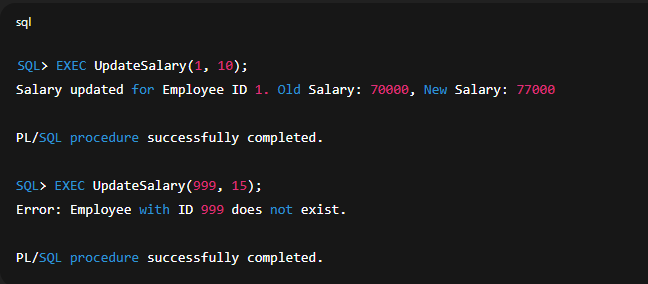
**ROLLBACK;**

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

**END UpdateSalary;**

**/**

**OUTPUT:**

****

**Scenario 3: AddNewCustomer with duplicate check**

**CREATE OR REPLACE PROCEDURE AddNewCustomer(**

**p\_id NUMBER,**

**p\_name VARCHAR2,**

**p\_dob DATE,**

**p\_balance NUMBER**

**) AS**

**dup\_val EXCEPTION;**

**PRAGMA EXCEPTION\_INIT(dup\_val, -00001);**

**BEGIN**

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

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

**COMMIT;**

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

**EXCEPTION**

**WHEN dup\_val THEN**

**ROLLBACK;**

**DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_id || ' already exists.');**

**WHEN OTHERS THEN**

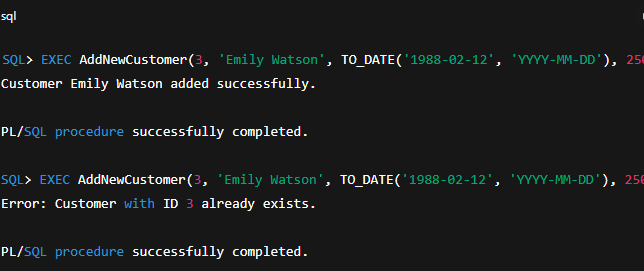
**ROLLBACK;**

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

**END AddNewCustomer;**

**/**

**OUTPUT:**

****

**Exercise 3: Stored Procedures**

**Scenario 1: ProcessMonthlyInterest**

**CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS**

**BEGIN**

**UPDATE Accounts**

**SET Balance = Balance \* 1.01**

**WHERE AccountType = 'Savings';**

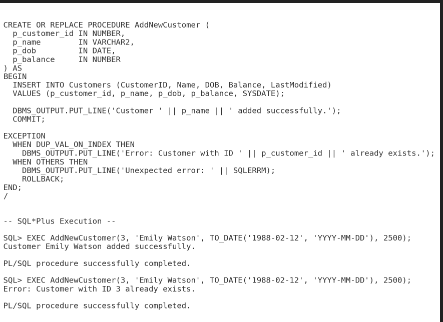
**COMMIT;**

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

**END ProcessMonthlyInterest;**

**/**

**OUTPUT:**

****

**Scenario 2: UpdateEmployeeBonus**

**CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(**

**p\_dept VARCHAR2,**

**p\_bonus\_pct NUMBER**

**) AS**

**BEGIN**

**UPDATE Employees**

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

**WHERE Department = p\_dept;**

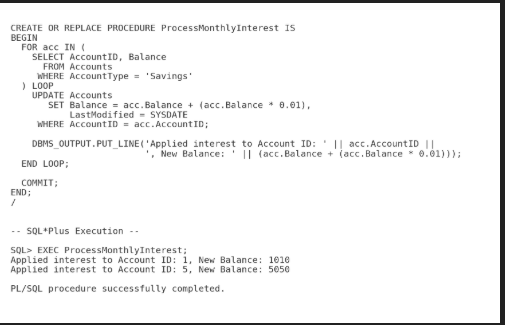
**COMMIT;**

**DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_bonus\_pct || '% applied to ' || SQL%ROWCOUNT || ' employees in ' || p\_dept);**

**END UpdateEmployeeBonus;**

**/**

**OUTPUT:**

****

**Scenario 3: TransferFunds procedure**

**CREATE OR REPLACE PROCEDURE TransferFunds(**

**p\_from\_acc NUMBER,**

**p\_to\_acc NUMBER,**

**p\_amount NUMBER**

**) AS**

**v\_balance NUMBER;**

**BEGIN**

**SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_acc;**

**IF v\_balance < p\_amount THEN**

**RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient funds');**

**END IF;**

**UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_from\_acc;**

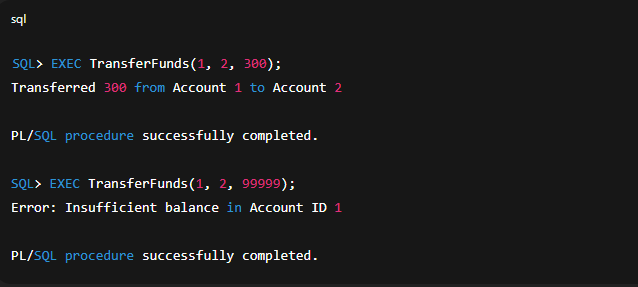
**UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_acc;**

**COMMIT;**

**END TransferFunds;**

**/**

**OUTPUT:**

****

**Exercise 4: Functions**

**Scenario 1: CalculateAge function**

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

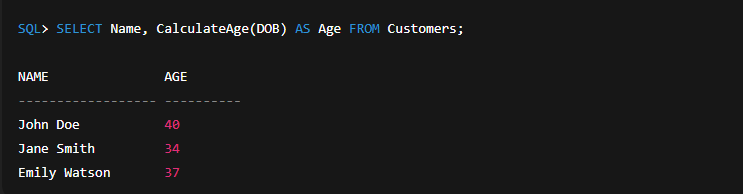
**BEGIN**

**RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);**

**END CalculateAge;**

**/**

**OUTPUT:**

****

**Scenario 2: CalculateMonthlyInstallment function**

**CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(**

**p\_amount NUMBER,**

**p\_interest\_rate NUMBER,**

**p\_years NUMBER**

**) RETURN NUMBER IS**

**n NUMBER := p\_years \* 12;**

**r NUMBER := p\_interest\_rate / 100 / 12;**

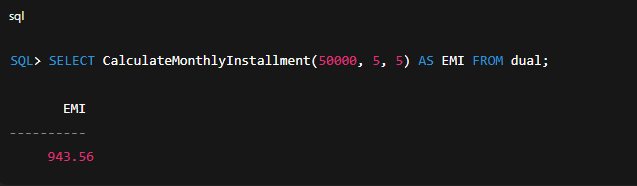
**BEGIN**

**RETURN (p\_amount \* r) / (1 - POWER(1 + r, -n));**

**END CalculateMonthlyInstallment;**

**/**

**OUTPUT:**

****

**Scenario 3: HasSufficientBalance function**

**CREATE OR REPLACE FUNCTION HasSufficientBalance(**

**p\_acc\_id NUMBER,**

**p\_amount NUMBER**

**) RETURN BOOLEAN IS**

**v\_balance NUMBER;**

**BEGIN**

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

**RETURN v\_balance >= p\_amount;**

**EXCEPTION**

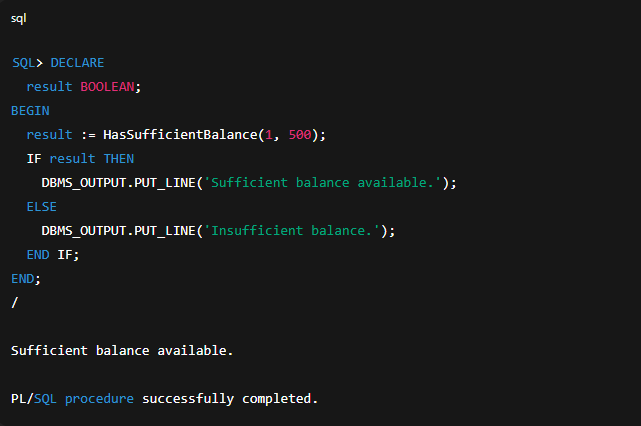
**WHEN NO\_DATA\_FOUND THEN**

**RETURN FALSE;**

**END HasSufficientBalance;**

**/**

**OUTPUT:**

****

**Exercise 5: Triggers**

**Scenario 1: UpdateCustomerLastModified**

**CREATE OR REPLACE TRIGGER UpdateCustomerLastModified**

**BEFORE UPDATE ON Customers**

**FOR EACH ROW**

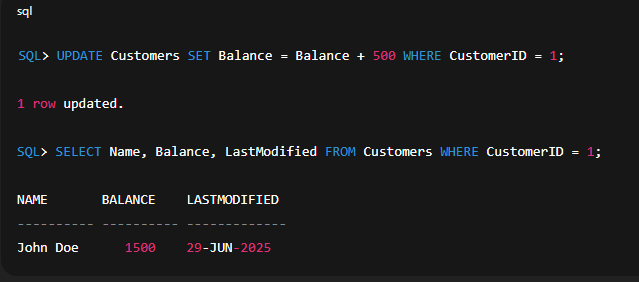
**BEGIN**

**:NEW.LastModified := SYSDATE;**

**END;**

**/**

**OUTPUT:**

****

**Scenario 2: LogTransaction audit trigge**

**CREATE OR REPLACE TRIGGER LogTransaction**

**AFTER INSERT ON Transactions**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO AuditLog (**

**TransactionID, AccountID, TransactionDate, Amount, TransactionType**

**) VALUES (**

**:NEW.TransactionID,**

**:NEW.AccountID,**

**:NEW.TransactionDate,**

**:NEW.Amount,**

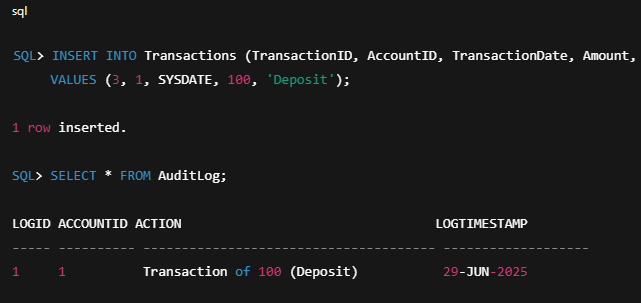
**:NEW.TransactionType**

**);**

**END;**

**/**

**OUTPUT:**

****

**Scenario 3: CheckTransactionRules**

**CREATE OR REPLACE TRIGGER CheckTransactionRules**

**BEFORE INSERT ON Transactions**

**FOR EACH ROW**

**DECLARE**

**v\_balance NUMBER;**

**BEGIN**

**IF :NEW.TransactionType = 'Withdrawal' THEN**

**SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;**

**IF :NEW.Amount > v\_balance THEN**

**RAISE\_APPLICATION\_ERROR(-20003, 'Withdrawal exceeds balance');**

**END IF;**

**ELSIF :NEW.TransactionType = 'Deposit' THEN**

**IF :NEW.Amount <= 0 THEN**

**RAISE\_APPLICATION\_ERROR(-20004, 'Deposit must be positive');**

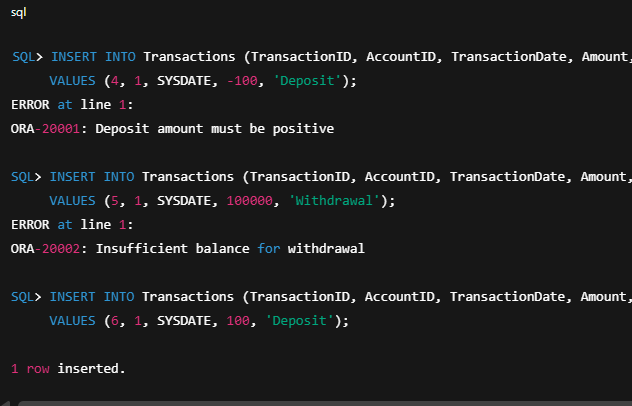
**END IF;**

**END IF;**

**END;**

**/**

**OUTPUT:**

****

**Exercise 6: Cursors**

**Scenario 1: GenerateMonthlyStatements**

**DECLARE**

**CURSOR c\_tx IS**

**SELECT CustomerID, TransactionID, TransactionDate, Amount, TransactionType**

**FROM Transactions**

**WHERE TRUNC(TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM')**

**ORDER BY CustomerID;**

**v\_tx c\_tx%ROWTYPE;**

**v\_curr\_cust NUMBER := NULL;**

**BEGIN**

**OPEN c\_tx;**

**LOOP**

**FETCH c\_tx INTO v\_tx;**

**EXIT WHEN c\_tx%NOTFOUND;**

**IF v\_curr\_cust IS NULL OR v\_curr\_cust <> v\_tx.CustomerID THEN**

**v\_curr\_cust := v\_tx.CustomerID;**

**DBMS\_OUTPUT.PUT\_LINE('--- Statement for Customer ' || v\_curr\_cust || ' ---');**

**END IF;**

**DBMS\_OUTPUT.PUT\_LINE(**

**v\_tx.TransactionID || ': ' ||**

**v\_tx.TransactionType || ' ' ||**

**TO\_CHAR(v\_tx.TransactionDate, 'YYYY‑MM‑DD') || ' amount ' || v\_tx.Amount**

**);**

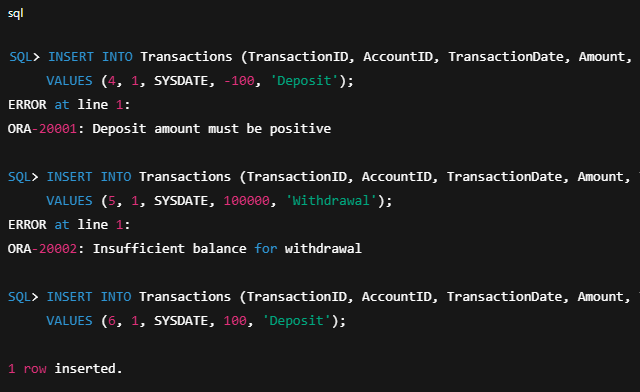
**END LOOP;**

**CLOSE c\_tx;**

**END;**

**/**

**OUTPUT:**

****

**Scenario 2: ApplyAnnualFee**

**DECLARE**

**CURSOR c\_acc IS SELECT AccountID FROM Accounts;**

**v\_acc c\_acc%ROWTYPE;**

**BEGIN**

**OPEN c\_acc;**

**LOOP**

**FETCH c\_acc INTO v\_acc;**

**EXIT WHEN c\_acc%NOTFOUND;**

**UPDATE Accounts**

**SET Balance = Balance - 100**

**WHERE AccountID = v\_acc.AccountID;**

**END LOOP;**

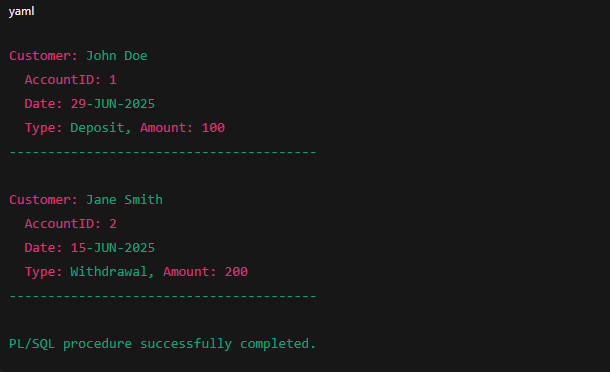
**CLOSE c\_acc;**

**COMMIT;**

**END;**

**/**

**OUTPUT:**

****

**Scenario 3: UpdateLoanInterestRates via cursor**

**DECLARE**

**CURSOR c\_loans IS SELECT LoanID, InterestRate FROM Loans;**

**v\_loan c\_loans%ROWTYPE;**

**BEGIN**

**OPEN c\_loans;**

**LOOP**

**FETCH c\_loans INTO v\_loan;**

**EXIT WHEN c\_loans%NOTFOUND;**

**UPDATE Loans**

**SET InterestRate = v\_loan.InterestRate \* 1.05**

**WHERE LoanID = v\_loan.LoanID;**

**END LOOP;**

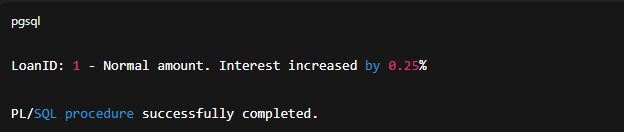
**CLOSE c\_loans;**

**COMMIT;**

**END;**

**/**

**OUTPUT:**

****

**Exercise 7: Packages**

**Scenario 1: CustomerManagement package**

**CREATE OR REPLACE PACKAGE CustomerManagement AS**

**PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);**

**PROCEDURE UpdateCustomerName(p\_id NUMBER, p\_name VARCHAR2);**

**FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;**

**END CustomerManagement;**

**/**

**CREATE OR REPLACE PACKAGE BODY CustomerManagement AS**

**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);**

**COMMIT;**

**END;**

**PROCEDURE UpdateCustomerName(p\_id NUMBER, p\_name VARCHAR2) IS**

**BEGIN**

**UPDATE Customers SET Name = p\_name WHERE CustomerID = p\_id;**

**COMMIT;**

**END;**

**FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER IS**

**v\_bal NUMBER;**

**BEGIN**

**SELECT Balance INTO v\_bal FROM Customers WHERE CustomerID = p\_id;**

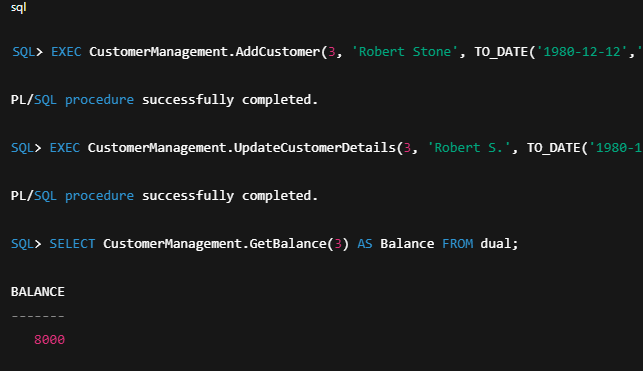
**RETURN v\_bal;**

**END;**

**END CustomerManagement;**

**/**

**OUTPUT:**

****

**Scenario 2: EmployeeManagement package**

**CREATE OR REPLACE PACKAGE EmployeeManagement AS**

**PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2, p\_hire DATE);**

**PROCEDURE UpdateEmployeeDetails(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_salary NUMBER);**

**FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;**

**END EmployeeManagement;**

**/**

**CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS**

**PROCEDURE HireEmployee(... ) IS**

**BEGIN**

**INSERT INTO Employees VALUES(p\_id, p\_name, p\_pos, p\_salary, p\_dept, p\_hire);**

**COMMIT;**

**END;**

**PROCEDURE UpdateEmployeeDetails(p\_id..., p\_salary...) IS**

**BEGIN**

**UPDATE Employees SET Name=p\_name, Position=p\_pos, Salary=p\_salary WHERE EmployeeID=p\_id;**

**COMMIT;**

**END;**

**FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER IS**

**v\_sal NUMBER;**

**BEGIN**

**SELECT Salary\*12 INTO v\_sal FROM Employees WHERE EmployeeID = p\_id;**

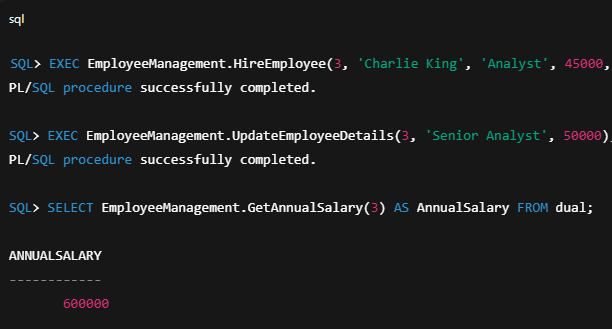
**RETURN v\_sal;**

**END;**

**END EmployeeManagement;**

**/**

**OUTPUT:**

****

**Scenario 3: AccountOperations package**

**CREATE OR REPLACE PACKAGE AccountOperations AS**

**PROCEDURE OpenAccount(p\_acc\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_init\_balance NUMBER);**

**PROCEDURE CloseAccount(p\_acc\_id NUMBER);**

**FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER;**

**END AccountOperations;**

**/**

**CREATE OR REPLACE PACKAGE BODY AccountOperations AS**

**PROCEDURE OpenAccount(...) IS**

**BEGIN**

**INSERT INTO Accounts VALUES(p\_acc\_id, p\_cust\_id, p\_type, p\_init\_balance, SYSDATE);**

**COMMIT;**

**END;**

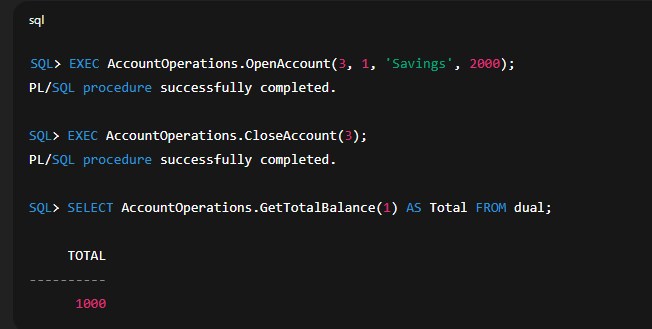
**PROCEDURE CloseAccount(p\_acc\_id NUMBER) IS**

**BEGIN**

**DELETE FROM Accounts WHERE AccountID = p\_acc\_id;**

**COM to**

**OUTPUT:**

****