**Excecise1;**

**Scenario 1**

begin

    for cus in(SELECT CUSTOMERID,DOB from CUSTOMERS )LOOP

        IF Months\_Between(SYSDATE,cus.DOB)/12< 60 THEN

            UPDATE LOANS

             Set INTERESTRATE=INTERESTRATE-1

            WHERE CUSTOMERID = cus.CUSTOMERID;

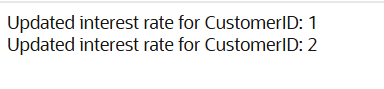
            DBMS\_OUTPUT.PUT\_LINE('Updated interest rate for CustomerID: ' || cus.CustomerID);

        end if;

     end LOOP;

end;

/

**Output**

**Scenario 2**

alter table Customers add IsVIP VARCHAR2(5);

BEGIN

    for cus in(SELECT BALANCE,CUSTOMERID from CUSTOMERS)LOOP

    if cus.BALANCE>10000 THEN

    UPDATE  CUSTOMERS set ISVIP="TRUE"

    where cus.CUSTOMERID=CUSTOMERID;

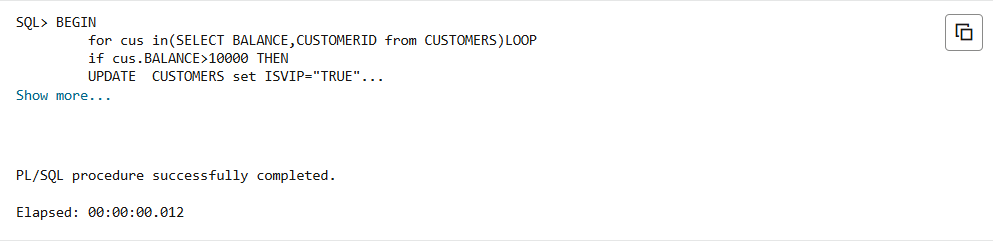
    end if;

    end loop;

    end;

    /

**OUTPUT**



**Scenario 3**

declare

var\_name varchar(50);

BEGIN

    for loan in(select LOANID,CUSTOMERID,ENDDATE from LOANS where ENDDATE between sysdate and sysdate+30)LOOP

    Select NAME into var\_name from CUSTOMERS where CUSTOMERID=loan.CUSTOMERID;

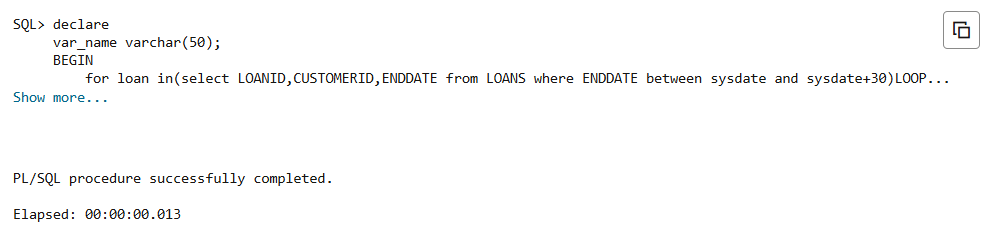
    DBMS\_OUTPUT.PUT\_LINE(loan.LOANID || chr(10) ||var\_name||chr(10)|| loan.ENDDATE);

    END loop;

    END;

    /

**OUTPUT**

****

**EXERCISE 2**

**Scenario 1**

create or replace PROCEDURE transfermoney(

    person\_from\_accountid in NUMBER,

    person\_to\_accountid in number,

    amount in NUMBER

)as v\_balance Number ;

begin

    SELECT Balance into v\_balance from ACCOUNTS

    where ACCOUNTID=person\_from\_accountid ;

    if v\_balance<amount then

        Raise\_application\_error(-20001,'Insufficient funds');

    end if;

    update ACCOUNTS set

    BALANCE =balance-amount

    where ACCOUNTID=person\_from\_accountid;

    update ACCOUNTS set

    BALANCE=BALANCE+amount

    where ACCOUNTID=person\_to\_accountid;

    COMMIT;

EXCEPTION

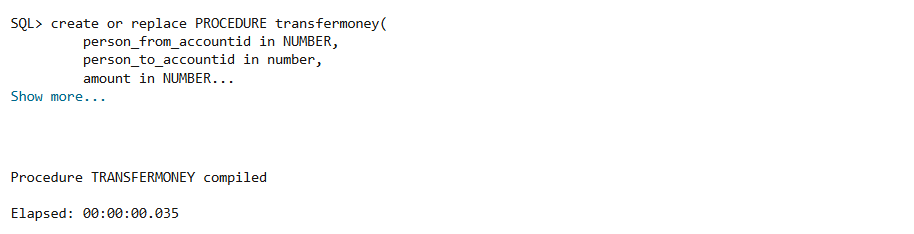
    when others THEN

    dbms\_output.PUT\_LINE('Transation failed'|| sqlerrm);

end;

/

**OUTPUT**

****

**Scenario 2**

CREATE OR REPLACE UpdateSalery(

    person\_emloyeeid in number,

    percentage in number

)as

begin

    update employees set

    salary=salary+(salary\*percentage/100)

    where EmployeeId=person\_emloyeeid;

    if sql%rowcount=0 then

      dbms\_output.put\_line('no employee found');

    end if;

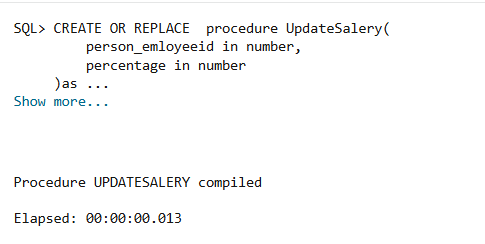
exception

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

**OUTPUT**

**Scenario 3**

create or replace procedure addnewcustomer(

    person\_customerid in number,

    person\_name in VARCHAR,

    person\_dob in date,

    person\_balance in number

)as

begin

    INSERT into CUSTOMERS(CUSTOMERID,NAME,DOB,BALANCE,LASTMODIFIED)

    VALUES(person\_customerid,person\_name,person\_dob,person\_balance,sysdate);

EXCEPTION

when dup\_val\_on\_index THEN

dbms\_output.put\_line('customer id already exists');

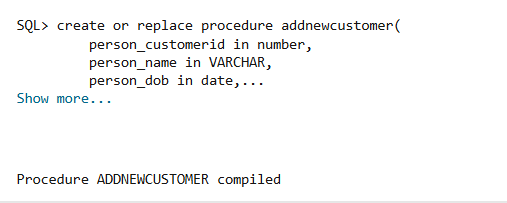
when others then

dbms\_output.PUT\_LINE(sqlerrm);

end;

/

**OUTPUT**

****

**EXERCISE 3**

**SCENARIO 1**

create or replace procedure proceessMonthlyintrest as updated\_rows number:=0;

begin update ACCOUNTS set

BALANCE=BALANCE\*1.01

where ACCOUNTTYPE='Savings';

updated\_rows:=sql%rowcount;

dbms\_output.PUT\_LINE(updated\_rows ||'have been updated');

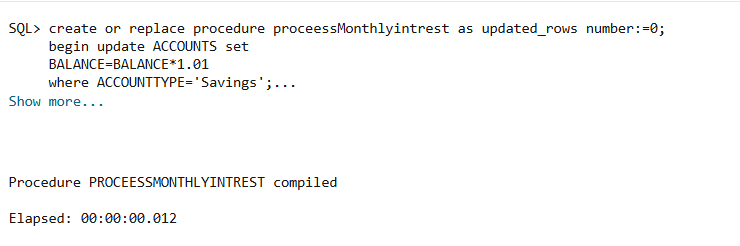
EXCEPTION

when others then

    dbms\_output.put\_line(sqlerrm);

end;

/

**OUTPUT:**

**SCENARIO 2**

create or replace PROCEDURE updateEnployeeBonus(

    department in VARCHAR,

    bonus in NUMBER

)as NO\_ROWS NUMBER :=0;

BEGIN

    update EMPLOYEES set

    SALARY=SALARY+(salary\*bonus/100)

    where DEPARTMENT=department;

    NO\_ROWS:=sql%rowcount;

    if NO\_ROWS =0 THEN

    dbms\_output.put\_line('no records found');

    ELSE

    dbms\_output.PUT\_LINE(No\_ROWS || ' have been updated with a bonus of'||bonus);

    end if;

EXCEPTION

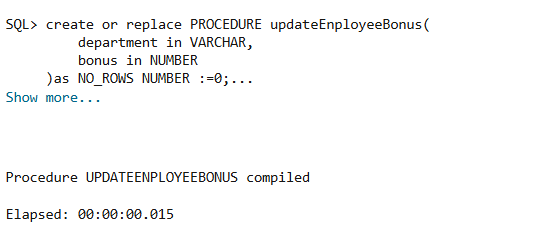
    when others THEN

    dbms\_output.put\_line(sqlerrm);

end;

/

**OUTPUT**



**SCENARIO 3**

create or replace PROCEDURE TransferFunds (

    person\_from\_accountid in NUMBER,

    person\_to\_accountid in number,

    amount in NUMBER

)as v\_balance Number ;

begin

    SELECT Balance into v\_balance from ACCOUNTS

    where ACCOUNTID=person\_from\_accountid ;

    if v\_balance<amount then

        Raise\_application\_error(-20001,'Insufficient funds');

    end if;

    update ACCOUNTS set

    BALANCE =balance-amount

    where ACCOUNTID=person\_from\_accountid;

    update ACCOUNTS set

    BALANCE=BALANCE+amount

    where ACCOUNTID=person\_to\_accountid;

    COMMIT;

EXCEPTION

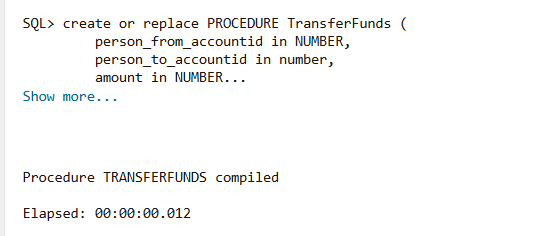
    when others THEN

    dbms\_output.PUT\_LINE('Transation failed'|| sqlerrm);

end;

/

**OUTPUT**

****

**EXERCISE 4**

**SCENARIO 1**

create or replace FUNCTION calculateage(dob in DATE) return NUMBER is

begin

    RETURN floor(MONTHS\_BETWEEN(sysdate,dob)/12);

end;

/

DECLARE

age number;

BEGIN

    age:=CALCULATEAGE(TO\_DATE('2005-03-28','YYYY-MM-DD'));

    dbms\_output.put\_line('age  ' ||age);

end;

/

**OUTPUT**

**SCENARIO 2**

create or replace function cacluatemmonthlyinstallments(

    amount in number,

    rate in number,

    years in number

) RETURN number is

rate\_monthly number:=rate /(12\*100);

total\_months number :=years\*12;

begin

    RETURN(amount\*rate\_monthly\*power(1+rate\_monthly,total\_months)/power(1+rate\_monthly,total\_months)-1);

end;

/

DECLARE

 emi number;

begin

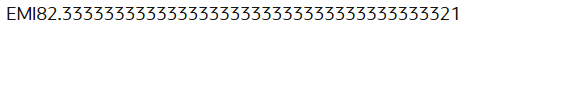
    emi:=CACLUATEMMONTHLYINSTALLMENTS(10000,10,3);

    dbms\_output.put\_line('EMI'  || emi);

end;

/

**OUTPUT**

**SCENARIO 3**

create or replace function has\_sufficient\_balance(

    person\_accountid in number,

    person\_amount in number

)RETURN boolean is T\_balance number;

begin

    SELECT Balance into T\_balance from ACCOUNTS where ACCOUNTID=person\_accountid;

    return T\_balance >person\_amount;

EXCEPTION

    when no\_data\_found then

    return false;

end;

/

declare

  a BOOLEAN;

BEGIN

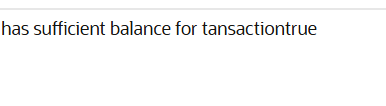
    a:=HAS\_SUFFICIENT\_BALANCE(1,100);

    dbms\_output.put\_line('has sufficient balance for tansaction '|| case when a then chr(10)|| ' true ' else chr(10)|| 'false' end );

end;

/

**OUTPUT**



**EXERCISE 5**

**SCENARIO 1**

create or replace trigger updatecustomerlastmodified before update on customers

for each row

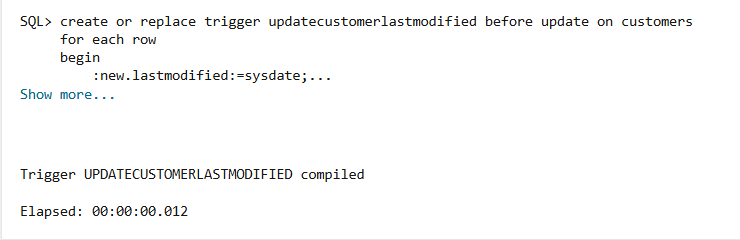
begin

    :new.lastmodified:=sysdate;

end ;

/

**OUTPUT**

****

**SCENARIO 2**

**AUDIT TABLE**

CREATE TABLE AuditLog (

  LogID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,

  TransactionID NUMBER,

  AccountID NUMBER,

  ActionType VARCHAR2(20),

  LogDate DATE

);

create or replace TRIGGER logtransaction after insert on transactions

for each ROW

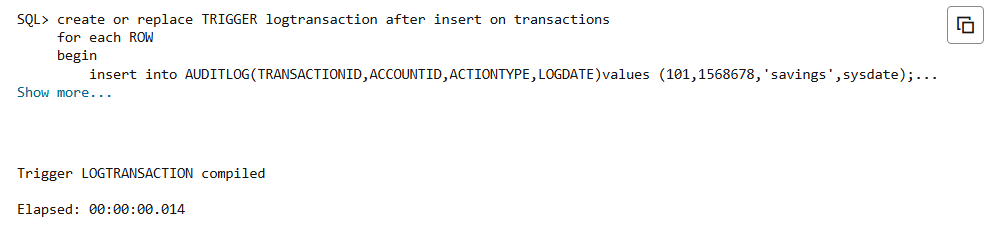
begin

    insert into AUDITLOG(TRANSACTIONID,ACCOUNTID,ACTIONTYPE,LOGDATE)values (101,1568678,'savings',sysdate);

end;

/

**OUTPUT**

****

**SCENARIO 3**

create or replace trigger checktansactions before insert on Transactions for each row

DECLARE

account\_balance number;

BEGIN

    SELECT balance into account\_balance FROM accounts

    WHERE ACCOUNTID = :new.ACCOUNTID

    for update;

    if :new.transactiontype ='withdrawal' and :new.amount>account\_balance then

    dbms\_output.PUT\_LINE('Insufficient balance');

    elsif :new.transactiontype ='deposit' and :new.amount<=0 THEN

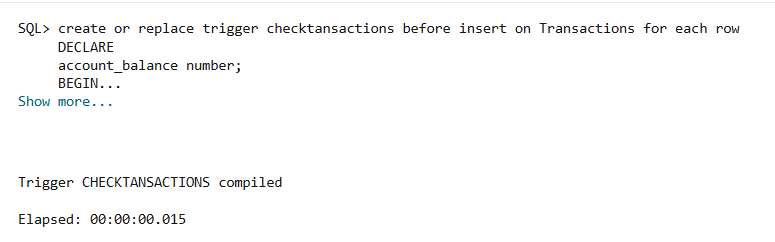
    dbms\_output.PUT\_LINE('amount must be in positive');

    end if;

    end;

    /

**OUTPUT**

**EXERCISE 6**

**SCENARIO 1**

Declare

  cursor transcation\_cursor is

    select a.CustomerID,

           t.TransactionID,

           t.TransactionDate,

           t.Amount,

           t.TransactionType

      from TRANSACTIONS t

      join ACCOUNTS a

        on t.ACCOUNTID = a.ACCOUNTID

     where trunc(t.TRANSACTIONDATE,'MM') = trunc(sysdate,'MM');

  transaction\_record transcation\_cursor%ROWTYPE;

BEGIN

  open transcation\_cursor;

  loop

    fetch transcation\_cursor into transaction\_record;

    exit when transcation\_cursor%notfound;

    DBMS\_OUTPUT.PUT\_LINE(

      'CUSTOMER ID'|| transaction\_record.CustomerID ||

      ' transcation id'|| transaction\_record.TransactionID ||

      ' date'|| transaction\_record.TransactionDate ||

      ' type '|| transaction\_record.TransactionType ||

      ' amount'||transaction\_record.Amount

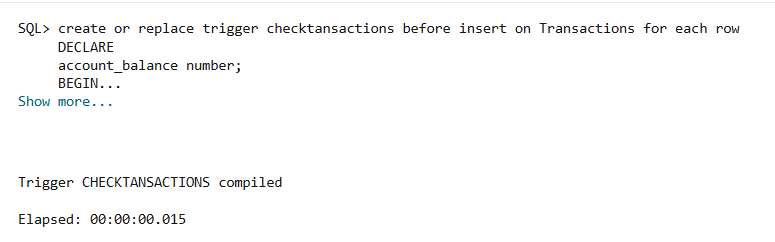
    );

  end loop;

  close transcation\_cursor;

END;

/

**OUTPUT**

**SCENARIO 2**

DECLARE

cursor annual\_fee is select accountid,BALANCE from accounts;

accounts\_recods annual\_fee%rowtype;

fee CONSTANT number:=100;

begin

    open annual\_fee;

    loop fetch annual\_fee into accounts\_recods;

    exit when annual\_fee%notfound;

    update accounts set balance =accounts\_recods.balance-fee where

    accountid=accounts\_recods.accountid;

    dbms\_output.put\_line('deduced '|| fee||'from account'||accounts\_recods.accountid||'with a new balance of'||accounts\_recods.balance);

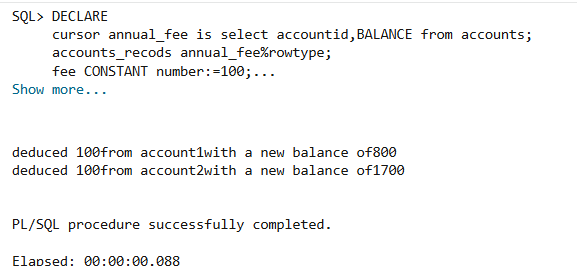
    end loop;

    close annual\_fee;

    end;

    /

**OUTPUT**

****

**SCENARIO 3**

declare

  CURSOR loan\_intrest is

    select loanid, INTERESTRATE from loans;

  loan\_rec loan\_intrest%rowtype;

  new\_rate number;

begin

  open loan\_intrest;

  loop

    fetch loan\_intrest into loan\_rec;

    exit when loan\_intrest%notfound;

    if loan\_rec.interestrate < 10 then

      new\_rate := loan\_rec.interestrate + 0.5;

    else

      new\_rate := loan\_rec.interestrate + 0.2;

    end if;

    update loans

       set INTERESTRATE = new\_rate

     where loanid = loan\_rec.loanid;

    dbms\_output.put\_line(

      'loan id :' || loan\_rec.loanid || ' has updated to ' || new\_rate

    );

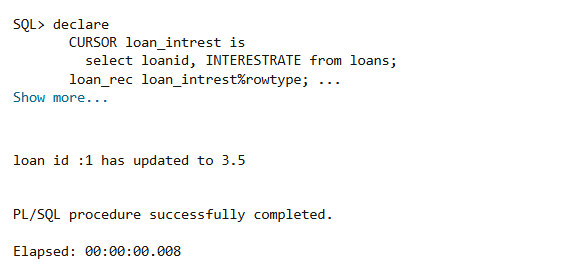
  end loop;

  CLOSE loan\_intrest;

end;

/

**OUTPUT**

****

**EXERCISE 7**

**SCENARIO 1**

create or replace PACKAGE customermangement as type customer\_record is RECORD(

    customer\_row customers%rowtype

);

PROCEDURE addcustomer( person\_record in customer\_record);

PROCEDURE updatecustomer( person\_record in customer\_record);

FUNCTION getbalance(person\_id in number) return number;

end customermangement;

/

CREATE OR REPLACE PACKAGE BODY customermangement AS

  PROCEDURE addcustomer(person\_record IN customer\_record) IS

  BEGIN

    INSERT INTO CUSTOMERS(CUSTOMERID, name, DOB, balance, LASTMODIFIED)

    VALUES (

      person\_record.customer\_row.CUSTOMERID,person\_record.customer\_row.name,person\_record.customer\_row.dob,person\_record.customer\_row.balance,SYSDATE

    );

  END addcustomer;

  PROCEDURE updatecustomer(person\_record IN customer\_record) IS

  BEGIN

    UPDATE CUSTOMERS

       SET name = person\_record.customer\_row.name,

           dob = person\_record.customer\_row.dob,

           balance = person\_record.customer\_row.balance,

           LASTMODIFIED = SYSDATE

     WHERE customerid = person\_record.customer\_row.CUSTOMERID;

  END updatecustomer;

  FUNCTION getbalance(person\_id IN NUMBER) RETURN NUMBER IS

    v\_balance NUMBER;

  BEGIN

    SELECT balance INTO v\_balance

      FROM CUSTOMERS

     WHERE CUSTOMERID = person\_id;

    RETURN v\_balance;

  EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

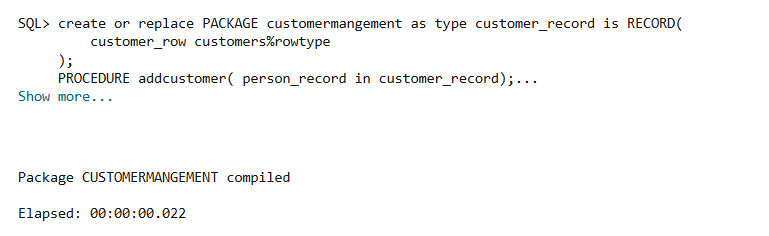
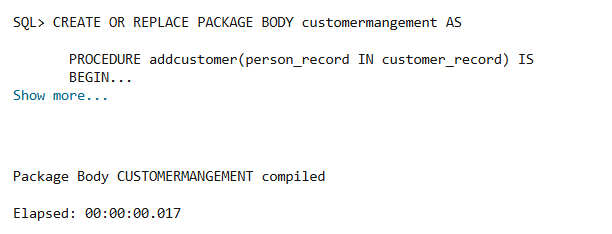
      dbms\_output.put\_line('no data found');

      RETURN NULL;

  END getbalance;

END customermangement;

/

**OUTPUT**

**SCENARIO 2**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

  TYPE EmployeeRec IS RECORD (

    employee\_row Employees%ROWTYPE

  );

  PROCEDURE HireEmployee(p\_rec IN EmployeeRec);

  PROCEDURE UpdateEmployee(p\_rec IN EmployeeRec);

  FUNCTION GetAnnualSalary(p\_employeeid IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

  PROCEDURE HireEmployee(p\_rec IN EmployeeRec) IS

  BEGIN

    INSERT INTO Employees

      (EmployeeID, Name, Position, Salary, Department, HireDate)

    VALUES

      (p\_rec.employee\_row.EmployeeID,p\_rec.employee\_row.Name,p\_rec.employee\_row.Position,p\_rec.employee\_row.Salary,p\_rec.employee\_row.Department,SYSDATE);

  END HireEmployee;

  PROCEDURE UpdateEmployee(p\_rec IN EmployeeRec) IS

  BEGIN

    UPDATE Employees

       SET Name= p\_rec.employee\_row.Name,Position= p\_rec.employee\_row.Position,Salary= p\_rec.employee\_row.Salary,Department = p\_rec.employee\_row.Department

     WHERE EmployeeID = p\_rec.employee\_row.EmployeeID;

  END UpdateEmployee;

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

    v\_salary NUMBER;

  BEGIN

    SELECT Salary INTO v\_salary FROM Employees

     WHERE EmployeeID = p\_employeeid;

    RETURN v\_salary \* 12;

  EXCEPTION

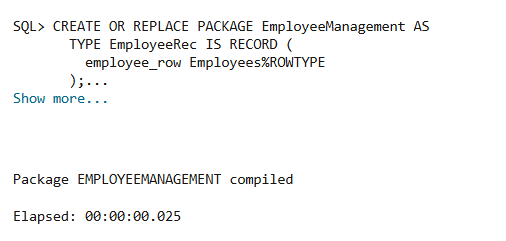
    WHEN NO\_DATA\_FOUND THEN

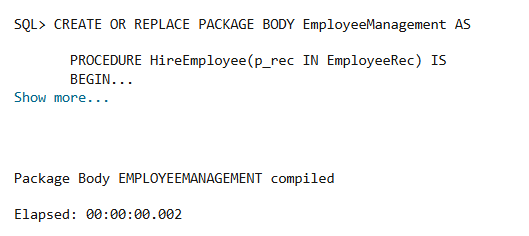
      RETURN NULL;

  END GetAnnualSalary;

END EmployeeManagement;

/

**OUTPUT**

****

**SCENARIO 3**

  CREATE OR REPLACE PACKAGE AccountOperations AS

  TYPE AccountRec IS RECORD (

    acct\_row Accounts%ROWTYPE

  );

  PROCEDURE OpenAccount(p\_rec IN AccountRec);

  PROCEDURE CloseAccount(p\_accountid IN NUMBER);

  FUNCTION GetTotalBalance(p\_customerid IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

  PROCEDURE OpenAccount(p\_rec IN AccountRec) IS

  BEGIN

    INSERT INTO Accounts

      (AccountID, CustomerID, AccountType, Balance, LastModified)

    VALUES

      (p\_rec.acct\_row.AccountID,p\_rec.acct\_row.CustomerID,p\_rec.acct\_row.AccountType,p\_rec.acct\_row.Balance,SYSDATE);

  END OpenAccount;

  PROCEDURE CloseAccount(p\_accountid IN NUMBER) IS

  BEGIN

    DELETE FROM Accounts WHERE AccountID = p\_accountid;

  END CloseAccount;

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

    v\_total NUMBER;

  BEGIN

    SELECT COALESCE(SUM(Balance), 0)

      INTO v\_total

      FROM Accounts

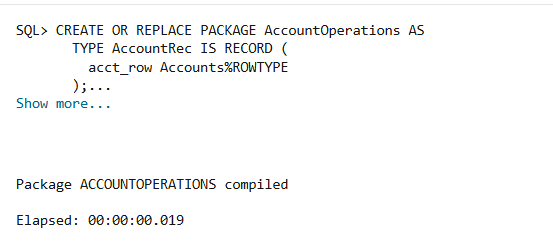
     WHERE CustomerID = p\_customerid;

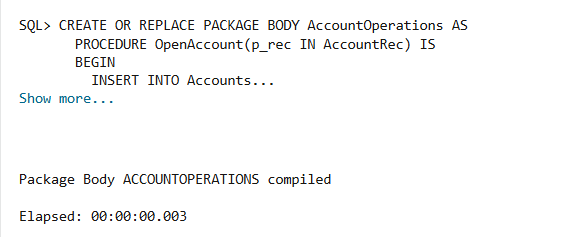
    RETURN v\_total;

  END GetTotalBalance;

END AccountOperations;

/

**OUTPUT**

****