**Exercise 1: Control Structures**

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

**CODE:**

SET SERVEROUTPUT ON;

DECLARE

CURSOR loan\_cursor IS

SELECT L.LOANID,L.INTERESTRATE,L.CUSTOMERID,C.DOB FROM LOANS L INNER JOIN CUSTOMERS C ON L.CUSTOMERID=C.CUSTOMERID;

BEGIN

    FOR loan\_rec IN loan\_cursor LOOP

        DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || loan\_rec.LOANID || ', DOB: ' || loan\_rec.DOB);

        IF ((SYSDATE - loan\_rec.DOB)/365)>60 THEN

            UPDATE LOANS

            SET INTERESTRATE=INTERESTRATE\*(0.99)

            WHERE LOANS.CUSTOMERID=loan\_rec.CUSTOMERID;

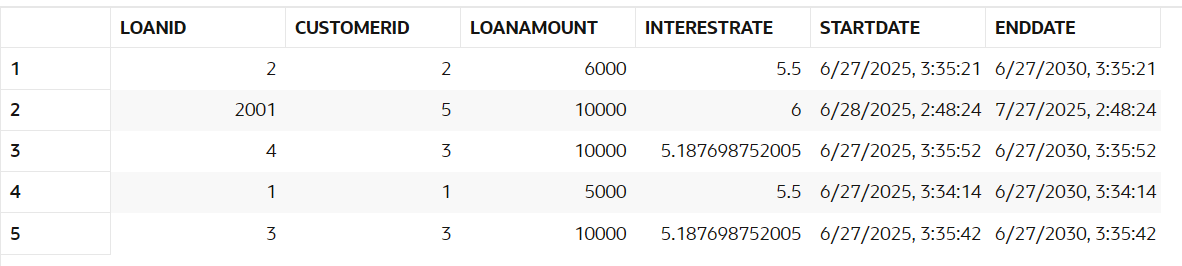
        END IF;

    END LOOP;

END;

SELECT \* FROM LOANS;

**OUTPUT:**



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

**CODE:**

ALTER TABLE CUSTOMERS

ADD (isVip NUMBER(1) DEFAULT 0 NOT NULL);

SELECT \* FROM CUSTOMERS;

DECLARE

CURSOR customer\_cursor IS (SELECT CUSTOMERID,BALANCE FROM CUSTOMERS);

BEGIN

    FOR customer IN customer\_cursor LOOP

        IF customer.BALANCE > 1000 THEN

            UPDATE CUSTOMERS

            SET ISVIP=0

            WHERE CUSTOMERID=customer.CUSTOMERID;

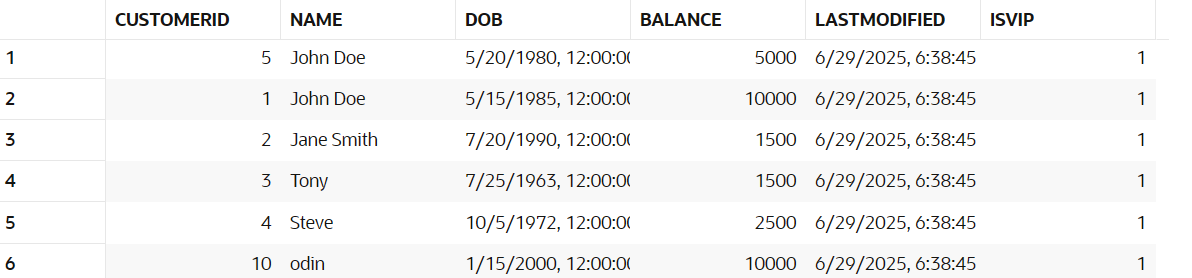
        END IF;

    END LOOP;

END;

SELECT \* FROM CUSTOMERS;

**OUTPUT:**

****

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

**CODE:**

ALTER TABLE CUSTOMERS

ADD (isVip NUMBER(1) DEFAULT 0 NOT NULL);

SELECT \* FROM CUSTOMERS;

DECLARE

CURSOR customer\_cursor IS (SELECT CUSTOMERID,BALANCE FROM CUSTOMERS);

BEGIN

    FOR customer IN customer\_cursor LOOP

        IF customer.BALANCE > 1000 THEN

            UPDATE CUSTOMERS

            SET ISVIP=0

            WHERE CUSTOMERID=customer.CUSTOMERID;

        END IF;

    END LOOP;

END;

SELECT \* FROM CUSTOMERS;

**OUTPUT:**

Loan ID 2001 for customer John Doe is due in less than 30 days (Due date: 27-07-2025).

**Exercise 2: Error Handling**

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

**CODE:**

SELECT \* FROM ACCOUNTS;

INSERT INTO ACCOUNTS (ACCOUNTID,CUSTOMERID,ACCOUNTTYPE,BALANCE,LASTMODIFIED) VALUES (1,1,'Checking',1000,SYSDATE);

SELECT \* FROM CUSTOMERS;

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

    N\_FROM\_ID ACCOUNTS.ACCOUNTID%TYPE,

    N\_TO\_ID   ACCOUNTS.ACCOUNTID%TYPE,

    AMOUNT    NUMBER

)

IS

    N\_BALANCE\_IN\_FROM\_ACCOUNT NUMBER;

BEGIN

    SAVEPOINT BEFORE\_TRANSACTION\_STATE;

    SELECT BALANCE INTO N\_BALANCE\_IN\_FROM\_ACCOUNT

    FROM ACCOUNTS

    WHERE ACCOUNTID = N\_FROM\_ID;

    IF N\_BALANCE\_IN\_FROM\_ACCOUNT < AMOUNT THEN

        RAISE\_APPLICATION\_ERROR(-20001, 'INSUFFICIENT FUNDS IN THE SENDER ACCOUNT');

    END IF;

    UPDATE ACCOUNTS

    SET BALANCE = BALANCE - AMOUNT

    WHERE ACCOUNTID = N\_FROM\_ID;

    UPDATE ACCOUNTS

    SET BALANCE = BALANCE + AMOUNT

    WHERE ACCOUNTID = N\_TO\_ID;

    COMMIT;

EXCEPTION

    WHEN OTHERS THEN

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

        ROLLBACK TO BEFORE\_TRANSACTION\_STATE;

END;

SHOW ERRORS PROCEDURE SafeTransferFunds;

EXEC SafeTransferFunds(2, 1, 500);

**OUTPUT:**

Error: ORA-20001: INSUFFICIENT FUNDS IN THE SENDER ACCOUNT

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

**CODE:**

CREATE OR REPLACE PROCEDURE updateSalary

(

    EMPID EMPLOYEES.EMPLOYEEID%TYPE,

    PERCENTAGE\_INCREASE NUMBER

)

IS

    EMPLOYEE\_DETAILS EMPLOYEES%ROWTYPE;

BEGIN

    SELECT \* INTO EMPLOYEE\_DETAILS FROM EMPLOYEES WHERE EMPLOYEEID=EMPID;

    IF EMPLOYEE\_DETAILS.EMPLOYEEID IS NOT NULL THEN

        UPDATE EMPLOYEES

        SET SALARY=SALARY\*(PERCENTAGE\_INCREASE/100)

        WHERE EMPLOYEEID=EMPID;

    END IF;

EXCEPTION

    WHEN OTHERS THEN

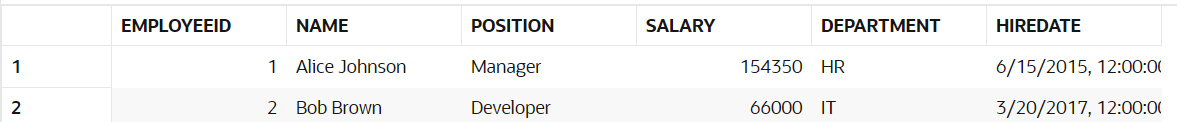
        DBMS\_OUTPUT.PUT\_LINE('EMPLOYEE ID NOT FOUND');

END;

EXEC updateSalary(1,105);

SELECT \* FROM EMPLOYEES;

**OUTPUT:**

****

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

**CODE:**

CREATE OR REPLACE PROCEDURE addnewcustomer

(

    customer\_id customers.customerid%TYPE,

    name customers.name%TYPE,

    dob customers.dob%TYPE,

    balance customers.balance%TYPE

)

IS

    n\_id NUMBER;

BEGIN

    BEGIN

        SELECT customerid INTO n\_id FROM customers WHERE customerid = customer\_id;

    EXCEPTION

        WHEN NO\_DATA\_FOUND THEN

            n\_id := NULL;

    END;

    IF n\_id IS NULL THEN

        INSERT INTO customers

        (

            customerid,

            name,

            dob,

            balance,

            lastmodified

        ) VALUES

        (

            customer\_id,

            name,

            dob,

            balance,

            SYSDATE

        );

    ELSE

        RAISE\_APPLICATION\_ERROR(-20001, 'Customer with ID ' || customer\_id || ' already exists.');

    END IF;

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

EXEC ADDNEWCUSTOMER(10,'odin',TO\_DATE('15-01-2000','DD-MM-YYYY'),10000);

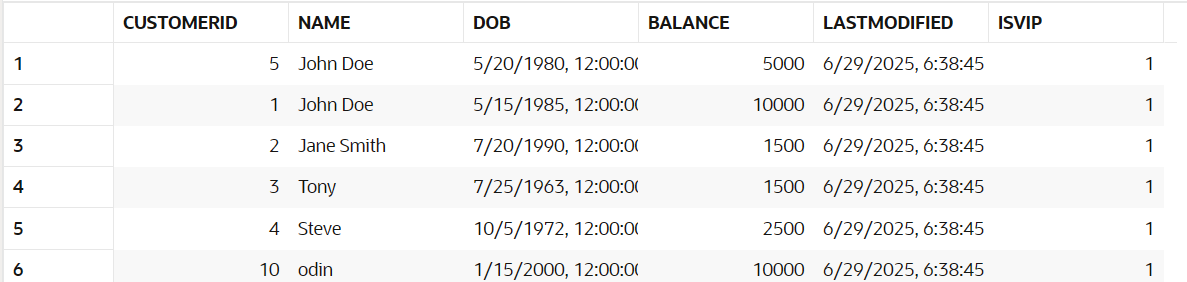
SELECT \* FROM CUSTOMERS;

EXEC ADDNEWCUSTOMER(1,'odin',TO\_DATE('15-01-2000','DD-MM-YYYY'),10000);

**OUTPUT:**

**ORA-20001: Customer with ID 10 already exists.**

**ORA-20001: Customer with ID 1 already exists.**

****

**Exercise 3: Stored Procedures**

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

**CODE:**

INSERT INTO accounts

VALUES (3, 4, 'Savings', 2500, SYSDATE);

SELECT \* FROM customers;

SELECT \* FROM accounts;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

    CURSOR savings\_account\_data IS

        SELECT \* FROM accounts WHERE accounttype = 'Savings';

BEGIN

    FOR account\_data IN savings\_account\_data LOOP

        UPDATE accounts

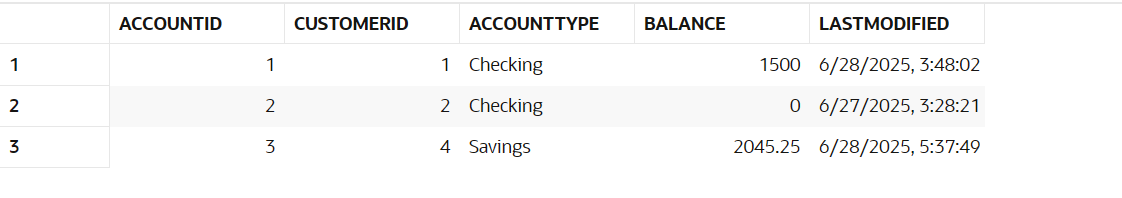
        SET balance = balance + (balance \* 0.01)

        WHERE accountid = account\_data.accountid;

    END LOOP;

END;

**OUTPUT:**

****

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

**CODE:**

create or replace procedure UpdateEmployeeBonus

( dep employees.department%type,bonus\_percentage number)

is

cursor emp\_data is select \* from employees where department=dep;

begin

    for emp in emp\_data loop

        update employees

        set salary=salary + (salary\*(bonus\_percentage/100))

        where employeeid=emp.employeeid;

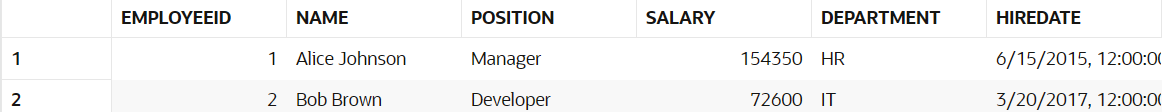
    end loop;

end;

exec UPDATEEMPLOYEEBONUS('IT',10);

select \* from employees;

**OUTPUT:**

****

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

**CODE:**

SELECT \* FROM ACCOUNTS;

INSERT INTO ACCOUNTS (ACCOUNTID,CUSTOMERID,ACCOUNTTYPE,BALANCE,LASTMODIFIED) VALUES (1,1,'Checking',1000,SYSDATE);

SELECT \* FROM CUSTOMERS;

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

    N\_FROM\_ID ACCOUNTS.ACCOUNTID%TYPE,

    N\_TO\_ID   ACCOUNTS.ACCOUNTID%TYPE,

    AMOUNT    NUMBER

)

IS

    N\_BALANCE\_IN\_FROM\_ACCOUNT NUMBER;

BEGIN

    SAVEPOINT BEFORE\_TRANSACTION\_STATE;

    SELECT BALANCE INTO N\_BALANCE\_IN\_FROM\_ACCOUNT

    FROM ACCOUNTS

    WHERE ACCOUNTID = N\_FROM\_ID;

    IF N\_BALANCE\_IN\_FROM\_ACCOUNT < AMOUNT THEN

        RAISE\_APPLICATION\_ERROR(-20001, 'INSUFFICIENT FUNDS IN THE SENDER ACCOUNT');

    END IF;

    UPDATE ACCOUNTS

    SET BALANCE = BALANCE - AMOUNT

    WHERE ACCOUNTID = N\_FROM\_ID;

    UPDATE ACCOUNTS

    SET BALANCE = BALANCE + AMOUNT

    WHERE ACCOUNTID = N\_TO\_ID;

    COMMIT;

EXCEPTION

    WHEN OTHERS THEN

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

        ROLLBACK TO BEFORE\_TRANSACTION\_STATE;

END;

SHOW ERRORS PROCEDURE SafeTransferFunds;

EXEC SafeTransferFunds(2, 1, 500);

**OUTPUT:**

Error: ORA-20001: INSUFFICIENT FUNDS IN THE SENDER ACCOUNT

**Exercise 4: Functions**

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

**Code:**

CREATE OR REPLACE FUNCTION calculateAge (

    dob IN customers.dob%TYPE

) RETURN NUMBER

IS

    age NUMBER;

BEGIN

    age := TRUNC((SYSDATE - dob) / 365);

    RETURN age;

END;

SELECT calculateAge(TO\_DATE('18-08-2005', 'DD-MM-YYYY')) AS age

FROM dual;

**Output:**

****

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

**Code:**

create or replace function CalculateMonthlyInstallment (

    principal\_amount number,

    intrest number,

    tenure number

) return number

is

    emi number;

    mi number;

begin

    mi :=intrest/12/100;

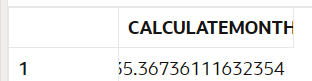
    emi :=(principal\_amount\*mi\*power(1+mi,tenure\*12))/(power(1+mi,tenure\*12) -1);

    return emi;

end;

select CalculateMonthlyInstallment(5000,12,2) from dual;

**Output:**

****

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

**Code:**

create or replace function CalculateMonthlyInstallment (

    principal\_amount number,

    intrest number,

    tenure number

) return number

is

    emi number;

    mi number;

begin

    mi :=intrest/12/100;

    emi :=(principal\_amount\*mi\*power(1+mi,tenure\*12))/(power(1+mi,tenure\*12) -1);

    return emi;

end;

select CalculateMonthlyInstallment(5000,12,2) from dual;

create or replace function HasSufficientBalance(account\_id number,amount number) return NUMBER

is

currbalance number;

begin

    select balance into currbalance from accounts where accountid=account\_id;

    if amount<=currbalance then

        return 1;

    ELSE

        return 0;

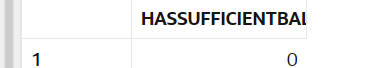
    end if;

end;

select HASSUFFICIENTBALANCE(1,2001) from dual;

select \* from accounts;

**Output:**



**Exercise 5: Triggers**

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

**CODE:**

create or replace trigger UpdateCustomerLastModified

before update on customers for each row

begin

    :NEW.lastmodified:=sysdate;

end;

select \* from customers;

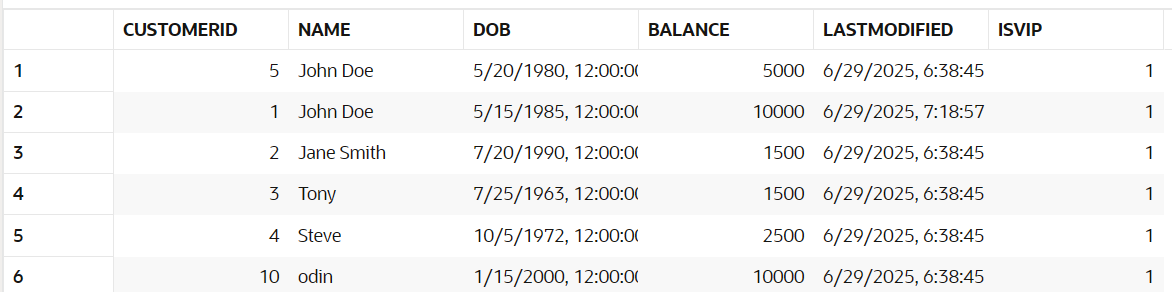
update CUSTOMERS

set balance=10000

where customerid=1;

select \* from customers;

**OUTPUT:**

****

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

**CODE:**

CREATE TABLE AuditLog (

    audit\_id      NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    table\_name    VARCHAR2(50),

    operation     VARCHAR2(10),

    record\_id     NUMBER,

    new\_data      CLOB,

    changed\_by    VARCHAR2(100),

    changed\_on    DATE DEFAULT SYSDATE

);

create or replace trigger logtransaction

after insert on transactions for each row

BEGIN

    insert into auditlog

    (

        table\_name,

        operation,

        record\_id,

        new\_data,

        CHANGED\_BY

    ) values (

        'transactions',

        'insert',

        :NEW.transactionid,

        :NEW.accountid || :NEW.amount || 'on' || :NEW.transactiondate || :NEW.transactiontype,

        user

    );

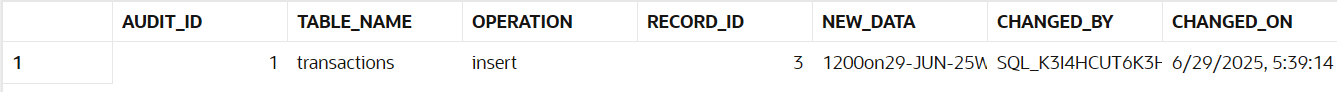
end;

select \* from transactions;

insert into TRANSACTIONS values(3,1,sysdate,200,'Withdrawal');

select \* from AUDITLOG;

**OUTPUT:**

****

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

**CODE:**

create or replace trigger checktransactionrules

before insert on transactions for each ROW

declare

 account\_balance number;

begin

    if :NEW.transactiontype='Withdrawal' then

        select balance into account\_balance from accounts where accountid=:NEW.accountid;

        if account\_balance<:NEW.amount then

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

        end if;

    end if;

    if :NEW.transactiontype='Deposit' then

        if :NEW.amount<=0 then

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

        end if;

    end if;

end;

insert into TRANSACTIONS values(4,1,sysdate,200000,'Withdrawal');

**OUTPUT:**

SQL> insert into TRANSACTIONS values(4,1,sysdate,200000,'Withdrawal')

[ORA-20001](https://docs.oracle.com/en/error-help/db/ora-20001): Insufficient balance for withdrawal  
ORA-06512: at "SQL\_K3I4HCUT6K3H0W4I6P5F5BU82G.CHECKTRANSACTIONRULES", line 8  
ORA-04088: error during execution of trigger 'SQL\_K3I4HCUT6K3H0W4I6P5F5BU82G.CHECKTRANSACTIONRULES'  
  
<https://docs.oracle.com/error-help/db/ora-20001/>

**Exercise 6: Cursors**

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

**CODE:**

DECLARE

    CURSOR GenerateMonthlyStatements IS

        SELECT t.transactionid, t.accountid, t.amount, t.transactiontype, t.transactiondate

        FROM transactions t

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

        ORDER BY t.accountid, t.transactiondate;

    prev\_account\_id transactions.accountid%TYPE := -1;  -- Set to a value no account will have

    first\_time BOOLEAN := TRUE;

BEGIN

    FOR tx IN GenerateMonthlyStatements LOOP

        -- Print account header when account changes OR on first iteration

        IF first\_time OR tx.accountid != prev\_account\_id THEN

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

            DBMS\_OUTPUT.PUT\_LINE('Monthly Statement for Account ID: ' || tx.accountid);

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

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

            prev\_account\_id := tx.accountid;

            first\_time := FALSE;

        END IF;

        DBMS\_OUTPUT.PUT\_LINE(

            TO\_CHAR(tx.transactiondate, 'DD-Mon') || ' | ' ||

            RPAD(tx.transactiontype, 10) || ' | ' ||

            tx.amount

        );

    END LOOP;

END;  
**OUTPUT:**

--------------------------------------

Monthly Statement for Account ID: 1

Date | Type | Amount

--------------------------------------

29-Jun | Withdrawal | 200

--------------------------------------

Monthly Statement for Account ID: 2

Date | Type | Amount

--------------------------------------

27-Jun | Withdrawal | 300

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

**CODE:**

DECLARE

    CURSOR ApplyAnnualFee IS

        SELECT accountid, balance FROM accounts;

    annual\_fee NUMBER := 500;

BEGIN

    FOR acc IN ApplyAnnualFee LOOP

        UPDATE accounts

        SET balance = acc.balance - annual\_fee

        WHERE accountid = acc.accountid;

        DBMS\_OUTPUT.PUT\_LINE(

            'Annual fee of ' || annual\_fee || ' applied to Account ID: ' || acc.accountid

        );

    END LOOP;

END;  
**OUTPUT:**

Annual fee of 500 applied to Account ID: 1

Annual fee of 500 applied to Account ID: 2

Annual fee of 500 applied to Account ID: 3

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

DECLARE

    CURSOR UpdateLoanInterestRates IS

        SELECT loanid, loanamount, interestrate FROM loans;

    new\_rate NUMBER;

BEGIN

    FOR loan IN UpdateLoanInterestRates LOOP

        IF loan.loanamount > 100000 THEN

            new\_rate := loan.interestrate + 1;

        ELSE

            new\_rate := loan.interestrate + 0.5;

        END IF;

        UPDATE loans

        SET interestrate = new\_rate

        WHERE loanid = loan.loanid;

        DBMS\_OUTPUT.PUT\_LINE(

            'Loan ID: ' || loan.loanid || ' → New Interest Rate: ' || new\_rate || '%'

        );

    END LOOP;

END;  
**OUTPUT:**

Loan ID: 2 → New Interest Rate: 6%

Loan ID: 2001 → New Interest Rate: 6.5%

Loan ID: 4 → New Interest Rate: 5.687698752005%

Loan ID: 1 → New Interest Rate: 6%

Loan ID: 3 → New Interest Rate: 5.687698752005%