**Exercise 1: Control Structures**

Table Creation and data insertion:

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Age NUMBER,

Balance NUMBER(10,2),

InterestRate NUMBER(5,2),

IsVIP VARCHAR2(5),

LoanDueDate DATE

);

INSERT INTO Customers VALUES (1, 'Alice', 65, 8000.00, 7.5, 'FALSE', SYSDATE + 10);

INSERT INTO Customers VALUES (2, 'Bob', 45, 12000.00, 8.0, 'FALSE', SYSDATE + 40);

INSERT INTO Customers VALUES (3, 'Charlie', 70, 9500.00, 6.5, 'FALSE', SYSDATE + 5);

INSERT INTO Customers VALUES (4, 'Diana', 30, 20000.00, 9.0, 'FALSE', SYSDATE + 25);

INSERT INTO Customers VALUES (5, 'Eve', 61, 15000.00, 7.2, 'FALSE', SYSDATE + 28);

COMMIT;

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

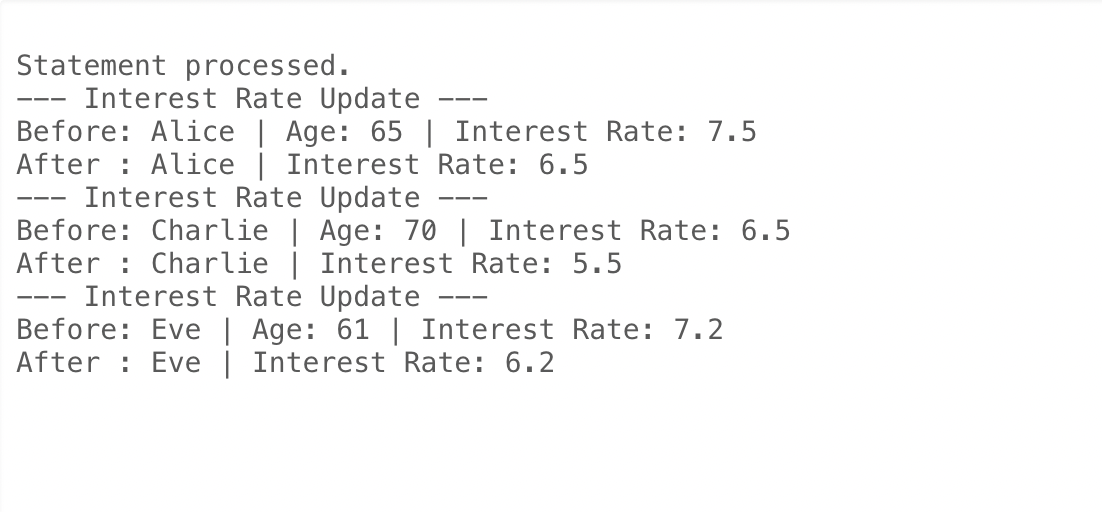
**Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

BEGIN FOR cust IN (SELECT \* FROM Customers WHERE Age > 60) LOOP DBMS\_OUTPUT.PUT\_LINE('--- Interest Rate Update ---'); DBMS\_OUTPUT.PUT\_LINE('Before: ' || cust.Name || ' | Age: ' || cust.Age || ' | Interest Rate: ' || cust.InterestRate);

UPDATE Customers  
 SET InterestRate = InterestRate - 1  
 WHERE CustomerID = cust.CustomerID;  
  
 -- Fetch updated value  
 SELECT InterestRate INTO cust.InterestRate  
 FROM Customers  
 WHERE CustomerID = cust.CustomerID;  
  
 DBMS\_OUTPUT.PUT\_LINE('After : ' || cust.Name || ' | Interest Rate: ' || cust.InterestRate);  
END LOOP;

END; /

Output:



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

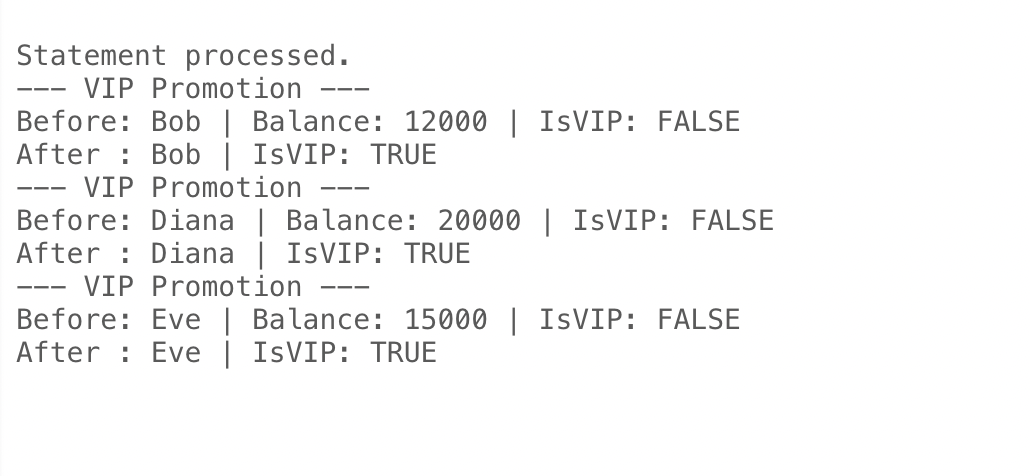
**Question: Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.**

BEGIN FOR cust IN (SELECT \* FROM Customers WHERE Balance > 10000) LOOP DBMS\_OUTPUT.PUT\_LINE('--- VIP Promotion ---'); DBMS\_OUTPUT.PUT\_LINE('Before: ' || cust.Name || ' | Balance: ' || cust.Balance || ' | IsVIP: ' || cust.IsVIP);

UPDATE Customers  
 SET IsVIP = 'TRUE'  
 WHERE CustomerID = cust.CustomerID;  
  
 SELECT IsVIP INTO cust.IsVIP  
 FROM Customers  
 WHERE CustomerID = cust.CustomerID;  
  
 DBMS\_OUTPUT.PUT\_LINE('After : ' || cust.Name || ' | IsVIP: ' || cust.IsVIP);  
END LOOP;

END; /

Output:



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

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

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Loan Due Reminders ---');

FOR cust IN ( SELECT Name, LoanDueDate

FROM Customers

WHERE LoanDueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

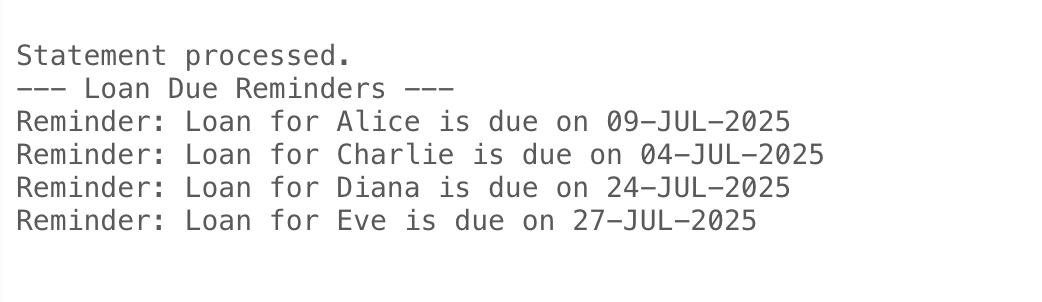
DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan for ' || cust.Name || ' is due on ' || TO\_CHAR(cust.LoanDueDate, 'DD-MON-YYYY'));

END LOOP;

END;

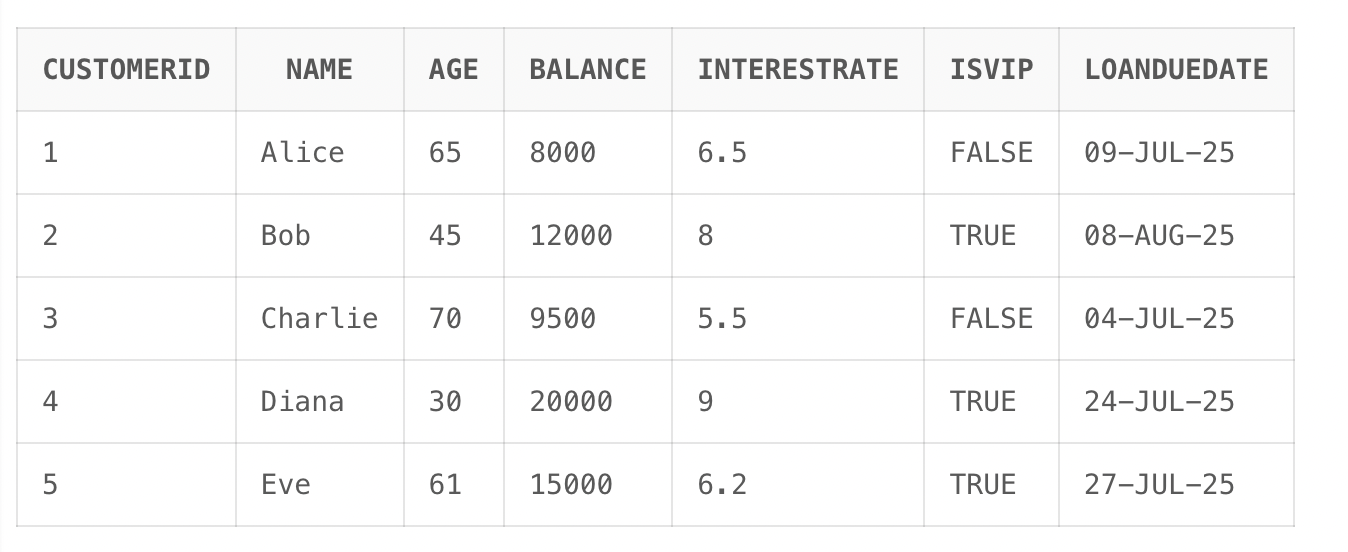
/

Output:



Final Output:

SELECT \* FROM Customers;



Exercise 2: Error Handling

Table Creation and data insertion:

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

HolderName VARCHAR2(50),

Balance NUMBER(10,2)

);

INSERT INTO Accounts VALUES (101, 'Alice', 5000.00);

INSERT INTO Accounts VALUES (102, 'Bob', 3000.00);

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

EmpID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Salary NUMBER(10,2)

);

INSERT INTO Employees VALUES (1, 'John', 50000);

INSERT INTO Employees VALUES (2, 'Jane', 60000);

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Age NUMBER,

Balance NUMBER(10,2)

);

Scenario 1: Handle exceptions during fund transfers between accounts.

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 OR REPLACE PROCEDURE SafeTransferFunds ( p\_from\_account\_id IN NUMBER, p\_to\_account\_id IN NUMBER, p\_amount IN NUMBER ) IS v\_from\_balance NUMBER; BEGIN SELECT Balance INTO v\_from\_balance FROM Accounts WHERE AccountID = p\_from\_account\_id;

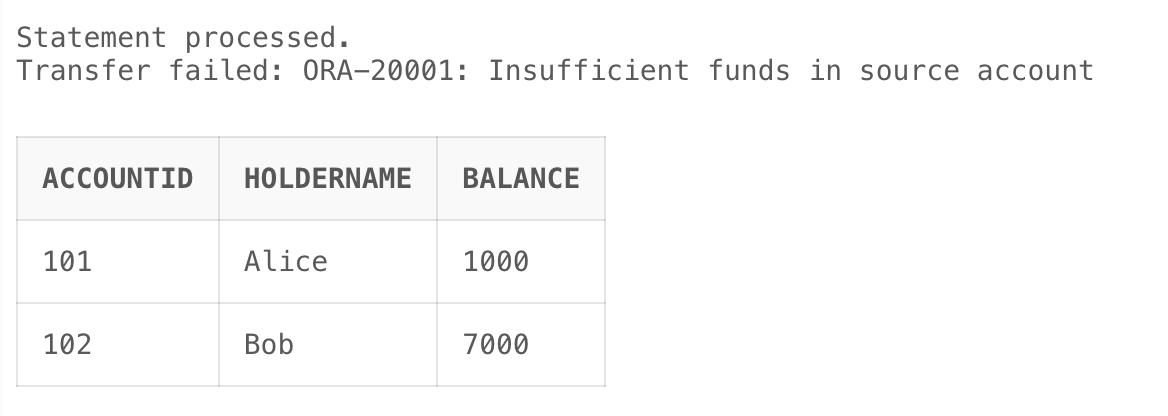
IF v\_from\_balance < p\_amount THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in source account');  
END IF;  
  
UPDATE Accounts  
SET Balance = Balance - p\_amount  
WHERE AccountID = p\_from\_account\_id;  
  
UPDATE Accounts  
SET Balance = Balance + p\_amount  
WHERE AccountID = p\_to\_account\_id;  
  
COMMIT;  
DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully.');

EXCEPTION WHEN OTHERS THEN ROLLBACK; DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM); END; /

OUTPUT:

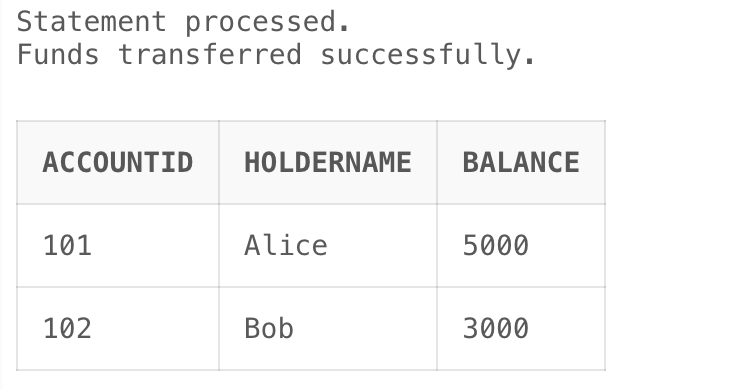
EXEC SafeTransferFunds(101, 102, 2000);

SELECT \* FROM Accounts WHERE AccountID IN (101, 102);



EXEC SafeTransferFunds(102, 101, 2000);

SELECT \* FROM Accounts WHERE AccountID IN (102, 101);



Scenario 2: Manage errors when updating employee salaries.

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\_emp\_id IN NUMBER, p\_percent IN NUMBER ) IS BEGIN UPDATE Employees SET Salary = Salary + (Salary \* p\_percent / 100) WHERE EmpID = p\_emp\_id;

IF SQL%ROWCOUNT = 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID not found');  
END IF;  
  
COMMIT;  
DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully.');

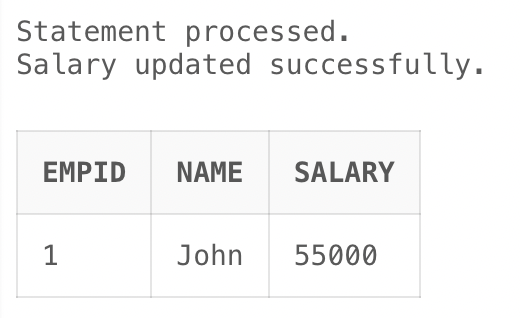
EXCEPTION WHEN OTHERS THEN ROLLBACK; DBMS\_OUTPUT.PUT\_LINE('Salary update failed: ' || SQLERRM);

END; /

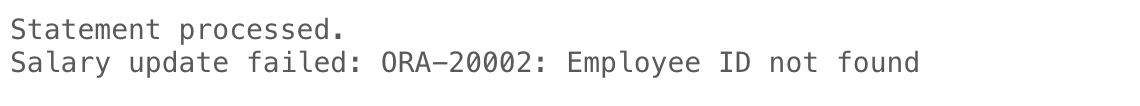
OUTPUT:

EXEC UpdateSalary(1, 10);

SELECT \* FROM Employees WHERE EmpID = 1;



EXEC UpdateSalary(999, 10);



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

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 NUMBER, p\_name IN VARCHAR2, p\_age IN NUMBER, p\_balance IN NUMBER ) IS BEGIN INSERT INTO Customers(CustomerID, Name, Age, Balance) VALUES (p\_customer\_id, p\_name, p\_age, p\_balance);

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

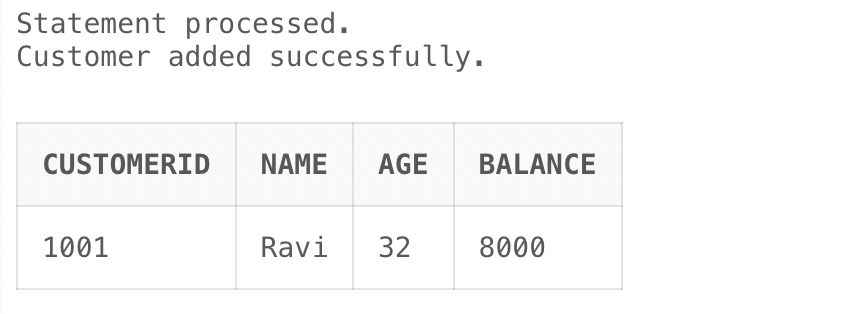
EXCEPTION WHEN DUP\_VAL\_ON\_INDEX THEN DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID already exists.'); ROLLBACK; WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM); ROLLBACK;

END; /

OUTPUT:

EXEC AddNewCustomer(1001, 'Ravi', 32, 8000);

SELECT \* FROM Customers WHERE CustomerID = 1001;



EXEC AddNewCustomer(1001, 'Sita', 28, 8500);

SELECT \* FROM Customers WHERE CustomerID = 1001;

