**Exercise 1: Control Structures**

**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.

**CODE:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers\_Test';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN

RAISE;

END IF;

END;

/

**-- Create new table**

CREATE TABLE Customers\_Test (

CustomerID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Age NUMBER,

InterestRate NUMBER(5,2)

);

**-- Insert sample data**

INSERT INTO Customers\_Test VALUES (101, 'Ravi Kumar', 65, 8.5);

INSERT INTO Customers\_Test VALUES (102, 'Anita Das', 45, 7.5);

INSERT INTO Customers\_Test VALUES (103, 'Sundar Iyer', 70, 9.0);

INSERT INTO Customers\_Test VALUES (104, 'Kavitha B', 60, 6.5);

COMMIT;

**-- PL/SQL block to apply 1% discount for age > 60**

BEGIN

FOR cust IN (

SELECT CustomerID, InterestRate

FROM Customers\_Test

WHERE Age > 60

) LOOP

UPDATE Customers\_Test

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

END LOOP;

COMMIT;

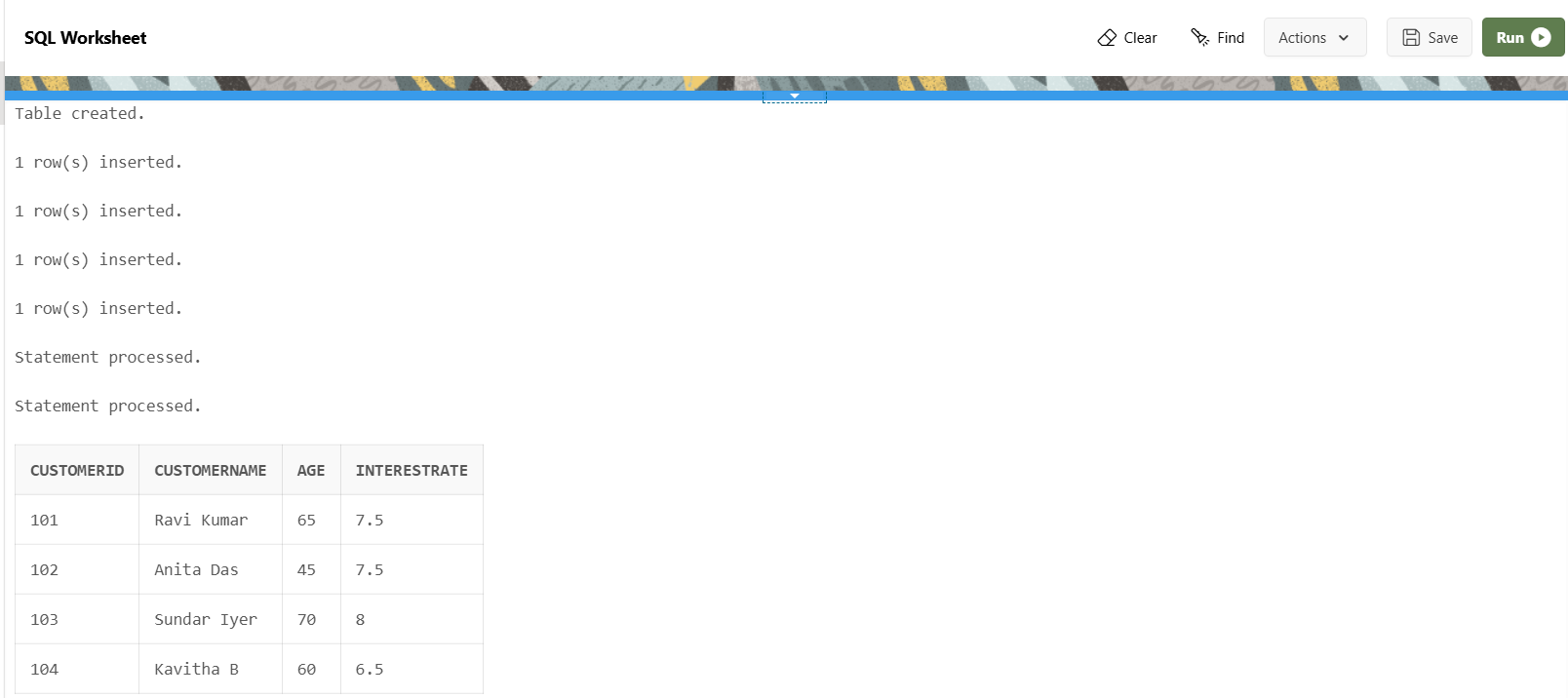
END;

/

**-- View updated data**

SELECT \* FROM Customers\_Test;

**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.

**CODE:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers\_VIP';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN

RAISE;

END IF;

END;

/

**-- Create table**

CREATE TABLE Customers\_VIP (

CustomerID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Balance NUMBER(10, 2),

IsVIP VARCHAR2(5)

);

**-- Insert data**

INSERT INTO Customers\_VIP VALUES (201, 'Kiran Raj', 15000.00, 'FALSE');

INSERT INTO Customers\_VIP VALUES (202, 'Divya Nair', 9800.00, 'FALSE');

INSERT INTO Customers\_VIP VALUES (203, 'Farhan A', 12500.00, 'FALSE');

INSERT INTO Customers\_VIP VALUES (204, 'Nisha R', 8700.00, 'FALSE');

COMMIT;

**-- Apply VIP logic**

BEGIN

FOR cust IN (

SELECT CustomerID, Balance

FROM Customers\_VIP

WHERE Balance > 10000

) LOOP

UPDATE Customers\_VIP

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

END LOOP;

COMMIT;

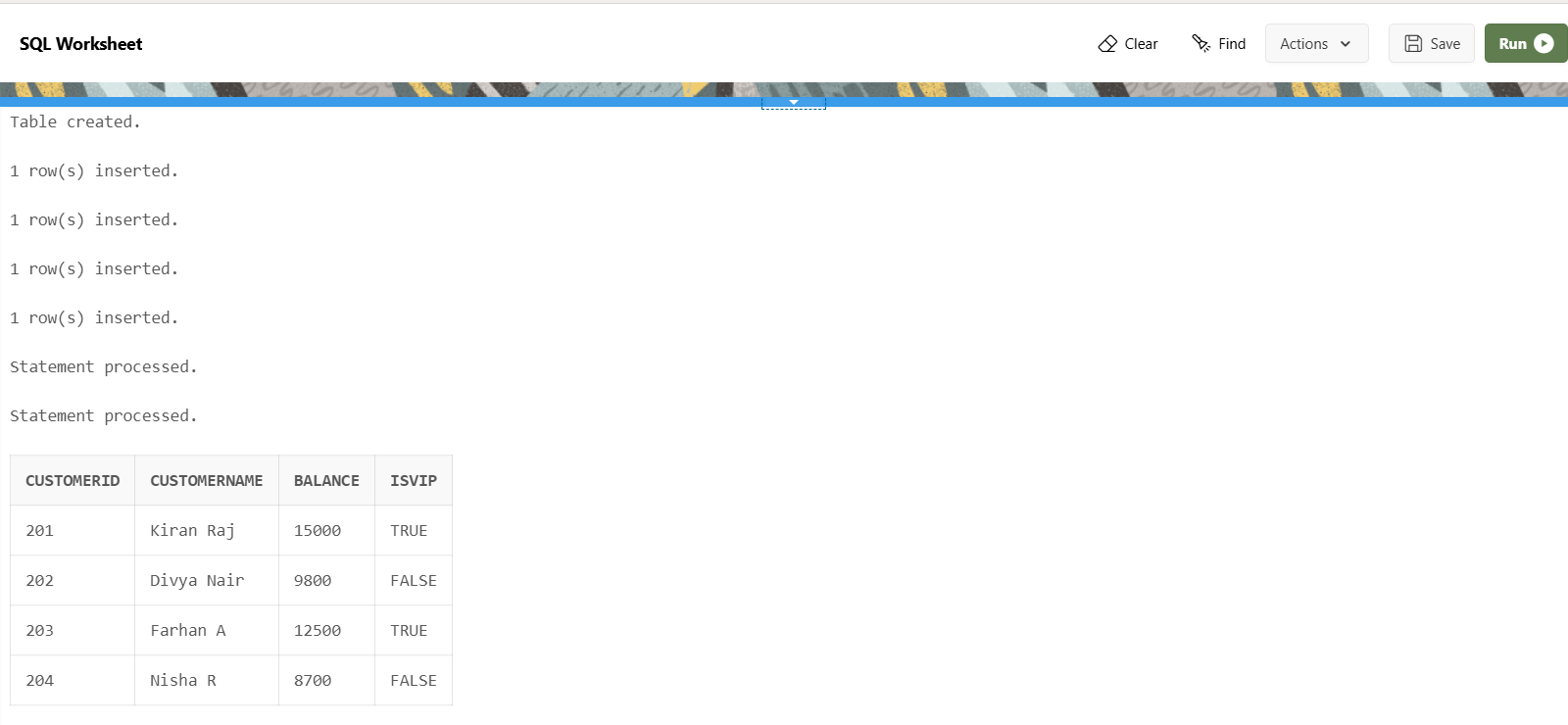
END;

/

**-- View result**

SELECT \* FROM Customers\_VIP;

**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.

**Code:**

**--Drop table if it already exists**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE LoanDetails';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN

RAISE;

END IF;

END;

/

**-- Create Table**

CREATE TABLE LoanDetails (

LoanID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

DueDate DATE

);

**-- Insert Sample Data**

INSERT INTO LoanDetails VALUES (1, 'Ravi Kumar', SYSDATE + 5);

INSERT INTO LoanDetails VALUES (2, 'Divya Nair', SYSDATE + 40);

INSERT INTO LoanDetails VALUES (3, 'Sundar Iyer', SYSDATE + 25);

INSERT INTO LoanDetails VALUES (4, 'Kavitha B', SYSDATE - 2);

COMMIT;

**-- PL/SQL Block to Show Reminders**

BEGIN

FOR loan IN (

SELECT LoanID, CustomerName, DueDate

FROM LoanDetails

WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

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

' for customer "' || loan.CustomerName ||

'" is due on ' || TO\_CHAR(loan.DueDate, 'DD-MON-YYYY'));

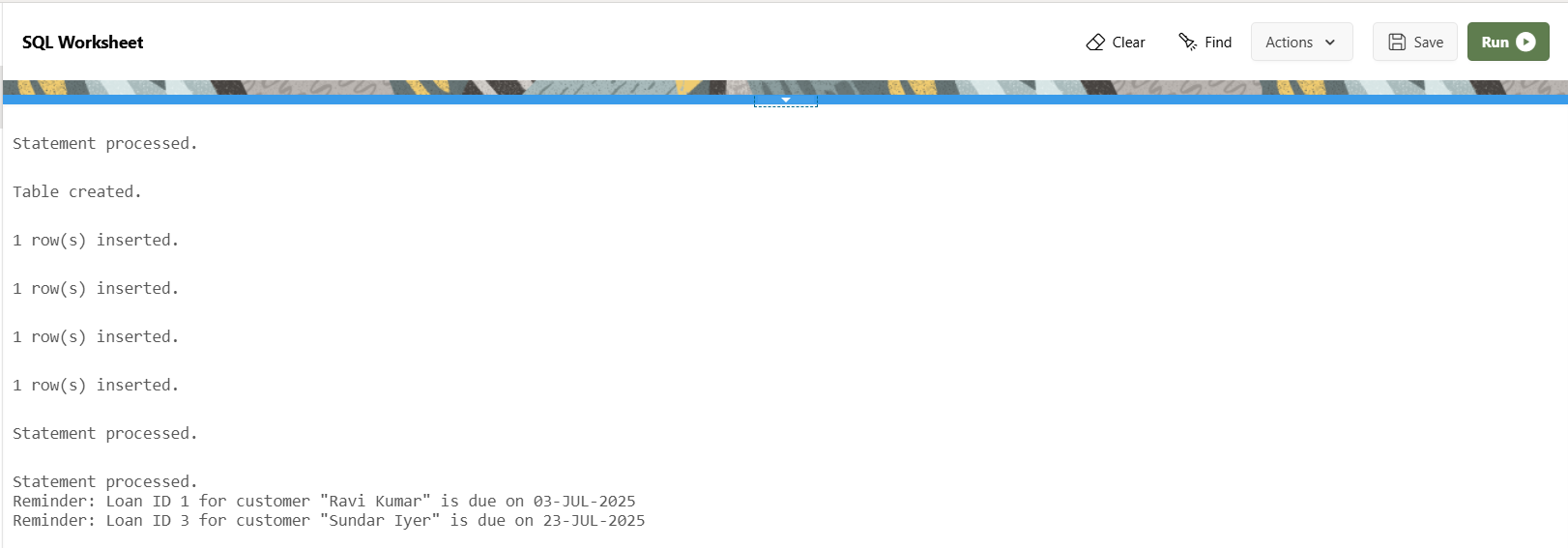
END LOOP;

END;

/

DROP TABLE LoanDetails;

**Output:**

**Exercise 2: Error Handling**

**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.

**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.

**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.

**Exercise 3: Stored Procedures**

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

* + **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.

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

* + **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.

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

* + **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.

**Exercise 4: Functions**

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

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

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

* + **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.

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

* + **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.

**Exercise 5: Triggers**

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

* + **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.

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

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

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

* + **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.

**Exercise 6: Cursors**

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

* + **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.

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

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

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

* + **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.

**Exercise 7: Packages**

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

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

**Scenario 2:** Create a package to manage employee data.

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

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

* + **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.

**Schema to be Created**

*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*

*);*

**Example Scripts for Sample Data Insertion**

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

*VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);*

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

*VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);*

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

*VALUES (1, 1, 'Savings', 1000, SYSDATE);*

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

*VALUES (2, 2, 'Checking', 1500, SYSDATE);*

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

*VALUES (1, 1, SYSDATE, 200, 'Deposit');*

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

*VALUES (2, 2, SYSDATE, 300, 'Withdrawal');*

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

*VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));*

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

*VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));*

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

*VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));*