

**Prof. Ramkrishna More**

**Arts, Commerce and Science College(Autonomous)**

**Akurdi Pradhikaran, Pune-44**

**M. SC - COMPUTER APPLICATIONS**



Semester-I

Lab Course: Advanced Databases Laboratory

(CAMAP-517)

WorkBook

**Name of Student: Ms. Shravani Kishor Hore**

**Roll No: 12**

**Academic Year: 2023-24**

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| 1. | Write a SQL query to retrieve all records from a table named "Customers." |
| 2. | Explain the basic structure of an SQL query and provide an example. |
| 3. | Create a table called "Employees" using the appropriate DDL command, specifying the necessary attributes and constraints. |
| 4. | Perform an UPDATE operation in SQL to modify the "Salary" column of the "Employees" table for all employees with a "JobTitle" of "Manager." |
| 5. | Write an SQL query to retrieve employee records where the "LastName" starts with the letter "S" and the "City" is either "New York" or "London." |
| 6. | Combine the results of two SQL queries using a set operation to retrieve the common records from two tables. |
| 7. | Calculate the average salary of all employees using aggregate operators and functions. |
| 8. | Retrieve all records from a table named "Orders" where the "OrderDate" is between '2022-01-01' and '2022-12-31.' |
| 9. | Use SQL date functions to extract the month and year from a given date column. |
| 10. | Write a nested subquery in SQL to retrieve the employees who earn a higher salary than the average salary of all employees. |
| 11. | Write a PL/pgSQL program that displays the message "Hello, World!" using the RAISE NOTICE statement. |
| 12. | Write a PL/pgSQL program that takes two numbers as input parameters and calculates their sum, difference, product, and quotient. Display the results using the RAISE NOTICE statement. |
| 13. | Write a PL/pgSQL program that takes a number as input and determines whether it is even or odd. Display the result using the RAISE NOTICE statement. |
| 14. | Write a PL/pgSQL program that takes a number as input and calculates its factorial. Display the result using the RAISE NOTICE statement. |
| 15. | Write a PL/pgSQL program that generates the Fibonacci series up to a given number. Display the series using the RAISE NOTICE statement. |
| 16. | Write a PL/pgSQL program that takes a year as input and determines whether it is a leap year or not. Display the result using the RAISE NOTICE statement. |
| 17. | Write a PL/pgSQL program that takes a number as input and checks whether it is a prime number. Display the result using the RAISE NOTICE statement. |
| 18. | Write a PL/pgSQL program that takes a string as input and checks whether it is a palindrome (reads the same forwards and backwards). Display the result using the RAISE NOTICE statement. |
| 19. | Write a PL/pgSQL program that takes a string as input and reverses it. Display the reversed string using the RAISE NOTICE statement. |
| 20. | Write a PL/pgSQL program that takes multiple numbers as input and determines the maximum and minimum numbers among them. Display the results using the RAISE NOTICE statement. |

**Certificate**

This is to certify that Mr./Ms. has successfully completed the programming assignments in the Advanced Database Laboratory.

(Dr. Santosh Jagtap)

**Lab In-charge H.O.D**

**Internal Examiner External Examiner**

**SQL**

**1. Write a SQL query to retrieve all records from a table named "Customers."**

**1.Create the "CompanyDatabase" database:**

create database CompanyDatabase;

use CompanyDatabase;

**2.Create the "Customers" table:**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

Phone VARCHAR(20)

);

This SQL query creates a "Customers" table with columns for CustomerID (as the primary key), FirstName, LastName, Email, and Phone.

**3.Insert a record into the "Customers" table:**

INSERT INTO Customers (CustomerID, FirstName, LastName, Email, Phone)

VALUES (1, 'John', 'Doe', 'johndoe@example.com', '+1234567890');

This query inserts a single record into the "Customers" table.

**4.Retrieve all records from the "Customers" table:**

SELECT \* FROM Customers;

This query will select all columns (\*) from the "Customers" table, returning all the records that have been inserted into the table.

**Output:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustomerID** | **FirstName** | **LastName** | **Email** | **Phone** |
| 1 | John | Doe | johndoe@example.com | +1234567890 |

**2.Explain the basic structure of an SQL query and provide an example.**

The basic structure of an SQL query consists of several clauses that specify what data you want to retrieve or manipulate from a database. Here's a breakdown of the essential components of an SQL query:

**SELECT Clause:** The SELECT clause is used to specify the columns or expressions you want to retrieve from the database. You can use an asterisk (\*) to select all columns or list specific column names.

**FROM Clause:** The FROM clause specifies the table or tables from which you want to retrieve data. It defines the data source for your query.

**WHERE Clause (Optional):** The WHERE clause is used to filter rows based on specified conditions. It allows you to retrieve only the rows that meet certain criteria.

**GROUP BY Clause (Optional):** The GROUP BY clause is used to group rows with similar values in one or more columns. It is often used with aggregate functions like SUM, COUNT, or AVG to perform calculations on grouped data.

**HAVING Clause (Optional):** The HAVING clause is used to filter groups of rows that result from the GROUP BY clause based on aggregate function results.

**ORDER BY Clause (Optional):** The ORDER BY clause is used to sort the result set based on one or more columns. You can specify ascending (ASC) or descending (DESC) sorting order.

**LIMIT Clause (Optional):** The LIMIT clause restricts the number of rows returned in the result set. It is often used for pagination or to limit the size of the result.

create database CompanyDatabase;

use CompanyDatabase;

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

Phone VARCHAR(20)

);

INSERT INTO Customers (CustomerID, FirstName, LastName, Email, Phone)

VALUES

('1','John', 'Doe', 'johndoe@example.com', '+1234567890'),

('2','Alice', 'Smith', 'alicesmith@example.com', '+9876543210'),

('3','Bob', 'Johnson', 'bobjohnson@example.com', '+5555555555'),

('4','Eva', 'Brown', 'evabrown@example.com', '+1111111111'),

('5','David', 'Lee', 'davidlee@example.com', '+9999999999');

SELECT CustomerID, FirstName, LastName

FROM Customers

WHERE CustomerID >2

ORDER BY LastName ASC;

**Output:**

CustomerID FirstName LastName

4 Eva Brown

3 Bob Johnson

5 David Lee

**3.Create a table called "Employees" using the appropriate DDL command, specifying the necessary attributes and constraints.**

To create a table called "Employees" with the appropriate attributes and constraints using SQL Data Definition Language (DDL), you can use the CREATE TABLE statement. Here's an example of creating an "Employees" table with commonly used attributes and constraints:

create database CompanyDatabase;

use CompanyDatabase;

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(50) NOT NULL,

Location VARCHAR(100)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE,

Phone VARCHAR(20),

HireDate DATE,

Salary DECIMAL(10, 2),

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

**Output :**

Commands completed successfully.

Completion time: 2023-09-27T12:52:41.8811314+05:30

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **EmployeeID** | **FirstName** | **LastName** | **Email** | **Phone** | **HireDate** | **Salary** | **DepartmentID** |

**4.** **Perform an UPDATE operation in SQL to modify the "Salary" column of the "Employees" table for all employees with a "JobTitle" of "Manager."**

create database CompanyDatabase;

use CompanyDatabase;

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(50) NOT NULL,

Location VARCHAR(100)

);

INSERT INTO Departments (DepartmentID, DepartmentName, Location)

VALUES

(1, 'HR Department', 'New York'),

(2, 'Marketing Department', 'Los Angeles'),

(3, 'IT Department', 'San Francisco');

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE,

Phone VARCHAR(20),

HireDate DATE,

Salary DECIMAL(10, 2),

JobTitle VARCHAR(50),

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Phone, HireDate, Salary, JobTitle, DepartmentID)

VALUES

(1, 'John', 'Doe', 'john.doe@example.com', '+1234567890', '2023-01-15', 60000.00, 'Manager', 1),

(2, 'Jane', 'Smith', 'jane.smith@example.com', '+9876543210', '2022-03-20', 55000.00, 'Sales Associate', 2),

(3, 'Mike', 'Johnson', 'mike.johnson@example.com', '+5555555555', '2023-04-10', 65000.00, 'Manager', 1),

(4, 'Alice', 'Brown', 'alice.brown@example.com', '+1111111111', '2022-12-05', 62000.00, 'HR Specialist', 3),

(5, 'Bob', 'Williams', 'bob.williams@example.com', '+9999999999', '2023-02-28', 58000.00, 'Sales Associate', 2),

(6, 'Eva', 'Davis', 'eva.davis@example.com', '+7777777777', '2023-05-15', 70000.00, 'Manager', 1),

(7, 'David', 'Clark', 'david.clark@example.com', '+8888888888', '2022-06-18', 63000.00, 'Sales Associate', 2),

(8, 'Linda', 'Lee', 'linda.lee@example.com', '+6666666666', '2022-11-30', 59000.00, 'HR Specialist', 3),

(9, 'Sarah', 'Turner', 'sarah.turner@example.com', '+4444444444', '2023-03-25', 66000.00, 'Sales Associate', 2),

(10, 'Kevin', 'Anderson', 'kevin.anderson@example.com', '+2222222222', '2022-07-10', 64000.00, 'Manager', 1);

UPDATE Employees

SET Salary = Salary + (Salary \* 10)

WHERE JobTitle = 'Manager';

select \* from Employees;

**Output:**

EmployeeID FirstName LastName Email Phone HireDate Salary

JobTitle DepartmentID

1 John Doe john.doe@example.com +1234567890 2023-01-15 660000.00 Manager 1

2 Jane Smith jane.smith@example.com +9876543210 2022-03-20 55000.00 Sales Associate 2

3 Mike Johnson mike.johnson@example.com +5555555555 2023-04-10 715000.00 Manager 1

4 Alice Brown alice.brown@example.com +1111111111 2022-12-05 62000.00 HR Specialist 3

5 Bob Williams bob.williams@example.com +9999999999 2023-02-28 58000.00 Sales Associate 2

6 Eva Davis eva.davis@example.com +7777777777 2023-05-15 770000.00 Manager 1

7 David Clark david.clark@example.com +8888888888 2022-06-18 63000.00 Sales Associate 2

8 Linda Lee linda.lee@example.com +6666666666 2022-11-30 59000.00 HR Specialist 3

9 Sarah Turner sarah.turner@example.com +4444444444 2023-03-25 66000.00 Sales Associate 2

10 Kevin Anderson kevin.anderson@example.com +2222222222 2022-07-10 704000.00 Manager 1

**5. Write an SQL query to retrieve employee records where the "LastName" starts with the letter "S" and the "City" is either "New York" or "London."**

create database CompanyDatabase;

use CompanyDatabase;

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(50) NOT NULL,

Location VARCHAR(100)

);

INSERT INTO Departments (DepartmentID, DepartmentName, Location)

VALUES

(1, 'HR Department', 'New York'),

(2, 'Marketing Department', 'Los Angeles'),

(3, 'IT Department', 'San Francisco');

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE,

Phone VARCHAR(20),

city VARCHAR(50),

HireDate DATE,

Salary DECIMAL(10, 2),

JobTitle VARCHAR(50),

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Phone, City, HireDate, Salary, JobTitle, DepartmentID)

VALUES

(1, 'John', 'Smith', 'john.smith@example.com', '555-123-4567', 'New York', '2023-01-15', 60000.00, 'Manager', 1),

(2, 'Jane', 'Doe', 'jane.doe@example.com', '555-234-5678', 'London', '2022-11-20', 55000.00, 'Manager', 2),

(3, 'Michael', 'Johnson', 'michael.johnson@example.com', '555-345-6789', 'New York', '2022-09-10', 52000.00, 'Analyst', 1),

(4, 'Emily', 'Brown', 'emily.brown@example.com', '555-456-7890', 'London', '2023-03-05', 48000.00, 'Analyst', 2),

(5, 'David', 'Wilson', 'david.wilson@example.com', '555-567-8901', 'New York', '2022-07-02', 55000.00, 'Manager', 1),

(6, 'Olivia', 'Lee', 'olivia.lee@example.com', '555-678-9012', 'London', '2023-02-10', 50000.00, 'Analyst', 2),

(7, 'James', 'Anderson', 'james.anderson@example.com', '555-789-0123', 'New York', '2022-12-18', 49000.00, 'Analyst', 1),

(8, 'Sophia', 'Martin', 'sophia.martin@example.com', '555-890-1234', 'London', '2022-08-25', 53000.00, 'Manager', 2),

(9, 'William', 'Clark', 'william.clark@example.com', '555-901-2345', 'New York', '2023-04-15', 52000.00, 'Analyst', 1),

(10, 'Ava', 'Turner', 'ava.turner@example.com', '555-012-3456', 'London', '2022-10-30', 48000.00, 'Analyst', 2);

select \* from Employees;

**Output 1:**

EmployeeID FirstName LastName Email Phone city HireDate Salary JobTitle DepartmentID

1 John Smith john.smith@example.com 555-123-4567 New York 2023-01-15 60000.00 Manager 1

2 Jane Doe jane.doe@example.com 555-234-5678 London 2022-11-20 55000.00 Manager 2

3 Michael Johnson michael.johnson@example.com 555-345-6789 New York 2022-09-10 52000.00 Analyst 1

4 Emily Brown emily.brown@example.com 555-456-7890 London 2023-03-05 48000.00 Analyst 2

5 David Wilson david.wilson@example.com 555-567-8901 New York 2022-07-02 55000.00 Manager 1

6 Olivia Lee olivia.lee@example.com 555-678-9012 London 2023-02-10 50000.00 Analyst 2

7 James Anderson james.anderson@example.com 555-789-0123 New York 2022-12-18 49000.00 Analyst 1

8 Sophia Martin sophia.martin@example.com 555-890-1234 London 2022-08-25 53000.00 Manager 2

9 William Clark william.clark@example.com 555-901-2345 New York 2023-04-15 52000.00 Analyst 1

10 Ava Turner ava.turner@example.com 555-012-3456 London 2022-10-30 48000.00 Analyst 2

Now,

SELECT \*

FROM Employees

WHERE LastName LIKE 'S%'

AND (City = 'New York' OR City = 'London');

**Output 2:**

EmployeeID FirstName LastName Email Phone city HireDate Salary JobTitle DepartmentID

1 John Smith john.smith@example.com 555-123-4567 New York 2023-01-15 60000.00 Manager 1

**6. Combine the results of two SQL queries using a set operation to retrieve the common records from two tables.**

To combine records from the "Employees" and "Departments" tables using the INTERSECT operator to retrieve common records, you would need to ensure that both tables have the same structure or at least share common columns. In your case, the "Employees" and "Departments" tables don't have identical columns for a straightforward INTERSECT.

However, if you want to retrieve common values from a specific column that exists in both tables, such as "DepartmentID," you can do so with a query like this:

-- Retrieve common DepartmentID values from Employees and Departments

SELECT DepartmentID

FROM Employees

INTERSECT

SELECT DepartmentID

FROM Departments;

**Output:**

**DepartmentID**

**1**

**2**

**7.Calculate the average salary of all employees using aggregate operators and functions.**

To calculate the average salary of all employees using aggregate operators and functions, you can use the SQL AVG function. Here's an example query:

SELECT AVG(Salary) AS AverageSalary

FROM Employees;

In this query:

AVG(Salary) calculates the average of the "Salary" column for all rows in the "Employees" table.

AS AverageSalary assigns the result of the AVG function to an alias called "AverageSalary" for better readability in the output.

**Output:**

**AverageSalary**

**52200.000000**

**8.Retrieve all records from a table named "Orders" where the "OrderDate" is between '2023-01-01' and '2023-12-31.'**

create database CompanyDatabase;

use CompanyDatabase;

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

TotalAmount DECIMAL(10, 2)

);

INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount)

VALUES

(1, 101, '2021-09-01', 150.99),

(2, 102, '2022-09-02', 220.50),

(3, 103, '2022-09-03', 75.75),

(4, 104, '2023-09-04', 320.25),

(5, 105, '2023-09-05', 180.00),

(6, 101, '2023-09-06', 85.50),

(7, 106, '2023-09-07', 420.99),

(8, 102, '2023-09-08', 130.75),

(9, 107, '2023-09-09', 240.25),

(10, 108, '2023-09-10', 300.00);

select \* from Orders;

**Ordertable:**

**OrderID CustomerID OrderDate TotalAmount**

1 101 2021-09-01 150.99

2 102 2022-09-02 220.50

3 103 2022-09-03 75.75

4 104 2023-09-04 320.25

5 105 2023-09-05 180.00

6 101 2023-09-06 85.50

7 106 2023-09-07 420.99

8 102 2023-09-08 130.75

9 107 2023-09-09 240.25

10 108 2023-09-10 300.00

Now,

SELECT \*

FROM Orders

WHERE OrderDate BETWEEN '2023-01-01' AND '2023-12-31';

**Output:**

**OrderID CustomerID OrderDate TotalAmount**

4 104 2023-09-04 320.25

5 105 2023-09-05 180.00

6 101 2023-09-06 85.50

7 106 2023-09-07 420.99

8 102 2023-09-08 130.75

9 107 2023-09-09 240.25

10 108 2023-09-10 300.00

**9.Use SQL date functions to extract the month and year from a given date column.**

You can use SQL date functions to extract the month and year from a given date column. SQL provides functions like MONTH() and YEAR() for this purpose. Here's an example:

Suppose you have a table called "Orders" with an "OrderDate" column, and you want to extract the month and year from it:

SELECT

OrderDate,

MONTH(OrderDate) AS OrderMonth,

YEAR(OrderDate) AS OrderYear

FROM Orders;

**Output:**

**OrderDate OrderMonth OrderYear**

2021-09-01 9 2021

2022-09-02 9 2022

2022-09-03 9 2022

2023-09-04 9 2023

2023-09-05 9 2023

2023-09-06 9 2023

2023-09-07 9 2023

2023-09-08 9 2023

2023-09-09 9 2023

2023-09-10 9 2023

**10. Write a nested subquery in SQL to retrieve the employees who earn a higher salary than the average salary of all employees.**

You can use a nested subquery in SQL to retrieve employees who earn a higher salary than the average salary of all employees. Here's an example query to do that:

SELECT EmployeeID, FirstName, LastName, Salary

FROM Employees

WHERE Salary > (

SELECT AVG(Salary)

FROM Employees

);

**Output :**

**EmployeeID FirstName LastName Salary**

1 John Smith 60000.00

2 Jane Doe 55000.00

5 David Wilson 55000.00

8 Sophia Martin 53000.00

**PL/pgSQL**

**11.Write a PL/pgSQL program that displays the message "Hello, World!" using the RAISE NOTICE statement.**

**Code:**

CREATE OR REPLACE FUNCTION hello\_world() RETURNS VOID AS $$

BEGIN

RAISE NOTICE 'Hello, World!';

END;

$$ LANGUAGE plpgsql;

SELECT hello\_world();

**Output:**

NOTICE: Hello, World!

Successfully run.

Total query runtime: 30 msec. 1 rows affected.

**12.Write a PL/pgSQL program that takes two numbers as input parameters and calculates their sum, difference, product, and quotient. Display the results using the RAISE NOTICE statement.**

**Code:**

CREATE OR REPLACE FUNCTION calculate\_operations(

IN num1 NUMERIC,

IN num2 NUMERIC

) RETURNS VOID AS $$

DECLARE

sum\_result NUMERIC;

difference\_result NUMERIC;

product\_result NUMERIC;

quotient\_result NUMERIC;

BEGIN

-- Calculate the results

sum\_result := num1 + num2;

difference\_result := num1 - num2;

product\_result := num1 \* num2;

-- Check for division by zero and calculate the quotient

IF num2 = 0 THEN

RAISE NOTICE 'Division by zero is not allowed.';

ELSE

quotient\_result := num1 / num2;

END IF;

-- Display the results using RAISE NOTICE

RAISE NOTICE 'Sum: %', sum\_result;

RAISE NOTICE 'Difference: %', difference\_result;

RAISE NOTICE 'Product: %', product\_result;

RAISE NOTICE 'Quotient: %', quotient\_result;

END;

$$ LANGUAGE plpgsql;

select calculate\_operations(20,5)

**Output:**

NOTICE: Sum: 25

NOTICE: Difference: 15

NOTICE: Product: 100

NOTICE: Quotient: 4.0000000000000000

Successfully run. Total query runtime: 56 msec.

1 rows affected.

**13.Write a PL/pgSQL program that takes a number as input and determines whether it is even or odd. Display the result using the RAISE NOTICE statement.**

**Code:**

CREATE OR REPLACE FUNCTION determine\_even\_or\_odd(

IN num\_input INTEGER

) RETURNS VOID AS $$

DECLARE

result\_text TEXT;

BEGIN

-- Determine if the number is even or odd

IF num\_input % 2 = 0 THEN

result\_text := 'Even';

ELSE

result\_text := 'Odd';

END IF;

-- Display the result using RAISE NOTICE

RAISE NOTICE 'The number % is %.', num\_input, result\_text;

END;

$$ LANGUAGE plpgsql;

SELECT determine\_even\_or\_odd(24);

**Output:**

NOTICE: The number 24 is Even.

Successfully run. Total query runtime: 33 msec.

1 rows affected.

**14.Write a PL/pgSQL program that takes a number as input and calculates its factorial. Display the result using the RAISE NOTICE statement.**

**Code:**

CREATE OR REPLACE FUNCTION calculate\_factorial(

IN num\_input INTEGER

) RETURNS BIGINT AS $$

DECLARE

result BIGINT;

BEGIN

-- Base case: Factorial of 0 is 1

IF num\_input = 0 THEN

result := 1;

ELSE

-- Recursive case

result := num\_input \* calculate\_factorial(num\_input - 1);

END IF;

-- Display the result using RAISE NOTICE

RAISE NOTICE 'The factorial of % is %.', num\_input, result;

RETURN result;

END;

$$ LANGUAGE plpgsql;

select calculate\_factorial(7);

**Output:**

NOTICE: The factorial of 0 is 1.

NOTICE: The factorial of 1 is 1.

NOTICE: The factorial of 2 is 2.

NOTICE: The factorial of 3 is 6.

NOTICE: The factorial of 4 is 24.

NOTICE: The factorial of 5 is 120.

NOTICE: The factorial of 6 is 720.

NOTICE: The factorial of 7 is 5040.

Successfully run. Total query runtime: 32 msec.

1 rows affected.

**15. Write a PL/pgSQL program that generates the Fibonacci series up to a given number. Display the series using the RAISE NOTICE statement**

**Code:**

CREATE OR REPLACE FUNCTION generate\_fibonacci\_series(

IN max\_value INTEGER

) RETURNS VOID AS $$

DECLARE

a BIGINT := 0;

b BIGINT := 1;

next\_term BIGINT;

BEGIN

-- Display the first two terms of the series

RAISE NOTICE 'Fibonacci Series:';

RAISE NOTICE '%', a;

RAISE NOTICE '%', b;

-- Generate and display the rest of the series

WHILE (a + b) <= max\_value LOOP

next\_term := a + b;

RAISE NOTICE '%', next\_term;

a := b;

b := next\_term;

END Loop;

END;

$$ LANGUAGE plpgsql;

SELECT generate\_fibonacci\_series(150);

**Output:**

NOTICE: Fibonacci Series:

NOTICE: 0

NOTICE: 1

NOTICE: 1

NOTICE: 2

NOTICE: 3

NOTICE: 5

NOTICE: 8

NOTICE: 13

NOTICE: 21

NOTICE: 34

NOTICE: 55

NOTICE: 89

NOTICE: 144

Successfully run. Total query runtime: 46 msec.

1 rows affected.

**16. Write a PL/pgSQL program that takes a year as input and determines whether it is a leap year or not. Display the result using the RAISE NOTICE statement.**

**Code:**

CREATE OR REPLACE FUNCTION check\_leap\_year(

IN input\_year INTEGER

) RETURNS VOID AS $$

DECLARE

is\_leap\_year BOOLEAN;

BEGIN

is\_leap\_year := (

(input\_year % 4 = 0 AND input\_year % 100 != 0) OR

(input\_year % 400 = 0)

);

IF is\_leap\_year THEN

RAISE NOTICE '% is a leap year.', input\_year;

ELSE

RAISE NOTICE '% is not a leap year.', input\_year;

END IF;

END;

$$ LANGUAGE plpgsql;

SELECT check\_leap\_year(2016);

**Output:**

NOTICE: 2016 is a leap year.

Successfully run. Total query runtime: 32 msec.

1 rows affect**17.Write a PL/pgSQL program that takes a number as input and checks whether it is a prime number. Display the result using the RAISE NOTICE statement.**

**Code:**

CREATE OR REPLACE FUNCTION is\_prime\_number(

IN num\_input INTEGER

) RETURNS VOID AS $$

DECLARE

is\_prime BOOLEAN := TRUE;

divisor INTEGER;

BEGIN

-- Check if num\_input is less than 2

IF num\_input <= 1 THEN

is\_prime := FALSE;

ELSE

-- Check for factors between 2 and the square root of num\_input

divisor := 2;

WHILE divisor \* divisor <= num\_input LOOP

IF num\_input % divisor = 0 THEN

is\_prime := FALSE;

EXIT;

END IF;

divisor := divisor + 1;

END Loop;

END IF;

IF is\_prime THEN

RAISE NOTICE '% is a prime number.', num\_input;

ELSE

RAISE NOTICE '% is not a prime number.', num\_input;

END IF;

END;

$$ LANGUAGE plpgsql;

SELECT is\_prime\_number(30);

**Output:**

NOTICE: 30 is not a prime number.

Successfully run. Total query runtime: 34 msec.

1 rows affected.

**18.Write a PL/pgSQL program that takes a string as input and checks whether it is a palindrome (reads the same forwards and backwards). Display the result using the RAISE NOTICE statement.**

**Code:**

CREATE OR REPLACE FUNCTION is\_palindrome(

IN input\_string TEXT

) RETURNS VOID AS $$

DECLARE

reversed\_string TEXT;

BEGIN

-- Reverse the input string

reversed\_string := REVERSE(input\_string);

-- Check if the reversed string is equal to the original input string

IF input\_string = reversed\_string THEN

RAISE NOTICE '% is a palindrome.', input\_string;

ELSE

RAISE NOTICE '% is not a palindrome.', input\_string;

END IF;

END;

$$ LANGUAGE plpgsql;

SELECT is\_palindrome('45654');

**Output:**

NOTICE: 45654 is a palindrome.

Successfully run. Total query runtime: 44 msec.

1 rows affected.

**19.Write a PL/pgSQL program that takes a string as input and reverses it. Display the reversed string using the RAISE NOTICE statement.**

**Code:**

CREATE OR REPLACE FUNCTION reverse\_string(

IN input\_string TEXT

) RETURNS TEXT AS $$

DECLARE

reversed\_string TEXT;

BEGIN

reversed\_string := REVERSE(input\_string);

-- Display the reversed string using RAISE NOTICE

RAISE NOTICE 'Original String: %', input\_string;

RAISE NOTICE 'Reversed String: %', reversed\_string;

RETURN reversed\_string;

END;

$$ LANGUAGE plpgsql;

SELECT reverse\_string('Shravani');

**Output:**

NOTICE: Original String: Shravani

NOTICE: Reversed String: inavarhS

Successfully run. Total query runtime: 49 msec.

1 rows affected.

**20.Write a PL/pgSQL program that takes multiple numbers as input and determines the maximum and minimum numbers among them. Display the results using the RAISE NOTICE statement.**

**Code:**

CREATE OR REPLACE FUNCTION find\_max\_min(

VARIADIC numbers NUMERIC[]

) RETURNS VOID AS $$

DECLARE

max\_num NUMERIC;

min\_num NUMERIC;

BEGIN

IF array\_length(numbers, 1) IS NULL OR array\_length(numbers, 1) = 0 THEN

RAISE NOTICE 'No numbers provided.';

ELSE

max\_num := numbers[1];

min\_num := numbers[1];

FOR i IN 2..array\_length(numbers, 1) LOOP

IF numbers[i] > max\_num THEN

max\_num := numbers[i];

ELSIF numbers[i] < min\_num THEN

min\_num := numbers[i];

END IF;

END LOOP;

RAISE NOTICE 'Maximum number: %', max\_num;

RAISE NOTICE 'Minimum number: %', min\_num;

END IF;

END;

$$ LANGUAGE plpgsql;

SELECT find\_max\_min(100, 55, 25, 15, 30);

**Output:**

NOTICE: Maximum number: 100

NOTICE: Minimum number: 15

Successfully run. Total query runtime: 48 msec.

1 rows affected.