Manav Rachna International Institute of Research and Studies

ORACLE LAB

Aqdas Ahmad 22/FCA/BCA(CS)/006

EXERCISE 1

AIM: Create the following table

Customer

Column_name	Data type	Size	Constraint
SID	Varchar2	4	Primary Key
First_Name	Char	20	
Last_name	Char	20	

Orders

Column_name	Data type	Size	Constraint
Order_ID	Varchar2	4	Primary Key
Order_date	Char	20	
Customer_SID	Varchar2	20	Foreign Key
Amount	Number		Check > 20000

Table Creation:

Programiz

Online SQL Editor

Customer [-]
- SID [varchar2(4)]
- First_Name [char(20)]
- Last_Name [char(20)]

Orders [-]
- Order_ID [varchar2(4)]
- Order_Date [char(20)]
- Customer_SID [varchar2(20)]
- Amount [number]

Input

Run SQL

```
-- Create Customer Table
CREATE TABLE Customer (
    SID          VARCHAR2(4) PRIMARY KEY,
    First_Name   CHAR(20),
    Last_Name    CHAR(20)
);
-- Create Orders Table
CREATE TABLE Orders (
    Order_ID      VARCHAR2(4) PRIMARY KEY,
    Order_Date    CHAR(20),
    Customer_SID  VARCHAR2(20),
    Amount        NUMBER CHECK (Amount > 20000),
    FOREIGN KEY (Customer_SID) REFERENCES Customer(SID)
);
```

Output

SQL query successfully executed. However, the result set is empty.

Output:

Customer

SID	First_Name	Last_Name
empty		

Orders

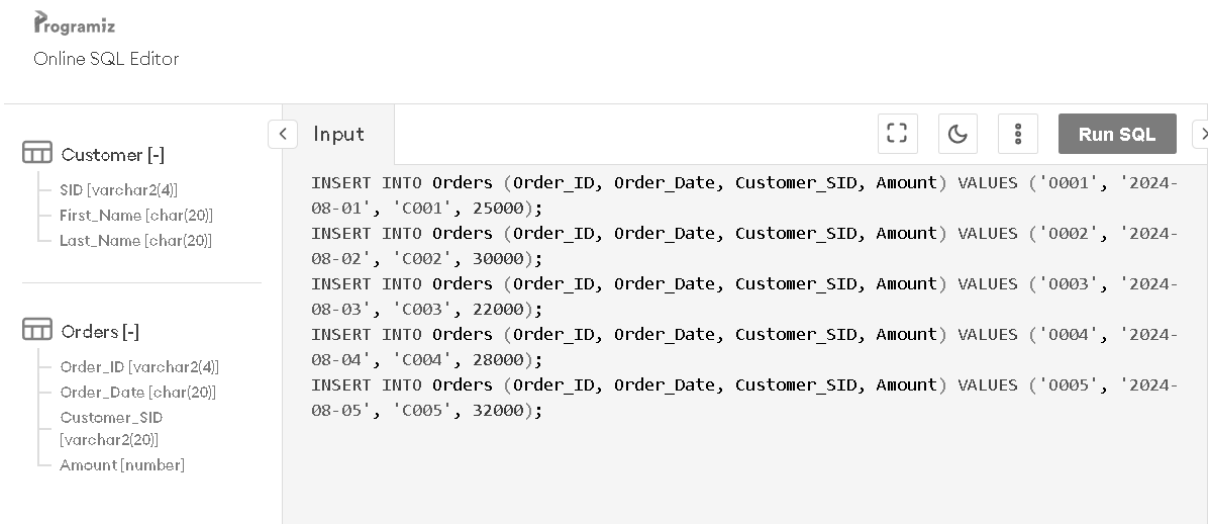
Order_ID	Order_Date	Customer_SID	Amount
empty			

EXERCISE 2

AIM: Insert 5 records for each table.

Inserting record in Customer ->

Inserting records Order ->



Output:

Table output for Customer

Customer

SID	First_Name	Last_Name
C001	Anant	Verma
C002	Virat	Sharma
C003	Mahika	Arora
C004	Walter	Black
C005	Emilia	Khan

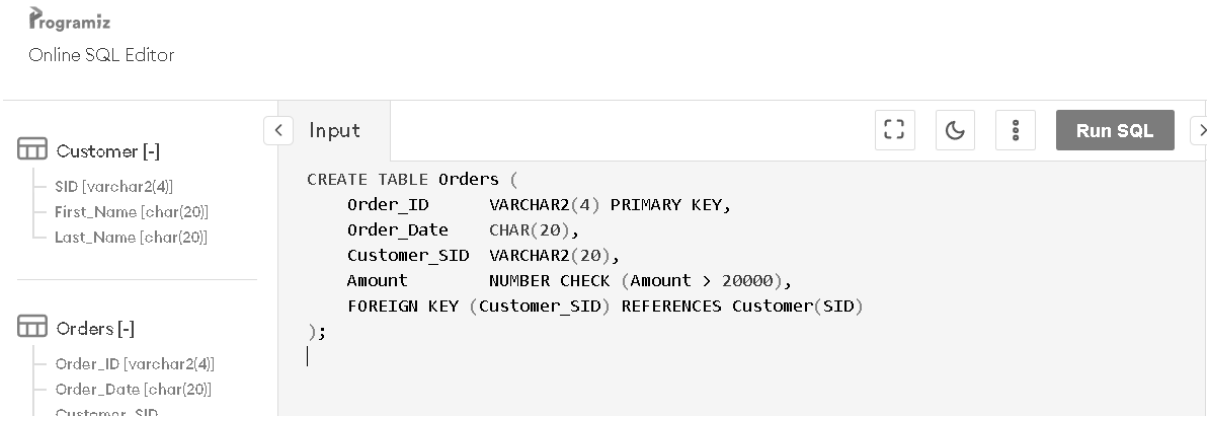
Table output for Orders

Orders

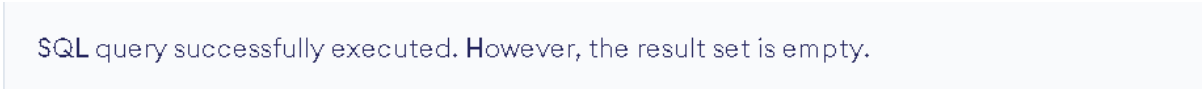
Order_ID	Order_Date	Customer_SID	Amount
O001	2024-08-01	C001	25000
O002	2024-08-02	C002	30000
O003	2024-08-03	C003	22000
O004	2024-08-04	C004	28000
O005	2024-08-05	C005	32000

EXERCISE 3

AIM: Customer SID column in the ORDERS table is a foreign key pointing to the SID column in the CUSTOMER table.



Output:

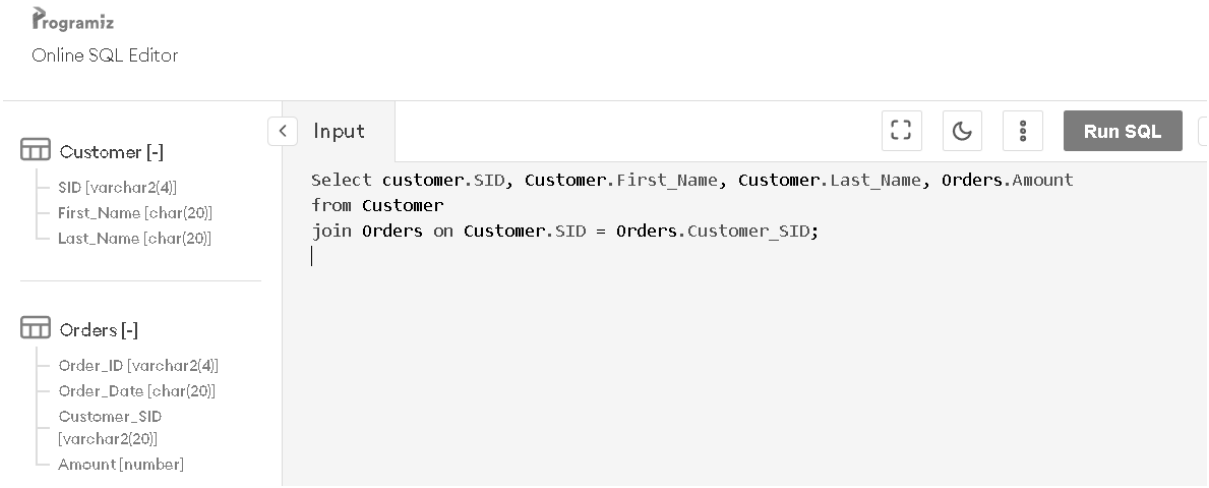


In this code:

- **The 'Customer_SID' column is a foreign key that links each order to a specific customer.**
- **The foreign key constraint ensures that any 'Customer_SID' in the Orders table must match an existing SID in the Customer table. This prevents orders from being associated with non-existent customers.**

EXERCISE 4

AIM: List the details of the customers along with the amount.

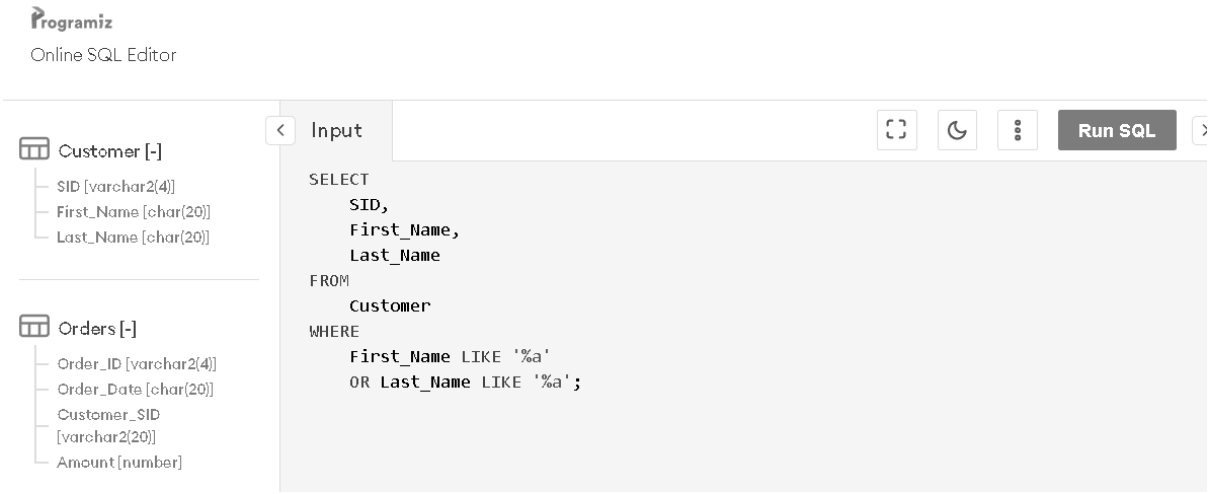


Output:

SID	First_Name	Last_Name	Amount
C001	Anant	Verma	25000
C002	Virat	Sharma	30000
C003	Mahika	Arora	22000
C004	Walter	Black	28000
C005	Emilia	Khan	32000

EXERCISE 5

AIM: List the customers whose names end with “a”.

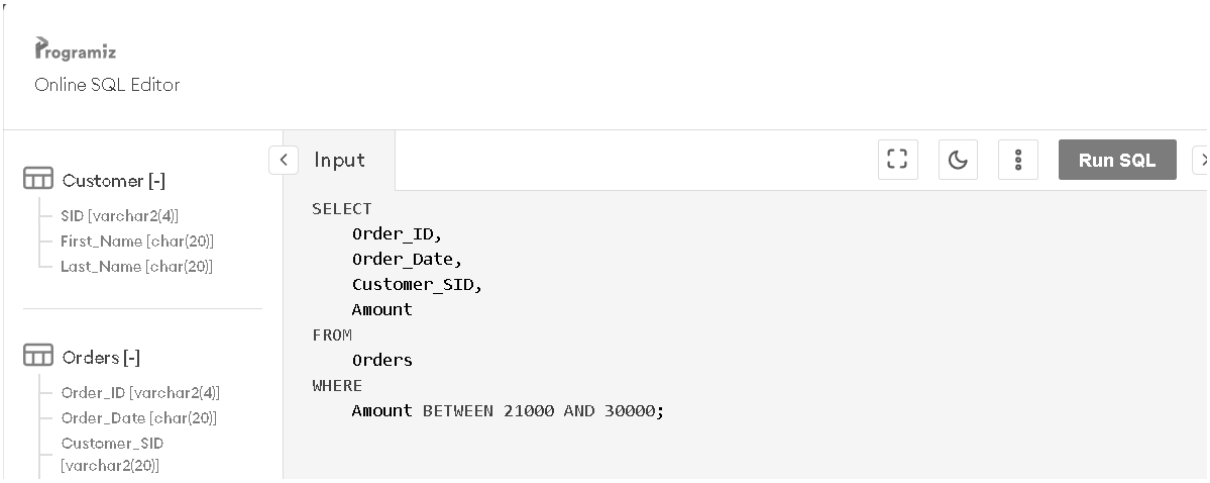


Output:

SID	First_Name	Last_Name
C001	Anant	Verma
C002	Virat	Sharma
C003	Mahika	Arora
C005	Emilia	Khan

EXERCISE 6

AIM: List the orders where amount is between 21000 and 30000

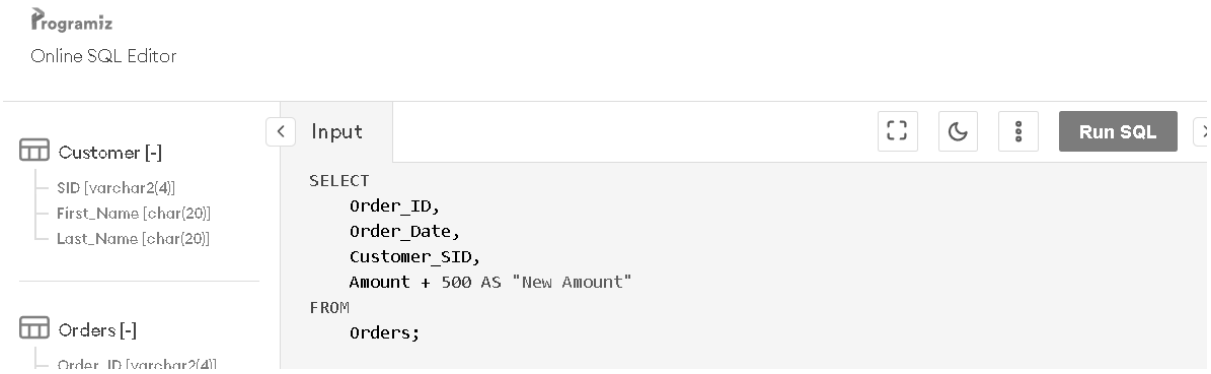


Output:

Order_ID	Order_Date	Customer_SID	Amount
O001	2024-08-01	C001	25000
O002	2024-08-02	C002	30000
O003	2024-08-03	C003	22000
O004	2024-08-04	C004	28000

EXERCISE 7

AIM: List the orders where amount is increased by 500 and replace with name “new amount”

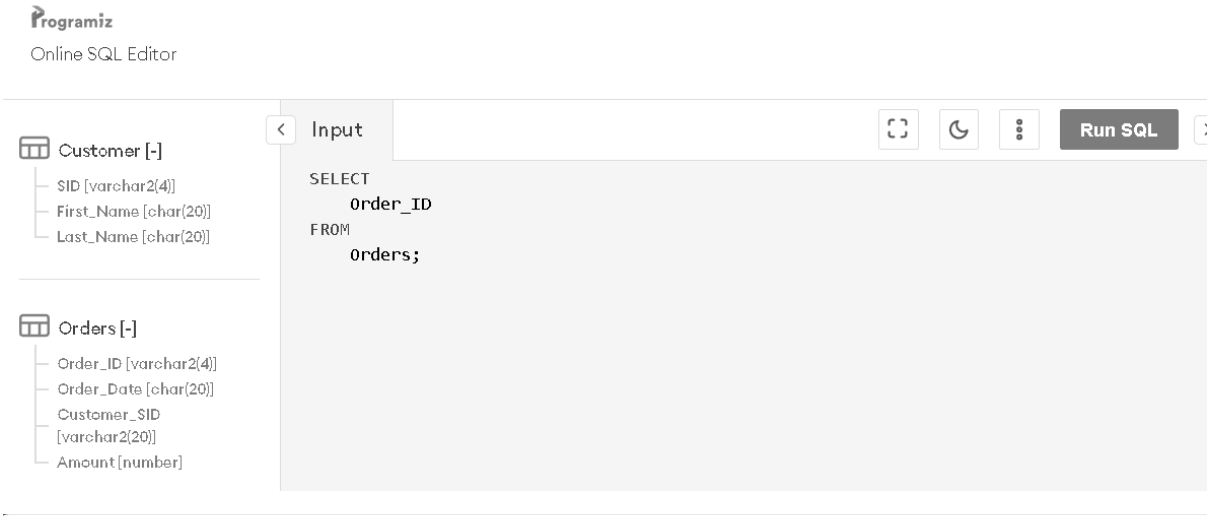


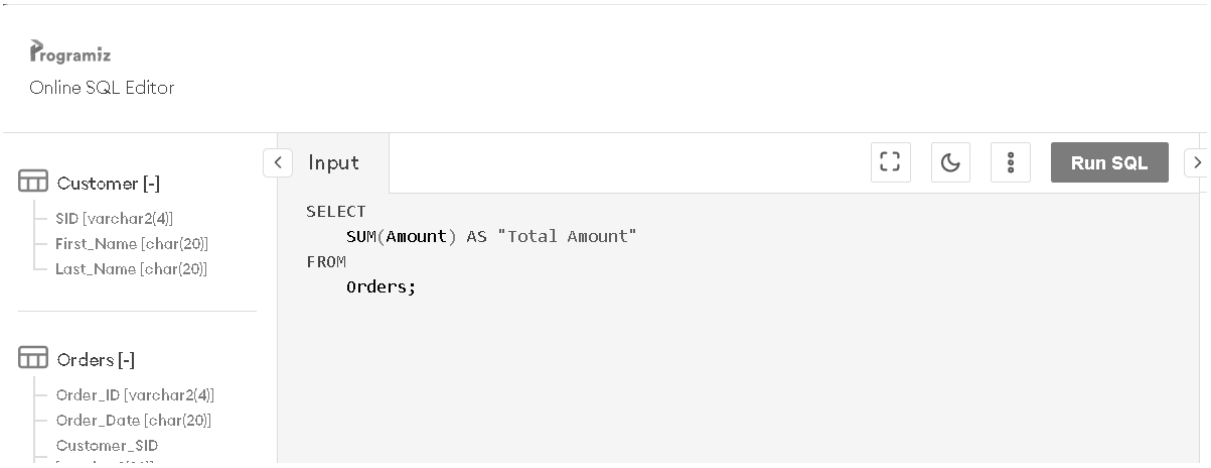
Output:

Order_ID	Order_Date	Customer_SID	New Amount
O001	2024-08-01	C001	25500
O002	2024-08-02	C002	30500
O003	2024-08-03	C003	22500
O004	2024-08-04	C004	28500
O005	2024-08-05	C005	32500

EXERCISE 8

AIM: Display the order_id and total amount of orders.





Output:

Table output for Order_ID from Orders ->

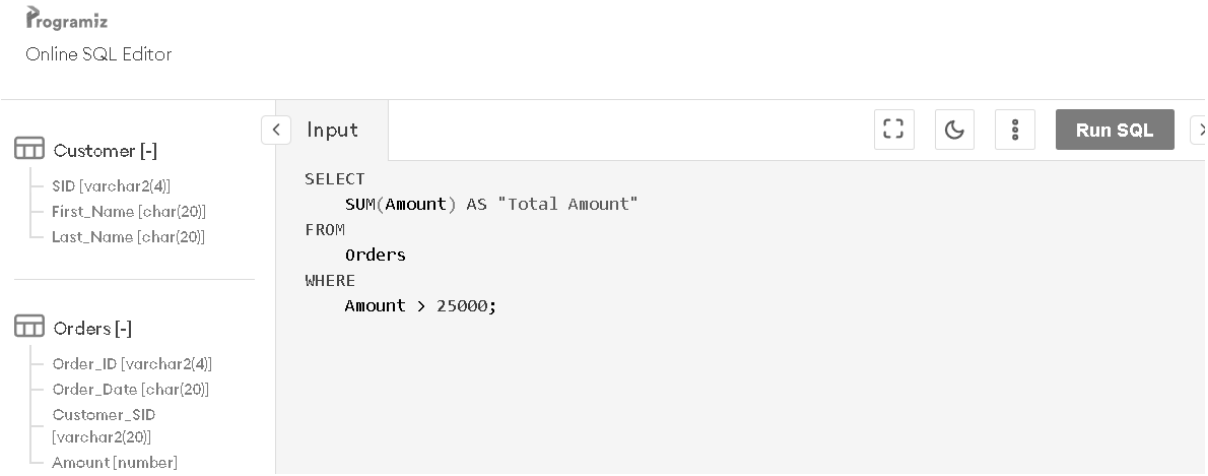
Order_ID
O001
O002
O003
O004
O005

Output for Total amount of all Orders ->

Total Amount
137000

EXERCISE 9

AIM: Calculate the total amount of orders that has more than 25000.



Output:

Total Amount
90000

EXERCISE 10

AIM: Display all the string functions used in SQL.

1. LENGTH (string)

- **Returns the length of a string.**

2. LOWER (string)

- **Converts all characters in a string to lowercase.**

3. UPPER (string)

- **Converts all characters in a string to uppercase.**

4. SUBSTR (string, start_position, length)

- **Extracts a substring from a string starting at a given position for a specified length.**

5. INSTR (string, substring)

- **Returns the position of the first occurrence of a substring within a string.**

6. TRIM ([LEADING | TRAILING | BOTH] trim_character FROM string)

- **Removes specified characters from the beginning (LEADING), end (TRAILING), or both ends (BOTH) of a string. By default, it removes spaces.**

7. LTRIM (string)

- **Removes leading spaces from a string.**

8. RTRIM (string)

- **Removes trailing spaces from a string.**

9. REPLACE (string, search_string, replace_string)

- **Replaces occurrences of a substring within a string with another substring.**

10. CONCAT (string1, string2)

- **Concatenates two or more strings together.**

11. LPAD (string, length, pad_string)

- **Pads the left side of a string with a specified character up to a certain length.**

12. RPAD (string, length, pad_string)

- **Pads the right side of a string with a specified character up to a certain length.**

13. LEFT (string, number_of_characters)

- **Returns the leftmost n characters from a string.**

14. RIGHT (string, number_of_characters)

- **Returns the rightmost n characters from a string.**

15. ASCII (character)

- **Returns the ASCII code of the first character in a string.**

16. CHR (ascii_code)

- **Converts an ASCII code to its corresponding character.**

17. INITCAP (string)

- **Converts the first letter of each word in a string to uppercase and the rest to lowercase.**

18. REVERSE (string) (Available in some SQL variants like SQL Server)

- **Reverses the characters in a string.**

19. POSITION (substring IN string)

- **Returns the position of the first occurrence of a substring within a string.**

20. SOUNDEX (string)

- **Returns a string's phonetic representation, useful for comparing words that sound alike.**

21. DIFFERENCE (string1, string2) (Available in SQL Server)

- **Compares two strings and returns a value based on their phonetic similarity.**

22. FORMAT (number, format) (Available in SQL Server)

- **Returns a number formatted as a string, according to a specified format.**

23. TRANSLATE (string, from_chars, to_chars) (Available in Oracle SQL)

- **Replaces characters in a string with other characters based on their position.**

24. REPEAT (string, number) (Available in MySQL)

- **Repeats a string a specified number of times.**

25. SPACE (number) (Available in SQL Server)

- **Returns a string of spaces with the specified length.**

These functions are commonly used for manipulating and querying string data across different SQL databases. The availability of these functions may vary depending on the specific SQL database you are using.

EXERCISE 11

AIM: Create the following table:

Student

Column_name	Data type	Size	Constraint
RollNo	Varchar2	20	Primary Key
Name	Char	20	
Class	Varchar2	20	
Marks	Number	6,2	

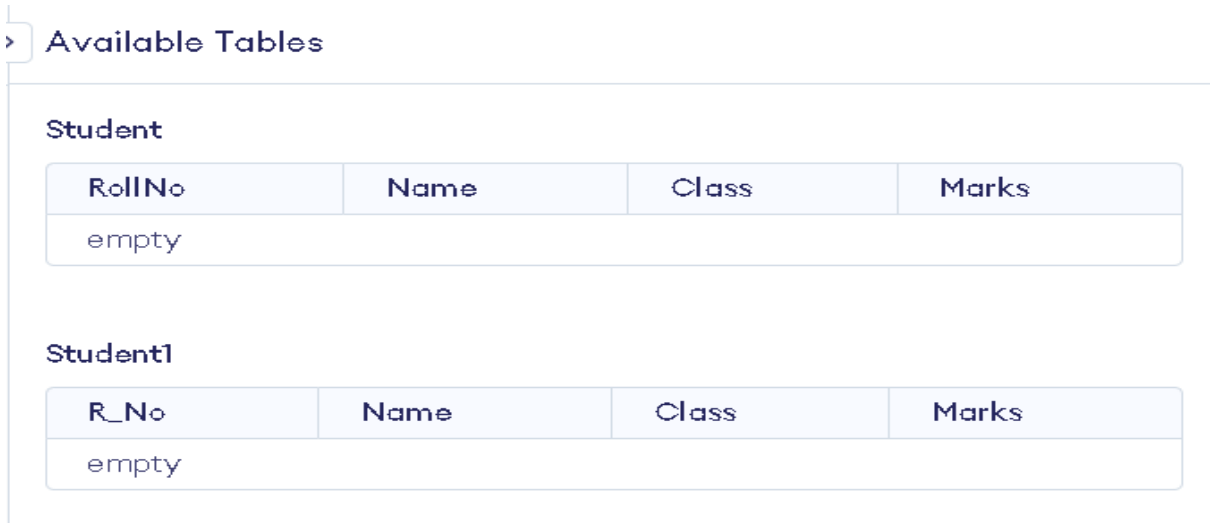
Student1

Column_name	Data type	Size	Constraint
R_No	Varchar2	20	Primary Key
Name	Char	20	
Class	Varchar2	20	
Marks	Number	6,2	

Table creation:

```
-- Create the Student table
CREATE TABLE IF NOT EXISTS Student (
    RollNo VARCHAR(20) PRIMARY KEY,  -- Primary key constraint
    Name CHAR(20),
    Class VARCHAR(20),
    Marks NUMBER(6, 2)
);

-- Create the Student1 table
CREATE TABLE IF NOT EXISTS Student1 (
    R_No VARCHAR(20) PRIMARY KEY,  -- Primary key constraint
    Name CHAR(20),
    Class VARCHAR(20),
    Marks NUMBER(6, 2)
);
```

Output:


Available Tables
Student
RollNo
Name
Class
Marks
empty
Student1
R_No
Name
Class
Marks
empty

EXERCISE 12

AIM: Display all the contents of student and student1 using union clause.

First insert 5 records in each table i.e. Student and Student1

```
-- Insert 5 records into the Student table
INSERT INTO Student (RollNo, Name, Class, Marks)
VALUES
    ('S101', 'John', '10th', 85.50),
    ('S102', 'Alice', '11th', 90.00),
    ('S103', 'Bob', '12th', 75.75),
    ('S104', 'Charlie', '10th', 88.00),
    ('S105', 'David', '11th', 92.50);

-- Insert 5 records into the Student1 table (with some common entries)
INSERT INTO Student1 (R_No, Name, Class, Marks)
VALUES
    ('S201', 'Eve', '10th', 80.25),
    ('S202', 'Frank', '12th', 70.50),
    ('S103', 'Bob', '12th', 75.75),   -- Common entry with Student
    ('S104', 'Charlie', '10th', 88.00), -- Common entry with Student
    ('S205', 'Isaac', '12th', 85.00);
```

Output:

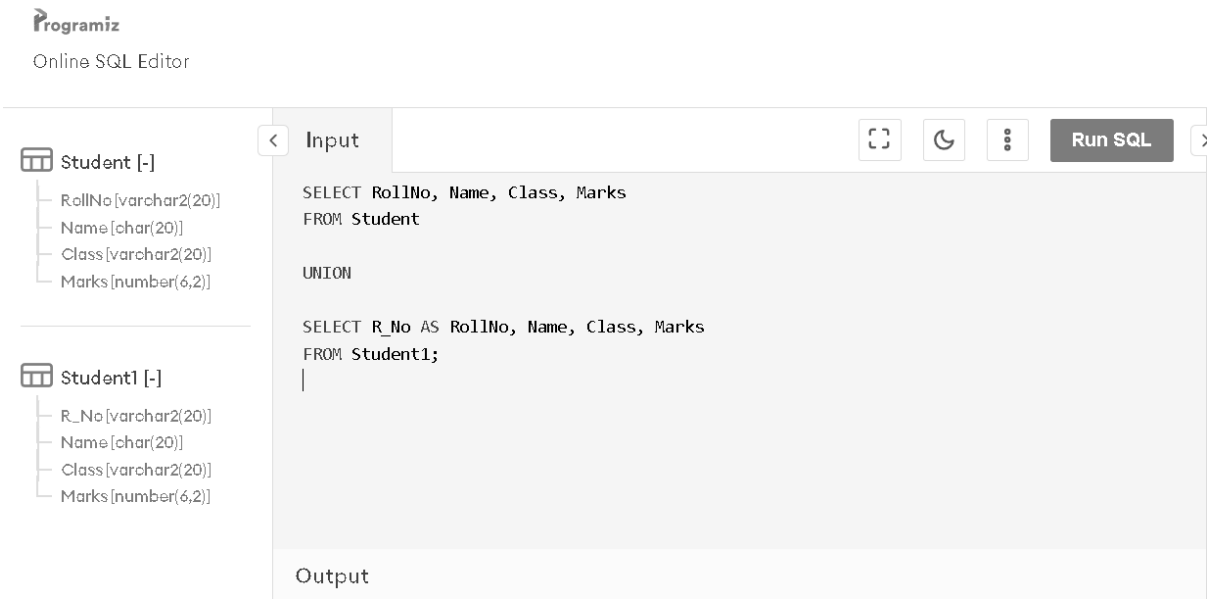
Student

RollNo	Name	Class	Marks
S101	John	10th	85.5
S102	Alice	11th	90
S103	Bob	12th	75.75
S104	Charlie	10th	88
S105	David	11th	92.5

Student1

R_No	Name	Class	Marks
S201	Eve	10th	80.25
S202	Frank	12th	70.5
S103	Bob	12th	75.75
S104	Charlie	10th	88
S205	Isaac	12th	85

Now union:



Output:

Output

RollNo	Name	Class	Marks
S101	John	10th	85.5
S102	Alice	11th	90
S103	Bob	12th	75.75
S104	Charlie	10th	88
S105	David	11th	92.5
S201	Eve	10th	80.25
S202	Frank	12th	70.5
S205	Isaac	12th	85

EXERCISE 13

AIM: Find out the intersection of student and student1 tables.

Input

Run SQL

```
SELECT RollNo, Name, Class, MArks
From Student

INTERSECT

SELECT R_No AS RollNo, Name, Class, Marks
FROM Student1;
```

Output | Available Tables

RollNo	Name	Class	Marks
S103	Bob	12th	75.75
S104	Charlie	10th	88

Programiz
Online SQL Editor

Student [-]
- RollNo [varchar2(20)]
- Name [char(20)]
- Class [varchar2(20)]
- Marks [number(6,2)]

Student1 [-]
- R_No [varchar2(20)]
- Name [char(20)]
- Class [varchar2(20)]
- Marks [number(6,2)]

Input

Run SQL

```
SELECT Student.Name AS Student_Name, Student1.Name AS Student1_Name
FROM Student
LEFT JOIN Student1 ON Student.RollNo = Student1.R_No;
```

Output

Student_Name	Student1_Name
Alice	
Bob	
Charlie	
David	
Eve	

EXERCISE 14

AIM: Display the names of student and student1 tables using left, right, inner and full join.

INNER JOIN

```
-- INNER JOIN to display names of students from both tables where there's a match
SELECT S.Name AS Student_Name, S1.Name AS Student1_Name
FROM Student S
INNER JOIN Student1 S1
ON S.Name = S1.Name;
```

Output

Student_Name	Student1_Name
Bob	Bob
Charlie	Charlie

LEFT JOIN AND RIGHT JOIN

```
-- LEFT JOIN to display names from Student and corresponding names from Student1
(if any)
SELECT S.Name AS Student_Name, S1.Name AS Student1_Name
FROM Student S
LEFT JOIN Student1 S1
ON S.Name = S1.Name;
```

```
-- RIGHT JOIN simulation: Swap tables and use LEFT JOIN to simulate RIGHT JOIN
SELECT S.Name AS Student_Name, S1.Name AS Student1_Name
FROM Student S
LEFT JOIN Student1 S1
ON S.Name = S1.Name;
```

Output

Student_Name	Student1_Name
John	
Alice	
Bob	Bob
Charlie	Charlie
David	

FULL JOIN

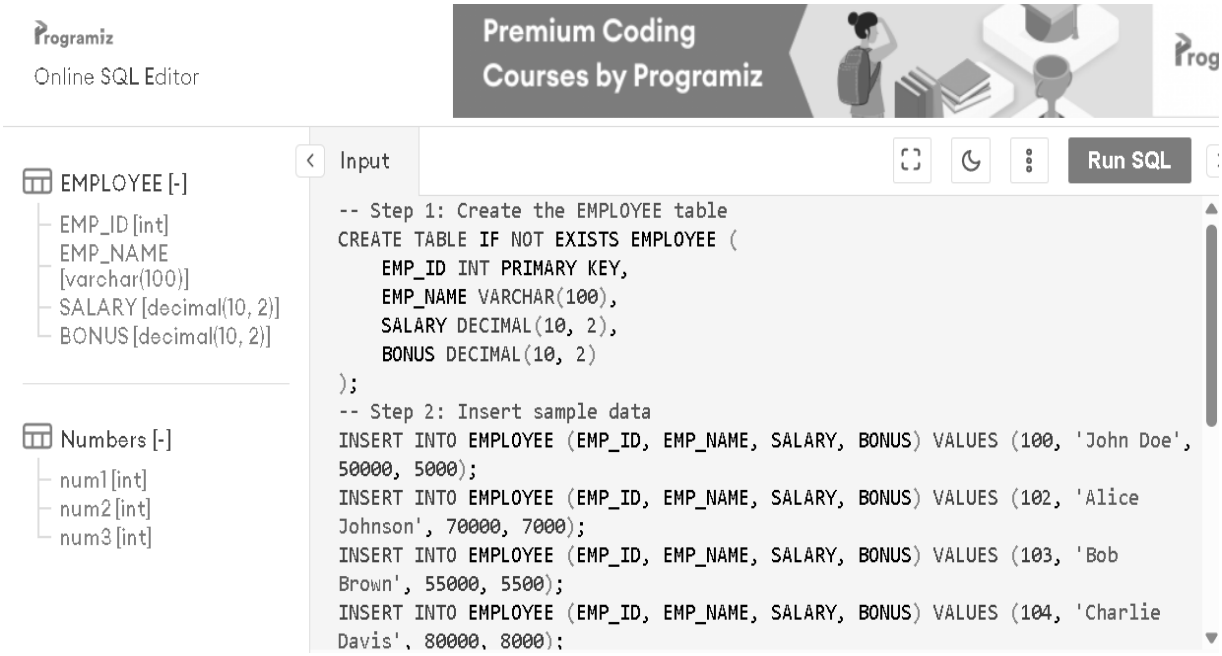
```
-- FULL JOIN simulation: Combine LEFT JOIN and RIGHT JOIN results
SELECT S.Name AS Student_Name, S1.Name AS Student1_Name
FROM Student S
LEFT JOIN Student1 S1
ON S.Name = S1.Name

UNION

SELECT S.Name AS Student_Name, S1.Name AS Student1_Name
FROM Student S
LEFT JOIN Student1 S1
ON S.Name = S1.Name;
```

Output

Student_Name	Student1_Name
Alice	
Bob	Bob
Charlie	Charlie
David	
John	

Exercise 15

AIM: To Write a PL/SQL block to calculate total salary of employee having employee number 100.




```
INSERT INTO EMPLOYEE (EMP_ID, EMP_NAME, SALARY, BONUS) VALUES (105, 'Diana Prince', 90000, 9000);
INSERT INTO EMPLOYEE (EMP_ID, EMP_NAME, SALARY, BONUS) VALUES (106, 'Evan Thomas', 45000, 4500);

-- Step 3: Calculate total salary for employee with EMP_ID = 100
SELECT
    EMP_ID,
    EMP_NAME,
    (SALARY + BONUS) AS TOTAL_SALARY
FROM
    EMPLOYEE
WHERE
    EMP_ID = 100;;
```

OUTPUT:

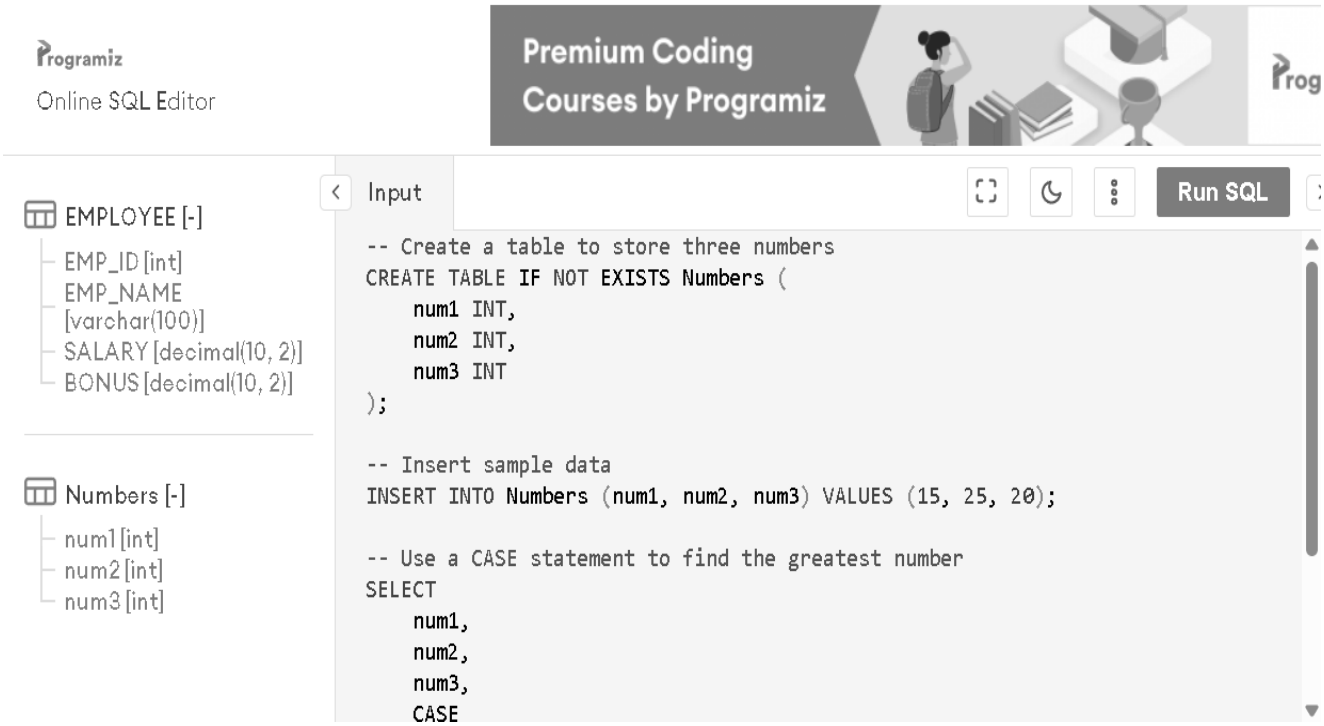
EMPLOYEE

EMP_ID	EMP_NAME	SALARY	BONUS
100	John Doe	50000	5000
102	Alice Johnson	70000	7000
103	Bob Brown	55000	5500
104	Charlie Davis	80000	8000
105	Diana Prince	90000	9000
106	Evan Thomas	45000	4500

EMP_ID	EMP_NAME	TOTAL_SALARY
100	John Doe	55000

EXERCISE 16

AIM: To Write a PL/SQL code to find the greatest of three numbers.




```
        WHEN num1 >= num2 AND num1 >= num3 THEN num1
        WHEN num2 >= num1 AND num2 >= num3 THEN num2
        ELSE num3
    END AS greatest_number
FROM
    Numbers;
```

OUTPUT:

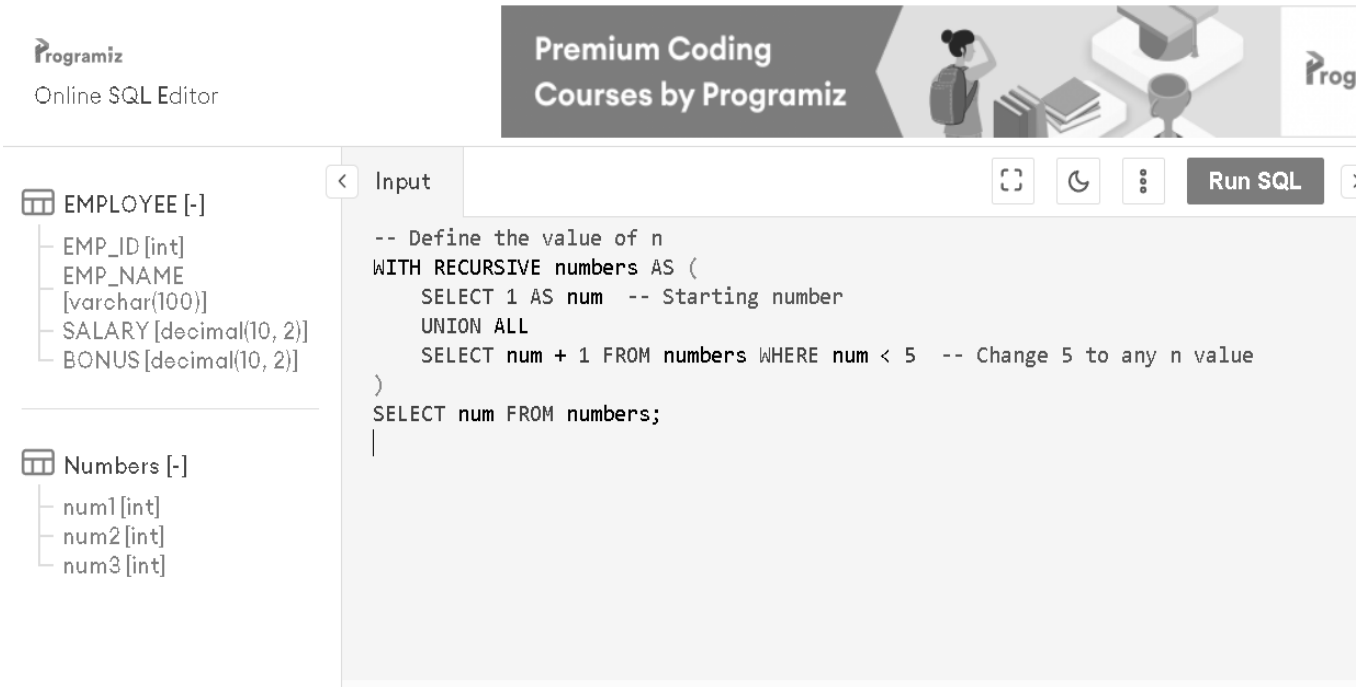
Numbers

num1	num2	num3
15	25	20

num1	num2	num3	greatest_number
15	25	20	25

EXERCISE 17

AIM: To Write a PL/SQL code to print the numbers from 1 to n.



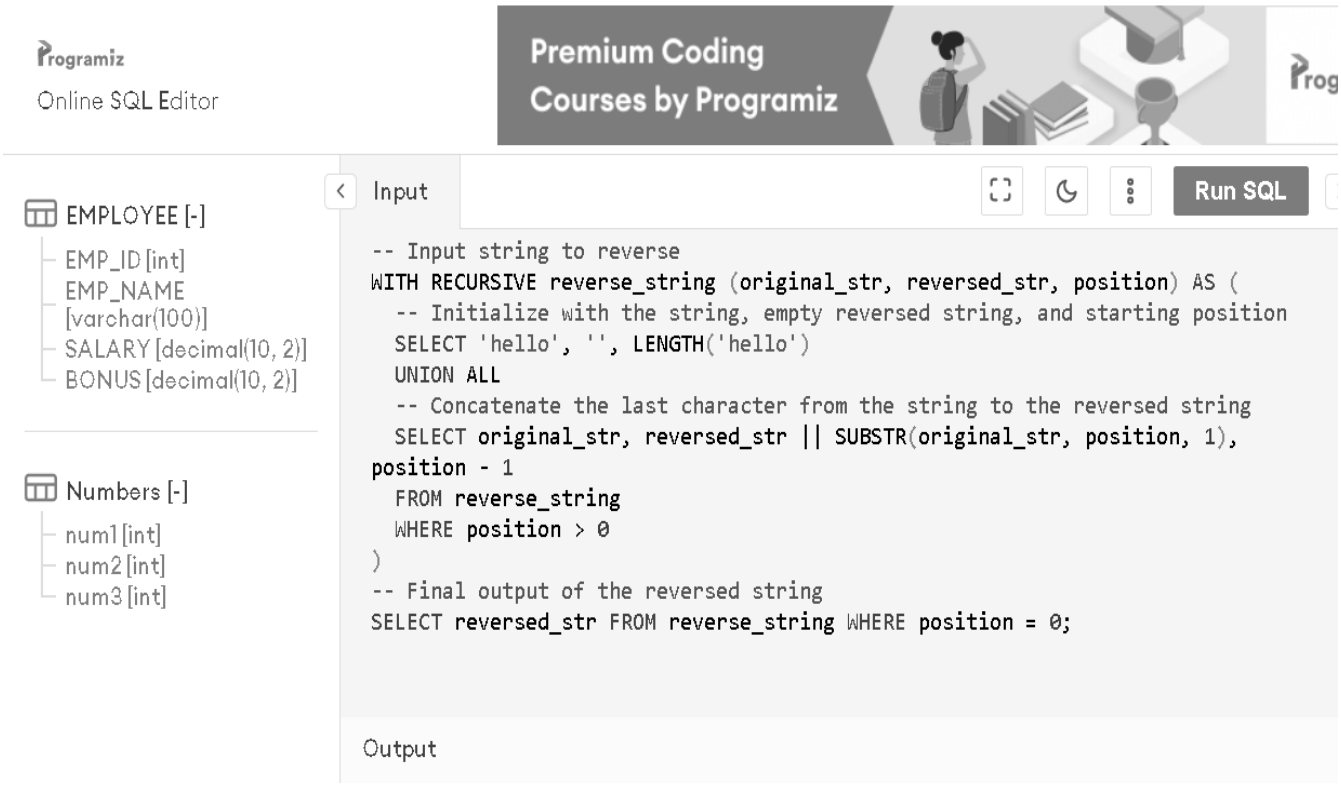
OUTPUT:

Output

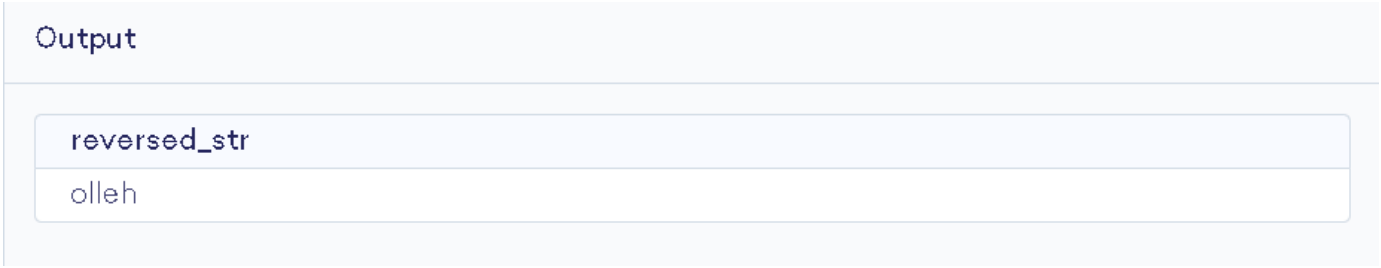
num
1
2
3
4
5

EXERCISE 18

AIM: To Write a PL/SQL code to reverse a string using for loop.

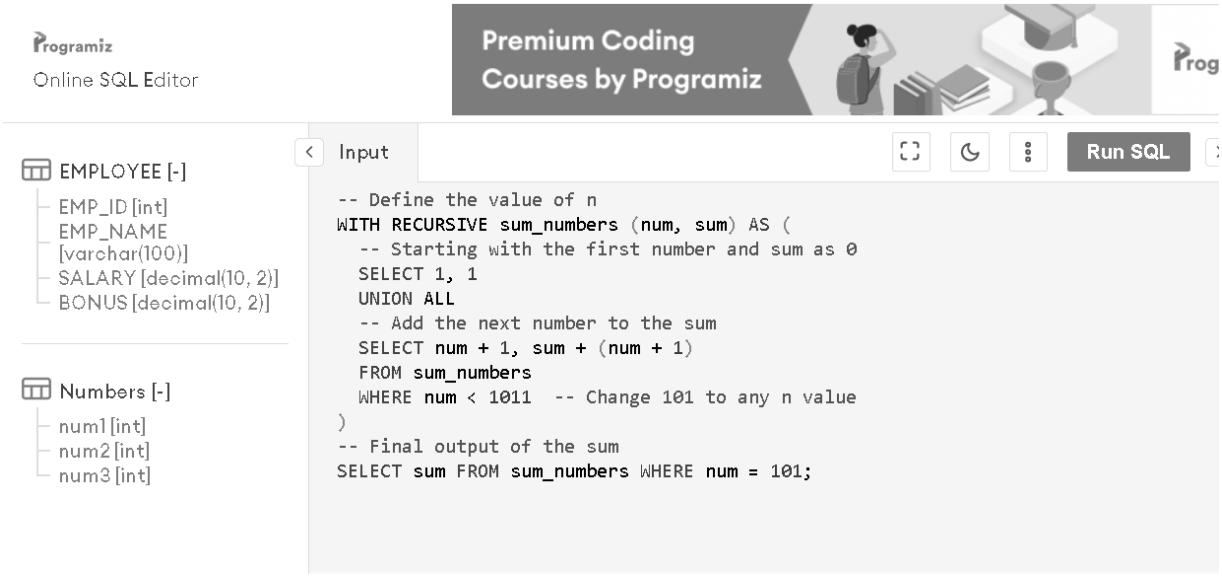


OUTPUT:



EXERCISE 19

AIM: To Write a PL/SQL code to find the sum of n numbers.



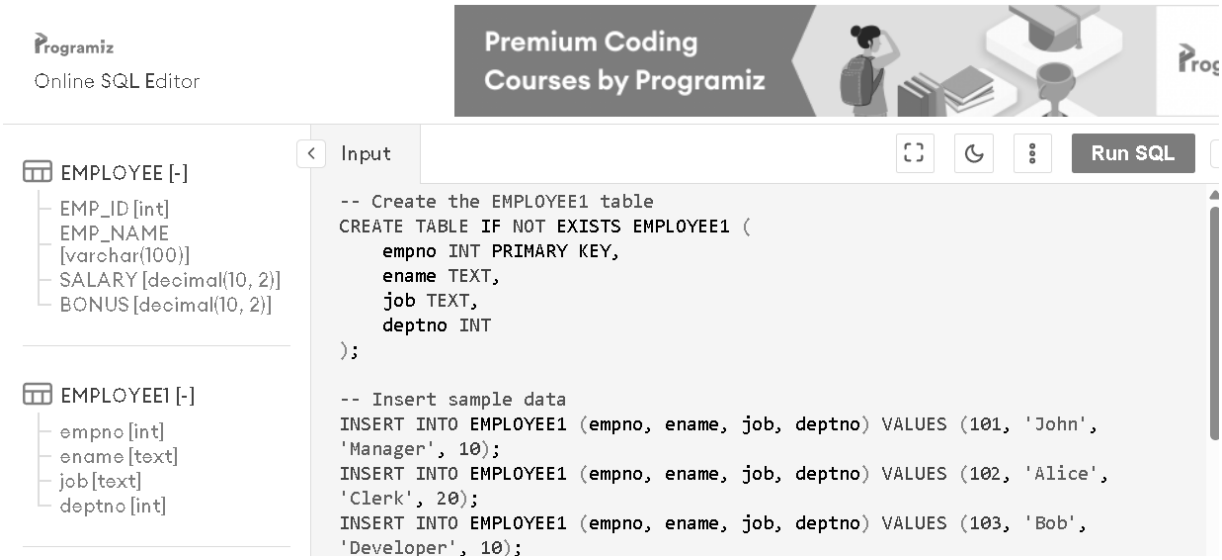
OUTPUT:

Output

sum
5151

EXERCISE 20

AIM: To Consider a PL/SQL code to display the empno, ename, job of employees of department number 10.




```
INSERT INTO EMPLOYEE1 (empno, ename, job, deptno) VALUES (104, 'Charlie',
'Analyst', 10);

-- Query to display empno, ename, and job for employees of department 10
SELECT empno, ename, job
FROM EMPLOYEE1
WHERE deptno = 10;
```

OUTPUT:

EMPLOYEE1

empno	ename	job	deptno
101	John	Manager	10
102	Alice	Clerk	20
103	Bob	Developer	10
104	Charlie	Analyst	10

Output

empno	ename	job
101	John	Manager
103	Bob	Developer
104	Charlie	Analyst

EXERCISE 21

AIM: To Consider a PL/SQL code to display the employee number & name of top five highest paid employees.

OUTPUT:

EMPLOYEE2

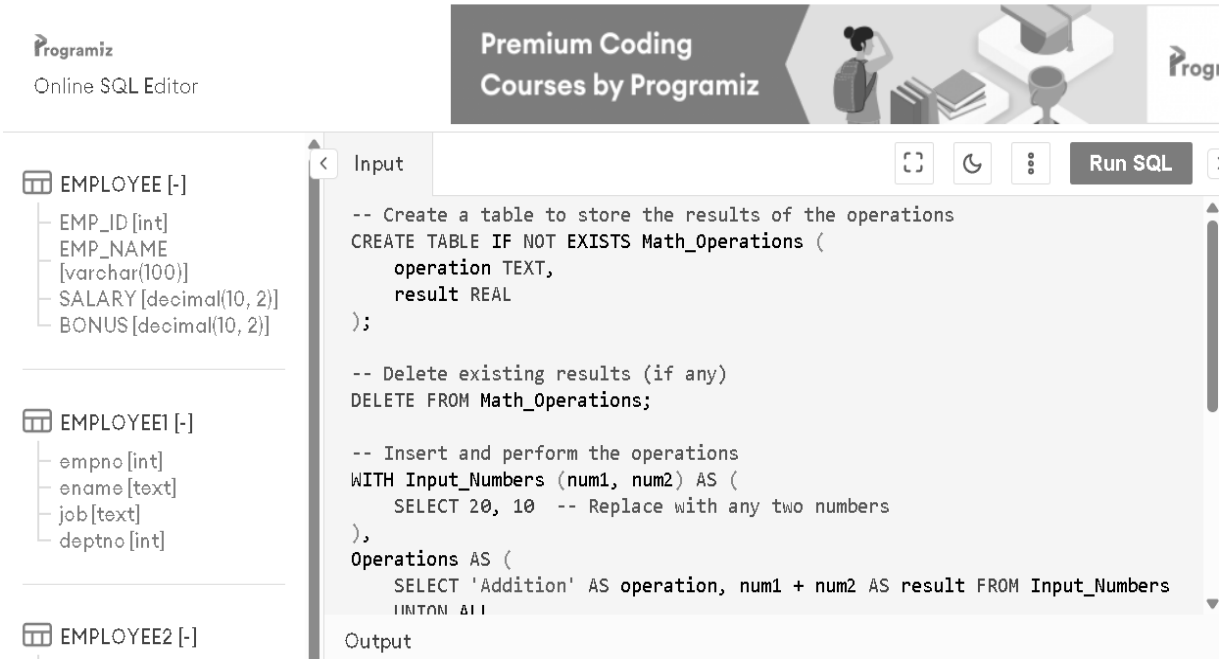
empno	ename	salary
101	John	50000
102	Alice	60000
103	Bob	75000
104	Charlie	80000
105	David	55000
106	Eva	70000

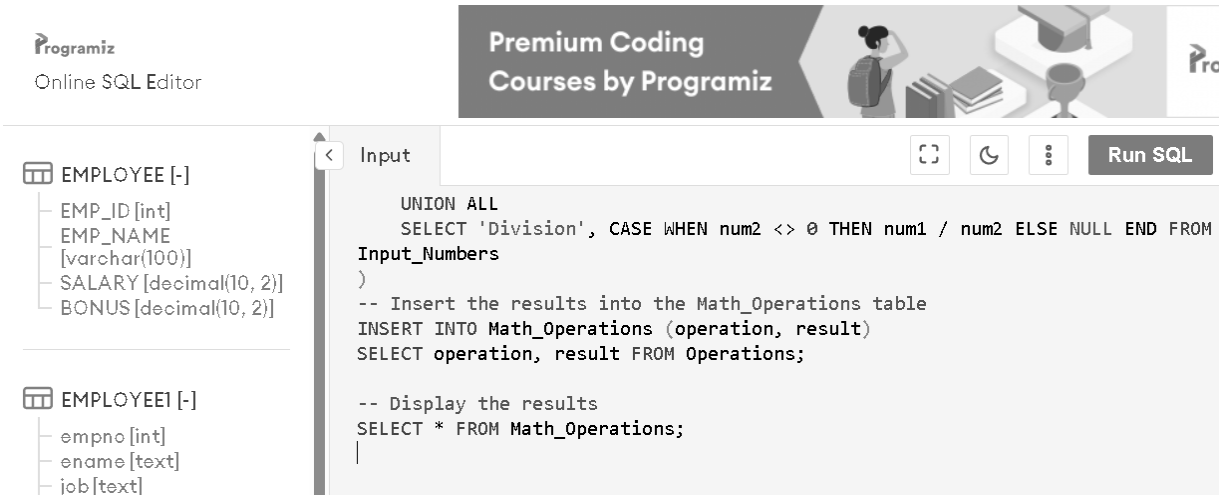
Output

empno	ename	salary
104	Charlie	80000
103	Bob	75000
106	Eva	70000
102	Alice	60000
105	David	55000

EXERCISE 22

AIM: To Consider a PL/SQL procedure that accepts 2 numbers & return addition, subtraction, multiplication & division of two numbers using stored procedure AND local procedure.





OUTPUT:

Math_Operations

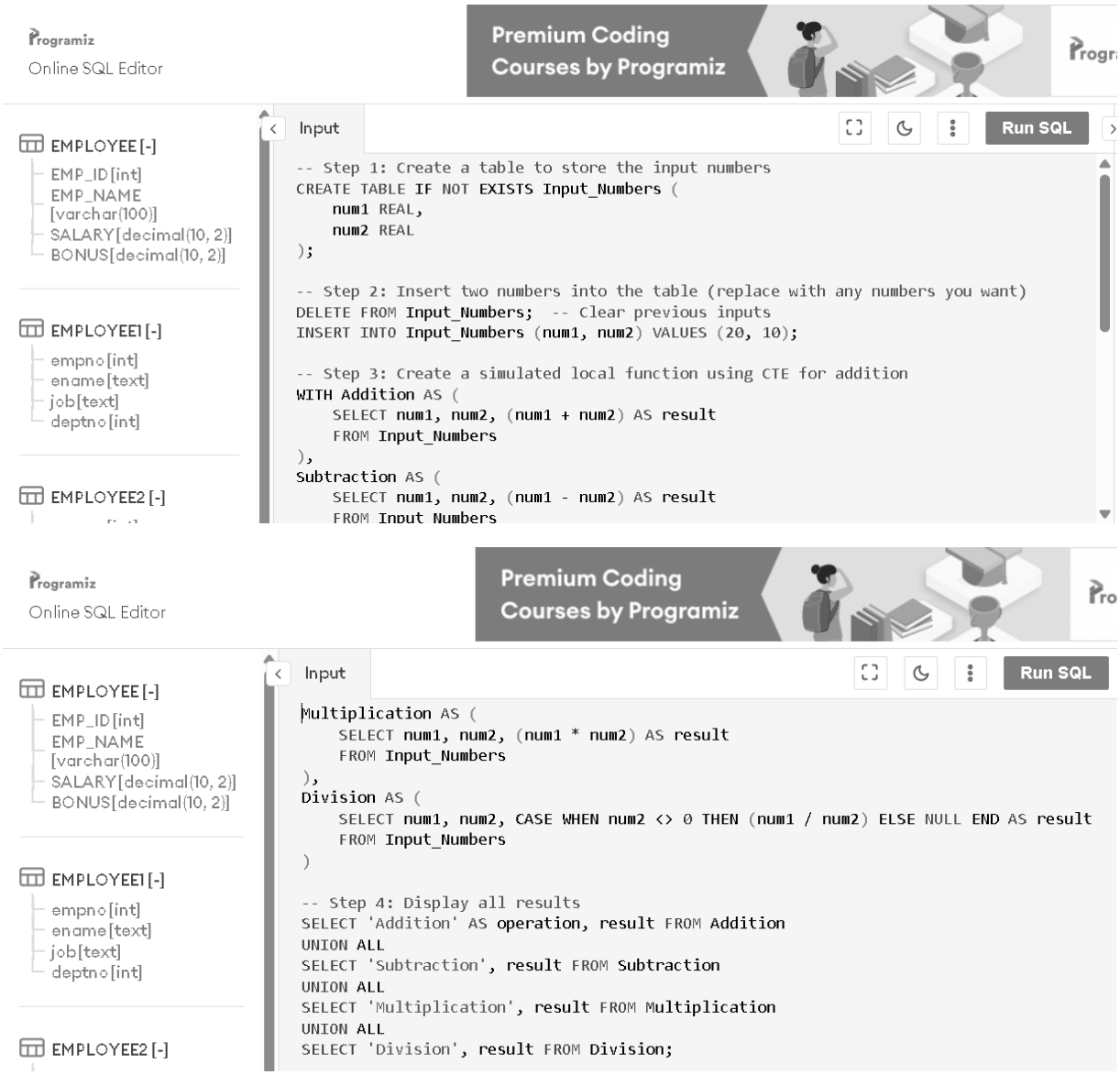
operation	result
Addition	30
Subtraction	10
Multiplication	200
Division	2

Output

operation	result
Addition	30
Subtraction	10
Multiplication	200
Division	2

EXERCISE 23

AIM: To Consider a PL/SQL code that accepts 2 numbers & return addition, subtraction, multiplication & division of two numbers using stored functions and local function.



OUTPUT:

Input_Numbers

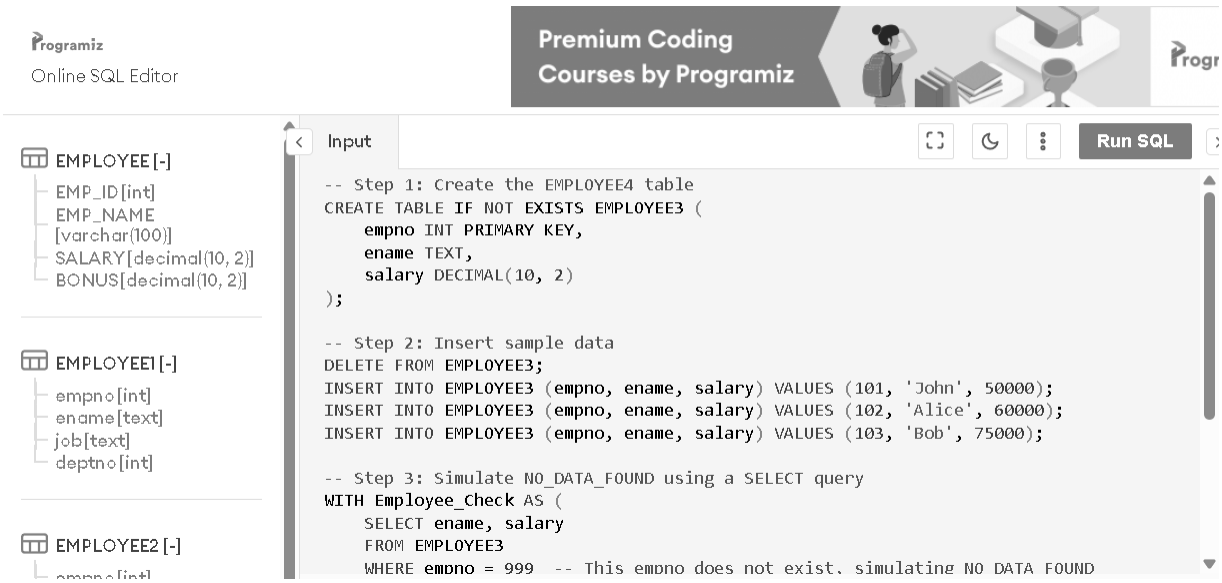
num1	num2
20	10

Output

operation	result
Addition	30
Subtraction	10
Multiplication	200
Division	2

EXERCISE 24

AIM: To Write a PL/SQL block to show the use of NO_DATA FOUND exception.




```
)
-- Check if the query returned any results
SELECT
    CASE
        WHEN EXISTS (SELECT 1 FROM Employee_Check)
        THEN (SELECT 'Employee Found: ' || ename || ', Salary: ' || salary FROM
Employee_Check)
        ELSE 'NO_DATA_FOUND: No employee found with the given employee number'
    END AS result;
```

OUTPUT:

EMPLOYEE3

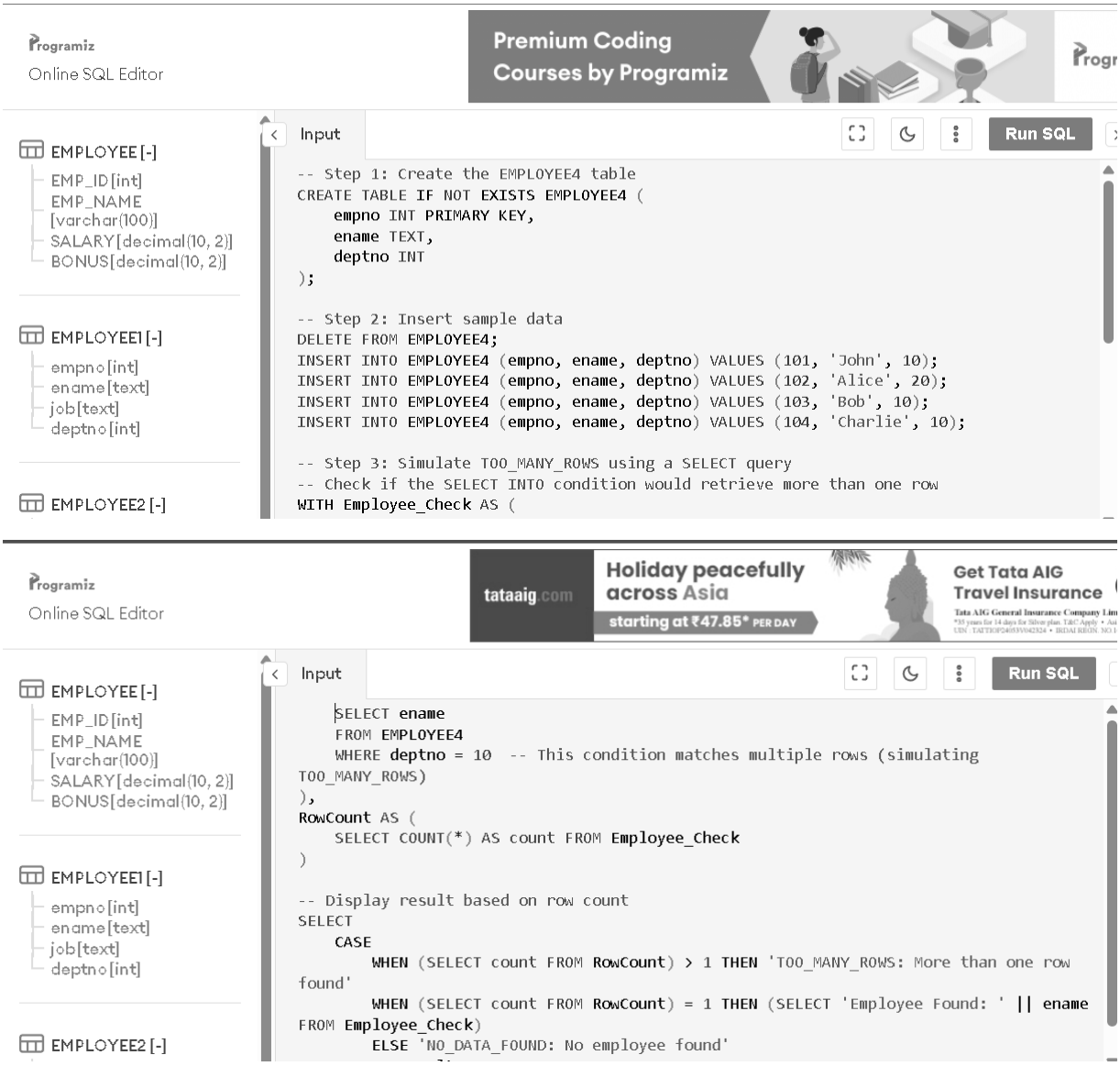
empno	ename	salary
101	John	50000
102	Alice	60000
103	Bob	75000

Output

result
NO_DATA_FOUND: No employee found with the given employee number

EXERCISE 25

AIM: To Write a PL/SQL block to show the use of TOO_MANY ROWS exception.



OUTPUT:

EMPLOYEE4

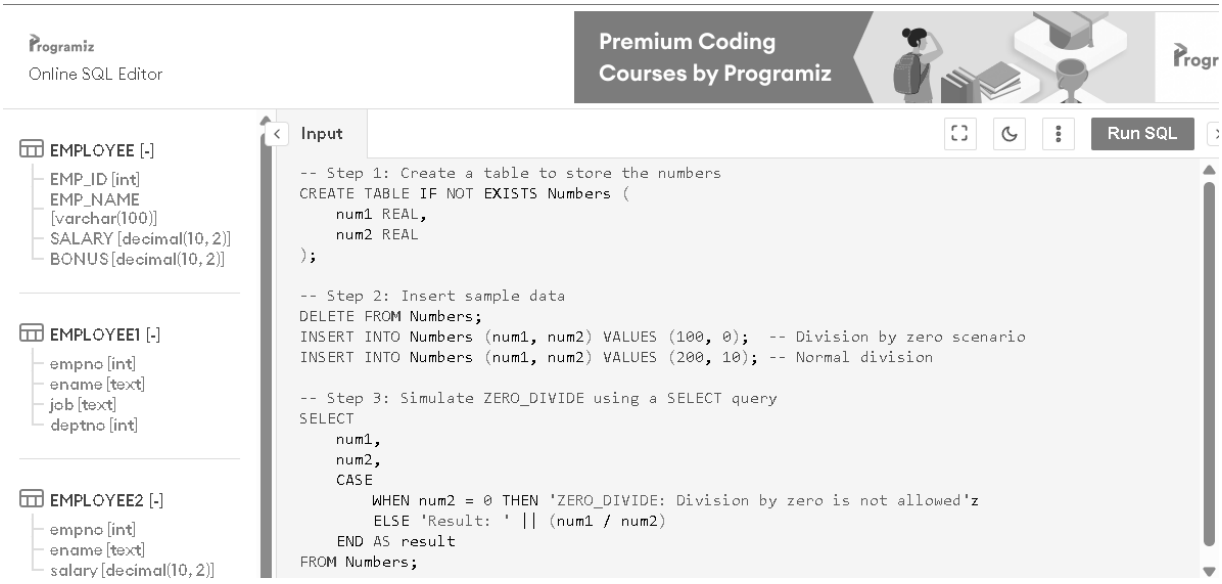
empno	ename	deptno
101	John	10
102	Alice	20
103	Bob	10
104	Charlie	10

Output

result
TOO_MANY_ROWS: More than one row found

EXERCISE 26

AIM: To Write a PL/SQL block to show the use of ZERO_DIVIDE exception.



OUTPUT:

Numbers

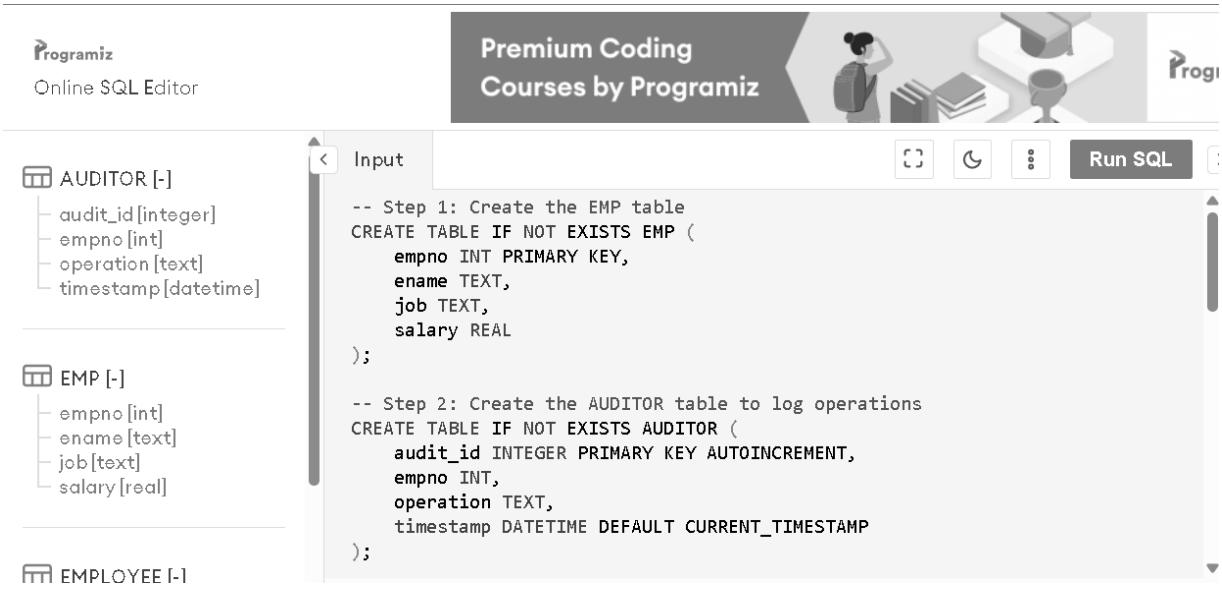
num1	num2	num3
100	0	
200	10	

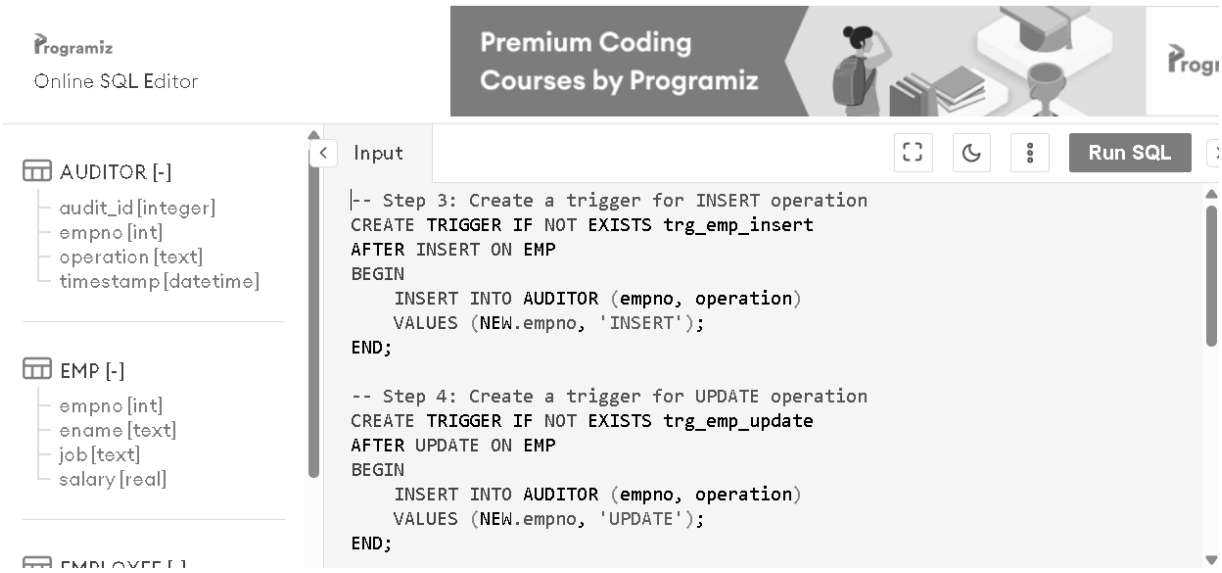
Output

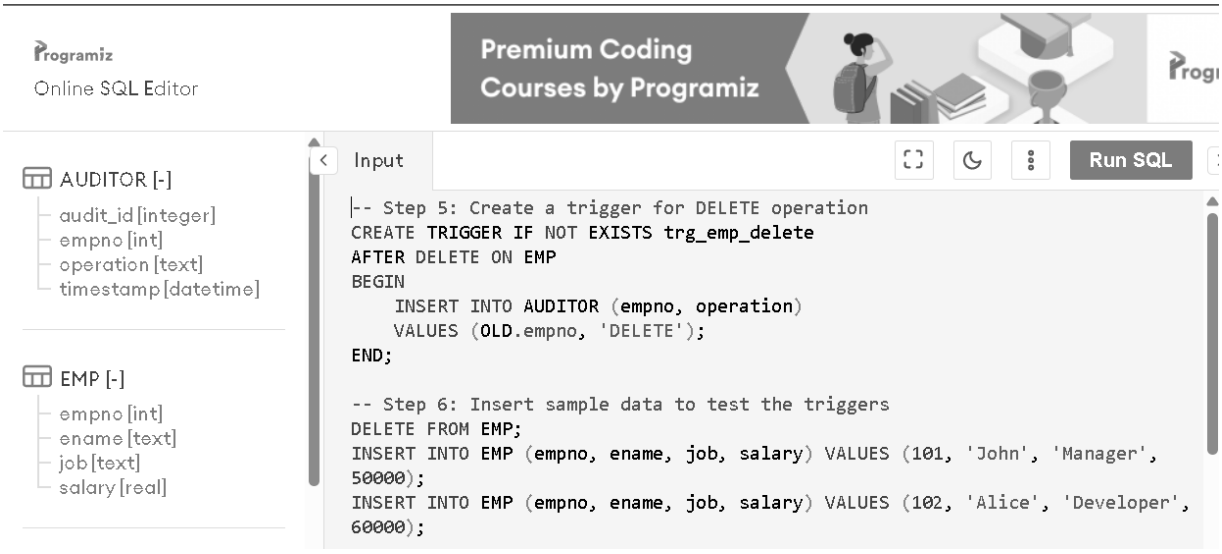
num1	num2	result
100	0	ZERO_DIVIDE: Division by zero is not allowed
200	10	Result: 20

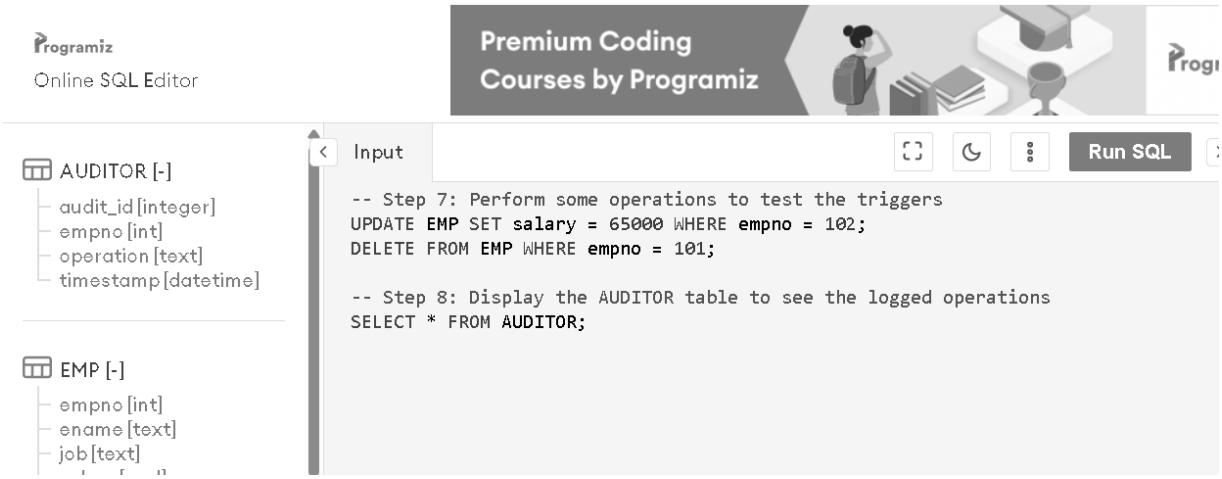
EXERCISE 27

AIM: To create a trigger on the emp table, which store the empno& operation in the table auditor for each operation i.e. Insert, Update & Delete.









OUTPUT:

AUDITOR

audit_id	empno	operation	timestamp
1	101	INSERT	2024-11-10 14:42:06
2	102	INSERT	2024-11-10 14:42:06
3	102	UPDATE	2024-11-10 14:42:06
4	101	DELETE	2024-11-10 14:42:06

EMP

empno	ename	job	salary
102	Alice	Developer	65000

Output

audit_id	empno	operation	timestamp
1	101	INSERT	2024-11-10 14:42:06
2	102	INSERT	2024-11-10 14:42:06
3	102	UPDATE	2024-11-10 14:42:06
4	101	DELETE	2024-11-10 14:42:06

EXERCISE 28

AIM: To create a trigger so that no operation can be performed on emp table.

AUDITOR [-]
- audit_id [integer]
- empno [int]
- operation [text]
- timestamp [datetime]

EMP [-]
- empno [int]
- ename [text]
- job [text]
- salary [real]

Input — Run SQL

```
-- Step 1: Create the EMP table
CREATE TABLE IF NOT EXISTS EMP (
    empno INT PRIMARY KEY,
    ename TEXT,
    job TEXT,
    salary REAL
);

-- Step 2: Create a trigger to block INSERT operation
CREATE TRIGGER IF NOT EXISTS trg_block_insert
BEFORE INSERT ON EMP
BEGIN
    SELECT RAISE(ABORT, 'INSERT operation is not allowed on EMP table');
END;
```

AUDITOR [-]
- audit_id [integer]
- empno [int]
- operation [text]
- timestamp [datetime]

EMP [-]
- empno [int]
- ename [text]
- job [text]
- salary [real]

Input — Run SQL

```
-- Step 3: Create a trigger to block UPDATE operation
CREATE TRIGGER IF NOT EXISTS trg_block_update
BEFORE UPDATE ON EMP
BEGIN
    SELECT RAISE(ABORT, 'UPDATE operation is not allowed on EMP table');
END;

-- Step 4: Create a trigger to block DELETE operation
CREATE TRIGGER IF NOT EXISTS trg_block_delete
BEFORE DELETE ON EMP
BEGIN
    SELECT RAISE(ABORT, 'DELETE operation is not allowed on EMP table');
END;
```

AUDITOR [-]
- audit_id [integer]
- empno [int]
- operation [text]
- timestamp [datetime]

EMP [-]
- empno [int]
- ename [text]
- job [text]
- salary [real]

Input — Run SQL

```
-- Step 5: Attempt to perform some operations to see the triggers in action

-- Attempt to insert a record (This should fail)
INSERT INTO EMP (empno, ename, job, salary) VALUES (101, 'John', 'Manager',
50000);

-- Attempt to update a record (This should fail)
UPDATE EMP SET salary = 60000 WHERE empno = 101;

-- Attempt to delete a record (This should fail)
DELETE FROM EMP WHERE empno = 101;
```

OUTPUT:

Output

Error: INSERT operation is not allowed on EMP table