ORACLE LAB

BCA-DS-552

Manav Rachna International Institute of Research and Studies

School of Computer Applications

Department of Computer Applications

Submitted By	
Student Name	**Archit Saxena**
Roll No	**22/FCA/BCA(AIML)/043**
Programme	**Bachelor of Computer Applications**
Semester	**5th Semester**
Section	**D**
Department	**Computer Applications**
Batch	**2022-25**
Submitted To	
Faculty Name	**Mrs. Neerja Negi**

SCHOOL OF COMPUTER APPLICATIONS

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

Output:

SQL Worksheet

```sql
CREATE TABLE Customer
    (
    SID VARCHAR2(4) PRIMARY KEY,
    First_Name CHAR(20),
    Last_name CHAR(20)
);

CREATE TABLE Orders
    (
    Order_ID VARCHAR2(4) PRIMARY KEY,
    Order_date CHAR(20),
    Customer_SID VARCHAR2(4),
    Amount NUMBER CHECK (Amount > 20000),
    FOREIGN KEY (Customer_SID) REFERENCES Customer(SID)
);
```

```
Table created.

Table created.
```

EXERCISE 2

AIM: Insert 5 records for each table.

Output:

SQL Worksheet

```sql
INSERT INTO Customer  VALUES ('C001', 'John', 'Doe');
INSERT INTO Customer  VALUES ('C002', 'Jane', 'Smith');
INSERT INTO Customer  VALUES ('C003', 'Alex', 'James');
INSERT INTO Customer  VALUES ('C004', 'Chris', 'Evans');
INSERT INTO Customer  VALUES ('C005', 'Emma', 'Watson');

INSERT INTO Orders VALUES ('O001', '2024-08-01', 'C001', 25000);
INSERT INTO Orders VALUES ('O002', '2024-08-02', 'C002', 22000);
INSERT INTO Orders VALUES ('O003', '2024-08-03', 'C003', 21000);
INSERT INTO Orders VALUES ('O004', '2024-08-04', 'C004', 30000);
INSERT INTO Orders VALUES ('O005', '2024-08-05', 'C005', 31000);
```

SQL Worksheet

SID	FIRST_NAME	LAST_NAME
C001	John	Doe
C002	Jane	Smith
C003	Alex	James
C004	Chris	Evans
C005	Emma	Watson

Download CSV

5 rows selected.

ORDER_ID	ORDER_DATE	CUSTOMER_SID	AMOUNT
O001	2024-08-01	C001	25000
O002	2024-08-02	C002	22000
O003	2024-08-03	C003	21000
O004	2024-08-04	C004	30000
O005	2024-08-05	C005	31000

Download CSV

EXERCISE 3

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

Output:

```
CREATE TABLE Orders
    (
    Order_ID VARCHAR2(4) PRIMARY KEY,
    Order_date CHAR(20),
    Customer_SID VARCHAR2(4),
    Amount NUMBER CHECK (Amount > 20000),
    FOREIGN KEY (Customer_SID) REFERENCES Customer(SID)
);
```

EXERCISE 4

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

Output:

SQL Worksheet

```
SELECT SID, First_Name, Last_name, Amount
FROM Customer
JOIN Orders ON Customer.SID = Orders.Customer_SID;
```

SID	FIRST_NAME	LAST_NAME	AMOUNT
C001	John	Doe	25000
C002	Jane	Smith	22000
C003	Alex	James	21000
C004	Chris	Evans	30000
C005	Emma	Watson	31000

Download CSV

EXERCISE 5

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

Output:

SQL Worksheet

```
select * from Customer  where trim(last_name) like '%s';
```

SID	FIRST_NAME	LAST_NAME
C003	Alex	James
C004	Chris	Evans

Download CSV

2 rows selected.

EXERCISE 6

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

Output:

SQL Worksheet

```
select * from Orders where Amount between 21000 and 30000;
```

ORDER_ID	ORDER_DATE	CUSTOMER_SID	AMOUNT
O001	2024-08-01	C001	25000
O002	2024-08-02	C002	22000
O003	2024-08-03	C003	21000
O004	2024-08-04	C004	30000

Download CSV

4 rows selected.

EXERCISE 7

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

Output:

SQL Worksheet

```sql
update Orders set Amount = Amount + 500;

select Order_ID, Amount as "New Amount" from Orders;
```

5 row(s) updated.

ORDER_ID	New Amount
O001	25500
O002	22500
O003	21500
O004	30500
O005	31500

Download CSV

EXERCISE 8

AIM: Display the order_id and total amount of orders.

Output:

SQL Worksheet

```sql
SELECT Order_ID, SUM(Amount) AS Total_Amount
FROM Orders
GROUP BY Order_ID;
```

ORDER_ID	TOTAL_AMOUNT
O001	25500
O002	22500
O003	21500
O004	30500
O005	31500

Download CSV

5 rows selected.

EXERCISE 9

AIM: Calculate the total amount of orders that has more than 15000.

Output:

SQL Worksheet

```
select sum(Amount) as Total_Amount from Orders where Amount > 15000;
```

TOTAL_AMOUNT
131500

Download CSV

EXERCISE 10

AIM: Display all the string functions used in SQL.

Output:

```
SELECT
   LOWER('ORACLE') AS "Lowercase",       -- Converts string to lowercase
   UPPER('oracle') AS "Uppercase",       -- Converts string to uppercase
   SUBSTR('ORACLE', 2, 3) AS "Substring", -- Extracts substring
   LENGTH('ORACLE') AS "Length”,         -- Returns length of string
   INSTR('ORACLE', 'A') AS "Position",   -- Returns position of a character
   LPAD('123', 5, '0') AS "Left Padding", -- Pads a string on the left
   RPAD('123', 5, '0') AS "Right Padding",-- Pads a string on the right
   TRIM('O' FROM 'ORACLE') AS "Trimmed"  -- Trims a specified character
FROM DUAL;
```

EXERCISE 11

AIM: Create the following tables.

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	

Output:

SQL Worksheet

```sql
create table Student
(
    RollNo varchar(20) primary key,
    Name char(20),
    Class varchar(20),
    Marks number(6,2)
);

create table Student1
(
    R_No varchar(20) primary key,
    Name char(20),
    Class varchar(20),
    Marks number(6,2)
);
```

```
Table created.

Table created.
```

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

```sql
-- 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:

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

```sql
SELECT RollNo, Name, Class, Marks
FROM Student

UNION

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

Output

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

```sql
-- 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

```sql
-- 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

```sql
-- 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.

Programiz

Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

Numbers [-]
- num1 [int]
- num2 [int]
- num3 [int]

Input

Run SQL

```sql
-- Step 1: Create the EMPLOYEE table
CREATE TABLE IF NOT EXISTS EMPLOYEE (
    EMP_ID INT PRIMARY KEY,
    EMP_NAME VARCHAR(100),
    SALARY DECIMAL(10, 2),
    BONUS DECIMAL(10, 2)
);
-- Step 2: Insert sample data
INSERT INTO EMPLOYEE (EMP_ID, EMP_NAME, SALARY, BONUS) VALUES (100, 'John Doe',
50000, 5000);
INSERT INTO EMPLOYEE (EMP_ID, EMP_NAME, SALARY, BONUS) VALUES (102, 'Alice
Johnson', 70000, 7000);
INSERT INTO EMPLOYEE (EMP_ID, EMP_NAME, SALARY, BONUS) VALUES (103, 'Bob
Brown', 55000, 5500);
INSERT INTO EMPLOYEE (EMP_ID, EMP_NAME, SALARY, BONUS) VALUES (104, 'Charlie
Davis', 80000, 8000);
```

```sql
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.

Programiz
Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

Numbers [-]
- num1 [int]
- num2 [int]
- num3 [int]

Input

Run SQL

```sql
-- Create a table to store three numbers
CREATE TABLE IF NOT EXISTS Numbers (
    num1 INT,
    num2 INT,
    num3 INT
);

-- Insert sample data
INSERT INTO Numbers (num1, num2, num3) VALUES (15, 25, 20);

-- Use a CASE statement to find the greatest number
SELECT
    num1,
    num2,
    num3,
    CASE
        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.

Programiz

Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

Numbers [-]
- num1 [int]
- num2 [int]
- num3 [int]

Input

Run SQL

```
-- Define the value of n
WITH RECURSIVE numbers AS (
    SELECT 1 AS num  -- Starting number
    UNION ALL
    SELECT num + 1 FROM numbers WHERE num < 5  -- Change 5 to any n value
)
SELECT num FROM numbers;
```

OUTPUT:

Output

num
1
2
3
4
5

EXERCISE 18

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

Programiz

Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

Numbers [-]
- num1 [int]
- num2 [int]
- num3 [int]

Input

Run SQL

```sql
-- Input string to reverse
WITH RECURSIVE reverse_string (original_str, reversed_str, position) AS (
  -- Initialize with the string, empty reversed string, and starting position
  SELECT 'hello', '', LENGTH('hello')
  UNION ALL
  -- Concatenate the last character from the string to the reversed string
  SELECT original_str, reversed_str || SUBSTR(original_str, position, 1),
position - 1
  FROM reverse_string
  WHERE position > 0
)
-- Final output of the reversed string
SELECT reversed_str FROM reverse_string WHERE position = 0;
```

Output

OUTPUT:

Output

reversed_str
olleh

EXERCISE 19

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

Programiz

Online SQL Editor

EMPLOYEE [-]

- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

Numbers [-]

- num1 [int]
- num2 [int]
- num3 [int]

Input

Run SQL

```
-- Define the value of n
WITH RECURSIVE sum_numbers (num, sum) AS (
  -- Starting with the first number and sum as 0
  SELECT 1, 1
  UNION ALL
  -- Add the next number to the sum
  SELECT num + 1, sum + (num + 1)
  FROM sum_numbers
  WHERE num < 1011  -- Change 101 to any n value
)
-- Final output of the sum
SELECT sum FROM sum_numbers WHERE num = 101;
```

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.

Programiz
Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

EMPLOYEE1 [-]
- empno [int]
- ename [text]
- job [text]
- deptno [int]

Input

Run SQL

```sql
-- Create the EMPLOYEE1 table
CREATE TABLE IF NOT EXISTS EMPLOYEE1 (
    empno INT PRIMARY KEY,
    ename TEXT,
    job TEXT,
    deptno INT
);

-- Insert sample data
INSERT INTO EMPLOYEE1 (empno, ename, job, deptno) VALUES (101, 'John',
'Manager', 10);
INSERT INTO EMPLOYEE1 (empno, ename, job, deptno) VALUES (102, 'Alice',
'Clerk', 20);
INSERT INTO EMPLOYEE1 (empno, ename, job, deptno) VALUES (103, 'Bob',
'Developer', 10);
```

```sql
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.

Programiz
Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

EMPLOYEE1 [-]
- empno [int]
- ename [text]
- job [text]
- deptno [int]

Input — Run SQL

```sql
-- Create the EMPLOYEE2 table
CREATE TABLE IF NOT EXISTS EMPLOYEE2 (
    empno INT PRIMARY KEY,
    ename TEXT,
    salary DECIMAL(10, 2)
);

-- Insert sample data
INSERT INTO EMPLOYEE2 (empno, ename, salary) VALUES (101, 'John', 50000);
INSERT INTO EMPLOYEE2 (empno, ename, salary) VALUES (102, 'Alice', 60000);
INSERT INTO EMPLOYEE2 (empno, ename, salary) VALUES (103, 'Bob', 75000);
INSERT INTO EMPLOYEE2 (empno, ename, salary) VALUES (104, 'Charlie', 80000);
INSERT INTO EMPLOYEE2 (empno, ename, salary) VALUES (105, 'David', 55000);
INSERT INTO EMPLOYEE2 (empno, ename, salary) VALUES (106, 'Eva', 70000);
```


Programiz
Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

Input — Run SQL

```sql
-- Query to get the top five highest paid employees
SELECT empno, ename, salary
FROM EMPLOYEE2
ORDER BY salary DESC
LIMIT 5;
```

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.

Programiz
Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

EMPLOYEE1 [-]
- empno [int]
- ename [text]
- job [text]
- deptno [int]

EMPLOYEE2 [-]

Input — Run SQL

```sql
-- Create a table to store the results of the operations
CREATE TABLE IF NOT EXISTS Math_Operations (
    operation TEXT,
    result REAL
);

-- Delete existing results (if any)
DELETE FROM Math_Operations;

-- Insert and perform the operations
WITH Input_Numbers (num1, num2) AS (
    SELECT 20, 10  -- Replace with any two numbers
),
Operations AS (
    SELECT 'Addition' AS operation, num1 + num2 AS result FROM Input_Numbers
    UNION ALL
```

Output

Programiz
Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

EMPLOYEE1 [-]
- empno [int]
- ename [text]
- job [text]

Input — Run SQL

```sql
    UNION ALL
    SELECT 'Division', CASE WHEN num2 <> 0 THEN num1 / num2 ELSE NULL END FROM
Input_Numbers
)
-- Insert the results into the Math_Operations table
INSERT INTO Math_Operations (operation, result)
SELECT operation, result FROM Operations;

-- Display the results
SELECT * FROM Math_Operations;
```

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.

Programiz
Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

EMPLOYEE1 [-]
- empno [int]
- ename [text]
- job [text]
- deptno [int]

EMPLOYEE2 [-]

Input | Run SQL

```sql
-- Step 1: Create a table to store the input numbers
CREATE TABLE IF NOT EXISTS Input_Numbers (
    num1 REAL,
    num2 REAL
);

-- Step 2: Insert two numbers into the table (replace with any numbers you want)
DELETE FROM Input_Numbers;  -- Clear previous inputs
INSERT INTO Input_Numbers (num1, num2) VALUES (20, 10);

-- Step 3: Create a simulated local function using CTE for addition
WITH Addition AS (
    SELECT num1, num2, (num1 + num2) AS result
    FROM Input_Numbers
),
Subtraction AS (
    SELECT num1, num2, (num1 - num2) AS result
    FROM Input_Numbers
```

Programiz
Online SQL Editor

Premium Coding Courses by Programiz

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

EMPLOYEE1 [-]
- empno [int]
- ename [text]
- job [text]
- deptno [int]

EMPLOYEE2 [-]

Input | Run SQL

```sql
Multiplication AS (
    SELECT num1, num2, (num1 * num2) AS result
    FROM Input_Numbers
),
Division AS (
    SELECT num1, num2, CASE WHEN num2 <> 0 THEN (num1 / num2) ELSE NULL END AS result
    FROM Input_Numbers
)

-- Step 4: Display all results
SELECT 'Addition' AS operation, result FROM Addition
UNION ALL
SELECT 'Subtraction', result FROM Subtraction
UNION ALL
SELECT 'Multiplication', result FROM Multiplication
UNION ALL
SELECT 'Division', result FROM Division;
```

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.

Programiz
Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

EMPLOYEE1 [-]
- empno [int]
- ename [text]
- job [text]
- deptno [int]

EMPLOYEE2 [-]
- empno [int]

Input | Run SQL

```sql
-- Step 1: Create the EMPLOYEE4 table
CREATE TABLE IF NOT EXISTS EMPLOYEE3 (
    empno INT PRIMARY KEY,
    ename TEXT,
    salary DECIMAL(10, 2)
);

-- Step 2: Insert sample data
DELETE FROM EMPLOYEE3;
INSERT INTO EMPLOYEE3 (empno, ename, salary) VALUES (101, 'John', 50000);
INSERT INTO EMPLOYEE3 (empno, ename, salary) VALUES (102, 'Alice', 60000);
INSERT INTO EMPLOYEE3 (empno, ename, salary) VALUES (103, 'Bob', 75000);

-- Step 3: Simulate NO_DATA_FOUND using a SELECT query
WITH Employee_Check AS (
    SELECT ename, salary
    FROM EMPLOYEE3
    WHERE empno = 999  -- This empno does not exist, simulating NO_DATA_FOUND
)
-- 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.

Programiz
Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

EMPLOYEE1 [-]
- empno [int]
- ename [text]
- job [text]
- deptno [int]

EMPLOYEE2 [-]

Input — Run SQL

```sql
-- Step 1: Create the EMPLOYEE4 table
CREATE TABLE IF NOT EXISTS EMPLOYEE4 (
    empno INT PRIMARY KEY,
    ename TEXT,
    deptno INT
);

-- Step 2: Insert sample data
DELETE FROM EMPLOYEE4;
INSERT INTO EMPLOYEE4 (empno, ename, deptno) VALUES (101, 'John', 10);
INSERT INTO EMPLOYEE4 (empno, ename, deptno) VALUES (102, 'Alice', 20);
INSERT INTO EMPLOYEE4 (empno, ename, deptno) VALUES (103, 'Bob', 10);
INSERT INTO EMPLOYEE4 (empno, ename, deptno) VALUES (104, 'Charlie', 10);

-- Step 3: Simulate TOO_MANY_ROWS using a SELECT query
-- Check if the SELECT INTO condition would retrieve more than one row
WITH Employee_Check AS (
```

Programiz
Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

EMPLOYEE1 [-]
- empno [int]
- ename [text]
- job [text]
- deptno [int]

EMPLOYEE2 [-]

Input — Run SQL

```sql
    SELECT ename
    FROM EMPLOYEE4
    WHERE deptno = 10  -- This condition matches multiple rows (simulating
TOO_MANY_ROWS)
),
RowCount AS (
    SELECT COUNT(*) AS count FROM Employee_Check
)

-- Display result based on row count
SELECT
    CASE
        WHEN (SELECT count FROM RowCount) > 1 THEN 'TOO_MANY_ROWS: More than one row
found'
        WHEN (SELECT count FROM RowCount) = 1 THEN (SELECT 'Employee Found: ' || ename
FROM Employee_Check)
        ELSE 'NO_DATA_FOUND: No employee found'
```

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.

Programiz
Online SQL Editor

EMPLOYEE [-]
- EMP_ID [int]
- EMP_NAME [varchar(100)]
- SALARY [decimal(10, 2)]
- BONUS [decimal(10, 2)]

EMPLOYEE1 [-]
- empno [int]
- ename [text]
- job [text]
- deptno [int]

EMPLOYEE2 [-]
- empno [int]
- ename [text]
- salary [decimal(10, 2)]

Input

Run SQL

```sql
-- Step 1: Create a table to store the numbers
CREATE TABLE IF NOT EXISTS Numbers (
    num1 REAL,
    num2 REAL
);

-- Step 2: Insert sample data
DELETE FROM Numbers;
INSERT INTO Numbers (num1, num2) VALUES (100, 0);  -- Division by zero scenario
INSERT INTO Numbers (num1, num2) VALUES (200, 10); -- Normal division

-- Step 3: Simulate ZERO_DIVIDE using a SELECT query
SELECT
    num1,
    num2,
    CASE
        WHEN num2 = 0 THEN 'ZERO_DIVIDE: Division by zero is not allowed'z
        ELSE 'Result: ' || (num1 / num2)
    END AS result
FROM Numbers;
```

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.

Programiz
Online SQL Editor

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

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

EMPLOYEE [-]

Input | Run SQL

```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 the AUDITOR table to log operations
CREATE TABLE IF NOT EXISTS AUDITOR (
    audit_id INTEGER PRIMARY KEY AUTOINCREMENT,
    empno INT,
    operation TEXT,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Programiz
Online SQL Editor

Premium Coding Courses by Programiz

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

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

EMPLOYEE [-]

Input | Run SQL

```sql
-- Step 3: Create a trigger for INSERT operation
CREATE TRIGGER IF NOT EXISTS trg_emp_insert
AFTER INSERT ON EMP
BEGIN
    INSERT INTO AUDITOR (empno, operation)
    VALUES (NEW.empno, 'INSERT');
END;

-- Step 4: Create a trigger for UPDATE operation
CREATE TRIGGER IF NOT EXISTS trg_emp_update
AFTER UPDATE ON EMP
BEGIN
    INSERT INTO AUDITOR (empno, operation)
    VALUES (NEW.empno, 'UPDATE');
END;
```

Programiz
Online SQL Editor

Premium Coding Courses by Programiz

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

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

Input | Run SQL

```sql
-- Step 5: Create a trigger for DELETE operation
CREATE TRIGGER IF NOT EXISTS trg_emp_delete
AFTER DELETE ON EMP
BEGIN
    INSERT INTO AUDITOR (empno, operation)
    VALUES (OLD.empno, 'DELETE');
END;

-- Step 6: Insert sample data to test the triggers
DELETE FROM EMP;
INSERT INTO EMP (empno, ename, job, salary) VALUES (101, 'John', 'Manager',
50000);
INSERT INTO EMP (empno, ename, job, salary) VALUES (102, 'Alice', 'Developer',
60000);
```

Programiz
Online SQL Editor

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

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

Input

Run SQL

```sql
-- Step 7: Perform some operations to test the triggers
UPDATE EMP SET salary = 65000 WHERE empno = 102;
DELETE FROM EMP WHERE empno = 101;

-- Step 8: Display the AUDITOR table to see the logged operations
SELECT * FROM AUDITOR;
```

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.

Programiz
Online SQL Editor

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;
```

Programiz
Online SQL Editor

Premium Coding Courses by Programiz

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;
```

Programiz
Online SQL Editor

Premium Coding Courses by Programiz

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
```