Oracle Lab

Q1. Create the following tables

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

```
CREATE TABLE Customer (
   SID VARCHAR(4) PRIMARY KEY,
   First_name CHAR(20),
   Last_name CHAR(20)
);
```

Output

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

```
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.

Q2. Insert five records for each table

```
Insert into Customer(SID, First_name, Last_name) VALUES ("1","Elena","Greorge");
Insert into Customer(SID, First_name, Last_name) VALUES ("2","Adam","John");
Insert into Customer(SID, First_name, Last_name) VALUES ("3","Elijah","Mikaelson");
Insert into Customer(SID, First_name, Last_name) VALUES ("4","Stefen","Salvatore");
Insert into Customer(SID, First_name, Last_name) VALUES ("5","Klaus","Alexander");
```

```
Select * from Customer;
```

Output

SID	First_name	Last_name
1	Elena	Greorge
2	Adam	John
3	Elijah	Mikaelson
4	Stefen	Salvatore
5	Klaus	Alexander

```
Insert into Orders(Order_ID,Order_date,Customer_SID,Amount) VALUES ("01","08-03","1",21000);
Insert into Orders(Order_ID,Order_date ,Customer_SID,Amount )VALUES ("02","15-03","2",23000);
Insert into Orders(Order_ID,Order_date ,Customer_SID,Amount)VALUES ("03","07-07","3",26000);
Insert into Orders(Order_ID,Order_date ,Customer_SID,Amount) VALUES ("04","14-07","4",25000);
Insert into Orders(Order_ID,Order_date ,Customer_SID,Amount) VALUES ("05","23-09","5",29000);

Select * from Orders;
```

Output

Order_ID	Order_date	Customer_SID	Amount
01	08-03	1	21000
02	15-03	2	23000
03	07-07	3	26000
04	14-07	4	25000
05	23-09	5	29000

Q3.Customer_SID column in the ORDERS table is a foreign key pointing to the SID column in the CUSTOMER 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)
);
```

Q5. List the details of the customers along with the amount.

```
SELECT Customer.*, Orders.Amount FROM Customer INNER JOIN Orders ON Customer.SID =
Orders.Customer_SID;
```

Output

SID	First_name	Last_name	Amount
1	Elena	Greorge	21000
2	Adam	John	23000
3	Elijah	Mikaelson	26000
4	Stefen	Salvatore	25000
5	Klaus	Alexander	29000

Q6. List the customers whose names end with “s”.

```
SELECT * FROM Customer WHERE last_name LIKE '%s';
```

Output

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

Q7. List the orders where amount is between 21000 and 30000

```
SELECT * FROM Orders WHERE Amount BETWEEN 21000 AND 30000;
```

Output

Order_ID	Order_date	Customer_SID	Amount
01	08-03	1	21000
02	15-03	2	23000
03	07-07	3	26000
04	14-07	4	25000
05	23-09	5	29000

Q8. list the orders where amount is increased by 500 and replace with name “new amount”.

SELECT *, (Amount + 500) AS New_Amount FROM Orders;

Output

Order_ID	Order_date	Customer_SID	Amount	New_Amount
01	08-03	1	21000	21500
02	15-03	2	23000	23500
03	07-07	3	26000	26500
04	14-07	4	25000	25500
05	23-09	5	29000	29500

Q9.Display the order_id and total amount of orders

SELECT Order_id, SUM(Amount) AS Total_amount FROM Orders GROUP BY Order_id;

Order_ID	Total_amount
01	21000
02	23000
03	26000
04	25000
05	29000

Q10. Calculate the totaamount of orders that has more than 15000.

```
SELECT SUM(Amount) AS Total_amount FROM Orders WHERE Amount > 15000;
```

Output

Total_amount
124000

11. Display all the string functions used in SQL.

```sql
Select CONCAT('John','Doe') AS Concatenation;
Select LENGTH("Adam","John") AS Length;
Select LOWER("Adam","John") AS Lower;
Select UPPER("Adam","John") AS Upper;
Select SUBSTR("Adam","John") AS Substring;
Select TRIM("Adam","John") AS Trim;
Select LTRIM("Adam","John") AS LeftTrim;
Select RTRIM("Adam","John") AS RightTrim;
```

Q12. Create the following tables

```sql
CREATE TABLE Student (
   RollNumber VARCHAR2(20),
   Name VARCHAR2(20),
   Class VARCHAR2(20),
   Marks NUMBER(6,2)
);
```

Output

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

```
INSERT INTO Student (RollNumber, Name, Class, Marks)
VALUES ('S001', 'John Doe', '10A', 85.50);

INSERT INTO Student (RollNumber, Name, Class, Marks)
VALUES ('S002', 'Alice Smith', '10B', 92.75);

INSERT INTO Student (RollNumber, Name, Class, Marks)
VALUES ('S003', 'Bob Johnson', '10A', 78.00);

INSERT INTO Student (RollNumber, Name, Class, Marks)
VALUES ('S004', 'Charlie Brown', '10C', 88.25);

INSERT INTO Student (RollNumber, Name, Class, Marks)
VALUES ('S005', 'Diana Clark', '10B', 91.00);
```

```
CREATE TABLE Student1 (
   R_no VARCHAR2(20),
   Name VARCHAR2(20),
   Class VARCHAR2(20),
   Marks NUMBER(6,2)
);
```

Output

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

```
INSERT INTO Student1 (R_no, Name, Class, Marks)
VALUES ('S006', 'Eva White', '10C', 79.50);
```

```
INSERT INTO Student1 (R_no, Name, Class, Marks)
VALUES ('S007', 'Frank Green', '10A', 94.00);

INSERT INTO Student1 (R_no, Name, Class, Marks)
VALUES ('S008', 'Grace Lee', '10B', 83.25);

INSERT INTO Student1 (R_no, Name, Class, Marks)
VALUES ('S009', 'Hannah Scott', '10C', 77.50);

INSERT INTO Student1 (R_no, Name, Class, Marks)
VALUES ('S010', 'Isaac Taylor', '10A', 90.00);
```

Q13. Display all the contents of student and student1 using union clause.

```
SELECT RollNumber AS R_no, Name, Class, Marks
FROM Student
UNION
SELECT R_no, Name, Class, Marks
FROM Student1;
```

Output

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

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

```
SELECT Student.name, Student1.name
FROM Student
INNER JOIN Student1 ON Student.RollNumber = Student1.R_no;
```

```
SELECT Student.name, Student1.name
FROM Student
RIGHT JOIN Student1 ON Student.student_RollNumber = Student1.Student.R_no
```

```
SELECT Student.name, Student1.name
FROM Student
LEFT JOIN Student1 ON Student.Student_RollNumber = Student1.Student_R_no;
```

```
SELECT Student.name, Student1.name
FROM Student
FULL JOIN Student1 ON Student.Student_RollNumber = Student1.Student_R_no;
```

Q15. To Write a PL/SQL block to calculate total salary of employee having employee number 100.

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

Q16. To Write a PL/SQL code to find the greatest of three numbers.

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

Q17. To Write a PL/SQL code to find the greatest of three numbers.

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 SQ

```
-- 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

Q18. To Write a PL/SQL code to reverse a string using for loop.

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

```
-- 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

Q19. To Write a PL/SQL code to find the sum of n numbers.

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

Q20. To Consider a PL/SQL code to display the empno, ename, job of employees of department number 10.

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);
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

Q21. To Consider a PL/SQL code to display the employee number & name of top five highest paid employees.

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:

Output

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

EMPLOYEE2

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

Q22. To Consider a PL/SQL procedure that accepts 2 numbers & return addition, subtraction, multiplication & division of two numbers using stored procedure AND local procedure.

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

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

Q23. To Consider a PL/SQL code that accepts 2 numbers & return addition, subtraction, multiplication & division of two numbers using stored functions and local function.

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 [-]

Programiz
Online SQL Editor

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
```


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;
```

UTPUT:

Input_Numbers

num1	num2
20	10

Output

operation	result
Addition	30
Subtraction	10
Multiplication	200
Division	2

Q24.To Write a PL/SQL block to show the use of NO_DATA FOUND exception.

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 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

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

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 (
```


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

Q26. To Write a PL/SQL block to show the use of ZERO_DIVIDE exception.

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

```
-- 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

Q27. 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.

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
);
```


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

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

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);
```

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

Q28. To create a trigger so that no operation can be performed on emp table.

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

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

```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

Input

Run SQL

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

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

```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

Input

Run SQL

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

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

```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