

EXPERIMENT-1

1.CREATE Table:

The screenshot shows the 'Live SQL' interface with the 'SQL Worksheet' tab selected. On the left, there's a sidebar with options like 'Home', 'SQL Worksheet' (which is selected), 'My Session', 'Schema', 'Quick SQL', 'My Scripts', 'My Tutorials', and 'Code Library'. In the main workspace, the following SQL code is written:

```
1 v CREATE TABLE Student_info
2 (
3     College_Id number(2),
4     College_name varchar(30),
5     Branch varchar(10)
6 )
```

Below the code, the message 'Table created.' is displayed.

2.ALTER table:

The screenshot shows the 'Live SQL' interface with the 'SQL Worksheet' tab selected. The sidebar is identical to the previous screenshot. In the main workspace, the following SQL code is written:

```
1 v ALTER TABLE Student_info
2 ADD CGPA number
```

Below the code, the message 'Table altered.' is displayed.

3.Drop Table:

The screenshot shows an SQL Worksheet interface. At the top, there are navigation links for Feedback, Help, and a user account (224g1a0565@srit.ac.in). Below the header, the title "SQL Worksheet" is displayed. The main area contains a single line of code: "1 DROP TABLE Student_info". After executing the command, the message "Table dropped." is shown at the bottom.

4.TRUNCATE Table:

The screenshot shows the "My Session" history of SQL statements. It lists four statements:

- Statement 9: TRUNCATE TABLE Student_info; Result: Table truncated.
- Statement 8: ALTER TABLE Student_info ADD CGPA number; Result: Table altered.
- Statement 7: CREATE TABLE Student_info (College_id number(2), College_name varchar(30), Branch varchar(10)); Result: Table created.
- Statement 6: (No visible output)

Each statement row includes edit, run, and delete icons.

conclusion:

In this lab, we successfully practiced SQL queries to CREATE TABLES for various databases using DDL commands.

EXPERIMENT-2

1.CREATE TABLE for employees:

The screenshot shows a SQL Worksheet interface with the following details:

- Toolbar:** Feedback, Help, User (224g1a0565@srit.ac.in), Clear, Find, Actions, Save.
- Code Area:** Contains the SQL code for creating the 'employees' table:

```
1 v CREATE TABLE employees (
2     employee_id INT PRIMARY KEY,
3     first_name VARCHAR(50),
4     last_name VARCHAR(50),
5     salary DECIMAL(10, 2),
6     department_id INT
7 );
```
- Status Bar:** Shows the message "Table created."

2.CREATE TABLE for customers:

The screenshot shows a SQL Worksheet interface with the following details:

- Toolbar:** Feedback, Help, User (224g1a0565@srit.ac.in), Clear, Find, Actions, Save.
- Code Area:** Contains the SQL code for creating the 'customers' table:

```
1 v CREATE TABLE customers (
2     customer_id INT PRIMARY KEY,
3     first_name VARCHAR(50),
4     last_name VARCHAR(50),
5     email VARCHAR(100)
6 );
```
- Status Bar:** Shows the message "SQL Statement Output".

3.CREATE TABLE for products:

The screenshot shows a SQL Worksheet interface with a light gray header bar containing 'Feedback', 'Help', and a user email '224g1a0565@srit.ac.in'. Below the header is a toolbar with 'Clear', 'Find', 'Actions', and 'Save' buttons. The main area is titled 'SQL Worksheet' and contains the following SQL code:

```
1 v CREATE TABLE employee (
2     employee_id INT PRIMARY KEY,
3     first_name VARCHAR(50),
4     last_name VARCHAR(50),
5     salary DECIMAL(10, 2),
6     department_id INT
7 )
```

Table created.

The screenshot shows a SQL Worksheet interface with a light gray header bar containing 'Feedback', 'Help', and a user email '224g1a0565@srit.ac.in'. Below the header is a toolbar with 'Clear', 'Find', 'Actions', and 'Save' buttons. The main area is titled 'SQL Worksheet' and contains the following SQL code:

```
1 v CREATE TABLE products (
2     product_id INT PRIMARY KEY,
3     product_name VARCHAR(100),
4     category VARCHAR(50),
5     stock_quantity INT
6 )
```

A cursor icon is visible near the bottom left of the code area. Below the code, the message 'Table created.' is displayed.

4.CREATE TABLE for orders:

The screenshot shows a SQL Worksheet interface with a toolbar at the top featuring 'Feedback', 'Help', and a user account icon. Below the toolbar is a menu bar with 'Clear', 'Find', 'Actions', and 'Save'. The main area is titled 'SQL Worksheet' and contains the following SQL code:

```
1 v CREATE TABLE orders (
2   order_id INT PRIMARY KEY,
3   customer_id INT,
4   order_date DATE,
5   FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
6 );
```

Below the code, a status bar indicates 'SQL Statement Output'.

5.CREATE TABLE for logs:

The screenshot shows a SQL Worksheet interface with a toolbar at the top featuring 'Feedback', 'Help', and a user account icon. Below the toolbar is a menu bar with 'Clear', 'Find', 'Actions', and 'Save'. The main area is titled 'SQL Worksheet' and contains the following SQL code:

```
1 v CREATE TABLE logs (
2   log_id INT PRIMARY KEY,
3   log_message VARCHAR(255)
4 );
```

Below the code, a status bar indicates 'Table created.'

6.INSERT:

The screenshot shows a SQL Worksheet interface with a toolbar at the top featuring 'Live SQL', 'Feedback', 'Help', and a user account icon. Below the toolbar is a menu bar with 'Clear', 'Find', 'Actions', and 'Save'. The main area is titled 'SQL Worksheet' and contains the following SQL code:

```
1 v INSERT INTO customers (customer_id, first_name, last_name, email)
2 VALUES (1, 'John', 'Doe', 'john.doe@example.com');
```

Below the code, a status bar indicates '1 row(s) inserted.'

7.SELECT:

A screenshot of a SQL worksheet interface. At the top, there are navigation links for Feedback, Help, and a user account (224g1a0565@srit.ac.in). Below the header is a toolbar with Clear, Find, Actions, and Save buttons. The main area is titled "SQL Worksheet". A single line of SQL code is present: "1 SELECT * FROM employees;". Below the code, the output shows the message "no data found".

8.UPDATE:

A screenshot of a SQL worksheet interface. At the top, there are navigation links for Feedback, Help, and a user account (224g1a0565@srit.ac.in). Below the header is a toolbar with Clear, Find, Actions, and Save buttons. The main area is titled "SQL Worksheet". A single line of SQL code is present: "1 UPDATE employees SET salary = salary * 1.1 WHERE department_id = 2;". Below the code, the output shows the message "0 row(s) updated.".

9.DELETE:

A screenshot of a SQL worksheet interface. At the top, there are navigation links for Feedback, Help, and a user account (224g1a0565@srit.ac.in). Below the header is a toolbar with Clear, Find, Actions, and Save buttons. The main area is titled "SQL Worksheet". A single line of SQL code is present: "1 DELETE FROM customers WHERE customer_id = 1;". Below the code, the output shows the message "1 row(s) deleted.".

conclusion:

In this lab, we successfully practised SQL queries using DML commands (SELECT, UPDATE, INSERT, DELETE).

EXPERIMENT-3

1.CREATE TABLE:

The screenshot shows a SQL worksheet interface with a toolbar at the top. The code input area contains the following SQL statement:

```
1 v CREATE TABLE employees (
2   employee_id INT PRIMARY KEY,
3   first_name VARCHAR(50),
4   last_name VARCHAR(50),
5   department_id INT
6 );
```

Below the code, the message "Table created." is displayed.

2.CREATING A VIEW:

The screenshot shows a SQL worksheet interface with a toolbar at the top. The code input area contains the following SQL statement:

```
1 v CREATE VIEW employee_view AS
2 SELECT employee_id, first_name, last_name
3 FROM employees
4 WHERE department_id = 1;
```

Below the code, the message "View created." is displayed.

3.UPDATING A VIEW:

The screenshot shows a SQL worksheet interface with a toolbar at the top. The code input area contains the following SQL statement:

```
1 v UPDATE employees
2 SET department_id = 2
3 WHERE employee_id = 100;
```

Below the code, the message "0 row(s) updated." is displayed.

4.ALTERING A VIEW:

The screenshot shows a SQL worksheet interface with a toolbar at the top. The code in the worksheet is as follows:

```
1v CREATE OR REPLACE VIEW employee_view AS
2 SELECT employee_id, first_name, last_name
3 FROM employees
4 WHERE department_id = 2;
```

Below the code, the message "View created." is displayed.

5.DELETING A VIEW:

The screenshot shows a SQL worksheet interface with a toolbar at the top. The code in the worksheet is as follows:

```
1 DROP VIEW employee_view;
```

Below the code, the message "View dropped." is displayed.

conclusion:

In the lab, we successfully practiced SQL Queries to create VIEWS for various data base (CREATE VIEW, UPDATE VIEW, ALTER VIEW,, DELETE VIEW).

EXPERIMENT-4

1.creating a table1:

The screenshot shows a SQL worksheet interface with a toolbar at the top. The code input area contains the following SQL statement:

```
1 v CREATE TABLE table1 (
2   id INT,
3   name VARCHAR(50),
4   PRIMARY KEY (id)
5 );
```

Below the code, the message "Table created." is displayed.

2.inserting VALUES into table1:

The screenshot shows a SQL worksheet interface with a toolbar at the top. The code input area contains the following SQL statements:

```
1 INSERT INTO table1 (id, name) VALUES (1, 'John');
2 INSERT INTO table1 (id, name) VALUES (2, 'Alice');
3 INSERT INTO table1 (id, name) VALUES (3, 'Bob');
```

Below the code, three messages are displayed: "1 row(s) inserted.", "1 row(s) inserted.", and "1 row(s) inserted.".

3.creating a table2:

The screenshot shows a SQL worksheet interface with a toolbar at the top. The code input area contains the following SQL statement:

```
1 v CREATE TABLE table2 (
2   id INT,
3   name VARCHAR(50),
4   PRIMARY KEY (id)
5 );
```

Below the code, the message "Table created." is displayed.

4.Inserting VALUES into table2:

Live SQL

SQL Worksheet

```
1 INSERT INTO table2 (id, name) VALUES (2, 'Alice');
2 INSERT INTO table2 (id, name) VALUES (3, 'Bob');
3 INSERT INTO table2 (id, name) VALUES (4, 'Charlie');
```

Feedback Help 224g1a0565@srit.ac.in Run

Clear Find Actions Save

1 row(s) inserted.
1 row(s) inserted.
1 row(s) inserted.

5.UNION:

Live SQL

SQL Worksheet

```
1 v SELECT * FROM table1
2 UNION
3 SELECT * FROM table2
```

Feedback Help 224g1a0565@srit.ac.in Run

Clear Find Actions Save

ID	NAME
1	John
2	Alice
3	Bob
4	Charlie

6.UNION ALL:

Live SQL

SQL Worksheet

```
1 v SELECT * FROM table1
2 UNION ALL
3 SELECT * FROM table2
```

Feedback Help 224g1a0565@srit.ac.in Run

Clear Find Actions Save

ID	NAME
1	John
2	Alice
3	Bob
4	Charlie

7.INTERSECT:

Live SQL

SQL Worksheet

```
1 v SELECT * FROM table1
2 INTERSECT
3 SELECT * FROM table2;
```

ID	NAME
2	Alice
3	Bob

[Download CSV](#)

8.CROSS JOIN:

Feedback Help 224g1a0565@srit.ac.in Actions Save

My Session

```
SELECT * FROM table1
CROSS JOIN table2
```

ID	NAME	ID	NAME
1	John	2	Alice
1	John	3	Bob
1	John	4	Charlie
2	Alice	2	Alice
2	Alice	3	Bob
2	Alice	4	Charlie
3	Bob	2	Alice
3	Bob	3	Bob
3	Bob	4	Charlie

[Download CSV](#)

9 rows selected.

9.NATURAL JOIN:

The screenshot shows a SQL worksheet interface. At the top, there are navigation links for Feedback, Help, and a user account (224g1a0565@srit.ac.in). Below the header are buttons for Clear, Find, Actions, Save, and Run. The code input area contains the following SQL query:

```
1 ✓ SELECT * FROM table1
2 NATURAL JOIN table2;
```

Below the code, the results are displayed in a table:

ID	NAME
2	Alice
3	Bob

At the bottom left, there is a "Download CSV" button.

conclusion:

In this lab, we successfully practiced how to perform SQL Queries on RELATIONAL OPERATIONS (UNION, UNION ALL, CROSS JOIN, NATURAL JOIN).

EXPERIMENT-5

WRITE SQL QUERIES TO PERFORM AGREGATE FUNCTIONS (count, sum, avg, min, max)

Aim:

To implement SQL QUERIES to perform Aggregate Functions (count, sum, avg, min, max)

Procedure:

```
C:\Users\kumma>sqlplus  
SQL*Plus: Release 21.0.0.0.0 - Production on Thu Nov 9 11:53:35 2023  
Version 21.3.0.0.0  
  
Copyright (c) 1982, 2021, Oracle. All rights reserved.
```

```
Enter user-name: system  
Enter password:  
Last Successful login time: Sat Oct 14 2023 10:11:50 +05:30  
  
Connected to:  
Oracle Database 21c Express Edition Release 21.0.0.0.0 - Production  
Version 21.3.0.0.0
```

```
SQL> CREATE TABLE Emp1  
 2  (eid int,  
 3  eName VARCHAR(20),  
 4  eSalary int);  
  
Table created.
```

224g1a0565-k.pallavi

```
SQL> INSERT INTO Emp1 VALUES('1','Siddu','20000');

1 row created.

SQL> INSERT INTO Emp1 VALUES('2','shiva','30000');

1 row created.

SQL> INSERT INTO Emp1 VALUES('3','naveen','40000');

1 row created.

SQL> INSERT INTO Emp1 VALUES('3','Bhanu','50000');

1 row created.

SQL> INSERT INTO Emp1 VALUES('5','Uma','80000');

1 row created.
```

```
SQL> SELECT * from Emp1;

      EID ENAME          ESALARY
----- -----
      1 Siddu           20000
      2 shiva           30000
      3 naveen          40000
      3 Bhanu           50000
      5 Uma             80000
```

```
SQL> SELECT avg(eid)
  2  from Emp1;

AVG(EID)
-----
      2.8
```

224g1a0565-k.pallavi

```
SQL> SELECT min(eid)
  2  from Emp1;
```

```
MIN(EID)
```

```
-----  
1
```

```
SQL> SELECT max(eid)
  2  from Emp1;
```

```
MAX(EID)
```

```
-----  
5
```

```
SQL> SELECT count(*) eid
  2  from Emp1;
```

```
EID
```

```
-----  
5
```

```
SQL> SELECT sum(eid)
  2  from Emp1;
```

```
SUM(EID)
```

```
-----  
14
```

Conclusion:

In this lab, we implemented aggregate functions successfully.

EXPERIMENT-6

1.creating a employee table:

The screenshot shows a SQL worksheet interface with a header bar featuring 'Live SQL' and user information. Below the header is a toolbar with 'Clear', 'Find', 'Actions', 'Save', and a 'Run' button. The main area is titled 'SQL Worksheet'. A code editor contains the following SQL command:

```
1 v CREATE TABLE employees (
2     employee_id INT,
3     employee_name VARCHAR(50),
4     salary INT,
5     department VARCHAR(50),
6     PRIMARY KEY (employee_id)
7 );
```

Below the code, a message 'Table created.' is displayed.

2.inserting VALUES into employee:

The screenshot shows a session history interface with a header bar featuring 'Feedback', 'Help', user information, and a 'Run' button. Below the header is a toolbar with 'Actions' and 'Reset Session'. The main area is titled 'My Session'. It displays four rows of statements:

Statement	Action	Query	Result
1			Table created.
2		INSERT INTO employees (employee_id, employee_name, salary, department) VALUES (1,'abc',70000,'cse')	1 row(s) inserted.
3		INSERT INTO employees (employee_id, employee_name, salary, department) VALUES (2,'abc',63000,'eee')	1 row(s) inserted.
4		INSERT INTO employees (employee_id, employee_name, salary, department) VALUES (3,'abc',80000,'ece')	1 row(s) inserted.

3.Creating a department table:

The screenshot shows a SQL Worksheet interface with the following code:

```
1 CREATE TABLE departments (
2     department_id INT,
3     department_name VARCHAR(50),
4     PRIMARY KEY (department_id)
5 );
```

Below the code, the message "Table created." is displayed.

4.inserting VALUES into table :

The screenshot shows a SQL Worksheet interface with the following code:

```
1 INSERT INTO departments (department_id, department_name) VALUES (1, 'IT');
2 INSERT INTO departments (department_id, department_name) VALUES (2, 'HR');
3 INSERT INTO departments (department_id, department_name) VALUES (3, 'Finance');
```

1 row(s) inserted.
1 row(s) inserted.
1 row(s) inserted.

conclusion:

In this lab, we successfully practiced SQL Queries on JOIN OPERATIONS (CONDITIONAL JOIN, EQUI JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN).

EXPERIMENT-7

1.create department table:

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark mode icon.
- Toolbar:** Clear, Find, Actions, Save, Run button.
- SQL Worksheet Area:** Contains the following SQL code:

```
1 v CREATE TABLE DEPARTMENT
2 (DEPT_NAME VARCHAR2(20),
3 BUILDING VARCHAR2(15),
4 BUDGET NUMERIC(12,2) CHECK (BUDGET > 0),
5 PRIMARY KEY (DEPT_NAME)
6 );
```
- Output Area:** Displays the message "Table created."

2.create instructor table:

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark mode icon.
- Toolbar:** Clear, Find, Actions, Save, Run button.
- SQL Worksheet Area:** Contains the following SQL code:

```
1 v CREATE TABLE INSTRUCTOR
2 (ID VARCHAR2(5),
3 NAME VARCHAR2(20) NOT NULL,
4 DEPT_NAME VARCHAR2(20),
5 SALARY NUMERIC(8,2) CHECK (SALARY > 29000),
6 PRIMARY KEY (ID),
7 FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
8 ON DELETE SET NULL
9 );
```
- Output Area:** Displays the message "Table created."

3.instances of instructor:

The screenshot shows a SQL worksheet interface with the following content:

```

1 insert into department values ('Biology', 'Watson', '90000');
2 insert into department values ('Comp. Sci.', 'Taylor', '100000');
3 insert into department values ('Elec. Eng.', 'Taylor', '85000');
4 insert into department values ('Finance', 'Painter', '120000');
5 insert into department values ('History', 'Painter', '50000');
6 insert into department values ('Music', 'Packard', '80000');
7 insert into department values ('Physics', 'Watson', '70000');

```

Below the code, the output shows four rows inserted:

```

1 row(s) inserted.
1 row(s) inserted.
1 row(s) inserted.
1 row(s) inserted.

```

4.create course table:

The screenshot shows a SQL worksheet interface with the following content:

```

1 v CREATE TABLE COURSE
2 (COURSE_ID VARCHAR2(8),
3 TITLE VARCHAR2(50),
4 DEPT_NAME VARCHAR2(20),
5 CREDITS NUMERIC(2,0) CHECK (CREDITS > 0),
6 PRIMARY KEY (COURSE_ID),
7 FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
8 ON DELETE SET NULL
9 );

```

Below the code, the output shows the table was created:

```

Table created.

```

5.instances of COURSE table:

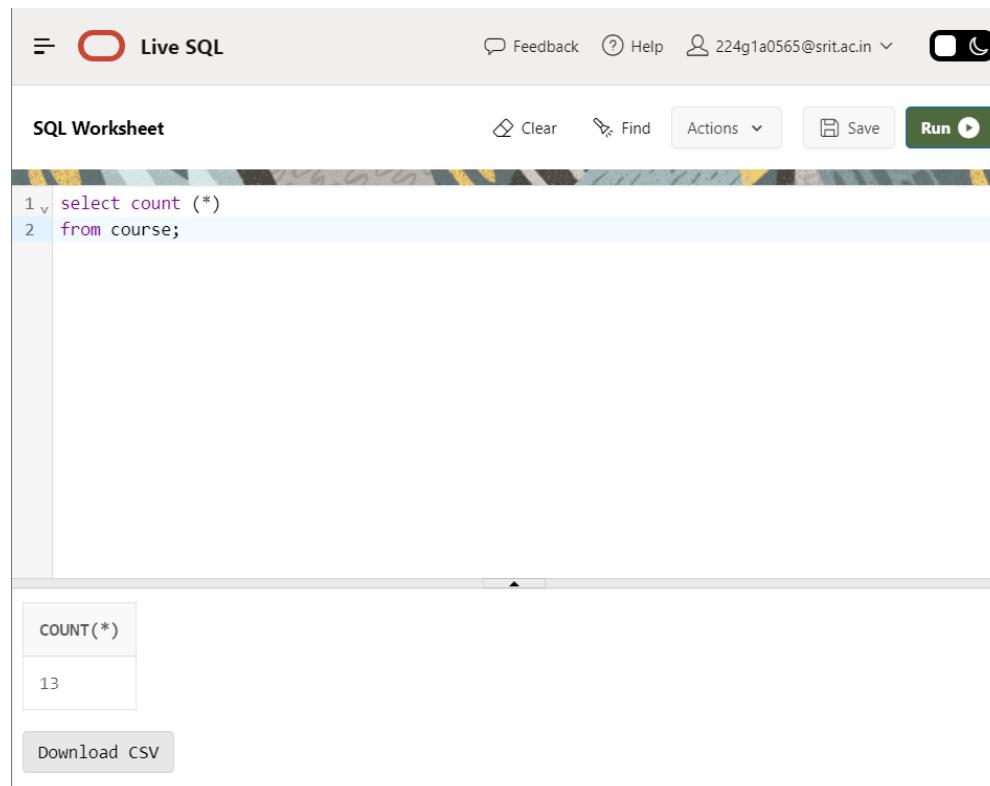
The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, Email (224g1a0565@srit.ac.in), Dark Mode toggle.
- Toolbar:** Clear, Find, Actions, Save, Run.
- SQL Worksheet Area:** Contains 13 numbered SQL insert statements for the COURSE table.
- Output Area:** Shows the confirmation message "1 row(s) inserted." repeated five times.

```
1 insert into course values ('BIO-101', 'Intro. to Biology', 'Biology', '4');
2 insert into course values ('BIO-301', 'Genetics', 'Biology', '4');
3 insert into course values ('BIO-399', 'Computational Biology', 'Biology', '3');
4 insert into course values ('CS-101', 'Intro. to Computer Science', 'Comp. Sci.', '4');
5 insert into course values ('CS-190', 'Game Design', 'Comp. Sci.', '4');
6 insert into course values ('CS-315', 'Robotics', 'Comp. Sci.', '3');
7 insert into course values ('CS-319', 'Image Processing', 'Comp. Sci.', '3');
8 insert into course values ('CS-347', 'Database System Concepts', 'Comp. Sci.', '3');
9 insert into course values ('EE-181', 'Intro. to Digital Systems', 'Elec. Eng.', '3');
10 insert into course values ('FIN-201', 'Investment Banking', 'Finance', '3');
11 insert into course values ('HIS-351', 'World History', 'History', '3');
12 insert into course values ('MU-199', 'Music Video Production', 'Music', '3');
13 insert into course values ('PHY-101', 'Physical Principles', 'Physics', '4');
```

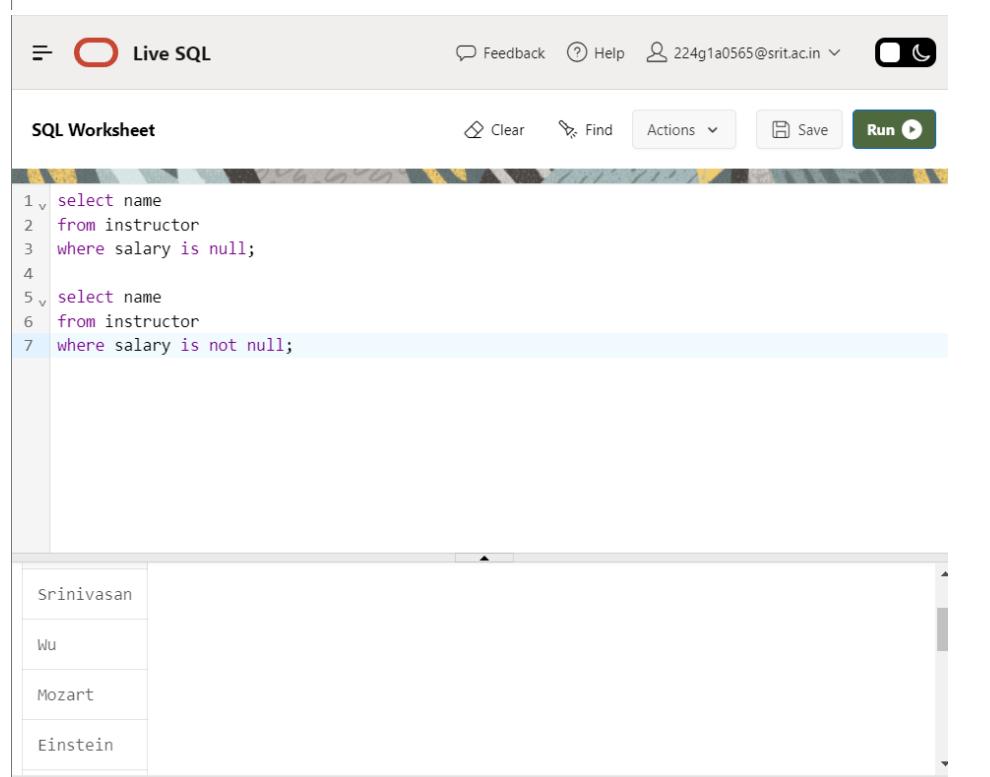
1 row(s) inserted.
1 row(s) inserted.
1 row(s) inserted.
1 row(s) inserted.

To find no.of course taught in the university:



The screenshot shows a SQL worksheet interface with the following details:

- Top Bar:** Includes "Live SQL" icon, "Feedback" button, "Help" button, user info "224g1a0565@srit.ac.in", and a toggle switch.
- SQL Worksheet:** Shows the query "select count (*) from course;"
- Result Area:** Displays the output of the query: "COUNT(*)" and "13".
- Buttons:** "Download CSV" button.



The screenshot shows a second SQL worksheet interface with the following details:

 - Top Bar:** Includes "Live SQL" icon, "Feedback" button, "Help" button, user info "224g1a0565@srit.ac.in", and a toggle switch.
 - SQL Worksheet:** Shows two queries:

```
1 v select name
2   from instructor
3  where salary is null;
4
5 v select name
6   from instructor
7  where salary is not null;
```
 - Result Area:** Displays the names of instructors with null and non-null salaries. The table has two columns: "name" and "salary".

name	salary
Srinivasan	NULL
Wu	NULL
Mozart	NULL
Einstein	NOT NULL

Conclusion:

In this lab, we successfully practiced SQL Queries to perform AGGREGATE operations.

EXPERIMENT-8

1. NUMBER FUNCTIONS:

The screenshot shows two separate sessions of a SQL worksheet interface. Both sessions are titled "Live SQL" and "SQL Worksheet". Each session has a toolbar with "Feedback", "Help", "Actions", "Save", and a "Run" button.

Session 1 (Top):

```
1 v SELECT ROUND(45.626,2)
2   FROM DUAL;
3 v SELECT ROUND(45.626,0)
4   FROM DUAL;
5 v SELECT ROUND(45.626,-1)
6   FROM DUAL;
7 v SELECT ROUND(45.626,-2)
8   FROM DUAL;
9
10 v SELECT TRUNC(45.626, 2)
11   FROM DUAL;
12 v SELECT TRUNC(45.626, 0)
13   FROM DUAL;
14 v SELECT TRUNC(45.626, -1)
15   FROM DUAL;
```

Output for Session 1:

ROUND(45.626,2)
45.63

Session 2 (Bottom):

```
1 v SELECT ROUND(45.626,2)
2   FROM DUAL;
3 v SELECT ROUND(45.626,0)
4   FROM DUAL;
5 v SELECT ROUND(45.626,-1)
6   FROM DUAL;
```

Output for Session 2:

ROUND(45.626,2)
45.63

ROUND(45.626,0)
46

ROUND(45.626,-1)
50

Live SQL Feedback Help 224g1a0565@srit.ac.in

SQL Worksheet Clear Find Actions Save Run

ROUND(45.626,-1)
50

Download CSV

ROUND(45.626,-2)
0

Download CSV

TRUNC(45.626,2)
45.62

Download CSV

▼ 2.D

Live SQL Feedback Help 224g1a0565@srit.ac.in

SQL Worksheet Clear Find Actions Save Run

TRUNC(45.626,-1)
40

Download CSV

TRUNC(45.626,-2)
0

Download CSV

MOD(1600,300)
100

Download CSV

2.Date Functions:

The screenshot shows two sessions of the Oracle Live SQL interface. Both sessions have the title "Live SQL" and "SQL Worksheet". The top session contains the following SQL code:

```

1 v SELECT SYSDATE
2 FROM DUAL;
3
4 v SELECT MONTHS_BETWEEN(SYSDATE, '15-FEB-20')
5 FROM DUAL;
6
7 v SELECT ADD_MONTHS(SYSDATE, 2)
8 FROM DUAL;
9
10 v SELECT NEXT_DAY(SYSDATE, 'THURSDAY')
11 FROM DUAL;
12
13 v SELECT LAST_DAY(SYSDATE)
14 FROM DUAL;
15

```

The results pane shows the output for the first query:

SYSDATE
07-JAN-24

A "Download CSV" button is present below the results.

The bottom session also has the title "Live SQL" and "SQL Worksheet". It contains the same SQL code as the top session. The results pane shows the output for the second query:

MONTHS_BETWEEN(SYSDATE, '15-FEB-20')
46.74911701015531660692951015531660692951

A "Download CSV" button is present below the results.

Both sessions also show the output for the third query:

ADD_MONTHS(SYSDATE,2)
07-MAR-24

A "Download CSV" button is present below the results.

Conclusion:

In this lab, we successfully practiced SQL Queries to perform ORACLE BUILT-IN Function.

224g1a0565-k.pallavi

EXPERIMENT-9

1.NOT NULL CONSTRAINT Example:

The screenshot shows a SQL worksheet interface with the following code:

```
1 v CREATE TABLE student (
2   ID int NOT NULL,
3   LastName varchar(255) NOT NULL,
4   FirstName varchar(255) NOT NULL,
5   Age int
6 );
7 v ALTER TABLE student
8   MODIFY Age int NOT NULL;
```

Below the code, the output is displayed in two parts:

Table created.
Table altered.

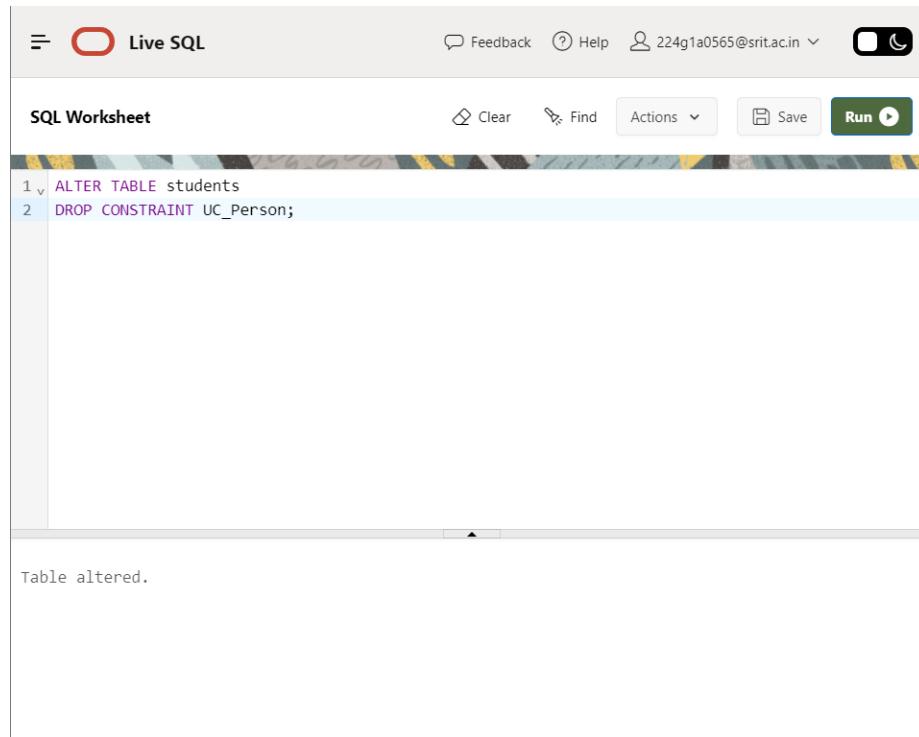
2.UNIQUE CONSTRAINT Example:

The screenshot shows a SQL worksheet interface with the following code:

```
1 v CREATE TABLE Students(
2   ID int NOT NULL,
3   LastName varchar(255) NOT NULL,
4   FirstName varchar(255),
5   Age int,
6   CONSTRAINT UC_Person UNIQUE (ID,LastName)
7 );
```

Below the code, the output is displayed:

Table created.

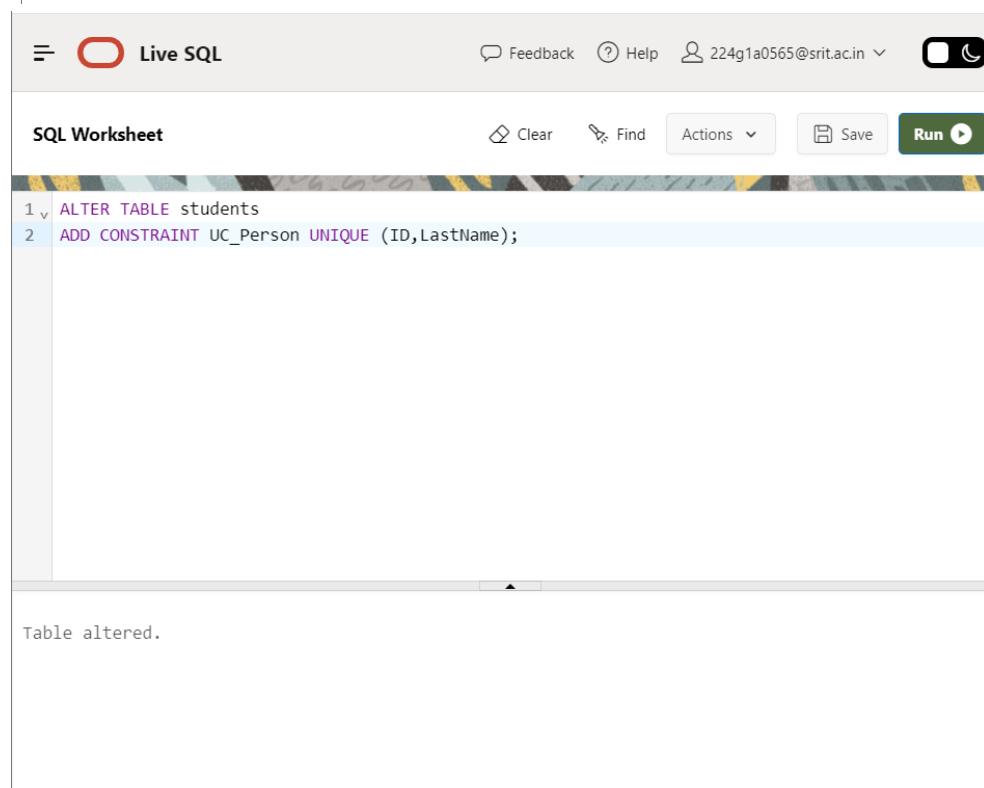


Live SQL

SQL Worksheet

```
1 v ALTER TABLE students
2   DROP CONSTRAINT UC_Person;
```

Table altered.



Live SQL

SQL Worksheet

```
1 v ALTER TABLE students
2   ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

Table altered.

3.PRIMARY KEY CONSTRAINT Example:

Live SQL

SQL Worksheet

```
1 ✓ CREATE TABLE Persons (
2   ID int NOT NULL,
3   LastName varchar(255) NOT NULL,
4   FirstName varchar(255),
5   Age int,
6   CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
7 );
```

Table created.

Live SQL

SQL Worksheet

```
1 ✓ ALTER TABLE Persons
2 DROP CONSTRAINT PK_Person;
```

SQL Statement Output

4.DEFAULT CONSTRAINTS Example:

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark mode switch.
- Toolbar:** Clear, Find, Actions, Save, Run.
- SQL Worksheet Area:** Contains the following SQL code:

```
1 v CREATE TABLE Person (
2 ID int NOT NULL,
3 LastName varchar(255) NOT NULL,
4 FirstName varchar(255),
5 Age int,
6 City varchar(255),
7 CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
8 );
9
```
- Output Area:** Displays the message "Table created."

Conclusion:

In this lab, we successfully executed SQL Queries to perform KEY CONSTRAINTS.

EXPERIMENT-10

1. Factorial Program

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark Mode toggle.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- Code Area:** A code editor containing PL/SQL code to calculate the factorial of 10. The code is as follows:

```
1 v DECLARE
2   fac NUMBER :=1;
3   n NUMBER := 10;
4 v BEGIN
5   WHILE n > 0 LOOP
6     fac:=n*fac;
7     n:=n-1;
8   END LOOP;
9   DBMS_OUTPUT.PUT_LINE(FAC);
10  END;
```

Output Area: Statement processed.
3628800

EXPERIMENT-11

1. Prime number or not Program:

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark mode switch.
- Toolbar:** Clear, Find, Actions, Save, Run button.
- Code Area:** A code editor containing PL/SQL code to check if the number 13 is prime. The code uses variables n, i, and temp, and a loop from 2 to n/2 to check for divisibility.

```
1 v DECLARE
2   n NUMBER;
3   i NUMBER;
4   temp NUMBER;
5 v BEGIN
6   n := 13;
7   i := 2;
8   temp := 1;
9 v FOR i IN 2..n/2
10 LOOP
11   IF MOD(n, i) = 0
12   THEN
13     temp := 0;
14   EXIT;
15 END IF;
```

- Output Area:** Displays the message "Statement processed." followed by "13 is a prime number".

conclusion:

In this lab, we successfully practiced a program for Finding the given number is prime or not.

EXPERIMENT-12

1.Fibonacci Series Program:

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark mode icon.
- Toolbar:** Clear, Find, Actions, Save, Run (highlighted).
- Code Area:** PL/SQL code for generating a Fibonacci series up to a given integer N (5 in this case). The code uses DBMS_OUTPUT.PUT_LINE to print the series.

```
1 v DECLARE
2   FIRST NUMBER := 0;
3   SECOND NUMBER := 1;
4   TEMP NUMBER;
5   N NUMBER := 5;
6   I NUMBER;
7 v BEGIN
8   DBMS_OUTPUT.PUT_LINE('SERIES:');
9   DBMS_OUTPUT.PUT_LINE(FIRST);
10  DBMS_OUTPUT.PUT_LINE(SECOND);
11 v FOR I IN 2..N
12 LOOP
13   TEMP:=FIRST+SECOND;
14   FIRST := SECOND;
15   SECOND := TEMP;
```

- Output Area:** Displays the processed statement and the resulting Fibonacci series: 0, 1, 1, 2, 3, 5.

Conclusion :

In this lab, we successfully practiced a program for displaying Fibonacci Series up to an integer.

Experiment-13

1.Create table:

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Logout.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- Code Area:** Contains the following SQL code:

```
1 CREATE TABLE SAILOR(ID NUMBER(10) PRIMARY KEY,NAME VARCHAR2(100));
2 v CREATE OR REPLACE PROCEDURE INSERTUSER
3 (ID IN NUMBER,
4 NAME IN VARCHAR2)
5 IS
6 BEGIN
7 INSERT INTO SAILOR VALUES(ID,NAME);
8 DBMS_OUTPUT.PUT_LINE('RECORD INSERTED SUCCESSFULLY');
9 END;
```

Table created.

Procedure created.

2.Execution Procedure:

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark mode switch.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- Code Area:** Contains the following PL/SQL code:

```
1 v DECLARE
2 CNT NUMBER;
3 v BEGIN
4 INSERTUSER(101, 'NARASIMHA');
5 SELECT COUNT(*) INTO CNT FROM SAILOR;
6 DBMS_OUTPUT.PUT_LINE(CNT||' RECORD IS INSERTED SUCCESSFULLY');
7 END;
```
- Output Area:** Displays the results of the execution:

```
Statement processed.
RECORD INSERTED SUCCESSFULLY
1 RECORD IS INSERTED SUCCESSFULLY
```

3.Drop Procedure:

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark mode switch.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- Code Area:** Shows the SQL command: `DROP PROCEDURE insertuser;`
- Output Area:** Displays the message: `Procedure dropped.`

Conclusion: In this lab, we successfully practiced How to write a program to implement Stored Procedure on table.

Experiment-14

1.PL/SQL Function:

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark mode switch.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- Code Area:** PL/SQL code for creating a function ADDER.

```
1 v CREATE OR REPLACE FUNCTION ADDER(N1 IN NUMBER, N2 IN NUMBER)
2   RETURN NUMBER
3   IS
4     N3 NUMBER(8);
5   BEGIN
6     N3 :=N1+N2;
7     RETURN N3;
8   END;
```

Function created.

2.Execution Procedure:

The screenshot shows the Oracle Live SQL interface. At the top, there are navigation links for Feedback, Help, and a user account (224g1a0565@srit.ac.in). On the right, there is a dark mode toggle switch. Below the header, the title "SQL Worksheet" is displayed, along with buttons for Clear, Find, Actions (with a dropdown arrow), Save, and Run (with a play icon).

The code area contains the following PL/SQL block:

```
1 v DECLARE
2   N3 NUMBER(2);
3 v BEGIN
4   N3 := ADDER(11,22);
5   DBMS_OUTPUT.PUT_LINE('ADDITION IS: ' || N3);
6 END;
7 /
```

After running the code, the output window displays the results:

```
Statement processed.
ADDITION IS: 33
```

3.Drop Procedure:

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark Mode toggle.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- Code Area:** A single line of SQL code: "1 DROP FUNCTION Adder;".
- Output Area:** The text "Function dropped." is displayed.

4.Recursive Function:

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark Mode toggle.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- Code Area:** A code editor containing the following SQL code:

```
1 v CREATE FUNCTION fact(x number)
2   RETURN number
3   IS
4     f number;
5   BEGIN
6     IF x=0 THEN
7       f := 1;
8     ELSE
9       f := x * fact(x-1);
10    END IF;
11    RETURN f;
12  END;
```
- Output Area:** A message indicating the function was created: "Function created."

5.Execution Procedure:

The screenshot shows the Oracle Live SQL interface. At the top, there's a navigation bar with 'Live SQL' and user information. Below it is a toolbar with 'SQL Worksheet' and various actions like 'Clear', 'Find', 'Actions', 'Save', and a prominent 'Run' button. The main area is a code editor with the following PL/SQL code:

```
1 v DECLARE
2 num number;
3 factorial number;
4 v BEGIN
5 num:= 6;
6 factorial := fact(num);
7 dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
8 END;
```

Statement processed.
Factorial 6 is 720

6.Drop Function:

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark Mode toggle.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- Code Area:** A single line of code: "1 DROP FUNCTION fact;".
- Output Area:** The message "Function dropped.".

Conclusion: In This Lab, we successfully practiced PL/SQL Program to implement Stored Function on Table

Experiment-15

1.Create Department Table

The screenshot shows a 'Live SQL' interface with a 'SQL Worksheet' tab. The code entered is:

```
1 v CREATE TABLE DEPARTMENT
2 (DEPT_NAME VARCHAR2(20),
3 BUILDING VARCHAR2(15),
4 BUDGET NUMERIC(12,2) CHECK (BUDGET > 0),
5 PRIMARY KEY (DEPT_NAME)
6 );
```

Below the code, the message 'Table created.' is displayed.

2.Instances of Department Table

The screenshot shows a 'Live SQL' interface with a 'SQL Worksheet' tab. The code entered is:

```
1 insert into department values ('Biology', 'Watson', '90000');
2 insert into department values ('Comp. Sci.', 'Taylor', '100000');
3 insert into department values ('Elec. Eng.', 'Taylor', '85000');
4 insert into department values ('Finance', 'Painter', '120000');
5 insert into department values ('History', 'Painter', '50000');
6 insert into department values ('Music', 'Packard', '80000');
7 insert into department values ('Physics', 'Watson', '70000');
```

Below the code, five messages indicating successful insertions are shown:

```
1 row(s) inserted.  
1 row(s) inserted.  
1 row(s) inserted.  
1 row(s) inserted.  
1 row(s) inserted.
```

3.Create Instructor Table

The screenshot shows a SQL worksheet interface with the following code:

```

1 v CREATE TABLE INSTRUCTOR
2   (ID VARCHAR2(5),
3    NAME VARCHAR2(20) NOT NULL,
4    DEPT_NAME VARCHAR2(20),
5    SALARY NUMERIC(8,2) CHECK (SALARY > 29000),
6    PRIMARY KEY (ID),
7      FOREIGN KEY (DEPT_NAME) REFERENCES DEPARTMENT(DEPT_NAME)
8    ON DELETE SET NULL
9 );

```

Table created.

4. Instances of Instructor Table

The screenshot shows a SQL worksheet interface with the following code:

```

1 insert into instructor values ('10101', 'Srinivasan', 'Comp. Sci.', '65000');
2 insert into instructor values ('12121', 'Wu', 'Finance', '90000');
3 insert into instructor values ('15151', 'Mozart', 'Music', '40000');
4 insert into instructor values ('22222', 'Einstein', 'Physics', '95000');
5 insert into instructor values ('32343', 'El Said', 'History', '60000');
6 insert into instructor values ('33456', 'Gold', 'Physics', '87000');
7 insert into instructor values ('45565', 'Katz', 'Comp. Sci.', '75000');
8 insert into instructor values ('58583', 'Califieri', 'History', '62000');
9 insert into instructor values ('76543', 'Singh', 'Finance', '80000');
10 insert into instructor values ('76766', 'Crick', 'Biology', '72000');
11 insert into instructor values ('83821', 'Brandt', 'Comp. Sci.', '92000');
12 insert into instructor values ('98345', 'Kim', 'Elec. Eng.', '80000');

```

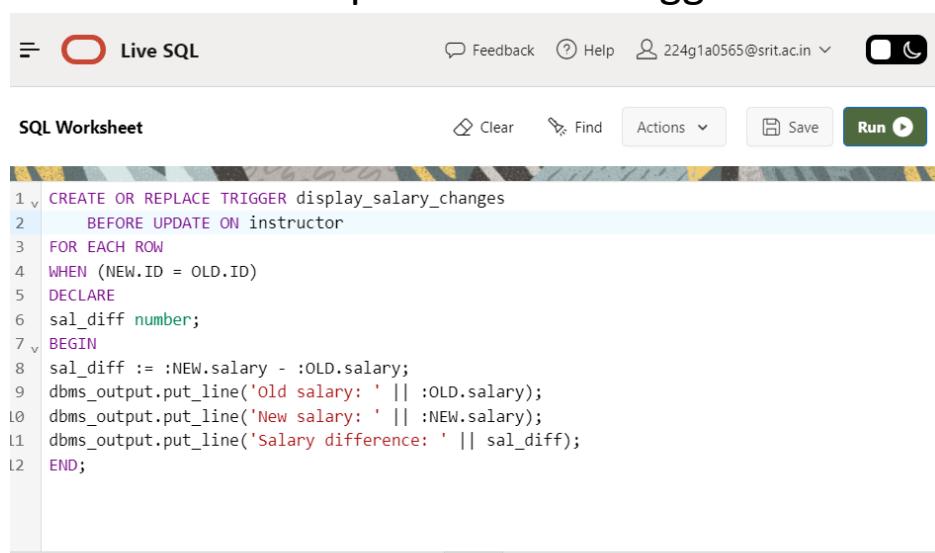
Output:

```

1 row(s) inserted.

```

5.Example to Create Trigger



The screenshot shows the Oracle SQL Developer Live Worksheet interface. The title bar says "Live SQL". The main area is a "SQL Worksheet" containing PL/SQL code to create a trigger. The code is as follows:

```
1 v CREATE OR REPLACE TRIGGER display_salary_changes
2       BEFORE UPDATE ON instructor
3       FOR EACH ROW
4       WHEN (NEW.ID = OLD.ID)
5       DECLARE
6           sal_diff number;
7       BEGIN
8           sal_diff := :NEW.salary - :OLD.salary;
9           dbms_output.put_line('Old salary: ' || :OLD.salary);
10          dbms_output.put_line('New salary: ' || :NEW.salary);
11          dbms_output.put_line('Salary difference: ' || sal_diff);
12      END;
```

Trigger created.

6.PL/SQL Program to Execute a trigger

The image displays two separate sessions of Oracle SQL Developer's "Live SQL" feature. Both sessions show the same PL/SQL code being run against a database.

Session 1 (Top):

```

1 v DECLARE
2 total_rows number(2);
3 v BEGIN
4 UPDATE instructor
5 SET salary = salary + 5000;
6 v IF sql%notfound THEN
7 dbms_output.put_line('no instructors updated');
8 v ELSIF sql%found THEN
9 total_rows := sql%rowcount;
10 dbms_output.put_line( total_rows || ' instructors updated ');
11 END IF;
12 END;

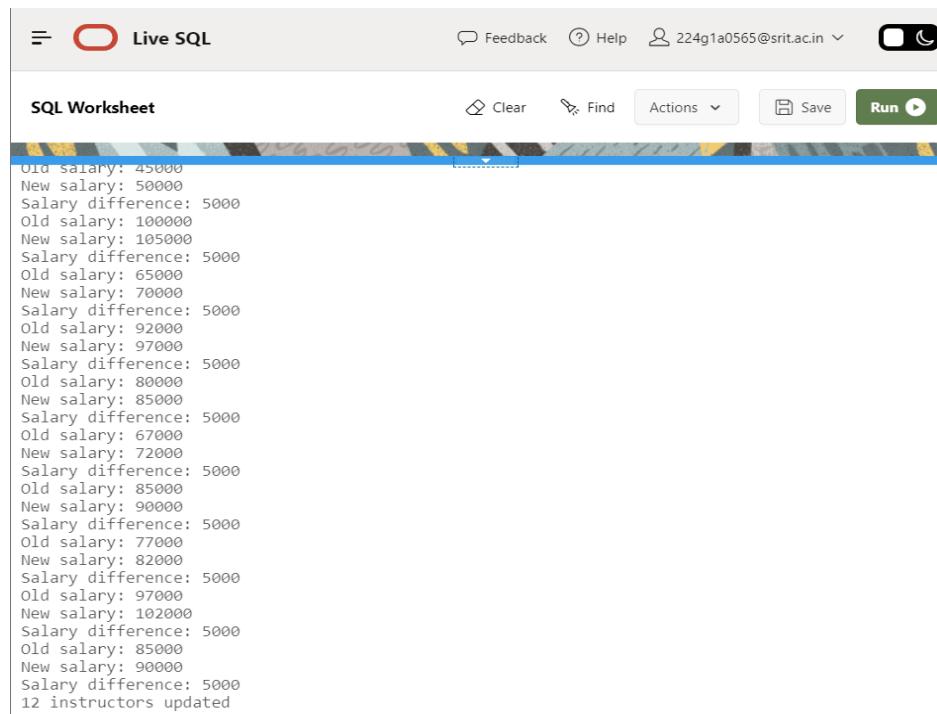
```

Session 2 (Bottom):

```

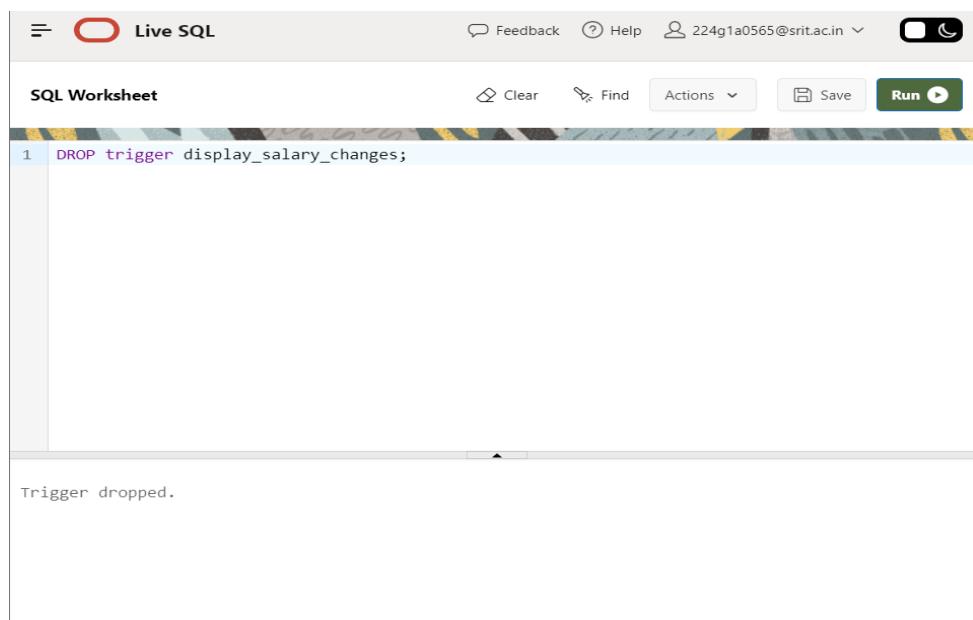
Statement processed.
Old salary: 65000
New salary: 70000
Salary difference: 5000
Old salary: 90000
New salary: 95000
Salary difference: 5000
Old salary: 40000
New salary: 45000
Salary difference: 5000
Old salary: 95000
Salary difference: 5000
Old salary: 100000
New salary: 105000
Salary difference: 5000
Old salary: 65000
New salary: 70000
Salary difference: 5000
Old salary: 92000
New salary: 97000
Salary difference: 5000
Old salary: 80000
New salary: 85000
Salary difference: 5000
Old salary: 67000
New salary: 72000
Salary difference: 5000
Old salary: 85000
New salary: 90000
Salary difference: 5000
Old salary: 77000
New salary: 82000

```



```
Old salary: 45000
New salary: 50000
Salary difference: 5000
Old salary: 100000
New salary: 105000
Salary difference: 5000
Old salary: 65000
New salary: 70000
Salary difference: 5000
Old salary: 92000
New salary: 97000
Salary difference: 5000
Old salary: 80000
New salary: 85000
Salary difference: 5000
Old salary: 67000
New salary: 72000
Salary difference: 5000
old salary: 85000
New salary: 90000
Salary difference: 5000
Old salary: 77000
New salary: 82000
Salary difference: 5000
Old salary: 97000
New salary: 102000
Salary difference: 5000
Old salary: 85000
New salary: 90000
Salary difference: 5000
12 instructors updated
```

7.Drop the Trigger



```
1 DROP trigger display_salary_changes;
```

Trigger dropped.

Conclusion:

In this lab, we successfully Executed PL/SQL program to implement Trigger on Table.

Experiment-16

1.create Table

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, Email (224g1a0565@srit.ac.in), Dark Mode icon.
- Toolbar:** Clear, Find, Actions, Save, Run button.
- SQL Worksheet Area:** Contains the following SQL code:

```
1 v CREATE TABLE customers(
2 ID NUMBER PRIMARY KEY,
3 NAME VARCHAR2(20) NOT NULL,
4 AGE NUMBER,
5 ADDRESS VARCHAR2(20),
6 SALARY NUMERIC(20,2));
```
- Output Area:** Displays the message "Table created."

2.Instances of Customers

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0565@srit.ac.in, Dark mode switch.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run (green button).
- SQL Statements:** Six INSERT INTO statements for the customers table, each inserting a new row with values corresponding to different customers and their details.
- Output:** Confirmation messages indicating 1 row(s) inserted for each of the six statements.

```
1 INSERT INTO customers VALUES(1, 'Ramesh', 23, 'Allabad', 25000);
2 INSERT INTO customers VALUES(2, 'Suresh', 22, 'Kanpur', 27000);
3 INSERT INTO customers VALUES(3, 'Mahesh', 24, 'Ghaziabad', 29000);
4 INSERT INTO customers VALUES(4, 'chandhan', 25, 'Noida', 31000);
5 INSERT INTO customers VALUES(5, 'Alex', 21, 'paris', 33000);
6 INSERT INTO customers VALUES(6, 'Sunita', 20, 'delhi', 35000);
```

1 row(s) inserted.
1 row(s) inserted.
1 row(s) inserted.
1 row(s) inserted.
1 row(s) inserted.

3.Create Update Procedure

The screenshot shows a "Live SQL" interface with a "SQL Worksheet". The code entered is:

```
1 v DECLARE
2   total_rows number(2);
3 v BEGIN
4   UPDATE customers
5   SET salary = salary + 5000;
6 v IF sql%notfound THEN
7   dbms_output.put_line('no customers updated');
8 v ELSIF sql%found THEN
9   total_rows := sql%rowcount;
10 dbms_output.put_line( total_rows || ' customers updated ');
11 END IF;
12 END;
13 /
```

After running the code, the output is displayed below the worksheet:

```
Statement processed.
6 customers updated
```

4.PL/SQL Program using Explicit Cursors

The screenshot shows a SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, Email (224g1a0565@srit.ac.in), Logout.
- Toolbar:** Clear, Find, Actions, Save, Run (green button).
- Worksheet Title:** SQL Worksheet.
- Code Area:** PL/SQL code listing numbered 1 to 15, which declares a cursor to select data from the customers table and prints it to the dbms_output.put_line function.
- Output Area:** Statement processed followed by the output of the printed data rows.

```
1 v DECLARE
2 c_id customers.id%type;
3 c_name customers.name%type;
4 c_addr customers.address%type;
5 v CURSOR c_customers IS
6 SELECT id, name, address FROM customers;
7 v BEGIN
8 OPEN c_customers;
9 LOOP
10 FETCH c_customers INTO c_id, c_name, c_addr;
11 EXIT WHEN c_customers%notfound;
12 dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
13 END LOOP;
14 CLOSE c_customers;
15 END;
```

Statement processed.
1 Ramesh Allabad
2 Suresh Kanpur
3 Mahesh Ghaziabad
4 chandhan Noida
5 Alex paris
6 Sunita delhi

Conclusion:

In this lab, we successfully Executed PL/SQL program to implement cursor on table.