

EXPERIMENT – 1

1.CREATE Table:

The screenshot shows a 'Live SQL' interface with a 'My Session' tab. In the statement list, there is one statement labeled 'Statement 14'. The code for this statement is:

```
CREATE TABLE Student_info
(
    College_Id number(2),
    College_name varchar(30),
    Branch varchar(10)
)
```

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

2.ALTER Table:

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

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

On the right side of the interface, there is a vertical toolbar with various icons for database management tasks.

At the bottom of the interface, the message 'Table altered.' is displayed.

3.DROP Table:

The screenshot shows a SQL interface with a statement history. Statement 17 contains the command `DROP TABLE Student_info`. The output below the statement shows the message `Table dropped.`

```
Statement 17
DROP TABLE Student_info

Table dropped.
```

4. TRUNCATE Table:

The screenshot shows a SQL interface with a session history. Statement 14 creates a table `Student_info` with columns `College_Id`, `College_name`, and `Branch`. Statement 15 alters the table to add a column `CGPA`. Statement 16 truncates the table. Statement 17 drops the table. The output for each statement includes the respective command and a success message: `Table created.`, `Table altered.`, `Table truncated.`, and `Table dropped.`.

```
Statement 14
CREATE TABLE Student_info
(
    College_Id number(2),
    College_name varchar(30),
    Branch varchar(10)
)

Table created.

Statement 15
ALTER TABLE Student_info
ADD CGPA number

Table altered.

Statement 16
TRUNCATE TABLE Student_info

Table truncated.

Statement 17
DROP TABLE Student_info

Table dropped.
```

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 "Live SQL" interface with a "SQL Worksheet" tab. The worksheet contains the following SQL code:

```
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 );
```

Below the worksheet, a message "Table created." is displayed.

2.CREATE TABLE for customers:

The screenshot shows a "Live SQL" interface with a "SQL Worksheet" tab. The worksheet contains the following SQL code:

```
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 );
```

Below the worksheet, a message "Table created." is displayed.

3.CREATE TABLE for products:

The screenshot shows a 'Live SQL' interface with two statements. Statement 25 creates the 'employees' table with columns: employee_id (INT PRIMARY KEY), first_name (VARCHAR(50)), last_name (VARCHAR(50)), salary (DECIMAL(10, 2)), and department_id (INT). Statement 26 creates the 'products' table with columns: product_id (INT PRIMARY KEY), product_name (VARCHAR(100)), category (VARCHAR(50)), and stock_quantity (INT). Both statements result in a 'Table created.' message.

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    salary DECIMAL(10, 2),
    department_id INT
)

CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category VARCHAR(50),
    stock_quantity INT
)
```

4.CREATE TABLE for orders:

The screenshot shows an 'SQL Worksheet' interface with one statement. Statement 1 creates the 'orders' table with columns: order_id (INT PRIMARY KEY), customer_id (INT), and order_date (DATE). A FOREIGN KEY constraint is defined for customer_id, referencing the customers table. The statement ends with a semicolon. A 'Table created.' message is displayed at the bottom.

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

5.CREATE TABLE for logs:

The screenshot shows the 'Live SQL' interface with the following details:

- Toolbar:** Includes 'Feedback', 'Help', a user profile for '224g1a0559@srit.ac.in', and a dark mode switch.
- SQL Worksheet:** Labeled 'SQL Worksheet' with a 'Run' button.
- Code:** The code to create a table named 'logs' is pasted:

```
1 v CREATE TABLE logs (
2   log_id INT PRIMARY KEY,
3   log_message VARCHAR(255)
4 );
```
- Output:** The message 'Table created.' is displayed below the code.

6.INSERT:

The screenshot shows the 'Live SQL' interface with the following details:

- Toolbar:** Includes 'Feedback', 'Help', a user profile for '224g1a0559@srit.ac.in', and a dark mode switch.
- SQL Worksheet:** Labeled 'SQL Worksheet' with a 'Run' button.
- Code:** The code to insert a row into the 'customers' table is pasted:

```
1 v INSERT INTO customers (customer_id, first_name, last_name, email)
2   VALUES (1, 'John', 'Doe', 'john.doe@example.com');
```
- Output:** The message '1 row(s) inserted.' is displayed below the code.

5.SELECT:

The screenshot shows a "Live SQL" interface with a "SQL Worksheet". The query entered is "SELECT * FROM employees;". The output area displays the message "no data found".

```
1 SELECT * FROM employees;
```

no data found

6.UPDATE:

The screenshot shows a "Live SQL" interface with a "SQL Worksheet". The query entered is "UPDATE employees SET salary = salary * 1.1 WHERE department_id = 2;". The output area displays the message "0 row(s) updated.".

```
1 UPDATE employees SET salary = salary * 1.1 WHERE department_id = 2;
```

0 row(s) updated.

7.DELETE:

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

```
1 DELETE FROM customers WHERE customer_id = 1;
```

The output below the query shows:

```
1 row(s) deleted.
```

Conclusion:

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

EXPERIMENT – 3

1.CREATE TABLE:

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

- Header:** Live SQL, Feedback, Help, User: 224g1a0559@srit.ac.in, Dark mode switch.
- Toolbar:** Clear, Find, Actions, Save, Run.
- SQL Worksheet Area:** Contains the following SQL code:

```
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 );
```
- Output Area:** Displays the message "Table created."

2.CREATING A VIEW:

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

- Header:** Live SQL, Feedback, Help, User: 224g1a0559@srit.ac.in, Dark mode switch.
- Toolbar:** Clear, Find, Actions, Save, Run.
- SQL Worksheet Area:** Contains the following SQL code:

```
1 v CREATE VIEW employee_view AS
2 SELECT employee_id, first_name, last_name
3 FROM employees
4 WHERE department_id = 1;
```
- Output Area:** Displays the message "View created."

3.UPDATING A VIEW:

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

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

The output below the code shows the result of the update:

```
0 row(s) updated.
```

4.ALTERING A VIEW:

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

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

The output below the code shows the result of the view creation:

```
View created.
```

5.DELETING A VIEW:

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

- Header:** Live SQL, Feedback, Help, User: 224g1a0559@srit.ac.in, Notifications.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- SQL Statement:** 1. `DROP VIEW employee_view;`
- Output:** View dropped.

Conclusion:

In this 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 'Live SQL' interface with a 'SQL Worksheet' tab. The code entered is:

```
1 v CREATE TABLE table1 (
2   id INT,
3   name VARCHAR(50),
4   PRIMARY KEY (id)
5 );
6
```

The output below the code area says "Table created."

2.inserting VALUES into table1:

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

```
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');
```

The output below the code area shows three separate messages indicating successful insertions:

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

3.creating a table2:

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

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

The output message at the bottom of the worksheet area is "Table created."

4.Inserting VALUES into table2:

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

```
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');
```

The output messages at the bottom of the worksheet area are:

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

5.UNION:

The screenshot shows a live SQL session with the following details:

- Session Information:** My Session, Statement 29.
- Feedback:** 1 row(s) inserted.
- SQL Query:**

```
SELECT * FROM table1
UNION
SELECT * FROM table2
```
- Result Table:**

ID	NAME
1	John
2	Alice
3	Bob
4	Charlie
- Actions:** Download CSV
- Feedback:** 4 rows selected.

6.UNION ALL:

The screenshot shows a live SQL session with the following details:

- Session Information:** My Session, Statement 30.
- Feedback:**
- SQL Query:**

```
SELECT * FROM table1
UNION ALL
SELECT * FROM table2
```
- Result Table:**

ID	NAME
1	John
2	Alice
3	Bob
2	Alice
3	Bob
4	Charlie
- Actions:** Download CSV
- Feedback:** 6 rows selected.

7. INTERSECT:

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

- Toolbar:** Includes "Live SQL" icon, "Feedback", "Help", user info (224g1a0559@srit.ac.in), and a toggle switch.
- SQL Worksheet:** Shows the following SQL code:

```
1 v SELECT * FROM table1
2 INTERSECT
3 SELECT * FROM table2;
```
- Result Table:** Displays two rows of data:

ID	NAME
2	Alice
3	Bob
- Buttons:** "Download CSV" and "2 rows selected."

8. CROSS JOIN:

The screenshot shows a session interface with the following details:

- Toolbar:** Includes "Live SQL" icon, "Feedback", "Help", user info (224g1a0559@srit.ac.in), and a toggle switch.
- Session Name:** "My Session".
- Statement List:** Shows a list of statements with icons for edit, run, and delete.
- SQL Worksheet:** Shows the following SQL code:

```
SELECT * FROM table1
CROSS JOIN table2;
```
- Result Table:** Displays 9 rows of data from the cross join:

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
- Buttons:** "Download CSV" and "9 rows selected."

9.NATURAL JOIN:

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

- Header:** Includes "Live SQL" mode, "Feedback", "Help", user info "224g1a0559@srit.ac.in", and a settings icon.
- Toolbar:** Buttons for "Clear", "Find", "Actions", "Save", and a prominent "Run" button.
- SQL Worksheet:** Displays the query:

```
1 v SELECT * FROM table1
2 NATURAL JOIN table2;
```
- Result Table:** Shows the output of the query:

	name
2	Alice
3	Bob
- Buttons:** "Download CSV" and a note "2 rows selected."

Conclusion:

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

EXPERIMENT – 5

1.creating a table:

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

```
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 );
```

The output below the code shows the confirmation: "Table created."

2.Inserting VALUES into table:

The screenshot shows a 'My Session' interface with a table of statements. Statement 2, 3, and 4 each show an 'INSERT INTO employees' query followed by a success message: "1 row(s) inserted."

Statement	Query / Result
Statement 2	INSERT INTO employees (employee_id, employee_name, salary, department) VALUES (1, 1 row(s) inserted.
Statement 3	INSERT INTO employees (employee_id, employee_name, salary, department) VALUES (2, 1 row(s) inserted.
Statement 4	INSERT INTO employees (employee_id, employee_name, salary, department) VALUES (3, 1 row(s) inserted.

3.IS NULL:

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

```
1 v SELECT * FROM employees
2 WHERE department IS NULL;
```

The output below the code area says "no data found".

4.BETWEEN:

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

```
1 v SELECT * FROM employees
2 WHERE salary BETWEEN 50000 AND 70000;
```

Below the code, a table is displayed with the following data:

EMPLOYEE_ID	EMPLOYEE_NAME	SALARY	DEPARTMENT
1	John	50000	IT
2	Alice	60000	HR
3	Bob	70000	Finance

5.LIKE:

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

```
1 v SELECT * FROM employees
2 WHERE employee_name LIKE 'A%';
```

The results are displayed in a table:

EMPLOYEE_ID	EMPLOYEE_NAME	SALARY	DEPARTMENT
2	Alice	60000	HR

[Download CSV](#)

6.IN:

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

```
1 v SELECT * FROM employees
2 WHERE department IN ('IT', 'HR');
```

The results are displayed in a table:

EMPLOYEE_ID	EMPLOYEE_NAME	SALARY	DEPARTMENT
1	John	50000	IT
2	Alice	60000	HR

[Download CSV](#)

Conclusion:

In this lab, we successfully practiced SQL Queries on SPECIAL OPERATIONS (IS NULL, BETWEEN, LIKE, IN).

EXPERIMENT – 6

1.creating a employee table:

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

```
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 );
```

The output below the code shows the confirmation: "Table created."

2.Inserting VALUES into employee:

The screenshot shows a "Live SQL" interface with a "My Session" tab. It displays four statements:

- Statement 2:** `INSERT INTO employees (employee_id, employee_name, salary, department) VALUES (1,` followed by a progress bar. Confirmation: "1 row(s) inserted."
- Statement 3:** `INSERT INTO employees (employee_id, employee_name, salary, department) VALUES (2,` followed by a progress bar. Confirmation: "1 row(s) inserted."
- Statement 4:** `INSERT INTO employees (employee_id, employee_name, salary, department) VALUES (3,` followed by a progress bar. Confirmation: "1 row(s) inserted."

3.Creating a department table:

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

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

The output below the code shows the confirmation: "Table created."

4.Inserting VALUES into table:

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

```
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');
```

The output below the code shows three rows inserted:

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

5. INNER JOIN (OR EQUI JOIN):

The screenshot shows the Live SQL interface with the following details:

- Toolbar:** Includes icons for Feedback, Help, User (224g1a0559@srit.ac.in), and a toggle switch.
- SQL Worksheet:** Labeled "SQL Worksheet". Contains the following SQL code:


```
1 v SELECT employees.*, departments.department_name
2   FROM employees
3     INNER JOIN departments ON employees.department_id = departments.department_id;
```
- Run Button:** A green button labeled "Run" with a play icon.
- Results:** A table titled "EMPLOYEE_ID", "EMPLOYEE_NAME", "DEPARTMENT_ID", and "DEPARTMENT_NAME". The data is:

EMPLOYEE_ID	EMPLOYEE_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	John	1	IT
2	Alice	2	HR
3	Bob	1	IT
4	Charlie	3	Finance

6. LEFT OUTER JOIN:

The screenshot shows the Live SQL interface with the following details:

- Toolbar:** Includes icons for Feedback, Help, User (224g1a0559@srit.ac.in), and a toggle switch.
- SQL Worksheet:** Labeled "SQL Worksheet". Contains the following SQL code:


```
1 v SELECT employees.*, departments.department_name
2   FROM employees
3     LEFT OUTER JOIN departments ON employees.department_id = departments.department_id;
```
- Run Button:** A green button labeled "Run" with a play icon.
- Results:** A table titled "EMPLOYEE_ID", "EMPLOYEE_NAME", "DEPARTMENT_ID", and "DEPARTMENT_NAME". The data is:

EMPLOYEE_ID	EMPLOYEE_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	John	1	IT
3	Bob	1	IT
2	Alice	2	HR
4	Charlie	3	Finance

7.RIGHT OUTER JOIN:

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

```
1 v SELECT employees.* , departments.department_name
2 FROM employees
3 RIGHT OUTER JOIN departments ON employees.department_id = departments.department_id;
```

The results are displayed in a table:

EMPLOYEE_ID	EMPLOYEE_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	John	1	IT
2	Alice	2	HR
3	Bob	1	IT
4	Charlie	3	Finance

8.FULL OUTER JOIN:

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

```
1 v SELECT employees.* , departments.department_name
2 FROM employees
3 FULL OUTER JOIN departments ON employees.department_id = departments.department_id;
```

The results are displayed in a table:

EMPLOYEE_ID	EMPLOYEE_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	John	1	IT
2	Alice	2	HR
3	Bob	1	IT
4	Charlie	3	Finance

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 "Live SQL" interface with a "SQL Worksheet" tab selected. The worksheet 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 );
```

The "Run" button at the top right is highlighted in green.

Table created.

2.Create INSTRUCTOR Table:

The screenshot shows a "Live SQL" interface with a "SQL Worksheet" tab selected. The worksheet 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 );
10
```

The "Run" button at the top right is highlighted in green.

Table created.

3. Instances of instructor:

The screenshot shows a SQL worksheet interface with the title "Live SQL". The "SQL Worksheet" tab is selected. The code area contains the following SQL statements:

```
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');
```

The output window below the code area displays the results of each insert statement, showing "1 row(s) inserted." for each of the 12 rows.

4. Create COURSE Table:

The screenshot shows a SQL worksheet interface with the title "Live SQL". The "SQL Worksheet" tab is selected. The code area contains the following SQL statement:

```
1 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 );
```

The output window below the code area displays the message "Table created.".

Table created.

5. Instances of COURSE Table:

The screenshot shows a "Live SQL" interface with a "SQL Worksheet" tab. The worksheet contains the following SQL code:

```
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');
```

The output below the code shows the results of the insertions:

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

To find no. of courses taught in the university:

The screenshot shows a "Live SQL" interface with a "SQL Worksheet" tab. The worksheet contains the following SQL code:

```
1 select count (*)
2 from course;
```

The output below the code shows the result of the query:

COUNT(*)
13

[Download CSV](#)

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

- Header:** Live SQL, Feedback, Help, Email (224g1a0559@srit.ac.in), Dark Mode toggle.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- Code Area:** Two queries are present:

```
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:** A table showing the names of instructors with non-null salaries:

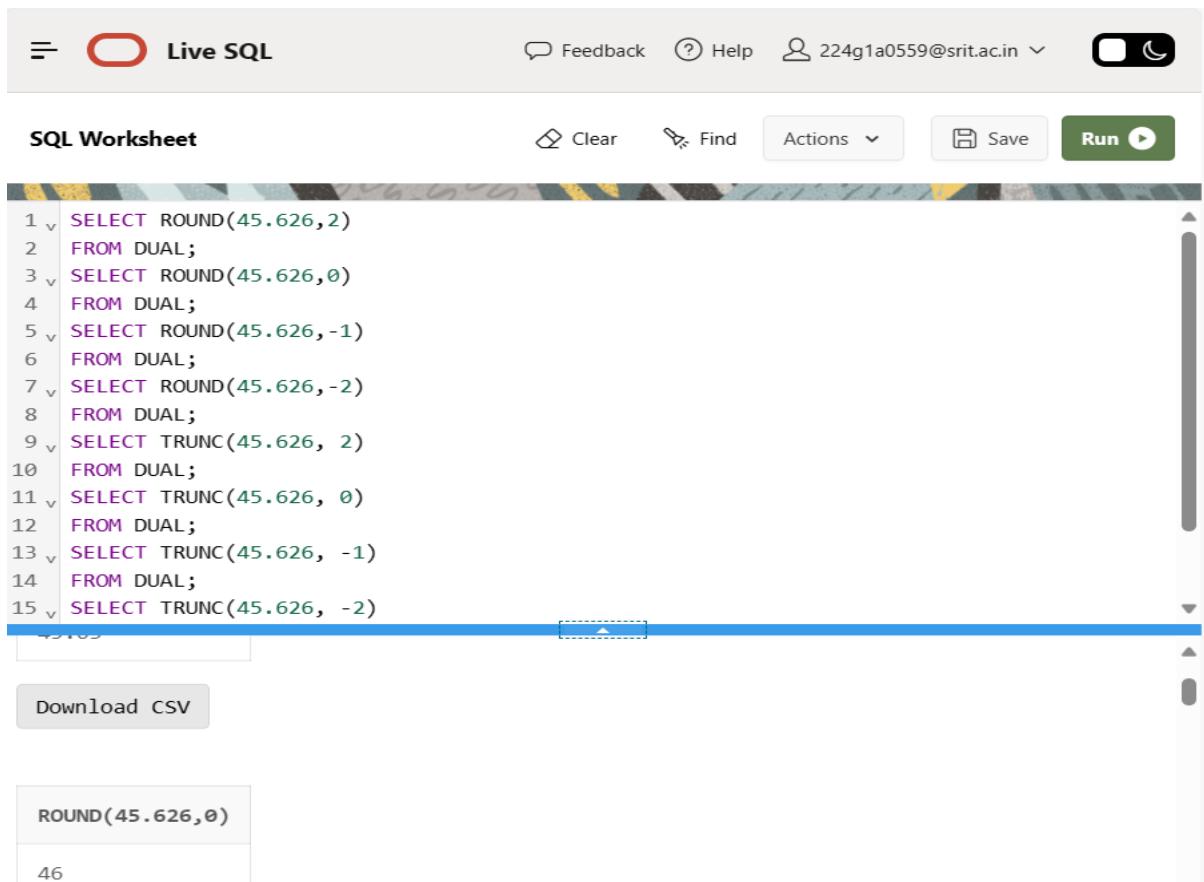
Srinivasan
Wu
Mozart
Einstein
El Said

Conclusion:

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

EXPERIMENT – 8

1. NUMBER Functions:



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

```

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 v SELECT TRUNC(45.626, 2)
10  FROM DUAL;
11 v SELECT TRUNC(45.626, 0)
12  FROM DUAL;
13 v SELECT TRUNC(45.626, -1)
14  FROM DUAL;
15 v SELECT TRUNC(45.626, -2)

```

ROUND(45.626,0)

46

Download CSV

ROUND(45.626,-1)

50

Download CSV

ROUND(45.626,-2)

0

Download CSV

Live SQL

Feedback Help 224g1a0559@srit.ac.in

SQL Worksheet Actions Save Run

ROUND(45.626,0)
46

Download CSV

ROUND(45.626,-1)
50

Download CSV

ROUND(45.626,-2)
0

Download CSV

Live SQL

Feedback Help 224g1a0559@srit.ac.in

SQL Worksheet 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 the Oracle Live SQL interface with the following details:

- Header:** Live SQL, Feedback, Help, Email (224g1a0559@srit.ac.in), Dark Mode icon.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- SQL Worksheet Content:**

```
1 v SELECT SYSDATE
2   FROM DUAL;
3 v SELECT MONTHS_BETWEEN(SYSDATE, '15-FEB-20')
4   FROM DUAL;
5 v SELECT ADD_MONTHS(SYSDATE, 2)
6   FROM DUAL;
7 v SELECT NEXT_DAY(SYSDATE, 'THURSDAY')
8   FROM DUAL;
9 v SELECT LAST_DAY(SYSDATE)
10  FROM DUAL;
11 v SELECT ROUND('25-JUL-03', 'MONTH')
12  FROM DUAL;
13 v SELECT ROUND('25-JUL-03', 'YEAR')
14  FROM DUAL;
15 v SELECT TRUNC('25-JUL-03', 'YEAR')
```
- Output Window:** Shows the result of the SYSDATE query:

SYSDATE
20-DEC-23

Download CSV button is present.
- Second Worksheet:** Shows the result of the MONTHS_BETWEEN query:

MONTHS_BETWEEN(SYSDATE, '15-FEB-20')
46.18043085424133811230585424133811230585

Download CSV button is present.
- Third Worksheet:** Shows the result of the ADD_MONTHS query:

ADD_MONTHS(SYSDATE, 2)
20-FEB-24

Download CSV button is present.

Conclusion:

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

EXPERIMENT – 9

1.NOT NULL CONSTRAINT Example:

The screenshot shows a SQL worksheet interface with the title "Live SQL". The code entered is:

```
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;
```

The output below the code shows two messages:

Table created.
Table altered.

2.UNIQUE CONSTRAINT Example:

The screenshot shows a SQL worksheet interface with the title "Live SQL". The code entered is:

```
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 );
```

The output below the code shows one message:

Table created.

The image shows two separate sessions of a SQL worksheet in a "Live SQL" interface. Both sessions have the same header: "Live SQL" with a red circular icon, "Feedback", "Help", and a user account for "224g1a0559@srit.ac.in". The first session (top) contains the following SQL code:

```
1 v ALTER TABLE students  
2 ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);  
3  
4
```

The output below the code is "Table altered." The second session (bottom) contains the following SQL code:

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

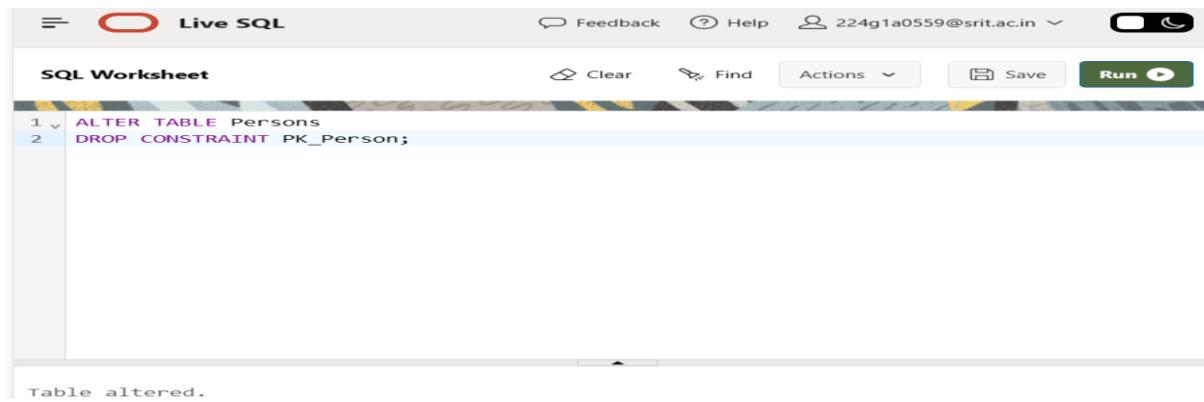
The output below the code is "Table altered."

3.PRIMARY KEY CONSTRAINT Example:

The image shows a single session of a SQL worksheet in a "Live SQL" interface. The header is identical to the previous screenshots. The SQL code in the worksheet is:

```
1 v 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 );
```

The output below the code is "Table created."



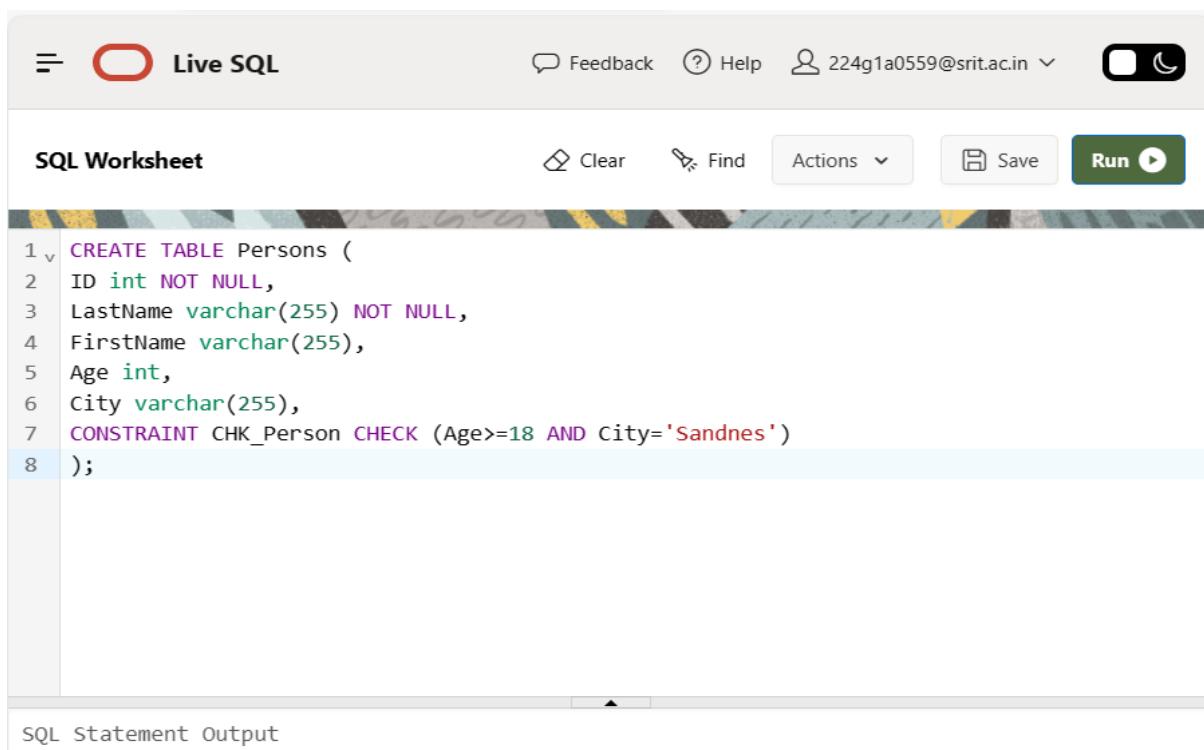
The screenshot shows the 'Live SQL' interface with the 'SQL Worksheet' tab selected. The code entered is:

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

The output below the code shows the result of the execution:

```
Table altered.
```

4. DEFAULT CONSTRAINTS Example:



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

```
1 v CREATE TABLE Persons (
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 );
```

The output below the code shows the result of the execution:

```
SQL Statement Output
```

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 elements:

- Header: Live SQL, Feedback, Help, Email (224g1a0559@srit.ac.in), and a dark mode switch.
- Toolbar: SQL Worksheet, Clear, Find, Actions (dropdown), Save, and Run (green button).
- Code Area:

```
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

Conclusion:

In this lab, We successfully practiced a program for calculating the factorial of a given number.

Experiment- 11

1. Prime number or Not Program:

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

- Header: Live SQL, Feedback, Help, User: 224g1a0559@srit.ac.in, Dark Mode toggle.
- Toolbar: SQL Worksheet, Clear, Find, Actions, Save, Run.
- Code Area:

```
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:

Statement processed.
13 is a prime number

Conclusion:

In this lab, we Successfully practiced a program for Finding the given Number is Prime number or not.

Experiment- 12

1. Fibonacci Series Program:

The screenshot shows a SQL Worksheet interface in Oracle SQL Developer. The top navigation bar includes 'Live SQL' (highlighted in red), 'Feedback', 'Help', a user profile for '224g1a0559@srit.ac.in', and a dark mode toggle. Below the bar, the worksheet title is 'SQL Worksheet' with buttons for 'Clear', 'Find', 'Actions', 'Save', and a large green 'Run' button with a play icon. The code area contains a PL/SQL block to generate a Fibonacci series up to 5. The output pane shows the processed statement and the resulting series values.

```
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;
```

Statement processed.
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 "Live SQL" interface with the following details:

- Toolbar:** Feedback, Help, User (224g1a0559@srit.ac.in), Dark Mode switch.
- Section:** SQL Worksheet
- Buttons:** Clear, Find, Actions (dropdown), Save, Run (green button).
- 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;
```
- Output:** Table created.
Procedure created.

2.Execution Procedure:

The screenshot shows a "Live SQL" interface with the following details:

- Toolbar:** Feedback, Help, User (224g1a0559@srit.ac.in), Dark Mode switch.
- Section:** SQL Worksheet
- Buttons:** Clear, Find, Actions (dropdown), Save, Run (green button).
- 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:** Statement processed.
RECORD INSERTED SUCCESSFULLY
1 RECORD IS INSERTED SUCCESSFULLY

3.DROP Procedure:

The screenshot shows a "Live SQL" interface with a "SQL Worksheet". The worksheet contains the following code:

```
1 DROP PROCEDURE insertuser;
```

Below the worksheet, a message indicates the procedure was dropped:

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 the Oracle Live SQL interface. The top navigation bar includes 'Live SQL', 'Feedback', 'Help', and a user account. Below the navigation is a toolbar with 'Clear', 'Find', 'Actions', 'Save', and a 'Run' button. The main area is titled 'SQL Worksheet'. The code entered is:

```
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;
```

After running the code, a message 'Function created.' is displayed.

2.Execution Procedure:

The screenshot shows the Oracle Live SQL interface. The top navigation bar includes 'Live SQL', 'Feedback', 'Help', and a user account. Below the navigation is a toolbar with 'Clear', 'Find', 'Actions', 'Save', and a 'Run' button. The main area is titled 'SQL Worksheet'. The code entered is:

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

After running the code, the output 'ADDITION IS: 33' is displayed.

Statement processed.
ADDITION IS: 33

3.DROP Function:

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

- Header:** Live SQL, Feedback, Help, User: 224g1a0559@srit.ac.in, a toggle switch.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- Code Area:** The code is as follows:

```
1 DROP FUNCTION Adder;
```
- Output Area:** The message "Function dropped." is displayed.

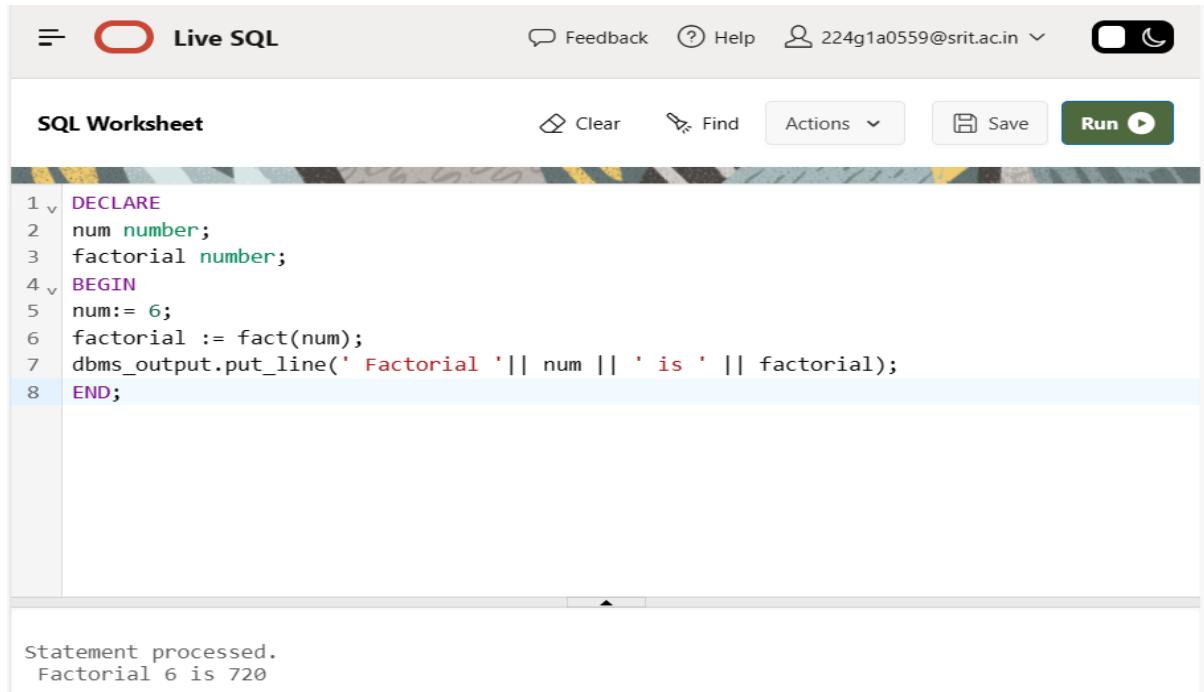
4.Recursive Function:

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

- Header:** Live SQL, Feedback, Help, User: 224g1a0559@srit.ac.in, a toggle switch.
- Toolbar:** SQL Worksheet, Clear, Find, Actions, Save, Run.
- Code Area:** The code is as follows:

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

5.Execution Procedure:



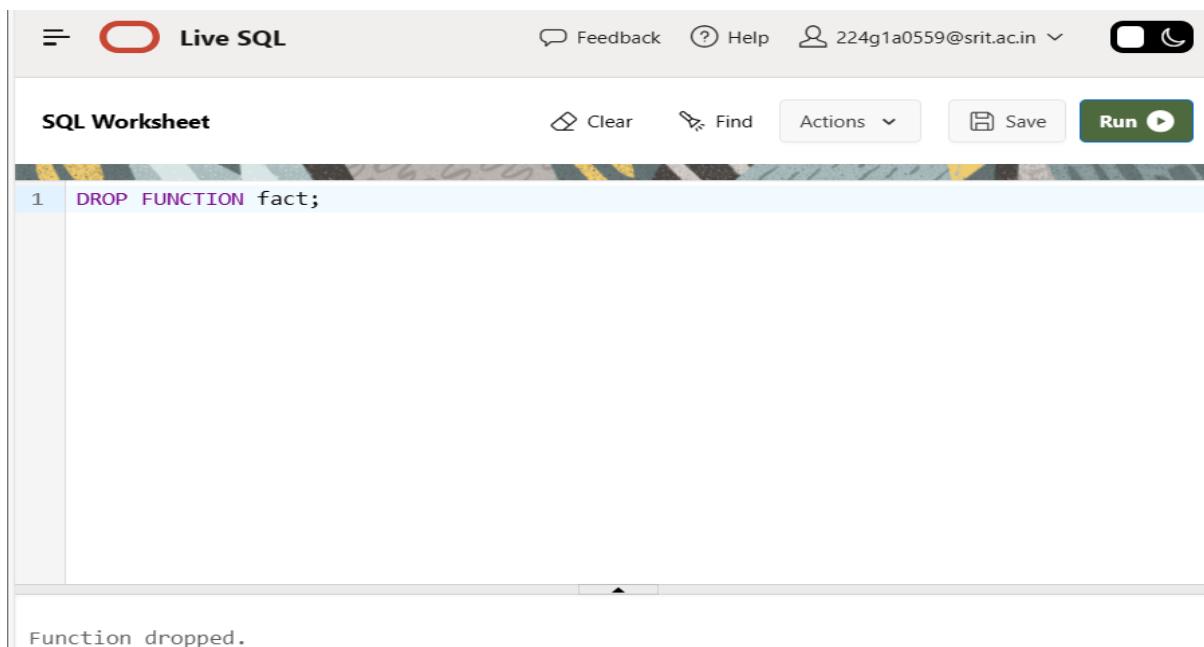
The screenshot shows a SQL Worksheet interface. The code entered is:

```
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;
```

The output below the code shows the result of running the statement:

```
Statement processed.
Factorial 6 is 720
```

6.DROP Function:



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

```
1 DROP FUNCTION fact;
```

The output below the code shows the result of running the statement:

```
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 SQL worksheet interface with the following details:

- Header:** Live SQL, Feedback, Help, User: 224g1a0559@srit.ac.in, Dark mode switch.
- Toolbar:** Clear, Find, Actions, Save, Run.
- SQL Worksheet:** 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:** Shows the message "Table created."

2.Instances of Department Table

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

- Header:** Live SQL, Feedback, Help, User: 224g1a0559@srit.ac.in, Dark mode switch.
- Toolbar:** Clear, Find, Actions, Save, Run.
- SQL Worksheet:** Contains the following SQL code:

```
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');
```
- Output:** Shows the message "1 row(s) inserted." repeated seven times, corresponding to the seven insert statements.

3.Create Instructor Table

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

```
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 );
```

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

4.Instances of Instructor Table

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

```
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');
```

Below the code, the message "1 row(s) inserted." is repeated six times, indicating successful insertions.

5.Example to Create Trigger

The screenshot shows a "Live SQL" interface with a "SQL Worksheet" tab. The worksheet contains the following PL/SQL code:

```
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 v 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;
```

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

6.PL/SQL Program to Execute a trigger

The screenshot shows a "Live SQL" interface with a "SQL Worksheet" tab. The worksheet contains the following PL/SQL code:

```
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;
```

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

- Header:** Live SQL, Feedback, Help, Email (224g1a0559@srit.ac.in), Dark Mode switch.
- Toolbar:** Clear, Find, Actions, Save, Run.
- Output Area:** Displays the results of a query or script execution, showing salary updates for 12 instructors. The output includes:
 - 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
 - New salary: 100000
 - Salary difference: 5000
 - Old salary: 60000
 - New salary: 65000
 - Salary difference: 5000
 - Old salary: 87000
 - New salary: 92000
 - Salary difference: 5000
 - Old salary: 75000
 - New salary: 80000
 - Salary difference: 5000
 - Old salary: 62000
 - New salary: 67000
 - Salary difference: 5000
 - Old salary: 80000
 - New salary: 85000
 - Salary difference: 5000
 - Old salary: 72000
 - New salary: 77000

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

- Header:** Live SQL, Feedback, Help, Email (224g1a0559@srit.ac.in), Dark Mode switch.
- Toolbar:** Clear, Find, Actions, Save, Run.
- Output Area:** Displays the results of a query or script execution, showing salary updates for 12 instructors. The output includes:
 - New salary: 45000
 - Salary difference: 5000
 - Old salary: 95000
 - New salary: 100000
 - Salary difference: 5000
 - Old salary: 60000
 - New salary: 65000
 - Salary difference: 5000
 - Old salary: 87000
 - New salary: 92000
 - Salary difference: 5000
 - Old salary: 75000
 - New salary: 80000
 - Salary difference: 5000
 - Old salary: 62000
 - New salary: 67000
 - Salary difference: 5000
 - Old salary: 80000
 - New salary: 85000
 - Salary difference: 5000
 - Old salary: 72000
 - New salary: 77000
 - Salary difference: 5000
 - Old salary: 92000
 - New salary: 97000
 - Salary difference: 5000
 - Old salary: 80000
 - New salary: 85000
 - Salary difference: 5000
 - 12 instructors updated

7.Drop the Trigger

The screenshot shows the Oracle SQL Developer interface. At the top, there's a toolbar with a refresh button, feedback, help, user information (224g1a0559@srit.ac.in), and a dark mode switch. Below the toolbar is a menu bar with 'SQL Worksheet' selected. On the right side of the menu bar are buttons for 'Clear', 'Find', 'Actions', 'Save', and a prominent green 'Run' button with a play icon. The main workspace contains a single line of SQL code: '1 DROP trigger display_salary_changes;'. Below the code, the output window displays the message '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 the Oracle Live SQL interface. At the top, there are navigation links: 'My Session \ Previous Sessions \ Previous Session' (highlighted in red), 'Delete Session', 'Re-Run', and 'Save this Session'. Below this is a toolbar with icons for copy, paste, and other functions. The main area displays a statement log. Statement 36 shows the creation of a 'customers' table with columns: ID (NUMBER PRIMARY KEY), NAME (VARCHAR2(20) NOT NULL), AGE (NUMBER), ADDRESS (VARCHAR2(20)), and SALARY (NUMERIC(20,2)). The output shows 'Table created.'.

```
CREATE TABLE customers(
ID NUMBER PRIMARY KEY,
NAME VARCHAR2(20) NOT NULL,
AGE NUMBER,
ADDRESS VARCHAR2(20),
SALARY NUMERIC(20,2))
```

Table created.

2.Instances of Customers

The screenshot shows the Oracle Live SQL interface. At the top, there are navigation links: 'My Session \ Previous Sessions \ Previous Session' (highlighted in red), 'Delete Session', 'Re-Run', and 'Save this Session'. Below this is a toolbar with icons for copy, paste, and other functions. The main area displays a statement log. Statements 37 through 41 show the insertion of five rows into the 'customers' table. Each statement includes the SQL query and the message '1 row(s) inserted.'.

```
INSERT INTO customers VALUES(1, 'Ramesh', 23, 'Allabad', 25000)
1 row(s) inserted.

INSERT INTO customers VALUES(2, 'Suresh', 22, 'Kanpur', 27000)
1 row(s) inserted.

INSERT INTO customers VALUES(3, 'Mahesh', 24, 'Ghaziabad', 29000)
1 row(s) inserted.

INSERT INTO customers VALUES(4, 'chandhan', 25, 'Noida', 31000)
1 row(s) inserted.

INSERT INTO customers VALUES(5, 'Alex', 21, 'paris', 33000)
```

The screenshot shows the Live SQL interface with the following session history:

- Statement 41: `INSERT INTO customers VALUES(5, 'Alex', 21, 'paris',33000)`
Result: 1 row(s) inserted.
- Statement 42: `INSERT INTO customers VALUES(6, 'Sunita',20,'delhi',35000)`
Result: 1 row(s) inserted.

3.Create Update Procedure

The screenshot shows the Live SQL interface with the following session history:

- Statement 42: `INSERT INTO customers VALUES(6, 'Sunita',20,'delhi',35000)`
Result: 1 row(s) inserted.
- Statement 43: A PL/SQL block to update the customers table by 5000 salary.

```
DECLARE
total_rows number(2);
BEGIN
UPDATE customers
SET salary = salary + 5000;
IF sql%notfound THEN
dbms_output.put_line('no customers updated');
ELSIF sql%found THEN
total_rows := sql%rowcount;
dbms_output.put_line( total_rows || ' customers updated ');
END IF;
END;
```


Result:
`Statement processed.
6 customers updated`

4.Display Created Customers Table

The screenshot shows a "Live SQL" interface with a session titled "My Session". The query "SELECT * FROM CUSTOMERS" is run, resulting in a table output:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allabad	30000
2	Suresh	22	Kanpur	32000
3	Mahesh	24	Ghaziabad	34000
4	chandhan	25	Noida	36000
5	Alex	21	paris	38000
6	Sunita	20	delhi	40000

6 rows selected.

5.PL/SQL Program using Explicit Cursors

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

```

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 v 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;
16 /

```

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 Practiced PL/SQL Program to implement Cursor On table.