EXPERIMENT-1

Aim: Write SQL queries to create for various databases using DDL commands (i.e. CREATE, ALTER, DROP, TRUNCATE).

**CREATE TABLE:**

Creates a table with specified constraints.

**Syntax:**

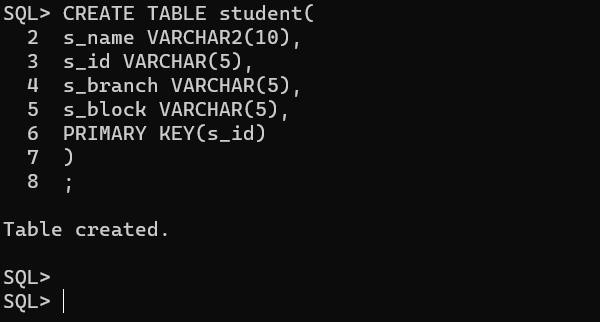
CREATE TABLE tablename (

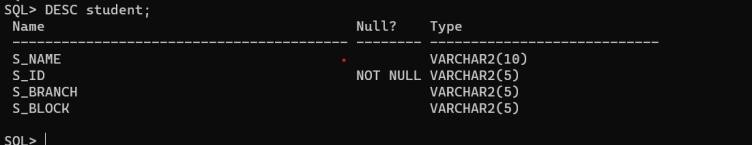
Column1 data\_type[constraint],

Column2 data\_type[constraint],

PRIMARY KEY (column1[, colunmn2]),

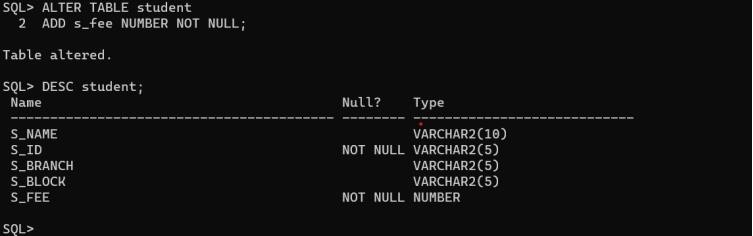
FOREIGN KEY (column1[, column2]) REFERENCES tablename] [, CONSTRAINT constraint]);





**ALTER TABLE:**

Used to add or modify table details like column names and data types, column constraints.

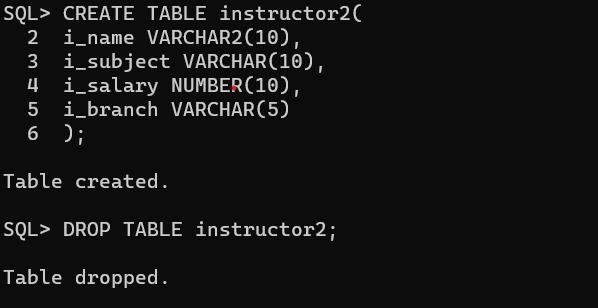


**DROP TABLE:**

Deletes the specified table.

**Syntax:**

DROP TABLE table\_name;

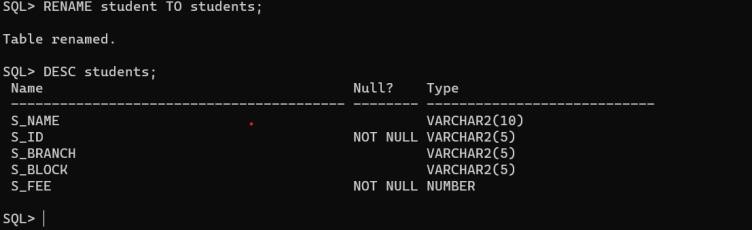


**RENAME TABLE:**

To rename table\_name, column\_name.

**Syntax:**

RENAME new\_table\_name TO old\_table\_name;

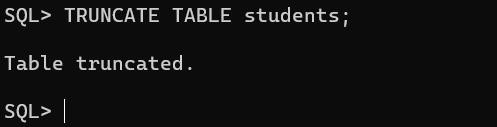


**TRUNCATE TABLE:**

To remove all rows in a specified table.

**Syntax:**

TRUNCATE TABLE table\_name;



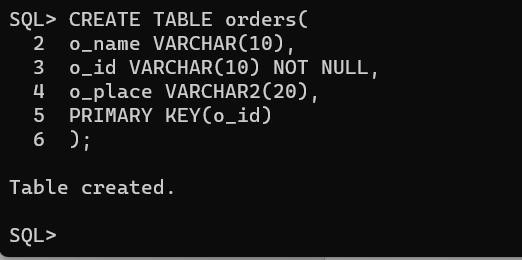
**Conclusion:**

In this lab, we have practiced CREATE, ALTER, DROP, and TRUNCATE commands for the user created table.

# EXPERIMENT-2

**AIM:** To write SQL queries to MANIPULATE TABLES for c=various databases using DML commands (i.e. INSERT, SELECT, UPDATE, DELETE).

**CREATING TABLE:**



**INSERT COMMAND:**

It is used to add values to a table.

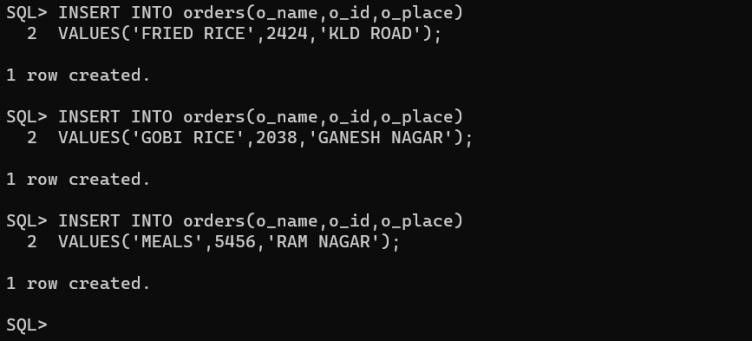
**Syntax:**

INSERT INTO table\_name

VALUES (value1, value2, …., valueN);

INSERT INTO table\_name (colunm1, colunm2, …., colunmN)

VALUES (value1, value2, ….., valueN);



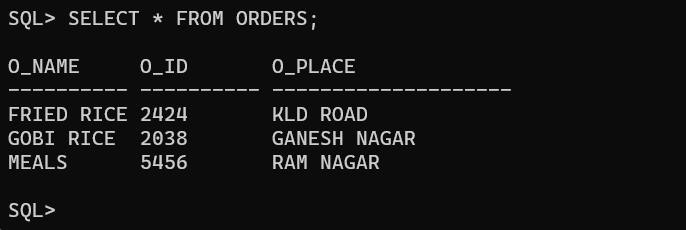
**SELECT COMMAND:**

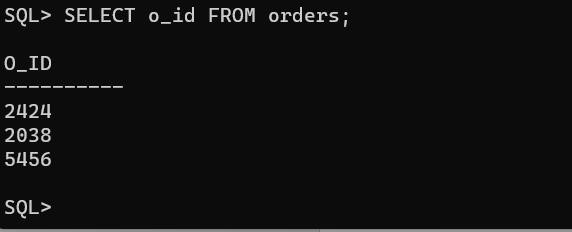
The SELECT command is used to list the contents of a table.

**Syntax:**

SELECT \* FROM table\_name:

SELECT column\_name FROM table\_name;





**UPDATE COMMAND:**

The UPDATE command is used to modify the contents of specified table.

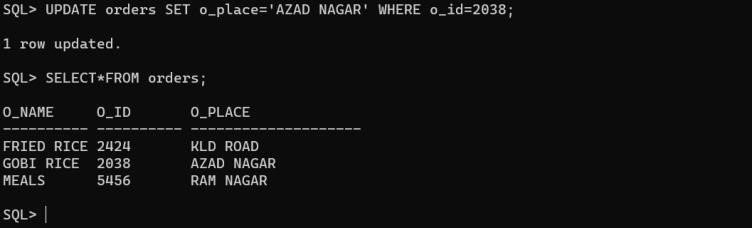
**Syntax:**

UPDATE table\_name

SET column\_name = value [,

Column\_name = value]

[WHERE condition\_list];

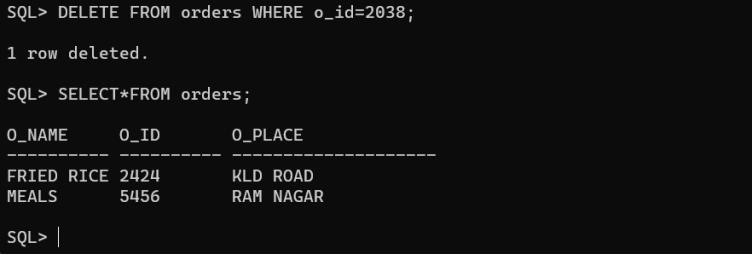


**DELETE COMMAND:**

To delete all rows or specified rows in a table.

**Syntax:**

DELETE FROM table\_name [ WHERE condition\_list];



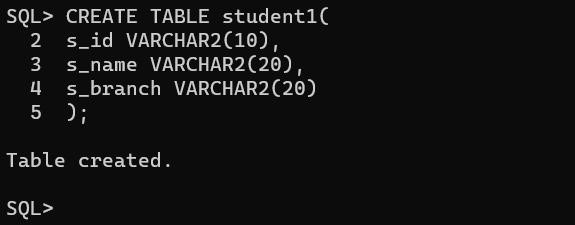
**CONCLUSION:**

In this lab, we have practiced INSERT, SELECT, UPDATE, and DELETE commands for user created table.

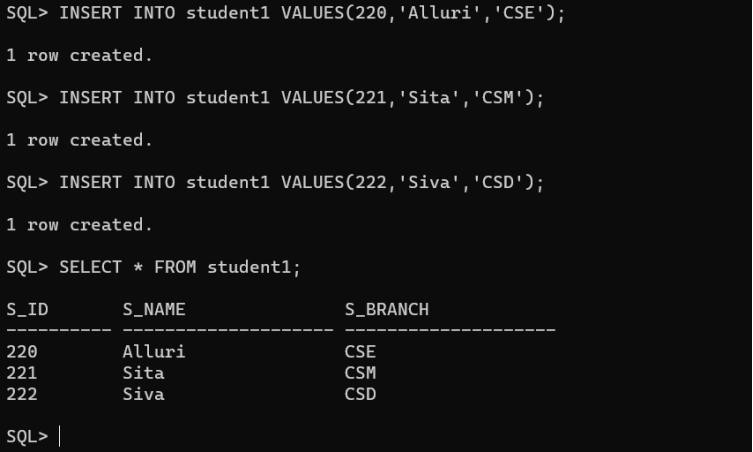
# EXPERIMENT-3

**AIM:** To implement view high level design for various views to CREATE VIEW, ALTER VIEW and DELETE VIEW using DDL commands.

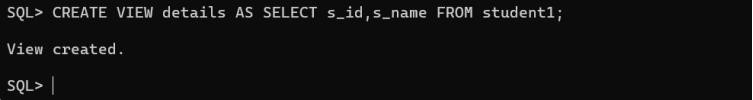
**CREATING TABLE:**



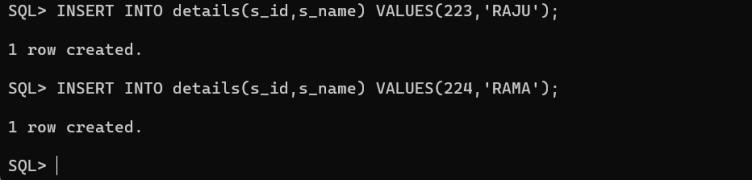
**INSERTING VALUES INTO TABLE student1:**

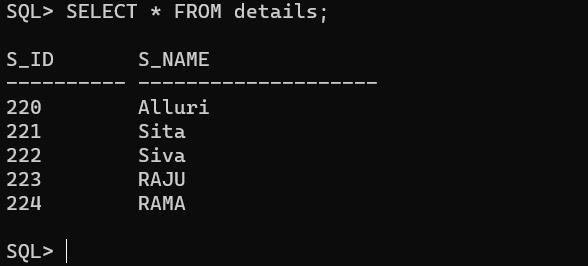


**CREATE VIEW:** Create a view details with attributes s\_id and s\_name.

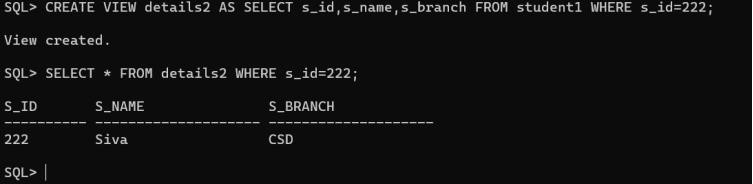


**INSERTING VALUES INTO details view:**

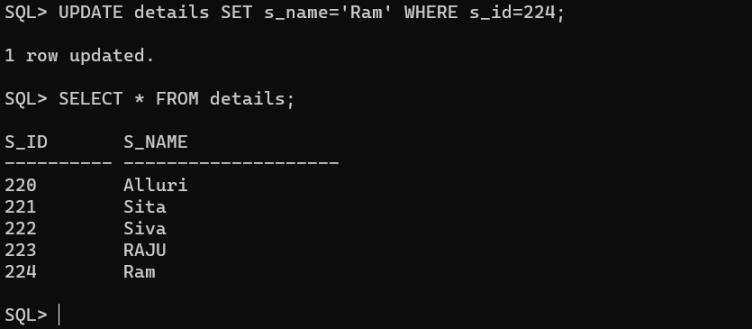




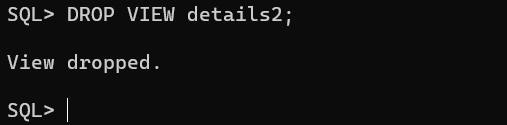
**ALTER VIEW:** Add an attribute branch to the view.



**UPDATE VIEW:** Update the existing name with the new name using UPDATE command.



**DROP VIEW:** Drop a view using DROP command.



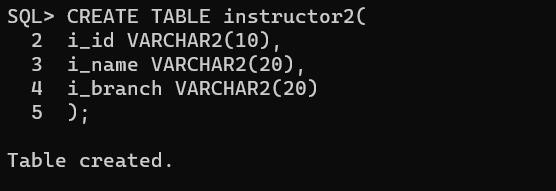
**Conclusion:**

In this lab, we have practiced how to CREATE VIEW, ALTER VIEW, UPDATE VIEW, and DELETE VIEW for user created table.

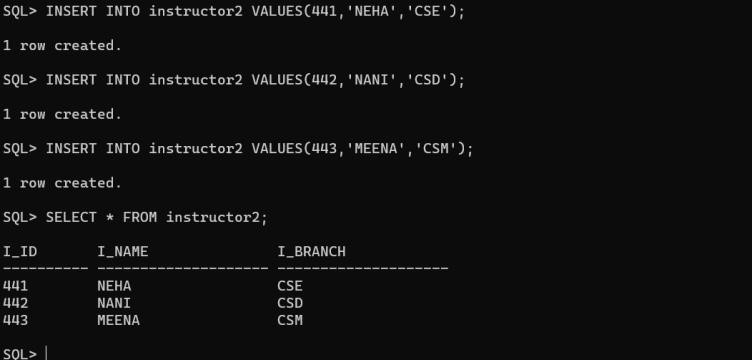
# EXPERIMENT-4

**AIM:** To implement SQL queries for set operations like UNION, UNION ALL, INTERSECT, INTERSECT ALL, MINUS, CROSS JOIN, NATURAL JOIN.

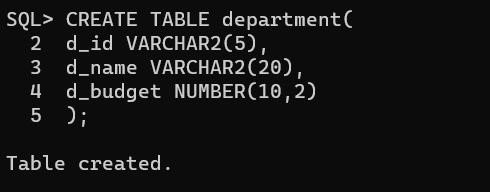
**CREATING A TABLE instructor2:**



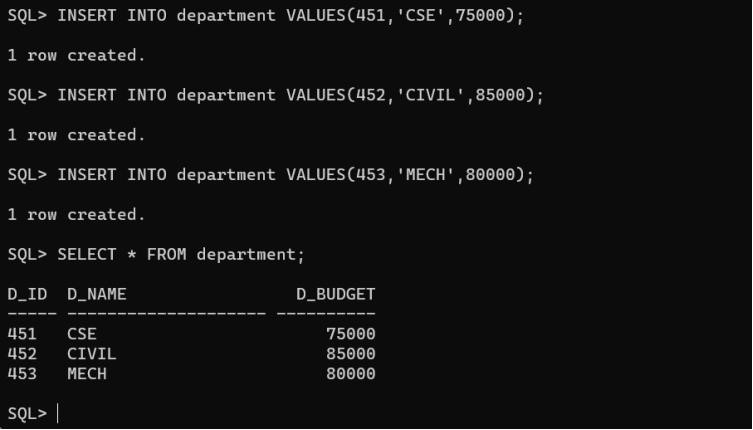
**INSERTING VALUES INTO TABLE instructor2:**



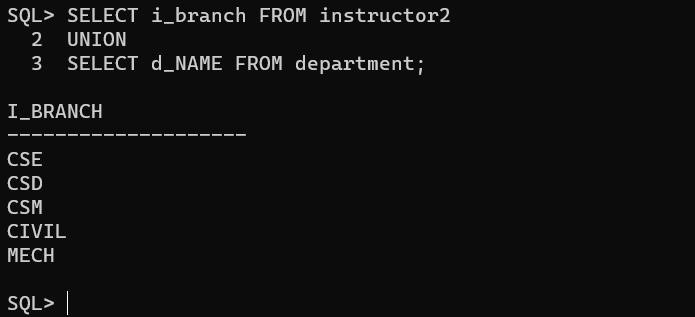
**CREATING TABLE department:**



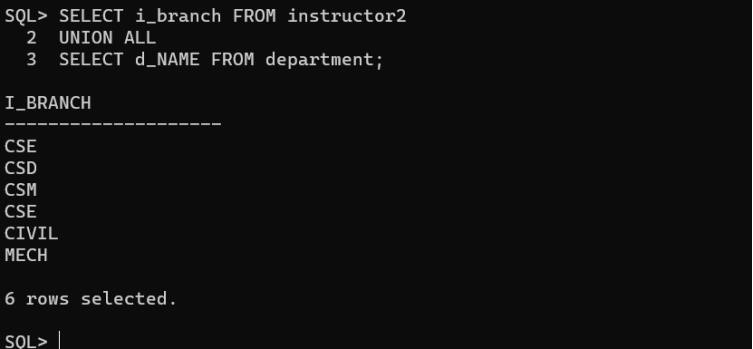
**INSERTING VALUES INTO department:**



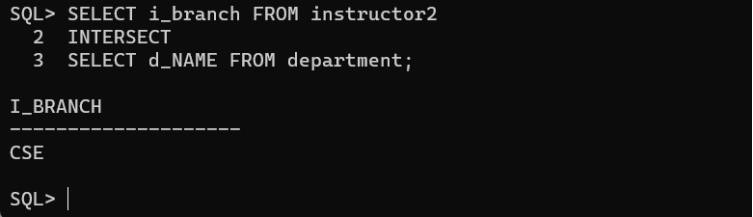
**UNION:** The attributes i\_branch from instructor2 and d\_name from department are joined using UNION command.



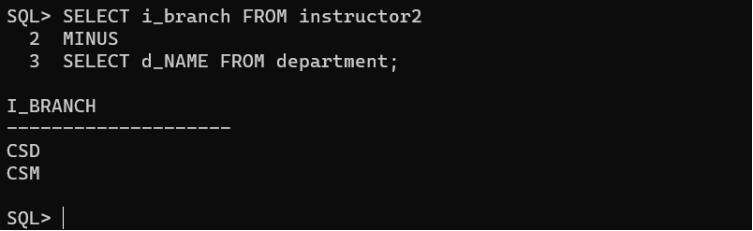
**UNION ALL:** The attributes i\_branch from instructor2 and d\_name from department are joined along with duplicates using UNION ALL command.



**INTERSECT:** Displays similar values in two or more attributes from department and instructor4 using INTERSECT command.



**MINUS:** It eliminates the same values of second column from the first column and represents the remaining values using command MINUS.



**CROSS JOIN:** Its cross products the all the attributes using CROSS JOIN command.



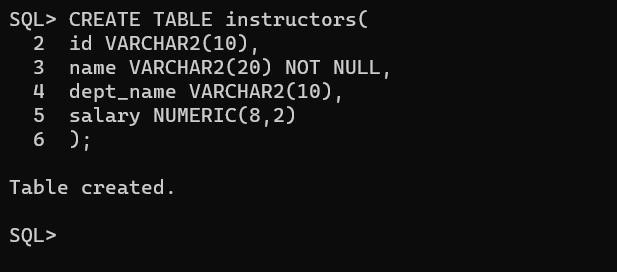
**Conclusion:**

In this lab, we have practiced the set operations like UNION, UNION ALL, INTERSECT, MINUS, CROSS JOIN on user created tables.

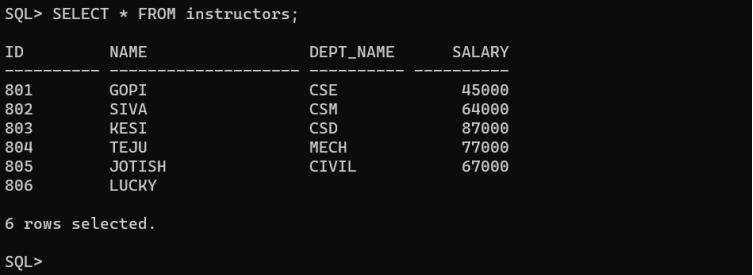
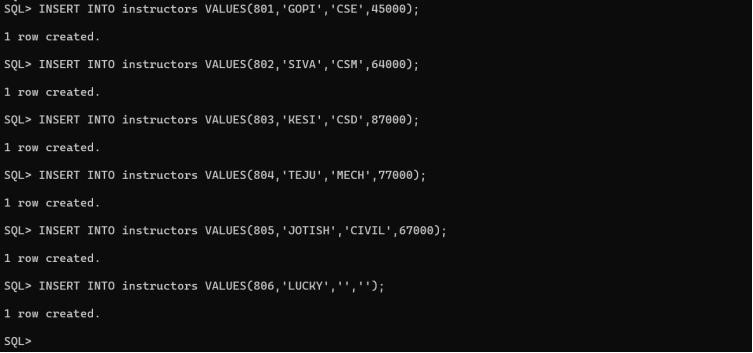
# EXPERIMENT-5

**AIM:**SQL queries to perform SPECIAL OPERATIONS (IS NULL, BETWEEN, LIKE, IN, EXISTS).

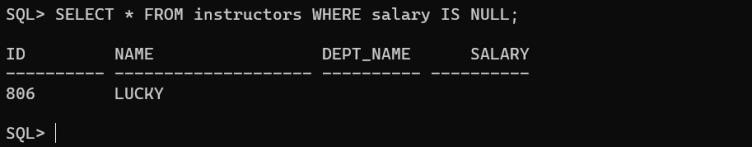
**CREATING TABLE-1:**



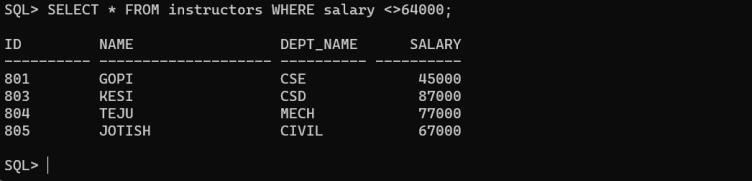
**INSERTING VALUES INTO THE TABLE:**



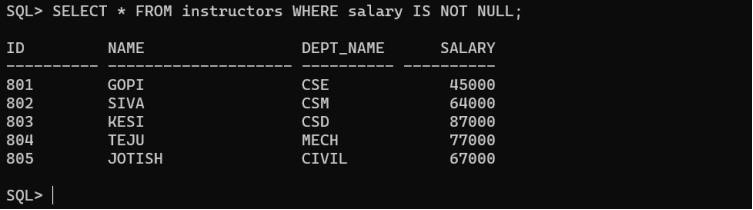
**IS NULL:** It is used to check null values and display null attributes. It displays attributes that have null values.



This command displays the salary that are not equal to 64000.

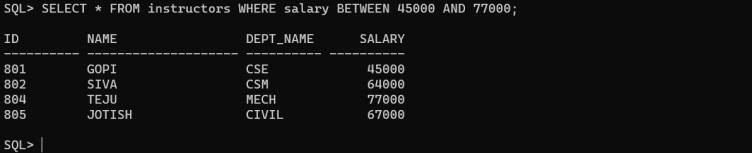


**IS NOT NULL:** It displays attributes that don’t have null values.

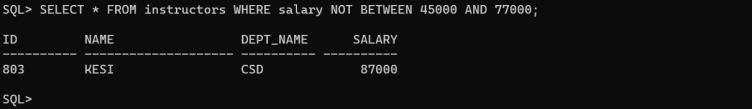


**BETWEEN:** This is used to check range of values.

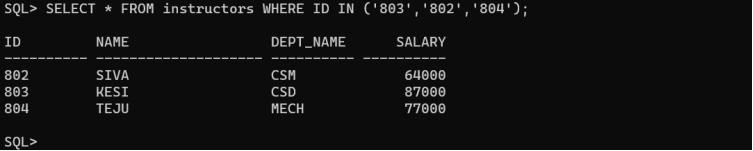
By following command, it displays all the attributes between 45000 and 77000.



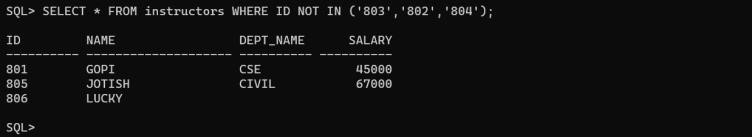
The following command displays salary that are not in between 45000 and 77000.



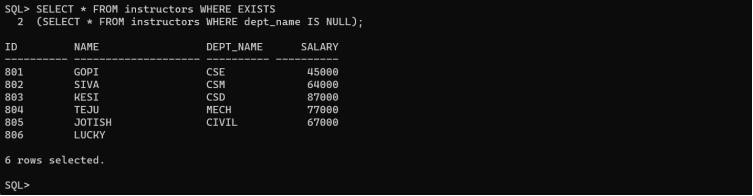
**IN:** This is used to check a member is in a set or not. It displays if the id’s are present in the table.



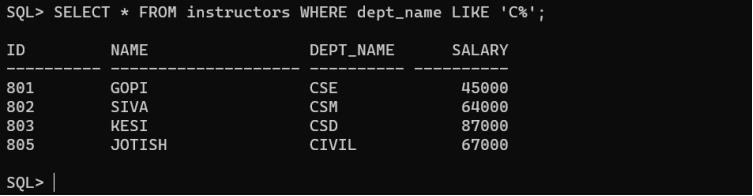
The following command displays all the attributes with id’s except the given id’s.



**EXISTS:** This is used to check whether given set is empty or not. It displays null attributes that are null according to the given condition.



**LIKE:** This is used to check given string is present or not. It displays all the attributes that start with character ‘c’.

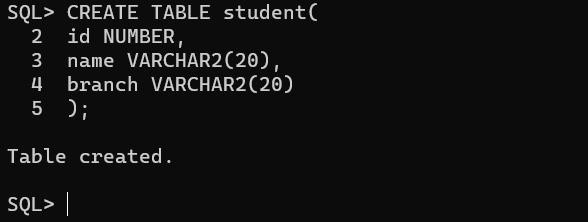


**Conclusion:** In this lab, we have practiced SPECIAL OPERATIONS IS NULL, BETWEEN, LIKE, IN, EXISTS on user created table.

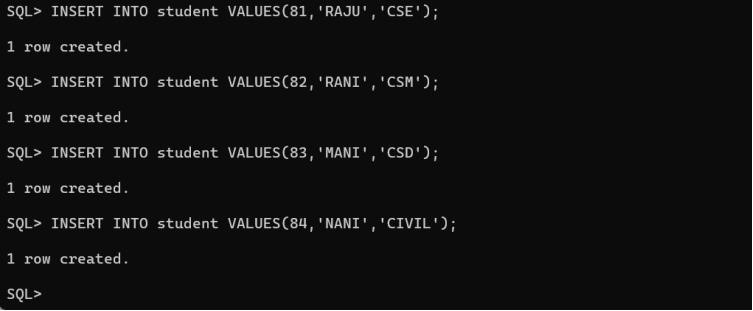
# EXPERIMENT-6

AIM: To implement SQL queries to perform JOIN OPERATIONS (CONDITIONAL JOIN, EQUI JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN).

CREATING TABLE-1:

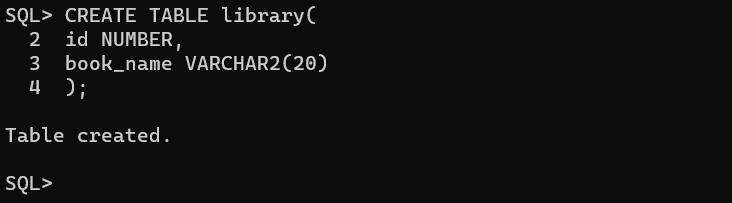


INSERTING VALUES INTO THE TABLE:

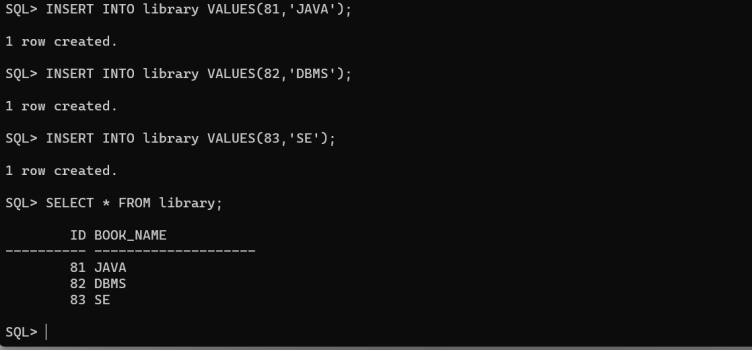




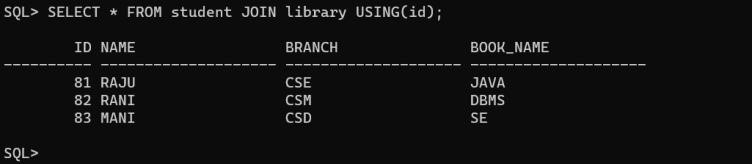
CREATING TABLE-2:



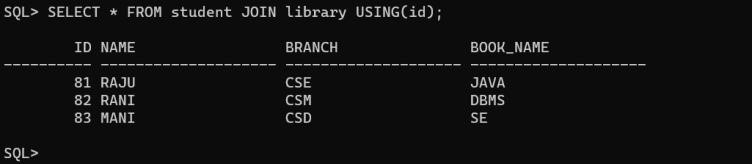
INSERTING VALUES INTO TABLE-2:



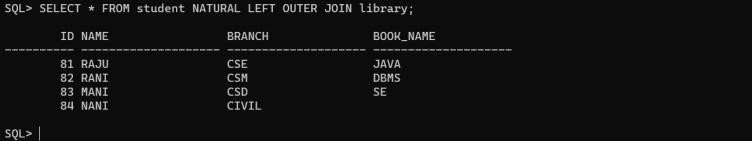
CONDITIONAL JOIN: It helps in retrieving the desired data and performing complex queries.



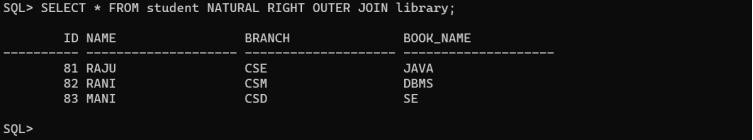
EQUI JOIN: It helps in retrieving related information from different tables by matching corresponding values.



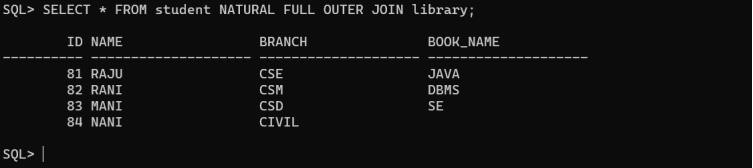
LEFT OUTER JOIN: It combines data from two or more tables based on the matching values in specified columns, but it also includes unmatched rows from the left table.



RIGHT OUTER JOIN: It combines data from two or more tables based on the matching values in specified columns, but also includes unmatched rows from he right table.



FULL OUTER JOIN: It includes all the rows from both the left and right tables, even if there is no match.

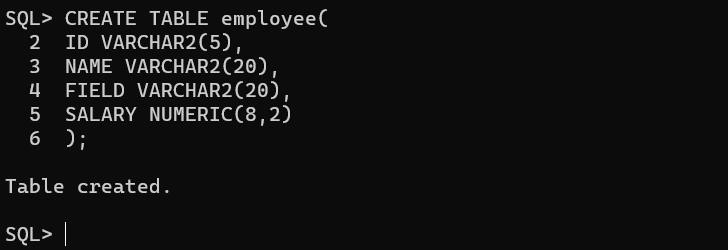


Conclusion: In this lab, we have practiced JOIN OPERATIONS CONDITIONAL JOIN, EQUI JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN on user created tables.

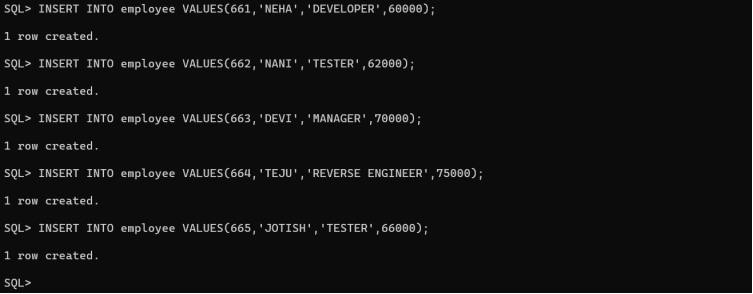
# EXPERIMENT-7

**AIM:** To implement SQL queries to perform AGGREGATE OPERATIONS (SUM, COUNT, AVG, MIN, MAX).

**CREATING A TABLE:**

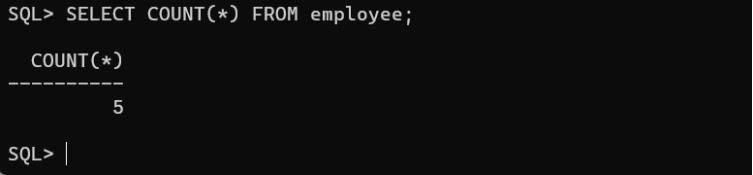


**INSERTING VALUES INTO TABLE:**

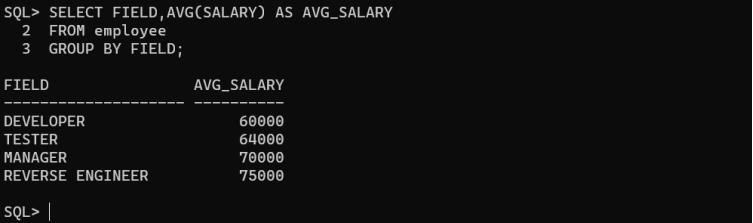




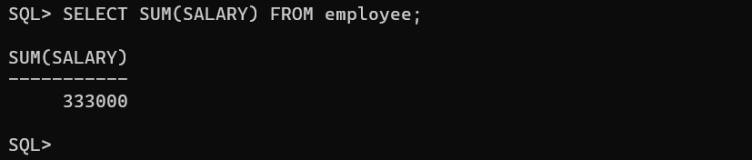
**COUNT:** It displays the count on members present in employee.



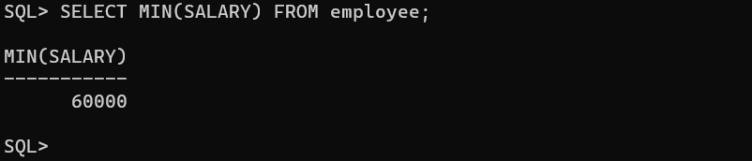
**AVERAGE(AVG):** It displays average salary of each employee.



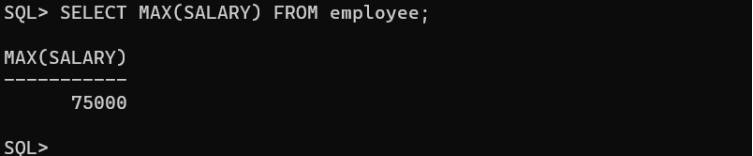
**SUM:** It displays sum of all the salaries from the table.



**MIN:** It displays the minimum salary from the table.



**MAX:** It displays the maximum salary from the table.



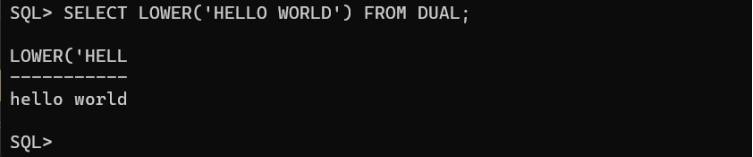
**Conclusion:** In this lab, we have practiced AGGREGATE OPERATIONS like SUM, COUNT, AVERAGE, MIN, MAX on user created table.

# EXPERIMENT-8

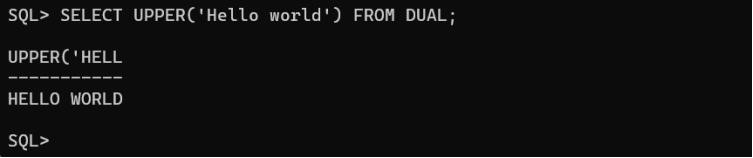
**AIM:** To implement SQL queries to perform BUILT-IN FUNCTIONS (DATE, TIME).

**CASE CONVERSION:**

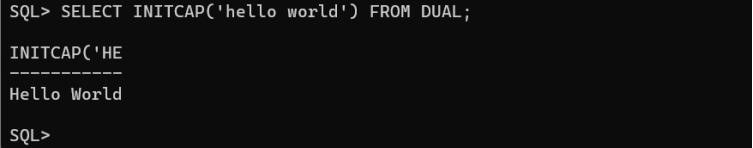
**LOWER ():** It converts a string into lowercase.



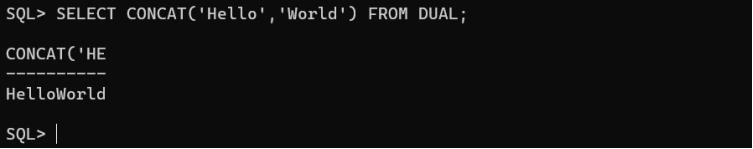
**UPPER ():** It converts a string into uppercase.



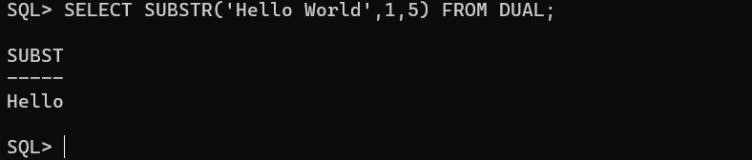
**INITCAP ():** It converts a string into camel case.



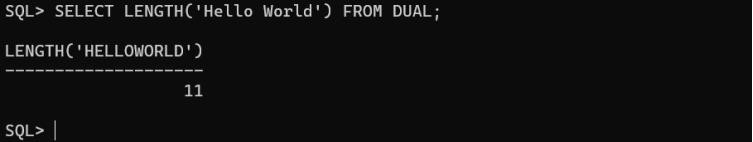
**CONCAT ():** It adds two or more expressions together.



**SUBSTR ():** It extracts a substring from a string.



**LENGTH ():** It returns the length of the given string.



**INSTR ():** It returns the position or the first occurrence of a string in another string.

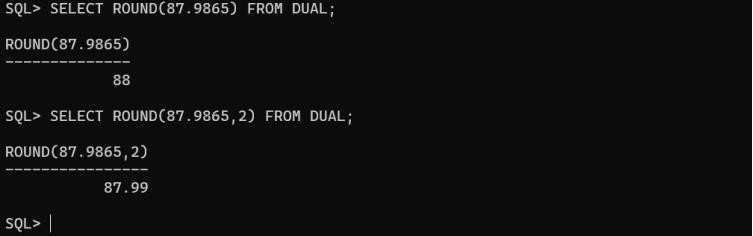


**TRIM ():** It removes the selected one from string.

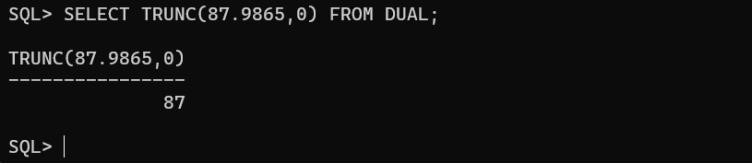


**NUMBER FUNCTIONS:**

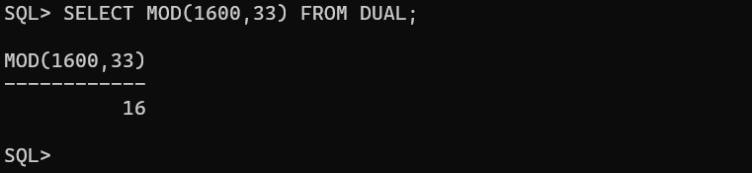
**ROUND ():** It returns the specified values.



**TRUNCATE ():** It removes the decimal values which are specified.

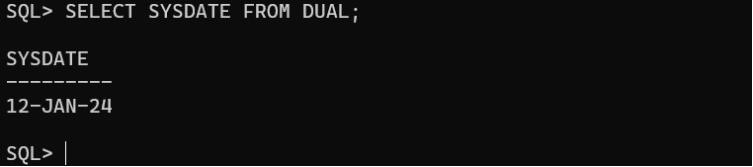


**MOD ():** It returns the remainder.

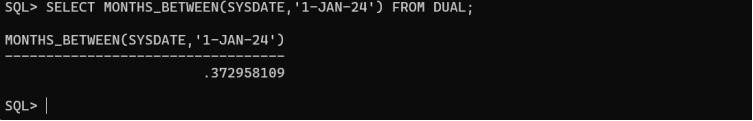


**DATE FUNCTIONS:**

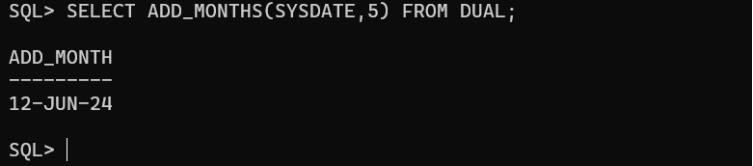
**SYSDATE ():**



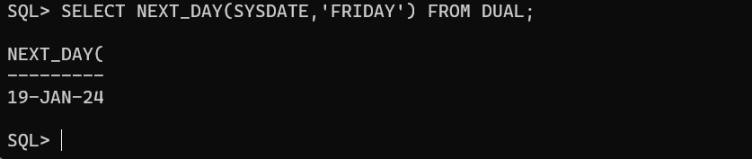
**MONTHS\_BETWEEN ():**



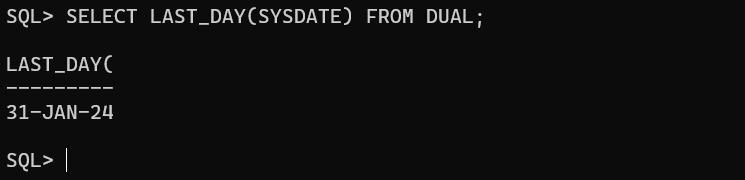
**ADD\_MONTHS ():**



**NEXT\_DAY ():**



**LAST\_DAY ():**



**TRUNC ():**

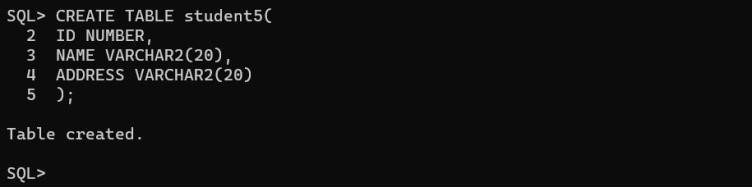


**Conclusion:** In this lab, we have practiced BUILT-IN FUNCTIONS like DATE AND TIME .

# EXPERIMENT-9

**AIM:** To implement SQL queries to perform KEY CONSTRAINTS (PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK, DEFAULT).

**PRIMARY KEY:** A primary key is a field which can uniquely identify each row in table and this constraint is used to specify a field as primary key.

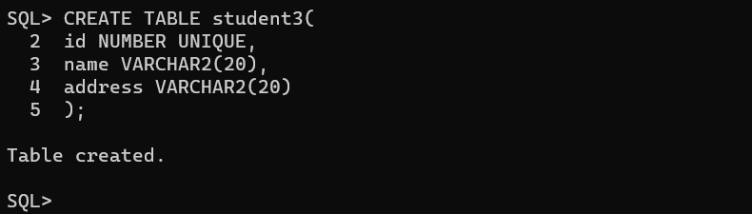


**FOREIGN KEY:** A foreign key is a field which can uniquely each row in another table.

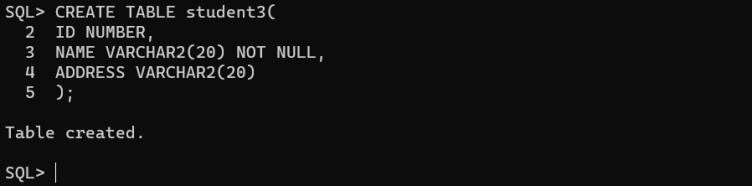




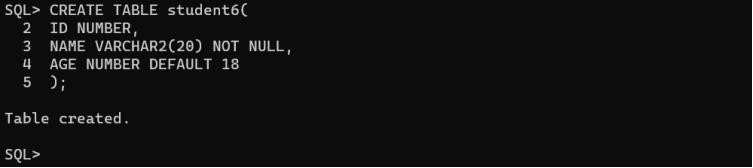
**UNIQUE:** This constraint when specified with a column, tells that the values in the column must be unique i.e., the values in any row of a column must not be repeated.



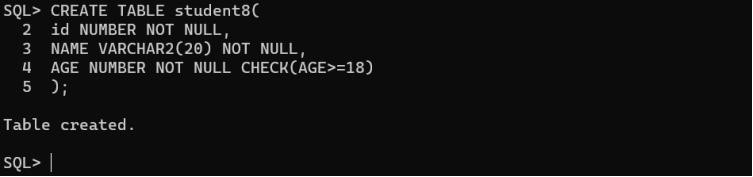
**NOT NULL:** This constraint tells that we cannot store a null value in a column.



**DEFAULT:** This constraint specifies a default value for the column when no value is specified by the user.



**CHECK:** This constraint helps to validate the value for the column to meet a particular condition i.e. it helps to ensure that the value stored in a column meets a specific condition.

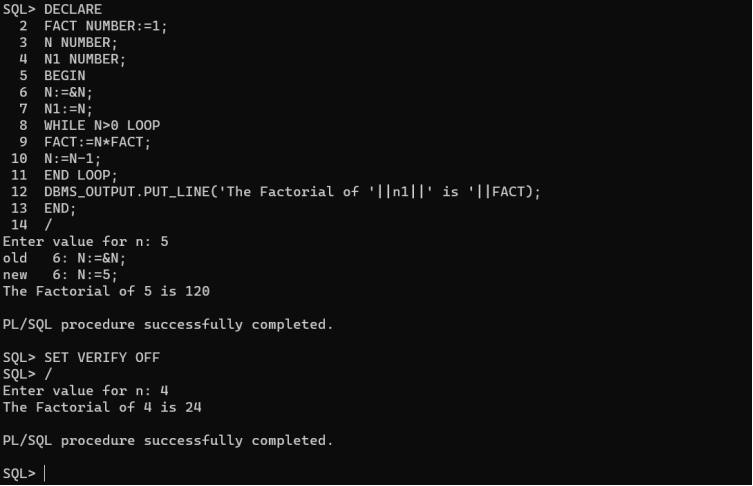


**Conclusion:** In this lab, we have practiced KEY CONSTRAINTS PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK, DEFAULT on user created tables.

# EXPERIMENT-10

**AIM:** To write a PL/SQL program for calculating the factorial of a given number.

**SOURCE CODE & OUTPUT:**



* To run the program ‘/’ is used.
* To display the output, we use “SET SERVEROUT ON”.
* To eliminate debugging message “SET VERIFY OFF” should be used.

**Conclusion:** In this lab, we have practiced a PL/SQL program to calculate factorial of given number.

# EXPERIMENT-11

**AIM:** To write a PL/SQL program for finding the given number is prime or not.

**SOURCE CODE & OUTPUT:**



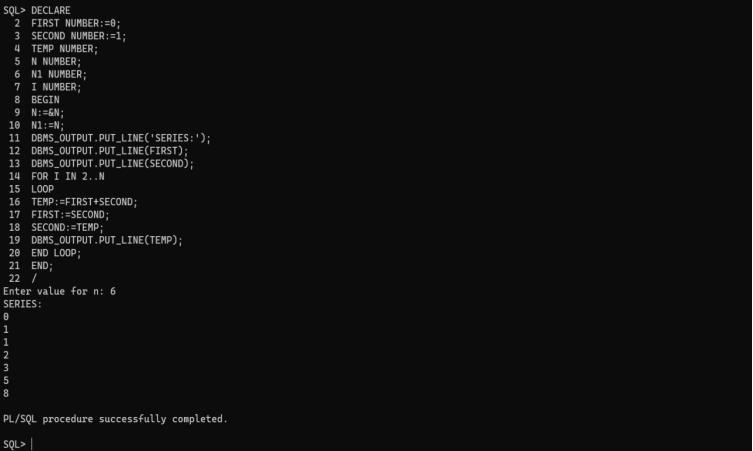
* To run the program ‘/’ is used.
* To display the output, we use “SET SERVEROUT ON”.
* To eliminate debugging message “SET VERIFY OFF” should be used.

**Conclusion:** In this lab, we have practiced a PL/SQL program for finding a given number is prime or not.

# EXPERIMENT-12

**AIM:** To write a PL/SQL program for displaying the Fibonacci series up to an integer.

**SOURCE CODE & OUTPUT:**



* To run the program ‘/’ is used.
* To display the output, we use “SET SERVEROUT ON”.
* To eliminate debugging message “SET VERIFY OFF” should be used.

**Conclusion:** In this lab, we have practiced a PL/SQL program for displaying Fibonacci series up to an integer.

# EXPERIMENT-13

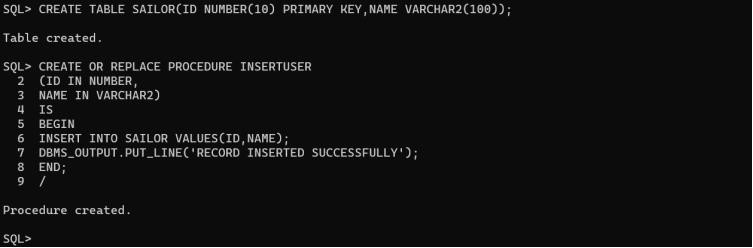
**AIM:** To write a PL/SQL program to implement Stored Procedure on table.

**PL/SQL Procedure:**

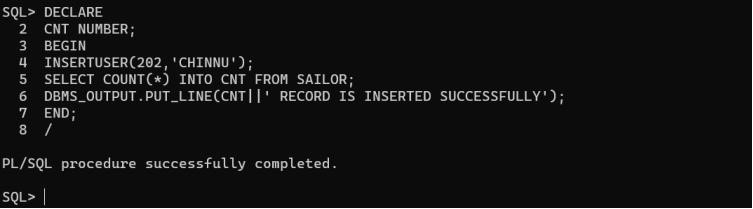
The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

**EXAMPLE-1:**



**EXECUTION PROCEDURE:**



**DROP PROCEDURE:**



**Conclusion:** In this lab, we have practiced a PL/SQL program to implement Stored Procedure on table.

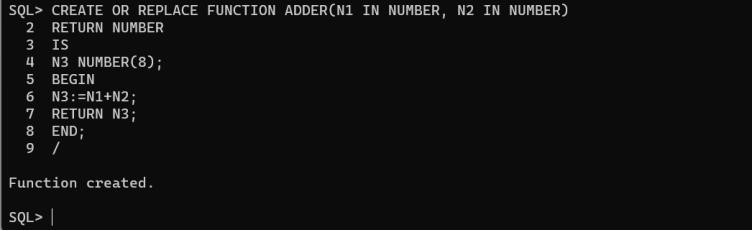
# EXPERIMENT-14

**AIM:** To write a PL/SQL program to implement Stored Function on table.

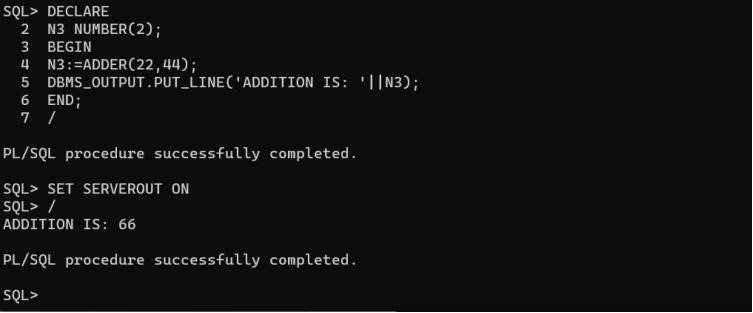
**PL/SQL Function:**

The Pl/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Expect this, all other things of PL/SQL procedure are true for PL/SQL function too.

**EXAMPLE-1:**

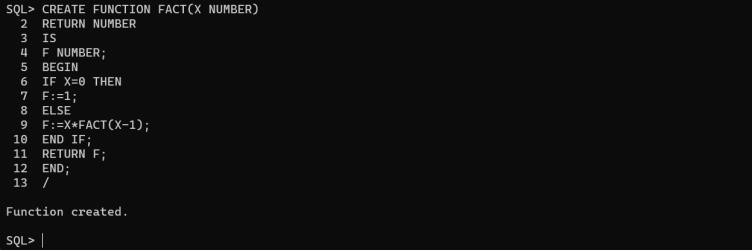


**EXECUTION PROCEDURE:**

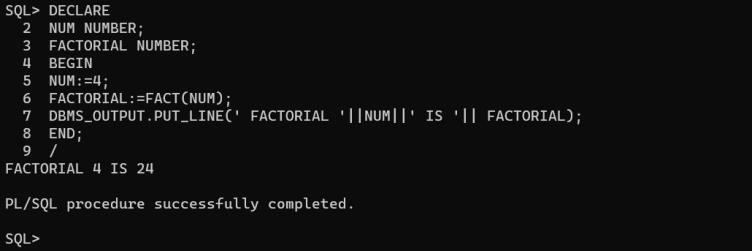




**EXAMPLE-2:**



**EXECUTION PROCEDURE:**





# EXPERIMENT-15

**AIM:** To write PL/SQL program to implement Trigger on table.

**Tigger:**

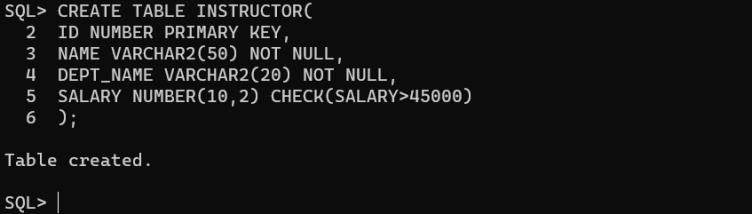
Trigger is invoked by Oracle engine automatically whenever a specified event occurs. Trigger is stored into database and invoked repeatedly, when specific condition match. Triggers are stored programs, which are automatically executed or fired when some event occurs. Triggers are written to be executed in response to any of the following events.

A database manipulation (DML) statement (DELETE, INSERT, UPDATE).

A database definition (DDL) statement (CREATE, ALTER, DROP).

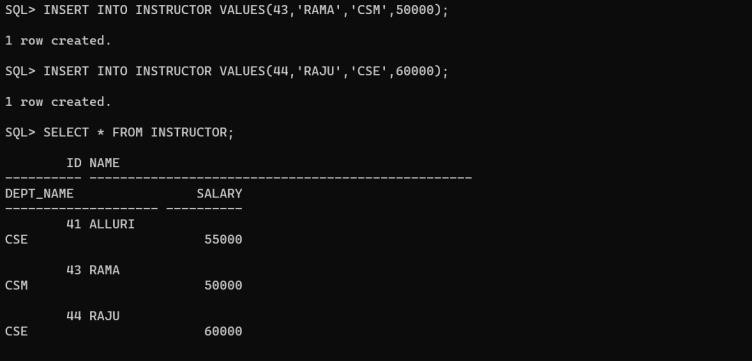
A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, SHUTDOWN).

**CREATING A TABLE:**

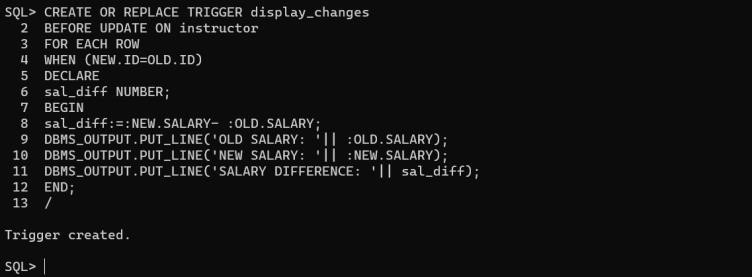


**INSERTING VALUES INTO THE TABLE:**

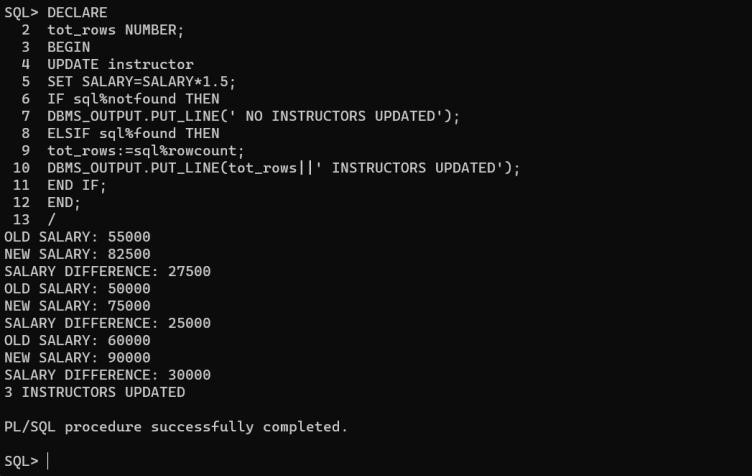




**AN EXAMPLE TO CREATE TRIGGER:**



**A PL/SQL Procedure to execute a trigger:**

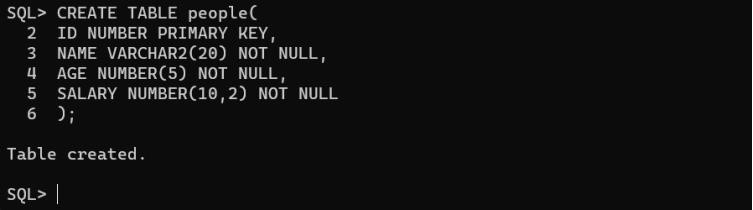


**Conclusion:** In this lab, we have practiced a PL/SQL program to implement Trigger on table.

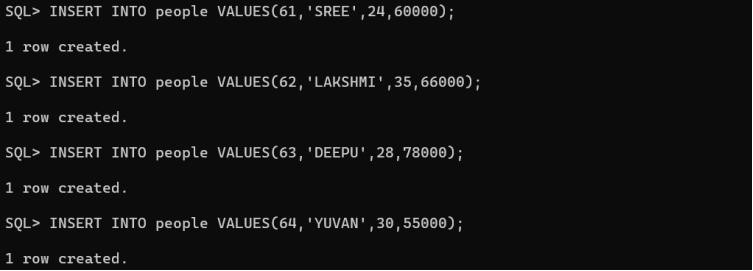
# EXPERIMENT-16

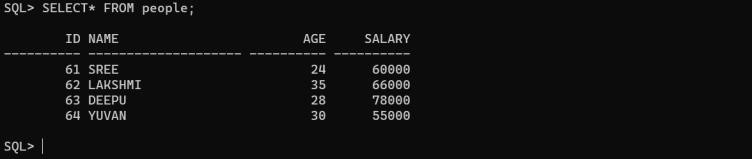
**AIM:** To write a PL/SQL program to implement Cursor on table.

**CREATING A TABLE:**



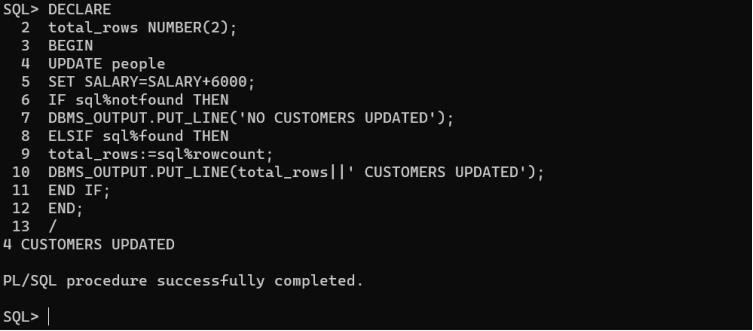
**INSERTING VALUES INTO TABLE:**



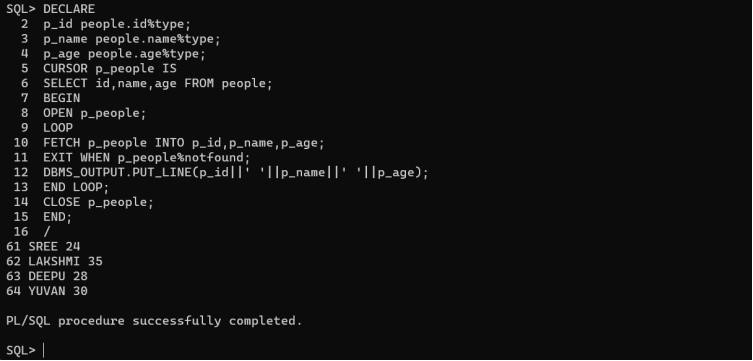


**CREATE UPDATE PROCEDURE:**

**CREATE PROCEDURE:**



**PL/SQL Program using Explicit Cursors:**



**Conclusion:** In this lab, we have practiced a PL/SQL program to implement Cursor on table.