

# ***ADVANCED DATABASE MANAGEMENT SYSTEM LAB***

***ETCS - 457***

Faculty Name:

Student Name:

Roll No.:

Semester: 7

Group:



Maharaja Agrasen Institute of Technology, PSP area,  
Sector - 22, Rohini, New Delhi - 110085

(Affiliated to Guru Gobind Singh Indraprastha  
University, New Delhi)

# **Introduction to Advanced DBMS Lab**

## **1.1 Objective**

This course aims to give students in depth information about database implementation techniques, data storage, representing data elements, database system architecture, the system catalog, query processing and optimization. Advanced DBMS lab provides a platform for practicing various software's such as Oracle, MySQL, PostgreSQL. All these require a thorough practice of various DDL, DCL and DML queries.

## **1.2 Course outcomes**

**At the end of the course, a student will be able to:**

ETCS457.1: To compare different Databases such as PostgreSQL, Oracle, IBM DB2, MySql.

ETCS457.2: Illustrate and install PostgreSql.

ETCC457.3: Design and develop advanced queries using Structured Query Language for PostgreSql.

ETCS457.4: Learn to build procedures, functions and triggers.

ETCS457.5: Learn the skills required to develop a complete data-intensive application.

## Department of Computer Science and Engineering

### Rubrics for Lab Assessment

Rubrics		0	1	2	3
		Missing	Inadequate	Needs Improvement	Adequate
R1	Is able to identify the problem to be solved and define the objectives of the experiment.	No mention is made of the problem to be solved.	An attempt is made to identify the problem to be solved but it is described in a confusing manner, objectives are not relevant, objectives contain technical/ conceptual errors or objectives are not measurable.	The problem to be solved is described but there are minor omissions or vague details. Objectives are conceptually correct and measurable but may be incomplete in scope or have linguistic errors.	The problem to be solved is clearly stated. Objectives are complete, specific, concise, and measurable. They are written using correct technical terminology and are free from linguistic errors.
R2	Is able to design a reliable experiment that solves the problem.	The experiment does not solve the problem.	The experiment attempts to solve the problem but due to the nature of the design the data will not lead to a reliable solution.	The experiment attempts to solve the problem but due to the nature of the design there is a moderate chance the data will not lead to a reliable solution.	The experiment solves the problem and has a high likelihood of producing data that will lead to a reliable solution.
R3	Is able to communicate the details of an experimental procedure clearly and completely.	Diagrams are missing and/or experimental procedure is missing or extremely vague.	Diagrams are present but unclear and/or experimental procedure is present but important details are missing.	Diagrams and/or experimental procedure are present but with minor omissions or vague details.	Diagrams and/or experimental procedure are clear and complete.
R4	Is able to record and represent data in a meaningful way.	Data are either absent or incomprehensible.	Some important data are absent or incomprehensible.	All important data are present, but recorded in a way that requires some effort to comprehend.	All important data are present, organized and recorded clearly.
R5	Is able to make a judgment about the results of the experiment.	No discussion is presented about the results of the experiment.	A judgment is made about the results, but it is not reasonable or coherent.	An acceptable judgment is made about the result, but the reasoning is flawed or incomplete.	An acceptable judgment is made about the result, with clear reasoning. The effects of assumptions and experimental uncertainties are considered.

## ADVANCED DBMS LAB PRACTICAL RECORD

PAPER CODE: ETCS – 457

BRANCH: CSE

GROUP:

S.No	EXPERIMENT NAME	DATE	MARKS					Total Marks	Signature
			R1	R2	R3	R4	R5		
1(a).	Introduction to PostgreSQL and its installation steps.								
1(b).	Compare different Databases such as PostgreSQL, Oracle, MySql and Maria DB.								
2.	Consider Schema: Student (student_name, enrollment_no, marks, area, branch). Write SQL queries on •DDL (create, alter, drop, rename, truncate), •DML (Insert, update, delete etc.), •DCL (Grant, revoke etc.) •Built-in Functions (sum, min, max, avg, count, lower, upper, trim, len etc.) •Indexes and views: Create and Drop								
3.	Write SQL queries on Nested queries and Join.								
4.	Write a program a. To calculate total students enrolled area-wise. b. To calculate average marks obtained by students residing in area that starts with "R"								
5.	Write the cursor to increase the marks of student by 10%.								

6.	Write a program to create exceptions if enrollment no. is not issued to student by the university and raise the exception explicitly by using raise command.								
7.	Procedure & Function a. Write the procedure to get the average marks of students for branch "CSE". b. Write a function that accepts branch and returns the total no. of students of the branch.								
8.	Trigger a. Create a trigger on table after inserting a new student into table. b. Write a row trigger to insert the existing values of the student table in to a new table when the marks of student is updated.								

# **EXPERIMENT – 1**

**AIM:** Introduction to PostgreSQL and its installation steps.

## **Theory:**

PostgreSQL, often simply Postgres, is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards compliance. As a database server, its primary functions are to store data securely and return that data in response to requests from other software applications. It can handle workloads ranging from small single-machine applications to large Internet-facing applications (or for data warehousing) with many concurrent users; on macOS Server, PostgreSQL is the default database; and it is also available for Microsoft Windows and Linux (supplied in most distributions).

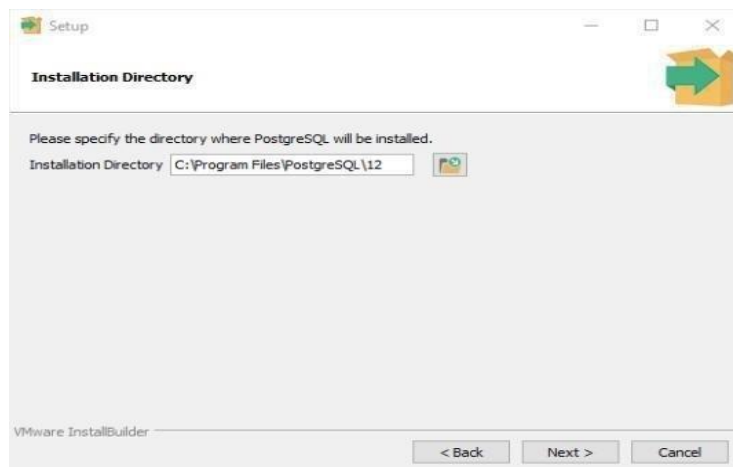
PostgreSQL is ACID-compliant and transactional. PostgreSQL has updatable views and materialized views, triggers, foreign keys; supports functions and stored procedures, and other expandability.

PostgreSQL is developed by the PostgreSQL Global Development Group, a diverse group of many companies and individual contributors. It is free and open-source, released under the terms of the PostgreSQL License, a permissive software license.

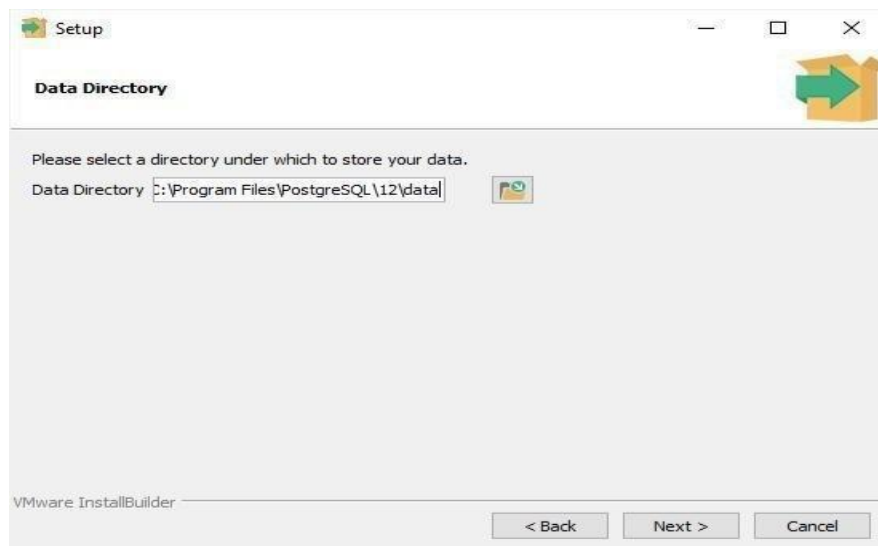
## **Installing PostgreSQL on Windows**

Following steps were to install PostgreSQL on Windows machine. It was made sure that Third Party Antivirus has been off while installing.

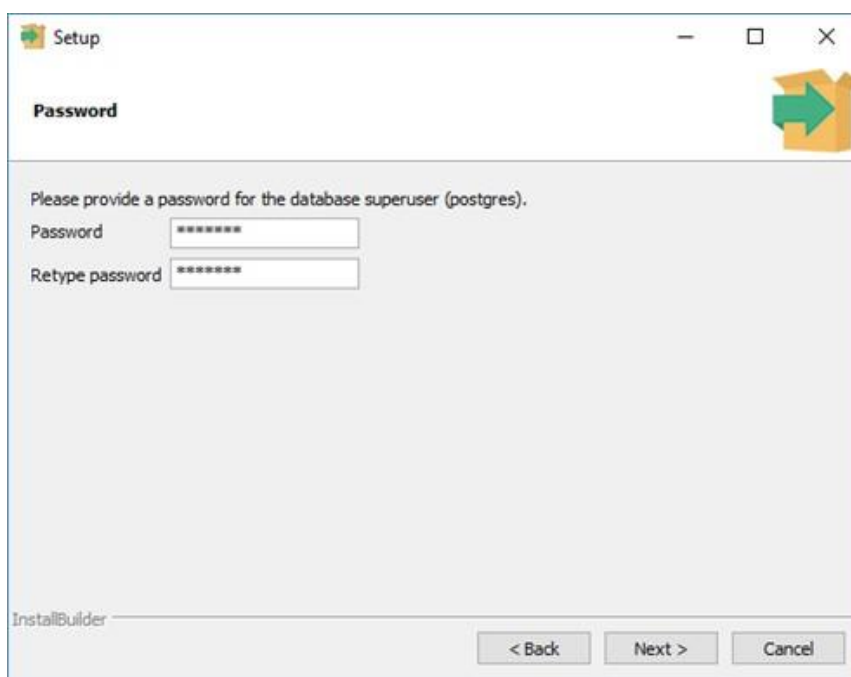
- Pick the version number of PostgreSQL you want and, as exactly as possible, the platform you want from a EnterpriseDB
- I download postgresql-14.0-1-windows.exe for my Windows PC running in 64 bit mode, so lets run postgresql-14.0-1-windows.exe as administrator to install PostgreSQL. Select the location where you want to install it. By default it is installed within Program Files folder.



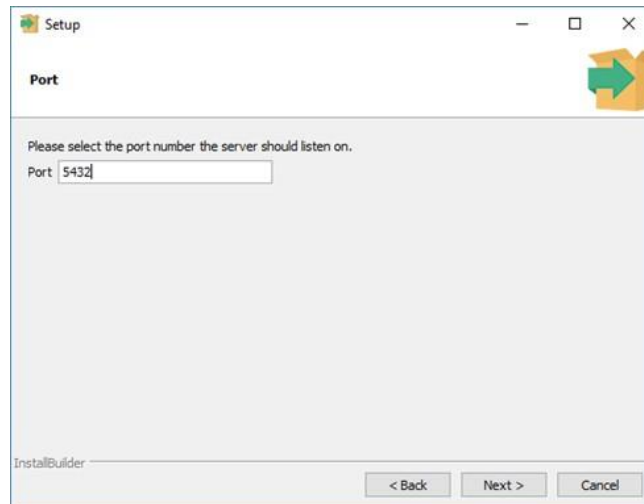
- The next step of the installation process would be to select the directory where data would be stored, by default it is stored under "data" directory



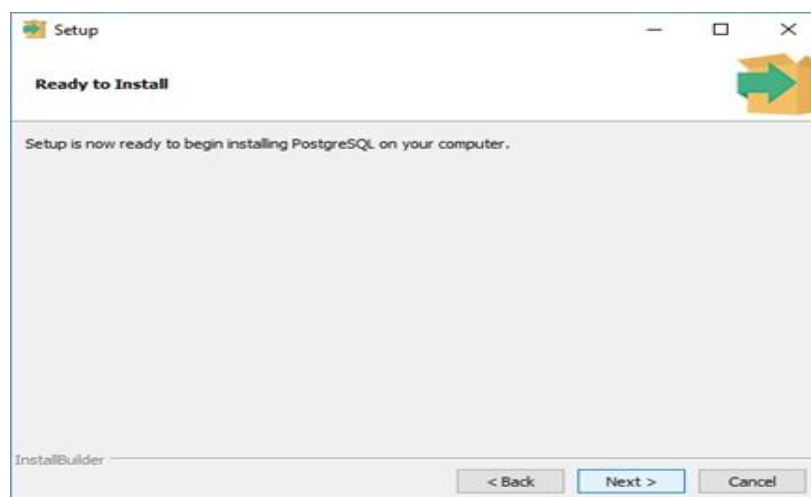
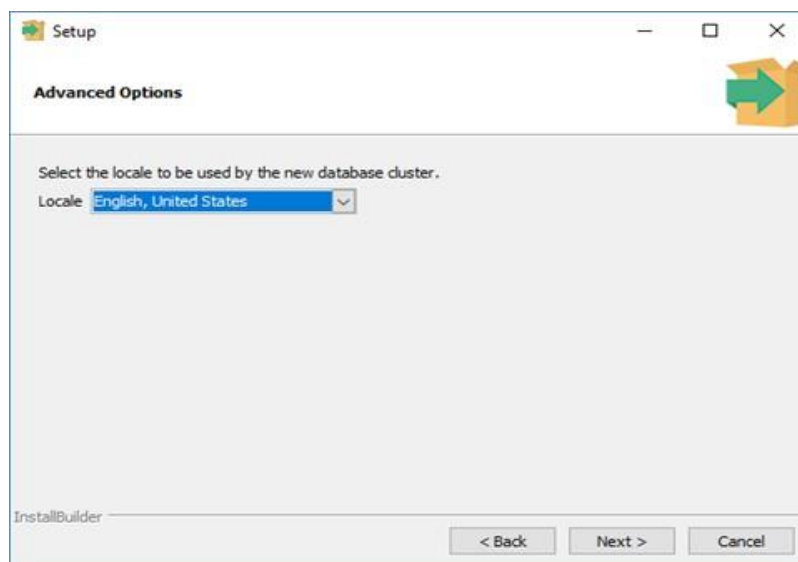
- The next step, setup asks for password, so you can use your favorite password



- The next step, keep the port as default.

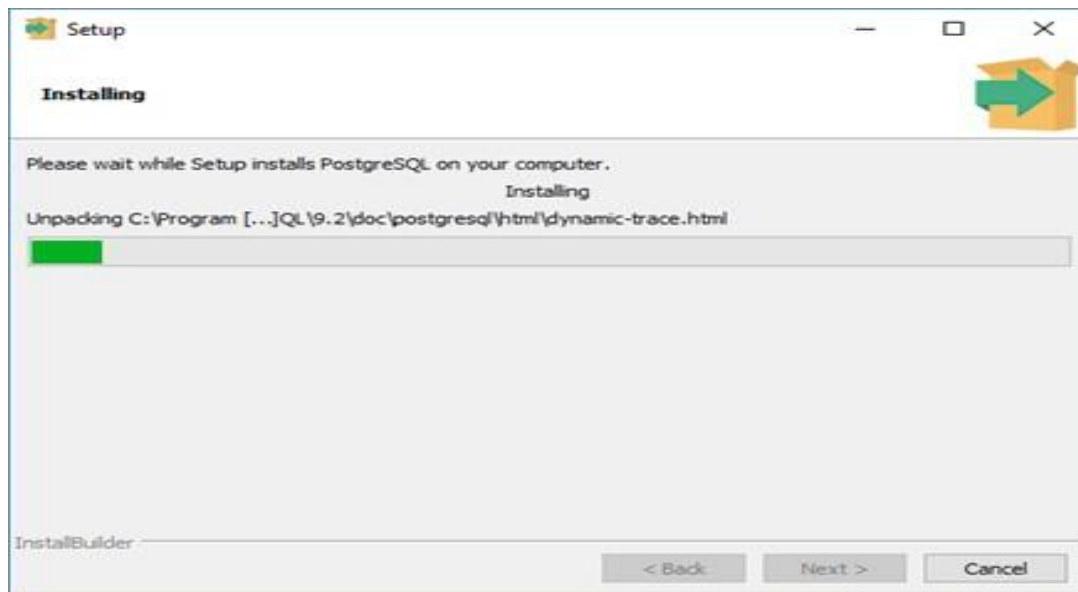


- The next step, when asked for "Locale", "English, United States" has been selected.





- It takes a while to install PostgreSQL on your system.



- On completion of the installation process, you will get the following screen. Uncheck the checkbox and click on Finish button.



## **EXPERIMENT – 1(b)**

**AIM:** Comparison of various Databases such as Postgre, Oracle, IBM DB2, MYSql and Maria database.

### **Theory:**

#### **Comparison between MariaDB and PostgreSQL:**

S.No	MariaDB	PostgreSQL
1	Developed by MariaDB Corporation Ab and MariaDB Foundation on 2009.	Developed By PostgreSQL Global Development Group on 1989
2	It is a MySQL application compatible open source RDBMS, enhanced with high availability, security, interoperability and performance capabilities.	It is widely used open source RDBMS
3	MariaDB is written in C and C++ languages.	PostgreSQL is written in C languages.
4	The primary database model for MariaDB is Relational DBMS.	Also the primary database model for PostgreSQL is Relational DBMS.
5	It has two Secondary database models – Document store and Graph DBMS.	It has Document store as Secondary database models.
6	It supports Server-side scripting.	It has user defined functions for Server-side scripts
7	It supports in-memory capabilities	It does not supports in-memory capabilities.

#### **Comparison between IBM DB2 and PostgreSQL:**

S.No	IBM DB2	PostgreSQL
1	IBM DB2 is a relational database model.	PostgreSQL is a object-relational database model
2	IBM DB2 was developed by IBM in 1983.	PostgreSQL was developed by PostgreSQL Global Development group in 1989.
3	In IBM DB2 partitioning is done by sharding..	In IBM DB2 partitioning is done by use of list, hash and range.
4	It has a commercial license	It is a open source software.

5	It is written in C and C++ languages	It is written in C++ language
6	It is a family of database management products given by IBM.	It is a advanced relational DBMS that is a extended form of SQL.
7	It has less availability as compared to PostgreSQL.	It has more availability as compared to IBM DB2.

### Comparison between Oracle and PostgreSQL:

S.No	Oracle	PostgreSQL
1	Oracle is a relational management system.It is first database designed for grid computing.	PostgreSQL is free open source relational-database management system emphasizing extensibility and SQL compliance.
2	Oracle is more secure than PostgreSQL	PostgreSQL provide good security but it is not secure as Oracle.
3	Oracle written in c and C++ language.	PostgreSQL written in C language.
4	Oracle required license.	PostgreSQL is open source.
5	Oracle support cost based.	PostgreSQL provide free support or option with paid support at low cost.

### Comparison between MySQL and PostgreSQL:

S.No	MySQL	PostgreSQL
1	It is the most popular Database.	It is the most advanced Database.
2	It is a relational based DBMS.	It is an object based relational DBMS
3	Implementation language is C/C++.	Implementation language is C.
4	It does not support CASCADE option.	CASCADE option is supported.
5	GUI tool provided is MySQL Workbench	PgAdmin is provided
6	It does not support partial, bitmap and expression indexes.	It supports all of these.
7	SQL only support Standard data types.	It support Advanced data types such as arrays, hstore and user defined types.

## **EXPERIMENT – 2**

**AIM:** Consider Schema: Student (student\_name, enrollment\_no, marks, area, branch).

Write SQL queries on

- DDL (create, alter, drop, rename, truncate),
- DML (Insert, update, delete etc.),
- DCL (Grant, revoke etc.)
- Built-in Functions (sum, min, max, avg, count, lower, upper, trim, len etc.)
- Indexes and views: Create and Drop

### **Theory:**

**1. Data Definition Language (DDL):** DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc. All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE It is used to create a new table in the database.
- DROP: It is used to delete both the structure and record stored in the table.
- ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.
- TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.
- RENAME TABLE helps in changing the name of the table.

**2. Data Manipulation Language (DML):** DML commands are used to modify the database. It is responsible for all form of changes in the database. The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

- INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.
- UPDATE: This command is used to update or modify the value of a column in the table.
- DELETE: It is used to remove one or more row from a table.

### **3. Data Control Language (DCL)**

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant: It is used to give user access privileges to a database.
- Revoke: It is used to take back permissions from the user.

### **4. Built In Functions**

- SQL SUM function is used to find out the sum of a field in various records.
- SQL MIN function is used to find out the record with minimum value among a record set.

- SQL MAX function is used to find out the record with maximum value among a record set.
- SQL AVG function is used to find out the average of a field in various records.
- SQL COUNT function is the simplest function and very useful in counting the number of records, which are expected to be returned by a SELECT statement.
- SQL LOWER function helps to convert all the letters of the given string to lowercase.
- The UPPER function converts a string to upper-case.
- The TRIM function removes the space character OR other specified characters from the start or end of a string. By default, the TRIM function removes leading and trailing spaces from a string.
- The LEN function returns the length of a string.

**5. Views:** In SQL, a view is a virtual table based on the result-set of an SQL statement.\ A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. User can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

- Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables or another view.
- A view is deleted using the DROP VIEW command.

## • DDL COMMANDS:

### CREATE TABLE:

The screenshot displays a SQL IDE interface. At the top, there are two tabs: 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, showing a SQL statement to create a table named 'student'. The statement is as follows:

```
1 CREATE TABLE student
2 (student_name varchar(20),enrollment_no varchar(11),
3 marks int,area varchar(20),branch varchar(30));
4
```








Below the query editor, there are four tabs: 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the execution result:

```
CREATE TABLE







Query returned successfully in 57 msec.
```

## ALTER TABLE:

Query: ALTER TABLE student ADD email varchar(255);

Data Output		Explain	Messages	Notifications		
	student_name character varying (20) 	enrollment_no character varying (11) 	marks integer 	area character varying (20) 	branch character varying (30) 	email character varying (255) 
1	Aakash	1	100	Delhi	CSE	[null]
2	Harsh	2	95	Delhi	CSE	[null]
3	Siddharth	3	98	Punjab	IT	[null]
4	Bhavay	4	89	Delhi	ECE	[null]
5	Rohan	5	80	Haryana	EEE	[null]
6	Abhishek	6	92	Agra	IT	[null]
7	Aditya	7	85	Noida	MAE	[null]
8	Sarthak	8	99	Kanpur	CSE	[null]
9	Nischay	9	94	Chandigarh	ECE	[null]
10	Muskan	10	97	Faridabad	EEE	[null]

Query: ALTER TABLE student DROP COLUMN email;

Data Output		Explain	Messages	Notifications	
	student_name character varying (20) 	enrollment_no character varying (11) 	marks integer 	area character varying (20) 	branch character varying (30) 
1	Aakash	1	100	Delhi	CSE
2	Harsh	2	95	Delhi	CSE
3	Siddharth	3	98	Punjab	IT
4	Bhavay	4	89	Delhi	ECE
5	Rohan	5	80	Haryana	EEE
6	Abhishek	6	92	Agra	IT
7	Aditya	7	85	Noida	MAE
8	Sarthak	8	99	Kanpur	CSE
9	Nischay	9	94	Chandigarh	ECE
10	Muskan	10	97	Faridabad	EEE

Query: ALTER TABLE student RENAME TO student\_table; (**RENAME TABLE**)

Query Editor

Query History

1

SELECT\* FROM student\_table;

Data Output

Explain

Messages

Notifications

	student_name character varying (20)	enrollment_no character varying (11)	marks integer	area character varying (20)	branch character varying (30)
1	Aakash	1	100	Delhi	CSE
2	Harsh	2	95	Delhi	CSE
3	Siddharth	3	98	Punjab	IT
4	Bhavay	4	89	Delhi	ECE
5	Rohan	5	80	Haryana	EEE
6	Abhishek	6	92	Agra	IT
7	Aditya	7	85	Noida	MAE
8	Sarthak	8	99	Kanpur	CSE
9	Nischay	9	94	Chandigarh	ECE
10	Muskan	10	97	Faridabad	EEE

## TRUNCATE TABLE:

Query Editor

Query History

1

TRUNCATE TABLE student;

2

SELECT \* from student;

Data Output

Explain

Messages

Notifications

student\_name

character varying (20)

enrollment\_no

character varying (11)

marks

integer

area

character varying (20)

branch

character varying (30)

## DROP TABLE:

Query Editor	Query History
1 DROP TABLE student;	
Data Output	Explain
	Messages
Notifications	
DROP TABLE	
Query returned successfully in 40 msec.	

- **DML COMMANDS:**

## INSERT INTO TABLE

Query:

Query Editor	Query History
1	<b>INSERT into</b> student <b>VALUES</b> ('Aakash','1', <b>100</b> ,'Delhi','CSE');
2	<b>INSERT into</b> student <b>VALUES</b> ('Harsh','2', <b>95</b> ,'Delhi','CSE');
3	<b>INSERT into</b> student <b>VALUES</b> ('Siddharth','3', <b>98</b> ,'Punjab','IT');
4	<b>INSERT into</b> student <b>VALUES</b> ('Bhavay','4', <b>89</b> ,'Delhi','ECE');
5	<b>INSERT into</b> student <b>VALUES</b> ('Rohan','5', <b>80</b> ,'Haryana','EEE');
6	<b>INSERT into</b> student <b>VALUES</b> ('Abhishek','6', <b>92</b> ,'Agra','IT');
7	<b>INSERT into</b> student <b>VALUES</b> ('Aditya','7', <b>85</b> ,'Noida','MAE');
8	<b>INSERT into</b> student <b>VALUES</b> ('Sarthak','8', <b>99</b> ,'Kanpur','CSE');
9	<b>INSERT into</b> student <b>VALUES</b> ('Nischay','9', <b>94</b> ,'Chandigarh','ECE');
10	<b>INSERT into</b> student <b>VALUES</b> ('Muskan','10', <b>97</b> ,'Faridabad','EEE');

Data Output		Explain	Messages	Notifications	
	student_name character varying (20)	enrollment_no character varying (11)	marks integer	area character varying (20)	branch character varying (30)
1	Aakash	1	100	Delhi	CSE
2	Harsh	2	95	Delhi	CSE
3	Siddharth	3	98	Punjab	IT
4	Bhavay	4	89	Delhi	ECE
5	Rohan	5	80	Haryana	EEE
6	Abhishek	6	92	Agra	IT
7	Aditya	7	85	Noida	MAE
8	Sarthak	8	99	Kanpur	CSE
9	Nischay	9	94	Chandigarh	ECE
10	Muskan	10	97	Faridabad	EEE



## UPDATE TABLE:

Query Editor

Query History

1

UPDATE student SET student\_name = 'Aakash Garg' WHERE enrollment\_no = '1';

2

SELECT \* FROM student;

Data Output

Explain

Messages

Notifications

	student_name character varying (20)	enrollment_no character varying (11)	marks integer	area character varying (20)	branch character varying (30)
1	Harsh	2	95	Delhi	CSE
2	Siddharth	3	98	Punjab	IT
3	Bhavay	4	89	Delhi	ECE
4	Rohan	5	80	Haryana	EEE
5	Abhishek	6	92	Agra	IT
6	Aditya	7	85	Noida	MAE
7	Sarthak	8	99	Kanpur	CSE
8	Nischay	9	94	Chandigarh	ECE
9	Muskan	10	97	Faridabad	EEE
10	Aakash Garg	1	100	Delhi	CSE

## DELETE TABLE:

Query Editor

Query History

1

DELETE FROM student WHERE marks = 80;

2

SELECT \* FROM student;

Data Output

Explain

Messages

Notifications

	student_name character varying (20)	enrollment_no character varying (11)	marks integer	area character varying (20)	branch character varying (30)
1	Harsh	2	95	Delhi	CSE
2	Siddharth	3	98	Punjab	IT
3	Bhavay	4	89	Delhi	ECE
4	Abhishek	6	92	Agra	IT
5	Aditya	7	85	Noida	MAE
6	Sarthak	8	99	Kanpur	CSE
7	Nischay	9	94	Chandigarh	ECE
8	Muskan	10	97	Faridabad	EEE
9	Aakash Garg	1	100	Delhi	CSE

- **DCL COMMANDS:**

**GRANT COMMAND:**

SYNTAX: grant privilege\_name on object\_name to {user\_name | public | role\_name}

**REVOKE COMMAND:**

SYNTAX: revoke privilege\_name on object\_name from {user\_name | public | role\_name}

- **BUILT IN FUNCTIONS:**

**SUM:**

Query Editor		Query History	
1	SELECT SUM(marks) FROM student;		
Data Output		Explain	Messages
sum bigint			
1	849		

**MIN:**

Query Editor		Query History	
1 SELECT MIN(marks) FROM student;			
Data Output		Explain	Messages
min integer			
1	85		


**MAX:**

Query Editor		Query History	
1 SELECT MAX(marks) FROM student;			
Data Output		Explain	Messages
max integer			
1	100		



## AVG:

Query Editor		Query History	
1	SELECT AVG(marks) FROM student;		
Data Output		Explain	Messages
avg			
	numeric		
1	94.33333333333333		



## COUNT:

Query Editor		Query History		
1	SELECT COUNT(*) FROM student;			
Data Output		Explain	Messages	Notifications
	count bigint			
1	9			

## LOWER:

Query Editor		Query History		
1	SELECT LOWER(student_name) FROM student;			
Data Output		Explain	Messages	Notifications
	lower text 			
1	harsh			
2	siddharth			
3	bhavay			
4	abhishek			
5	aditya			
6	sarthak			
7	nischay			
8	muskan			
9	aakash garg			

## UPPER:

Query Editor		Query History		
1	SELECT UPPER(student_name) FROM student;			
Data Output		Explain	Messages	Notifications
	upper text			
1	HARSH			
2	SIDDHARTH			
3	BHAVAY			
4	ABHISHEK			
5	ADITYA			
6	SARTHAK			
7	NISCHAY			
8	MUSKAN			
9	AAKASH GARG			

## TRIM:

Query Editor		Query History	
<pre>1 SELECT TRIM('A ' FROM (SELECT student_name from student where student_name = 'Aakash Garg'));</pre>			
Data Output		Explain	Messages
btrim text			
1	akash Garg		

- **VIEWS:**

## CREATE VIEW:

Query Editor		Query History	
1	CREATE VIEW	student_view	AS SELECT student_name, enrollment_no FROM student;
	2	SELECT *	FROM student_view;
Data Output		Explain	Messages
student_name character varying (20)		enrollment_no character varying (11)	
1	Harsh	2	
2	Siddharth	3	
3	Bhavay	4	
4	Abhishek	6	
5	Aditya	7	
6	Sarthak	8	
7	Nischay	9	
8	Muskan	10	
9	Aakash Garg	1	

**DROP VIEW:**

```
Query Editor  Query History

1  DROP VIEW student_view;

Data Output  Explain  Messages  Notifications

DROP VIEW

Query returned successfully in 39 msec.
```

## **EXPERIMENT – 3**

**AIM:** Write SQL queries on Nested queries and Join.

### **Theory:**

#### **Nested Queries:**

A Subquery or Inner query or a Nested query is a query within another Postgresqlquery and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

#### **Joins:**

PostgreSQL join is used to combine columns from one (self-join) or more tables based on the values of the common columns between related tables. The common columns are typically the primary key columns of the first table and foreign key columns of the second table.

PostgreSQL supports inner join, left join, right join, full outer join, cross join, natural join, and a special kind of join called self-join.

- **Nested Queries**

Query Editor	Query History
1	SELECT *
2	FROM students
3	WHERE GPA > (
4	SELECT AVG(GPA)
5	FROM students);

Our subquery here returns a single value (i.e. a table with a single column and a single row). This is important for the comparison operator to work. With the average GPA score returned by the inner query, the outer query can select the students who satisfy our filter condition (i.e. a GPA score above average).

And here is the result:

Data Output

Explain

Messages

Notifications

	id integer	name character varying (20)	class_id integer	gpa double precision
1	1	Jack Black	3	3.45
2	3	Katherine Star	1	3.85

- **Joins**

Suppose we have two tables called basket\_a and basket\_b that store fruits:

Query Editor    Query History

```

1 CREATE TABLE basket_a (
2     a INT PRIMARY KEY,
3     fruit_a VARCHAR (100) NOT NULL
4 );
5
6 CREATE TABLE basket_b (
7     b INT PRIMARY KEY,
8     fruit_b VARCHAR (100) NOT NULL
9 );
10
11 INSERT INTO basket_a (a, fruit_a)
12 VALUES (1, 'Apple'), (2, 'Orange'), (3, 'Banana'), (4, 'Cucumber');
13
14 INSERT INTO basket_b (b, fruit_b)
15 VALUES (1, 'Orange'), (2, 'Apple'), (3, 'Watermelon'), (4, 'Pear');
```

	a integer	fruit_a character varying (100)
1	1	Apple
2	2	Orange
3	3	Banana
4	4	Cucumber






	b integer	fruit_b character varying (100)
1	1	Orange
2	2	Apple
3	3	Watermelon
4	4	Pear

## PostgreSQL inner join

The following statement joins the first table (basket\_a) with the second table (basket\_b) by matching the values in the fruit\_a and fruit\_b columns:

Query Editor    Query History

```
1  SELECT
2      a, fruit_a, b, fruit_b
3  FROM
4      basket_a
5  INNER JOIN basket_b
6      ON fruit_a = fruit_b;
7
8
```

Data Output		Explain	Messages	Notifications	
	<b>a</b> integer 	<b>fruit_a</b> character varying (100) 	<b>b</b> integer 	<b>fruit_b</b> character varying (100) 	
1	1	Apple		2	Apple
2	2	Orange		1	Orange

The inner join examines each row in the first table (basket\_a). It compares the value in the fruit\_a column with the value in the fruit\_b column of each row in the second table (basket\_b). If these values are equal, the inner join creates a new row that contains columns from both tables and adds this new row to the result set.

## PostgreSQL left join

The following statement uses the left join clause to join the basket\_a table with the basket\_b table. In the left join context, the first table is called the left table and the second table is called the right table.

Query Editor    Query History

```
1  SELECT
2      a, fruit_a, b, fruit_b
3  FROM
4      basket_a
5  LEFT JOIN basket_b
6      ON fruit_a = fruit_b;
```



	Data Output	Explain	Messages	Notifications
	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange
3	3	Banana	[null]	[null]
4	4	Cucumber	[null]	[null]

The left join starts selecting data from the left table. It compares values in the fruit\_a column with the values in the fruit\_b column in the basket\_b table.

If these values are equal, the left join creates a new row that contains columns of both tables and adds this new row to the result set. (see the row #1 and #2 in the result set).

In case the values do not equal, the left join also creates a new row that contains columns from both tables and adds it to the result set. However, it fills the columns of the right table (basket\_b) with null. (see the row #3 and #4 in the result set).

## PostgreSQL right join

The right-join is a reversed version of the left join. The right join starts selecting data from the right table. It compares each value in the fruit\_b column of every row in the right table with each value in the fruit\_a column of every row in the fruit\_a table.

If these values are equal, the right join creates a new row that contains columns from both tables.

In case these values are not equal, the right join also creates a new row that contains columns from both tables. However, it fills the columns in the left table with NULL.

The following statement uses the right join to join the basket\_a table with the basket\_b table:

```

Query Editor  Query History
1  SELECT
2      a, fruit_a, b, fruit_b
3  FROM
4      basket_a
5  RIGHT JOIN basket_b ON fruit_a = fruit_b;
6

```

	Data Output	Explain	Messages	Notifications
	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	2	Orange	1	Orange
2	1	Apple	2	Apple
3	[null]	[null]	3	Watermelon
4	[null]	[null]	4	Pear

## PostgreSQL full outer join

The full outer join or full join returns a result set that contains all rows from both left and right tables, with the matching rows from both sides if available. In case there is no match, the columns of the table will be filled with NULL.

Query Editor    Query History

```

1  SELECT
2      a, fruit_a, b, fruit_b
3  FROM
4      basket_a
5  FULL OUTER JOIN basket_b
6      ON fruit_a = fruit_b;

```

	Data Output	Explain	Messages	Notifications
	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange
3	3	Banana	[null]	[null]
4	4	Cucumber	[null]	[null]
5	[null]	[null]	3	Watermelon
6	[null]	[null]	4	Pear

## **EXPERIMENT – 4**

**AIM:** Write a program

- a) To calculate total students enrolled area-wise.
- b) To calculate average marks obtained by students residing in area that starts with "R".

**Theory:**

### **Count Function:**

The COUNT() function is an aggregate function that allows you to get the number of rows that match a specific condition of a query. The COUNT(\*) function returns the number of rows returned by a SELECT statement, including NULL and duplicates.

When you apply the COUNT(\*) function to the entire table, PostgreSQL has to scan the whole table sequentially. If you use the COUNT(\*) function on a big table, the query will be slow. This is related to the PostgreSQL MVCC implementation. Because multiple transactions see different states of data at the same time, there is no direct way for COUNT(\*) function to count across the whole table, therefore PostgreSQL must scan all rows.

### **Like Function:**

The PostgreSQL LIKE operator is used to match text values against a pattern using wildcards. If the search expression can be matched to the pattern expression, the LIKE operator will return true, which is 1.

There are two wildcards used in conjunction with the LIKE operator –

- The percent sign (%)
- The underscore (\_)



The percent sign represents zero, one, or multiple numbers or characters. The underscore represents a single number or character. These symbols can be used in combinations.

If either of these two signs is not used in conjunction with the LIKE clause, then the LIKE acts like the equals operator.

## Main Table:

	student_name character varying (20)	enrollment_no character varying (11)	marks integer	area character varying (20)	branch character varying (30)
1	Ayush	1	100	Faridabad	CSE
2	Rhythm	2	80	Narula	IT
3	Charu	3	75	Faridabad	IT
4	Ishaan	4	99	Narula	MAE
5	Hardik	5	88	Rohini	CSE
6	Himanshu	6	94	Rohini	MAE
7	Aakash	7	50	Narula	CSE
8	Aditi	8	92	Rohini	MAE
9	Piyush	9	40	Faridabad	CSE
10	Anuj	10	94	Rohini	IT

a)

Query Editor   Query History

1   **SELECT** area, **COUNT**(\*) **FROM** Student **GROUP BY** area

	area character varying (20)	count bigint
1	Faridabad	3
2	Rohini	4
3	Narula	3

b)

	avg numeric
1	92.0000000000000000

## **EXPERIMENT – 5**

**AIM:** Write the cursor to increase the marks of student by 10%

### **Theory:**

**Cursor** is a Temporary Memory or Temporary Work Station. It is allocated by Database Server at the Time of Performing DML (Data Manipulation Language) operations on Table by User. Cursors are used to store Database Tables. There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors.

#### **Implicit Cursors:**

Implicit Cursors are also known as Default Cursors of SQL SERVER. These cursors are allocated by SQL SERVER when the user performs DML operations.

#### **Explicit Cursors:**

Explicit Cursors are created by Users whenever the user requires them. Explicit cursors are used for Fetching data from Table in Row-By-Row Manner.

#### **How to create explicit cursor**

##### **Declare Cursor Object.**

DECLARE cursor\_name CURSOR FOR SELECT \* FROM table\_name

##### **Open Cursor Connection.**

Syntax: OPEN cursor\_connectionx:

##### **Fetch Data from cursor.**

There are total 6 methods to access data from cursor. They are as follows:

- **FIRST** is used to fetch only the first row from cursor table.
- **LAST** is used to fetch only last row from cursor table.
- **NEXT** is used to fetch data in forward direction from cursor table.
- **PRIOR** is used to fetch data in backward direction from cursor table.
- **ABSOLUTE** n is used to fetch the exact nth row from cursor table.
- **RELATIVE** n is used to fetch the data in incremental way as well as decremental way.

Syntax: FETCH NEXT/FIRST/LAST/PRIOR/ABSOLUTE n/RELATIVE n FROM cursor\_name

##### **Close cursor connection.**

Syntax: CLOSE cursor\_name

##### **Deallocate cursor memory.**

Syntax: DEALLOCATE cursor\_name

## CODE:

```
Query Editor
1  create or replace function get_film_title(p_year integer)
2  returns text as $$
3  declare
4    titles text default '';
5    rec_films record;
6    incr_marks int default 10;
7    cur_films cursor
8    for select *
9    from student
10   for update;
11 begin
12 open cur_films;
13 loop
14   fetch cur_films into rec_films;
15   exit when not found;
16   Update student
17   set marks=marks+marks*0.10
18   WHERE CURRENT OF cur_films;
19   titles=titles || rec_films.marks;
20 end loop;
21 close cur_films;
22 return titles;
23 end; $$
24 language plpgsql;
25 select get_film_title(2006);
26 select * from student;
```

## OUTPUT:

	student_name character varying (20)	enrollment_no character varying (11)	marks integer	area character varying (20)	branch character varying (30)
1	Ayush	1	100	Faridabad	CSE
2	Rhythm	2	80	Narula	IT
3	Charu	3	75	Faridabad	IT
4	Ishaan	4	99	Narula	MAE
5	Hardik	5	88	Rohini	CSE
6	Himanshu	6	94	Rohini	MAE
7	Aakash	7	50	Narula	CSE
8	Aditi	8	92	Rohini	MAE
9	Piyush	9	40	Faridabad	CSE
10	Anuj	10	94	Rohini	IT

## **EXPERIMENT – 6**

**AIM:** Write a program to create exceptions if enrollment no. is not issued to student by the university and raise the exception explicitly by using raise command.

### **Theory:**

When an error occurs in a block, PostgreSQL will abort the execution of the block and also the surrounding transaction. To recover from the error, user can use the exception clause in between the begin and end block.

**The following illustrates the syntax of the exception clause:**

```
<<label>>
declare
begin
statements;
exception
when condition [or condition...] then
handle_exception;
    [when condition [or condition...] then
handle_exception;]
    [when others then
handle_other_exceptions;
    ]
end;
```

How it works:

1. First, when an error occurs between the begin and exception, PL/pgSQL stops the execution and passes the control to the exception list.
2. Second, PL/pgSQL searches for the first condition that matches the occurring error.
3. Third, if there is a match, the corresponding handle\_exception statements will execute. PL/pgSQL passes the control to the statement after the end keyword.
4. Finally, if no match found, the error propagates out and can be caught by the exception clause of the enclosing block. In case there is no enclosing block with the exception clause, PL/pgSQL will abort the processing.

The condition names can be no\_data\_found in case of a select statement return no rows or too\_many\_rows if the select statement returns more than one row.

It's also possible to specify the error condition by SQLSTATE code. For example, P0002 for no\_data\_found and P0003 for too\_many\_rows.

## CODE:

[Query Editor](#) [Query History](#)

```
1 CREATE OR REPLACE FUNCTION exception_example() RETURNS VOID LANGUAGE PLPGSQL AS $$
2 BEGIN
3 IF ((SELECT COUNT(*) FROM student WHERE enrollment_no ISNULL) > 0) THEN
4 RAISE EXCEPTION 'INVALID ENROLLMENT NUMBER' USING HINT = 'CHECK ENROLLMENT NUMBER';
5 END IF;
6 END; $$
7 SELECT exception_example();
```

## OUTPUT:

	student_name character varying (20)	enrollment_no character varying (11)	marks integer	area character varying (20)	branch character varying (30)
1	Aakash	1	100	Delhi	CSE
2	Harsh	2	90	Punjab	IT
3	Siddharth	3	99	Haryana	ECE
4	Bhavay	4	91	Kashmir	EEE
5	Rohan	5	98	Uttarakhand	MAE
6	Saksham	6	92	Indore	ME
7	Aakash	7	97	Chandigarh	CSE
8	Aditya	8	93	Goa	IT
9	Sarthak	9	96	Noida	ECE
10	Harshit	10	94	Ghaziabad	EEE
11	John	11	99	Kanpur	MAE
12	Alex	[null]	93	Rohini	CSE

[Data Output](#) [Explain](#) [Messages](#) [Notifications](#)

ERROR: INVALID ENROLLMENT NUMBER

HINT: CHECK ENROLLMENT NUMBER

CONTEXT: PL/pgSQL function exception\_example() line 4 at RAISE

SQL state: P0001



## **EXPERIMENT – 7**

### **AIM:** Procedure & Function

- a) Write the procedure to get the average marks of students for branch “CSE”.
- b) Write a function that accepts branch and returns the total no. of students of the branch.

### **Theory:**

**Functions:** These subprograms return a single value; mainly used to compute and return a value.

**Procedures:** These subprograms do not return a value directly; mainly used to perform an action.

Each PL/SQL subprogram has a name, and may also have a parameter list.

**Declarative Part:** It is an optional part. The declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms.

**Executable Part:** This is a mandatory part and contains statements that perform the designated action.

**Exception-handling:** This is an optional part. It contains the code that handles run-time errors.

### **Syntax for Creating a PROCEDURE**

```
CREATE [OR REPLACE] PROCEDURE procedure_name  
[(parameter_name[IN | OUT | IN OUT] type [...])]  
{IS | AS}  
BEGIN  
<procedure_body>  
END procedure_name;
```

### **Creating a FUNCTION**

A function is created using the CREATE FUNCTION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows –

```
CREATE [OR REPLACE] FUNCTION  
function_name [(parameter_name [IN | OUT | IN OUT] type [, ...])]  
RETURN return_datatype  
{IS | AS} BEGIN  
<function_body>  
END [function_name];
```

## CODE:

A. Write the procedure to get the average marks of students for branch “CSE”.

```
1 CREATE OR REPLACE PROCEDURE public.avg_marks_of_students(IN student_branch text, INOUT avg_marks real)
2 LANGUAGE 'plpgsql'
3 AS $BODY$
4 begin
5 select avg(marks) into avg_marks
6 from student
7 where branch = student_branch;
8 end;
9 $BODY$;
```

```
1 CALL avg_marks_of_students('CSE',0)
2
```

## OUTPUT:

avg_marks	
real	
1	76.5

B. Write a function that accepts branch and returns the total no. of students of the branch.

```
1 CREATE OR REPLACE FUNCTION public.get_student_count_by_branch(IN student_branch text, INOUT counts real)
2 LANGUAGE 'plpgsql'
3 AS $BODY$
4 begin
5 select count(student_name) into counts
6 from student
7 where branch = student_branch;
8 end;
9 $BODY$;
```

```
1 select get_student_count_by_branch('CSE',0)
2
```

## OUTPUT:

get_student_count_by_branch	
real	
1	4

## **EXPERIMENT – 8**

### **AIM:** Trigger

- a. Create a trigger on table after inserting a new student into table.
- b. Write a row trigger to insert the existing values of the student table in to a new table when the marks of student is updated.

### **Theory:**

A trigger is a named database object that is associated with a table, and it activates when a particular event (e.g. an insert, update or delete) occurs for the table/views. The statement CREATE TRIGGER creates a new trigger in PostgreSQL.

### **Syntax for creating a trigger**

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
Declaration-statements
BEGIN
Executable-statements
EXCEPTION
Exception-handling-statements
END;
```

## CODE:

A. Create a trigger on table after inserting a new student into table.

```
1 CREATE OR REPLACE FUNCTION rec_insert() RETURNS trigger AS $$
2 BEGIN
3 INSERT INTO student_added VALUES(NEW.s_id,NEW.s_name);
4 RETURN NEW;
5 END;
6 $$
7 LANGUAGE 'plpgsql';
8
```

Query Editor Query History

Data Output Explain Messages Notifications

CREATE FUNCTION

Query returned successfully in 137 msec.

Query Editor

```
1 CREATE TRIGGER
2 ins_same_rec AFTER INSERT ON student FOR EACH ROW
3 EXECUTE PROCEDURE rec_insert();
4
5
```

Query Editor Query History

Data Output Explain Messages Notifications

CREATE TRIGGER

Query returned successfully in 162 msec.

**B. Write a row trigger to insert the existing values of the student table in to a new table when the marks of student is updated.**

```
1 CREATE TRIGGER upd_same_rec
2 AFTER UPDATE
3 ON student_marks
4 FOR EACH ROW
5 EXECUTE PROCEDURE rec_update();
6
```

```
1 CREATE OR REPLACE FUNCTION rec_update() RETURNS trigger LANGUAGE 'plpgsql'
2 BEGIN
3 INSERT INTO student_modify VALUES(Old.s_id,Old.name,Old.marks);
4 RETURN NEW;
5 END;
```

t/postgres@PostgreSQL 12

Query Editor Query History

```
1 update student_marks
2 set marks = 67
3 where s_id = 2;
```

Data Output Explain Messages Notifications

UPDATE 1

Query returned successfully in 111 msec.

t/postgres@PostgreSQL 12

Query Editor Query History

```
1 select * from student_modify;
```

Data Output Explain Messages Notifications

	s_id integer	name text	marks real	
1	1	ruchir	98	
2	2	rohit	88	