

LAB FILE

ADVANCED DBMS LAB

(ETCS-457)

Submitted to:

Ms. Savita Sharma

Assistant Professor

CSE Dept., MAIT

Name: **Sarvesh Nath Tiwari**

Roll No.: 42814802718

Semester: 7th

Group: 7C-7



Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini New Delhi – 110085

S No.	List of experiments	Date	Remarks
1.	a. Learn to install PostgreSql. 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 a. DDL (eg create, alter, drop, rename, truncate), b. DML (eg. Insert, update, delete etc.), c. DCL (eg. Grant, revoke etc.) d. Built-in Functions (eg. Sum, min, max, avg, count, lower, upper, trim, len etc.) e. Indexes and views: Create and Drop		
3.	Write SQL queries on Nested queries and Join.		
4.	PL/SQL a. Write a program to calculate total students enrolled areawise. b. Write a program to calculate average marks obtained by students residing in area that starts with "R"		
5.	Cursor Write the cursor to increase the marks of student by 10%		
6.	Exception Handling Write a program to create exceptions if enrollment no. is not issued to students 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 the branch and returns the total no. of students of the branch.		
8.	Trigger c. Create a trigger on the table after inserting a new student into the table. d. Write a row trigger to insert the existing values of the student table into a new table when the marks of the student is updated.		

Experiment 1(a)

Aim : Learn to install PostgreSql.

Theory: PostgreSQL was developed for UNIX-like platforms, however, it was designed to be portable. It means that PostgreSQL can also run on other platforms such as Mac OS X, Solaris, and Windows.

PostgreSQL is an advanced, enterprise-class, and open-source relational database system. It supports both SQL (relational) and JSON (non-relational) querying. PostgreSQL is used as a primary database for many web applications as well as mobile and analytics applications.

Installation of PostgreSQL:

- We will download PostgreSQL installer for Windows
- First, we need to go to the download page of PostgreSQL installers on the EnterpriseDB.
- Then we will click the download link as shown below figure.
- It will take a few minutes to complete the download.

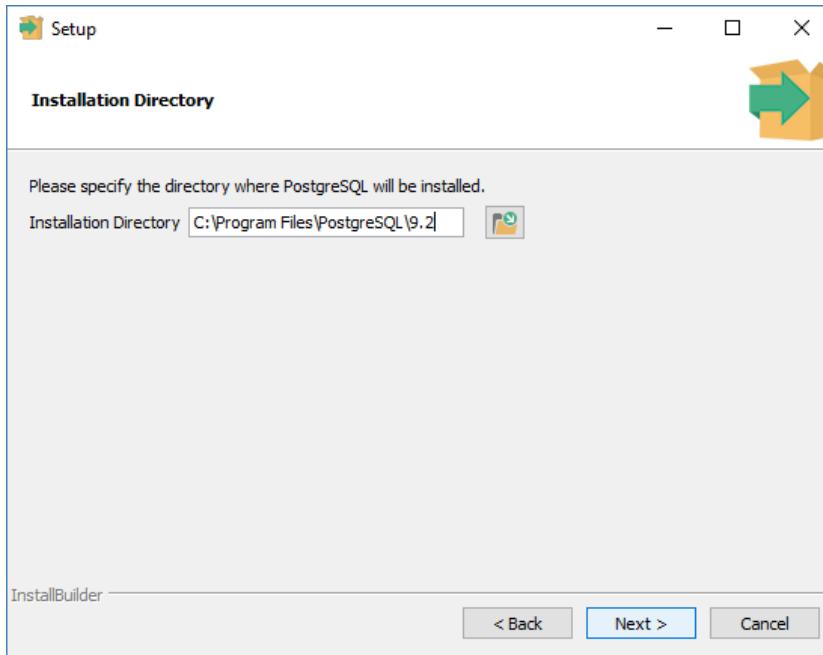
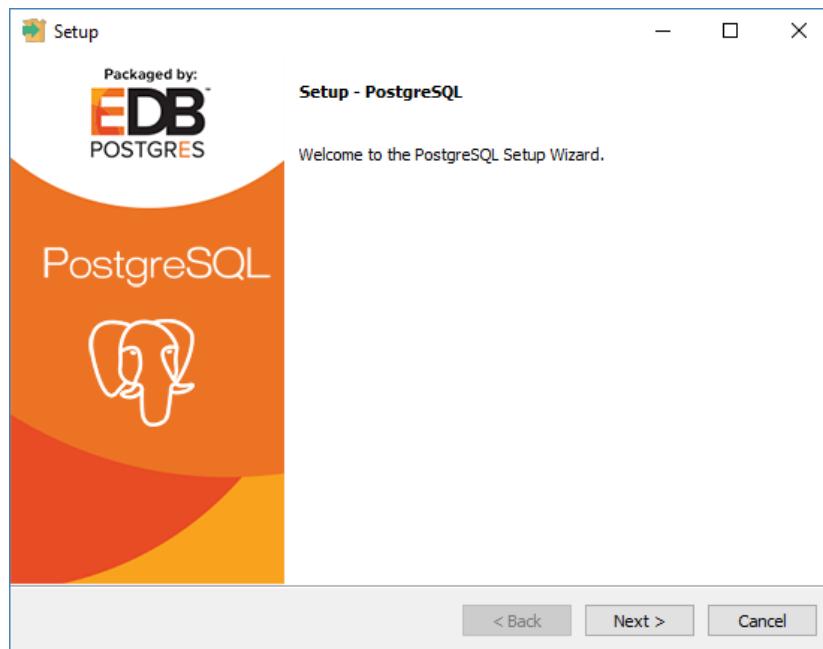
The screenshot shows the 'PostgreSQL Database Download' page on the EnterpriseDB website. The page features a navigation bar with links for EDB, Why EDB?, Products, Services, Support, Resources, Plans, Contact, Sign In, and Downloads (which is highlighted). Below the navigation is a search bar with placeholder text 'Search PostgreSQL products and services'. The main content area is titled 'PostgreSQL Database Download' and displays a table of download links for various PostgreSQL versions across different operating systems. The table has columns for Version, Linux x86-64, Linux x86-32, Mac OS X, Windows x86-64, and Windows x86-32. Each column contains a 'Download' button for each version listed. Versions available include 14, 13.4, 12.8, 11.13, 10.18, 9.6.23, 9.5.25 (Not Supported), 9.4.26 (Not Supported), and 9.3.25 (Not Supported).

Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
14	N/A	N/A	Download	Download	N/A
13.4	N/A	N/A	Download	Download	N/A
12.8	N/A	N/A	Download	Download	N/A
11.13	N/A	N/A	Download	Download	N/A
10.18	Download				
9.6.23	Download				
9.5.25 (Not Supported)	Download				
9.4.26 (Not Supported)	Download				
9.3.25 (Not Supported)	Download				

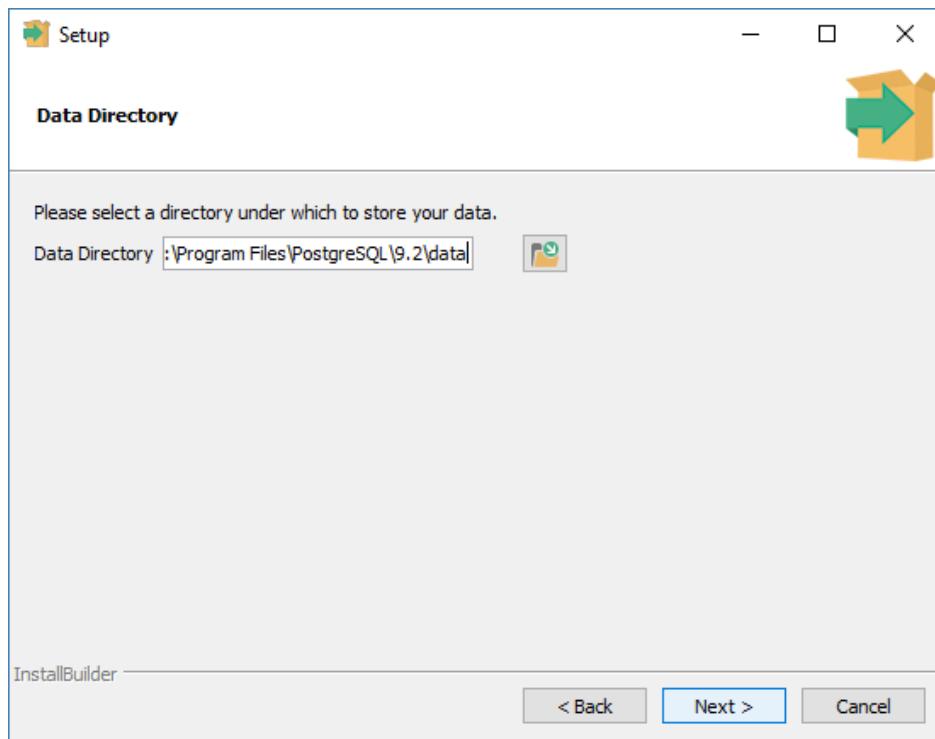
The further installation steps are:

- 1) Double click on the installer file, an installation wizard will appear and guide you through multiple steps where you can choose different options that you would like to have in PostgreSQL

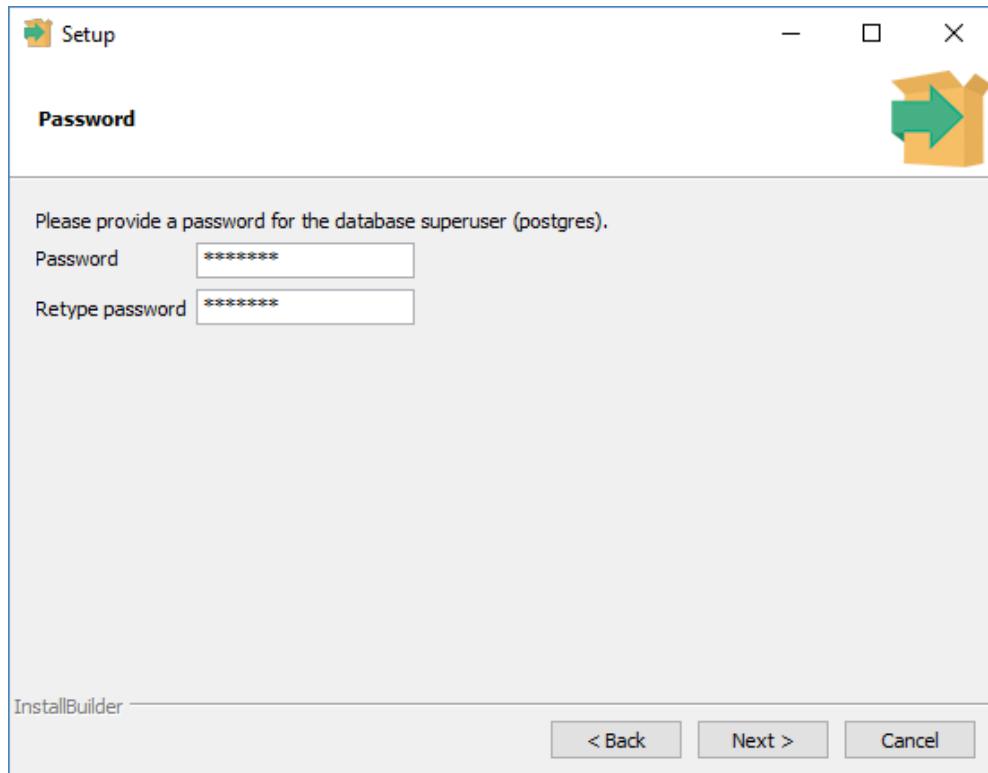
- 2) Click the Next button.



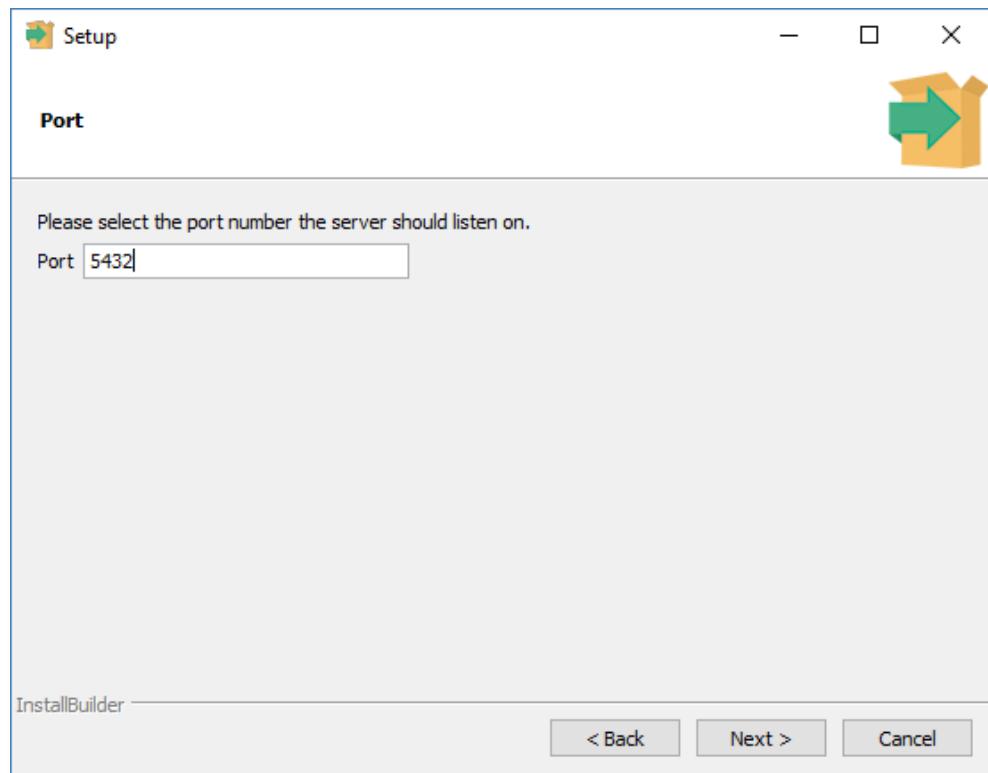
- 3) The next step of the installation process would be to select the directory where data would be stored, by default it is stored under "data" directory.



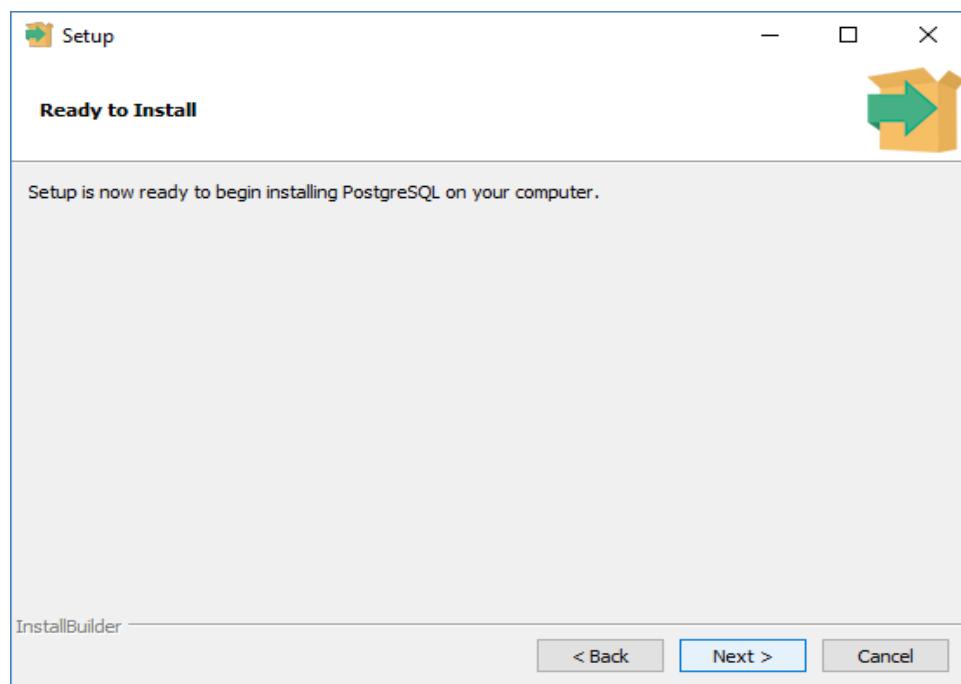
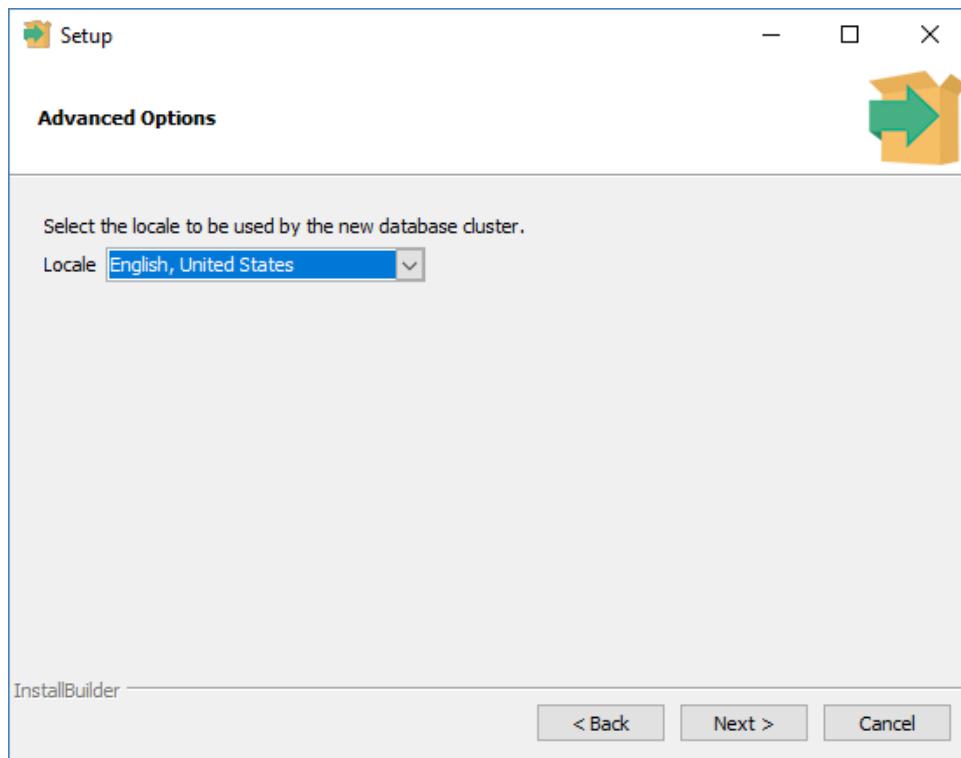
- 4) In the next step, setup asks for password, so you can use your favorite password

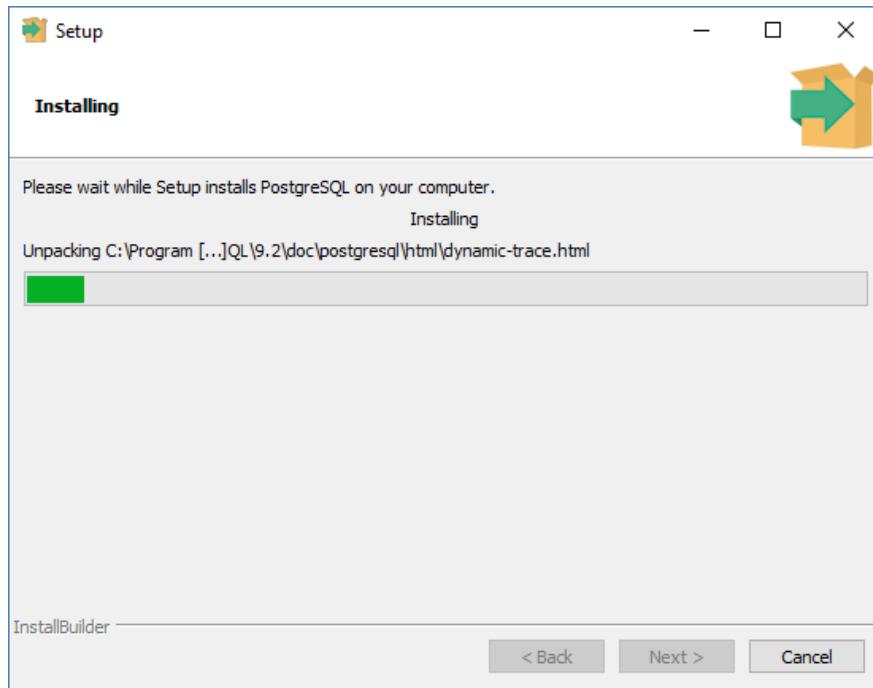


- 5) In the next step, keep the port as default.

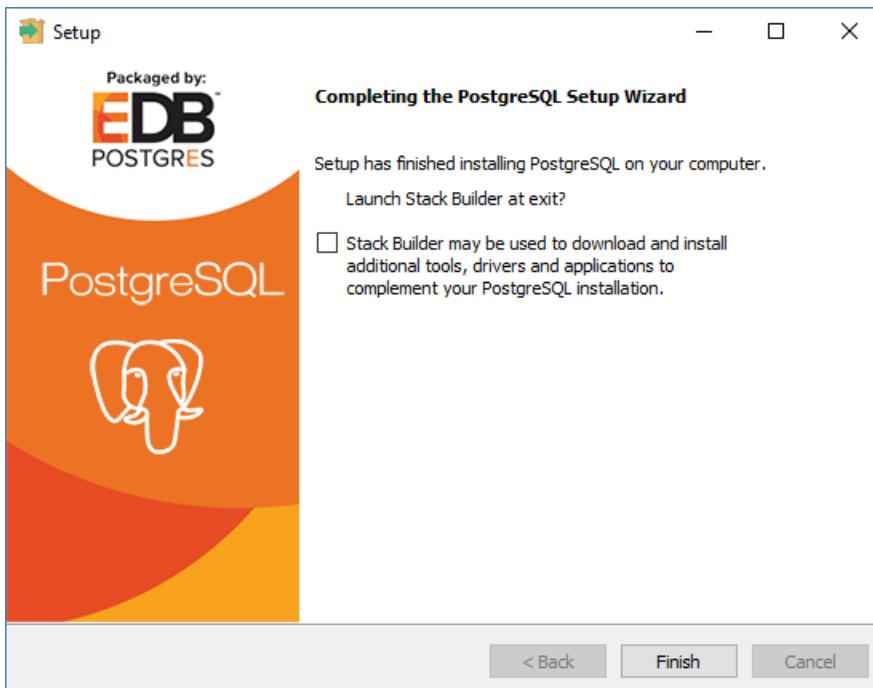


- 6) In the next step, when asked for "Locale", "English, United States" has been selected.



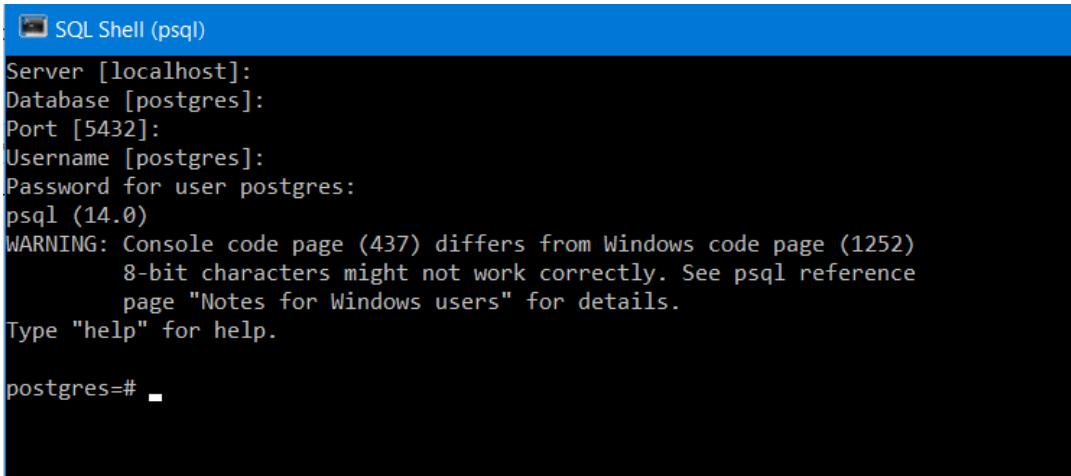


- 7) It takes a while to install PostgreSQL on your system. On completion of the installation process, you will get the following screen. Uncheck the checkbox and click on the Finish button.



Verify the Installation of PostgreSQL:

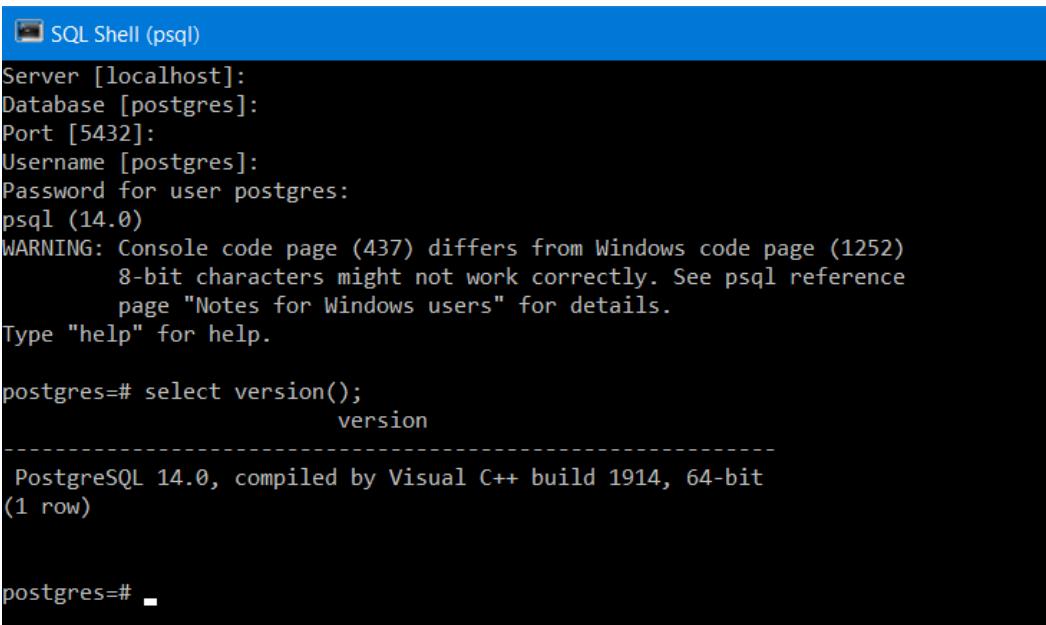
- 1) Click the **psql** (SQL SHELL) application to launch it. The psql command-line program will display. Enter all the necessary information such as the server, database, port, username, and password. To accept the default, you can press Enter. Note that you should provide the password that you entered during installing the PostgreSQL.



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (14.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=#
```

- 2) Enter the command `SELECT version()` and you will see the following output.



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (14.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# select version();
              version
-----
 PostgreSQL 14.0, compiled by Visual C++ build 1914, 64-bit
(1 row)

postgres=#
```

Experiment 1(b)

Aim : Compare different Databases such as PostgreSQL, Oracle, IBM DB2, MySql and Maria DB.

Theory: Let's look at some of the major database management systems currently available in regards to their pros and cons.

1. Oracle

- Closed-source; free version has very limited feature set
- Temporary tables persist across sessions, and must be removed by the user
- Support for four different character/string types: CHAR, VARCHAR2, NCHAR, NVARCHAR2
- Offers both table and row locking
- Extensive and flexible storage customization with commands like tablespace, synonym, and packages
- Extensive backup mechanisms
- Designed to manage tables and databases on a large-scale basis

2. MySQL

- Open-source
- Compatible with a wide range of engines and interfaces; one of the most mature databases on the market
- Lightweight
- One of the most popular database tools; easy to find support online
- Temporary tables are only visible within the current active session, and are removed automatically afterwards.
- Lacks ACID compliance
- Can partition tables via LIST, HASH, RANGE, and SET
- Support for two different character/string types: CHAR and VARCHAR
- Offers only table locking
- Lacks options for table views
- Limited storage customization
- Admin tools are incredibly powerful
- Two backup mechanisms: mysqlhotcopy and mysqldump
- Experiences significant performance degradation at high scale.
- Provides little in the way of performance optimization Issues with reliability
- Limited security compared to some other database systems.
- Designed for transactional workloads, and as such is ill-suited for analytical workloads

3. IBM DB2

- Closed-source. Enterprise version only available for a price
- Schema-based table management
- Does not support XML
- Can only partition tables via sharding
- No in-memory capabilities
- Designed for relational integrity
- More robust table/data management than MySQL
- Materialized table views
- Lacks native character/string support
- Multiple options for disaster recovery, availability, and scalability

4. PostgreSQL

- Open-source
- Adheres well to current SQL standards, and easier to learn as a result
- Large footprint makes it ill-suited for read-heavy operations
- Advanced business/location analytics features
- Rich variety of data and character types
- Fully ACID-compliant
- Designed for reliability and data-integrity; developer-focused
- Full-text search, support for powerful server-side procedural languages
- Full support for advanced SQL features such as table expressions and window functions
- Can efficiently join large numbers of tables
- Replication is poorly-implemented
- Not well-suited for low-concurrency projects

5. Maria DB

- MariaDB is forked out of MySQL. So, there are a lot of similarities between these two databases.
- MariaDB is an open-source database of type RDBMS and is compatible with MySQL for drop-in replacement of the MySQL database.
- Implemented in C, C++.
- Supports complex data transactions.
- Supports OLAP and OLTP systems.
- Relatively easy to scale up when compared to MySQL.
- Can be used for large sized data.
- MariaDB is faster in processing transactions and can also handle over 200,000+ connections which are over and above the capacity of MySQL.

Feature	PostgreSQL	MySQL
<i>Open Source</i>	Completely Open source	Open source, but owned by Oracle and offers commercial versions
<i>ACID Compliance</i>	Complete ACID Compliance	Some versions are compliant
<i>SQL Compliance</i>	Almost fully compliant	Some versions are compliant
<i>Concurrency Support</i>	MVCC implementation supports multiple requests without read locks	Support in some versions.
<i>Security</i>	Secure from ground up with SSL support	SSL support in some versions
<i>NoSQL/JSON Support</i>	Multiple supported features	JSON data support only
<i>Access Methods</i>	Supports all standards	Supports all standards
<i>Replication</i>	Multiple replication technologies available: Single master to one standby Single master to multiple standbys Hot Standby/Streaming Replication Bi-Directional replication Logical log streaming replication	Standard master-standby replication: Single master to one standby Single master to multiple standbys Single master to one standby to one or more standbys Circular replication (A to B to C and back to A) Master to master
<i>Materialized Views</i>	Supported	Not supported
<i>Temporary Tables</i>	Supported	Supported
<i>GeoSpatial Data</i>	Supported	Supported
<i>Programming Languages</i>	Supported	Not supported
<i>Extensible Type System</i>	Supported	Not supported

Experiment 2

Aim : Consider Schema: Student(student_name, enrollment_no, marks, area, branch)

Write SQL queries on

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

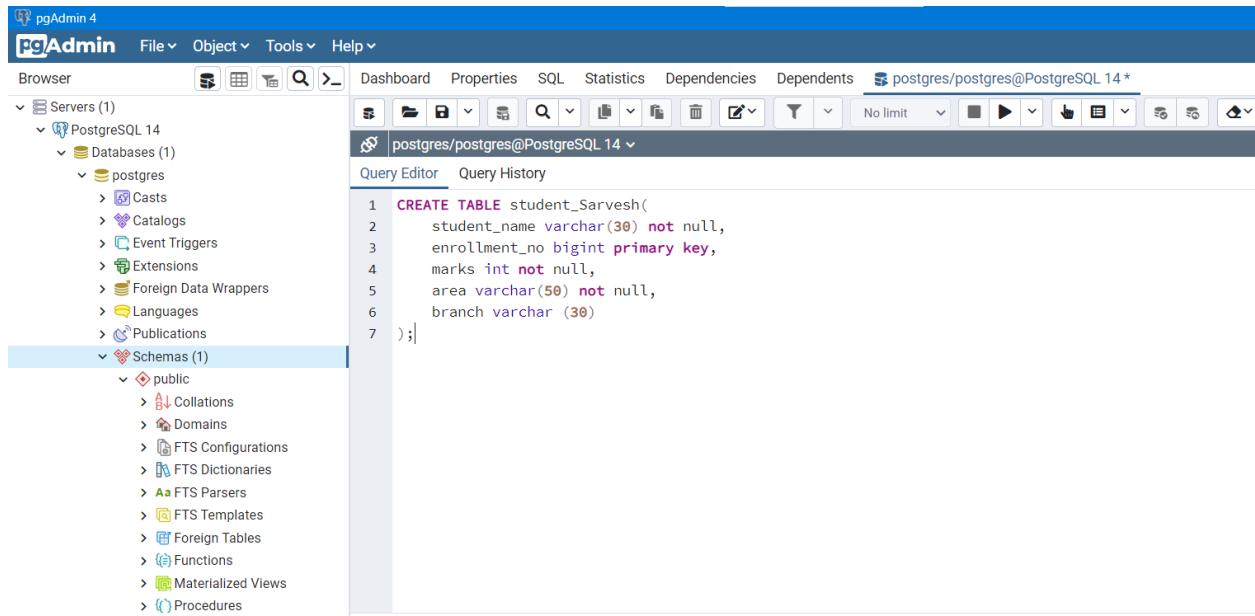
Theory:

- **Data Definition Language(DDL):** It consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

Let us build the table “Student”:

CREATE - It is used to create the database or its objects (like table, index, function, views, store procedure and triggers).

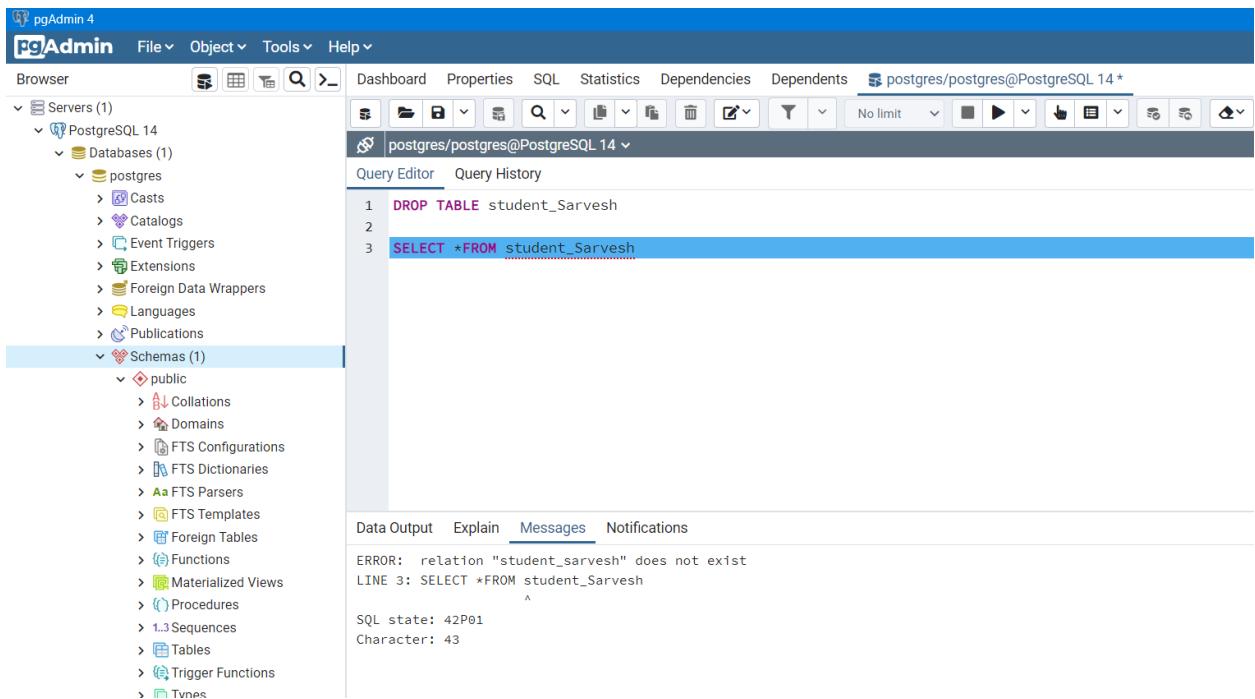
```
create table student_Sarvesh(  
    student_name varchar(30) not null,  
    enrollment_no bigint primary key,  
    marks int not null,  
    area varchar(50) not null,  
    branch varchar (30)  
);
```



The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays a tree structure of database objects under 'Servers (1) > PostgreSQL 14 > postres > Schemas (1) > public'. In the central 'Query Editor' window, the following SQL code is visible:

```
1 CREATE TABLE student_Sarvesh(
2     student_name varchar(30) not null,
3     enrollment_no bigint primary key,
4     marks int not null,
5     area varchar(50) not null,
6     branch varchar (30)
7 );
```

DROP- The DROP command is a type of SQL DDL command that is used to delete an existing database or an object within a database.



The screenshot shows the pgAdmin 4 interface. The 'Browser' pane is identical to the previous screenshot. In the 'Query Editor' window, the following SQL code is visible:

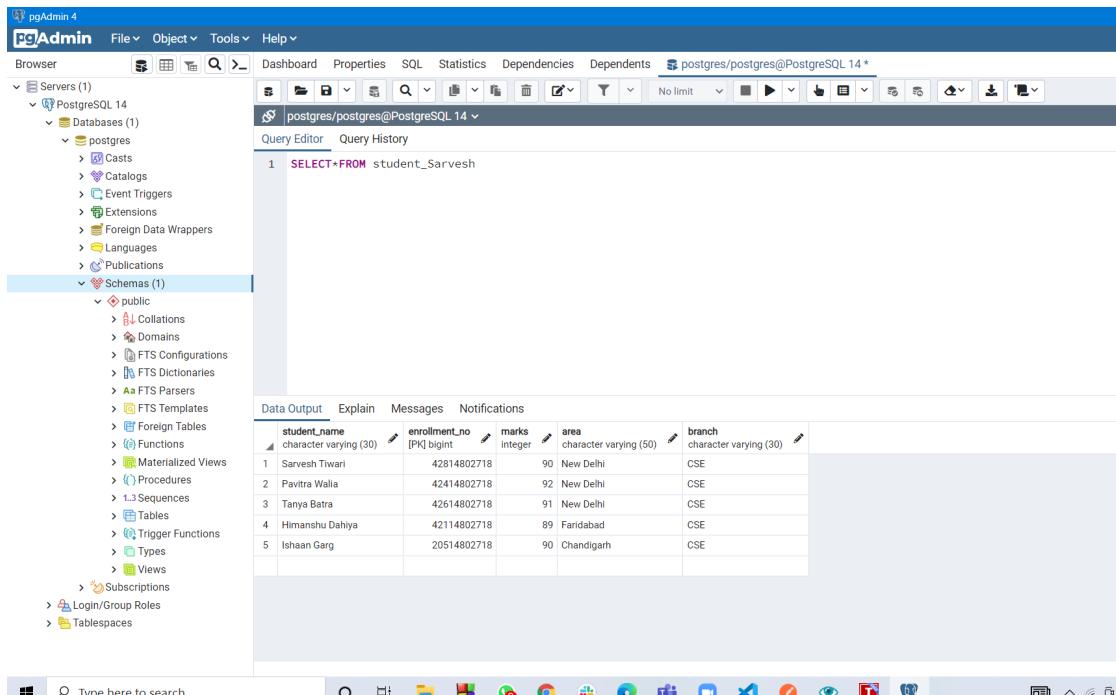
```
1 DROP TABLE student_Sarvesh
2
3 SELECT *FROM student_Sarvesh
```

Below the query editor, the 'Messages' tab is selected, showing the following error message:

```
ERROR: relation "student_sarvesh" does not exist
LINE 3: SELECT *FROM student_Sarvesh
          ^
SQL state: 42P01
Character: 43
```

TRUNCATE : The TRUNCATE command in SQL DDL is used to remove all the records from a table. Let's insert a few records in the Books table:

Before TRUNCATE:



The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure under 'Servers (1)'. The 'Schemas (1)' section is selected, showing the 'public' schema with various objects like Collations, Domains, FTS Configurations, etc. The main area shows a query editor with the following SQL query:

```
1 SELECT * FROM student_Sarvesh;
```

The results pane displays the data from the 'student_Sarvesh' table:

	student_name	enrollment_no	marks	area	branch
1	Sarvesh Tiwari	42814802718	90	New Delhi	CSE
2	Pavitra Walla	42414802718	92	New Delhi	CSE
3	Tanya Batra	42614802718	91	New Delhi	CSE
4	Himanshu Dahiya	42114802718	89	Faridabad	CSE
5	Ishaan Garg	20514802718	90	Chandigarh	CSE

After TRUNCATE:

pgAdmin 4

File Object Tools Help

Browser Dashboard Properties SQL Statistics Dependencies Dependents

Servers (1) PostgreSQL 14 Databases (1) postgres Schemas (1) public

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Publications

Collations Domains FTS Configurations FTS Dictionaries FTS Parsers FTS Templates Foreign Tables Functions Materialized Views Procedures

Query Editor Query History

1 `truncate table student_Sarvesh`

Data Output Explain Messages Notifications

TRUNCATE TABLE

Query returned successfully in 49 msec.

This screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure under 'Servers (1)'. The 'Databases' section shows 'postgres'. The 'Schemas' section is expanded, showing 'public' which contains various objects like Collations, Domains, and FTS configurations. The 'Query Editor' tab is active, containing the command 'truncate table student_Sarvesh'. The 'Messages' tab shows the result: 'TRUNCATE TABLE' and 'Query returned successfully in 49 msec.'

pgAdmin 4

File Object Tools Help

Browser Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres@PostgreSQL 14

Servers (1) PostgreSQL 14 Databases (1) postgres Schemas (1) public

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Publications

Collations Domains FTS Configurations FTS Dictionaries FTS Parsers FTS Templates Foreign Tables Functions Materialized Views Procedures

Query Editor Query History

1 `SELECT *FROM student_Sarvesh`

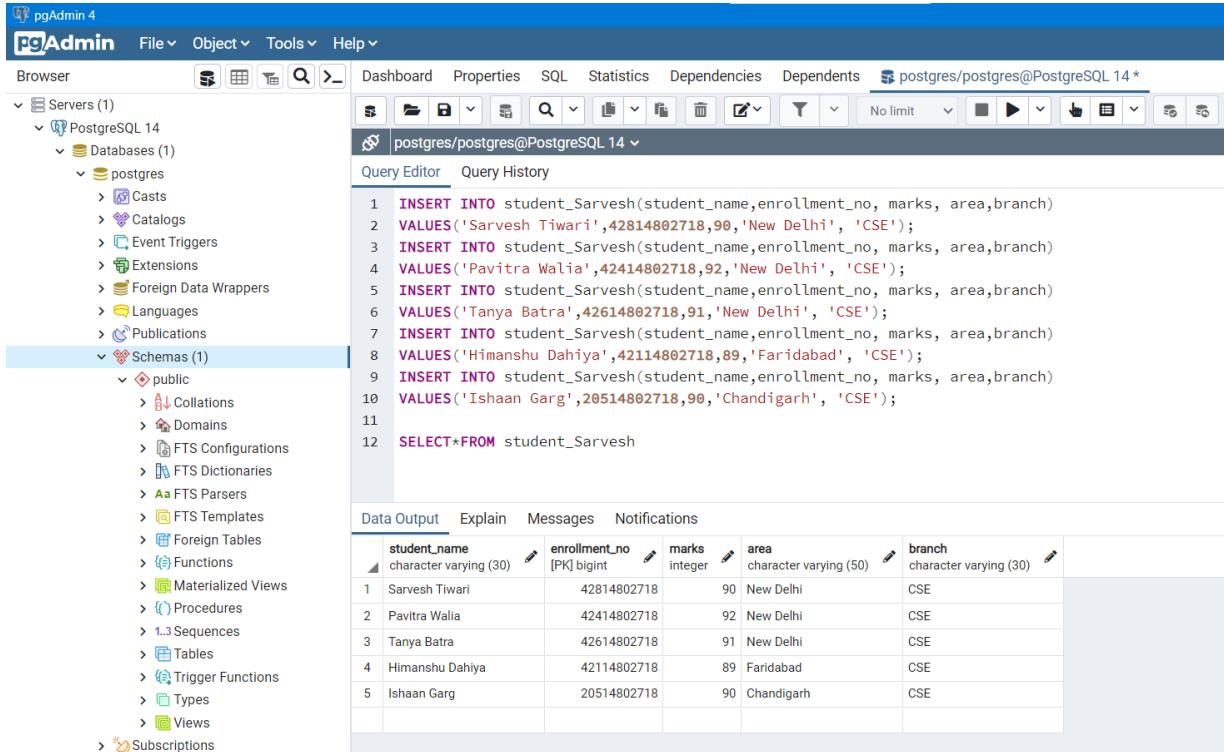
Data Output Explain Messages Notifications

student_name	enrollment_no	marks	area	branch

This screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure under 'Servers (1)'. The 'Databases' section shows 'postgres'. The 'Schemas' section is expanded, showing 'public'. The 'Query Editor' tab is active, containing the command 'SELECT *FROM student_Sarvesh'. The 'Data Output' tab is selected, showing a table with columns: student_name, enrollment_no, marks, area, and branch. The table currently has one empty row.

- **Data Manipulation Language (DML):** The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database.

INSERT : The INSERT statement is a SQL query used to insert data into the row of a table.



The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure under 'Servers (1)'. The 'Schemas (1)' section is selected, showing various schema components like public, Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Procedures, Sequences, Tables, Trigger Functions, Types, and Views. The main area is the 'Query Editor' tab, which contains the following SQL code:

```

1  INSERT INTO student_Sarvesh(student_name,enrollment_no, marks, area,branch)
2  VALUES('Sarvesh Tiwari',42814802718,90,'New Delhi', 'CSE');
3  INSERT INTO student_Sarvesh(student_name,enrollment_no, marks, area,branch)
4  VALUES('Pavitra Walia',42414802718,92,'New Delhi', 'CSE');
5  INSERT INTO student_Sarvesh(student_name,enrollment_no, marks, area,branch)
6  VALUES('Tanya Batra',42614802718,91,'New Delhi', 'CSE');
7  INSERT INTO student_Sarvesh(student_name,enrollment_no, marks, area,branch)
8  VALUES('Himanshu Dahiya',42114802718,89,'Faridabad', 'CSE');
9  INSERT INTO student_Sarvesh(student_name,enrollment_no, marks, area,branch)
10 VALUES('Ishaan Garg',20514802718,90,'Chandigarh', 'CSE');
11
12 SELECT*FROM student_Sarvesh

```

Below the code, the 'Data Output' tab is active, showing the results of the query in a table:

	student_name	enrollment_no	marks	area	branch
1	Sarvesh Tiwari	42814802718	90	New Delhi	CSE
2	Pavitra Walia	42414802718	92	New Delhi	CSE
3	Tanya Batra	42614802718	91	New Delhi	CSE
4	Himanshu Dahiya	42114802718	89	Faridabad	CSE
5	Ishaan Garg	20514802718	90	Chandigarh	CSE

DELETE: It is used to remove one or more rows from a table.

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays a tree structure of database objects under 'Servers (1)'. Under 'Databases (1)', there is a 'postgres' entry which contains various objects like Casts, Catalogs, Event Triggers, Extensions, Foreign Data Wrappers, Languages, Publications, and Schemas (1). The 'Schemas (1)' node is selected. It contains a single item: 'public'. Under 'public', there are entries for Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Procedures, and Sequences. On the right, the 'Query Editor' tab is active, showing the following SQL code:

```

1 DELETE FROM student_Sarvesh
2 WHERE enrollment_no = 20514802718;
3
4 SELECT * FROM student_Sarvesh

```

Below the Query Editor, the 'Data Output' tab is selected, displaying a table with the following data:

	student_name	enrollment_no	marks	area	branch
1	Sarvesh Tiwari	42814802718	90	New Delhi	CSE
2	Pavitra Walia	42414802718	92	New Delhi	CSE
3	Tanya Batra	42614802718	91	New Delhi	CSE
4	Himanshu Dahlya	42114802718	89	Faridabad	CSE

- **Data Control Language(DCL):** It includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

GRANT: It is used to give user access privileges to a database.

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Servers (1) > PostgreSQL 14 > Databases (1) > postgres > Schemas (1) > public', the 'privileges' icon is highlighted. The main window displays a query editor with the following SQL command:

```
1 GRANT SELECT, INSERT, UPDATE, DELETE ON student_Sarvesh TO Sarvesh
```

The status bar at the bottom indicates 'Query returned successfully in 65 msec.'

REVOKE: It is used to take back permissions from the user.

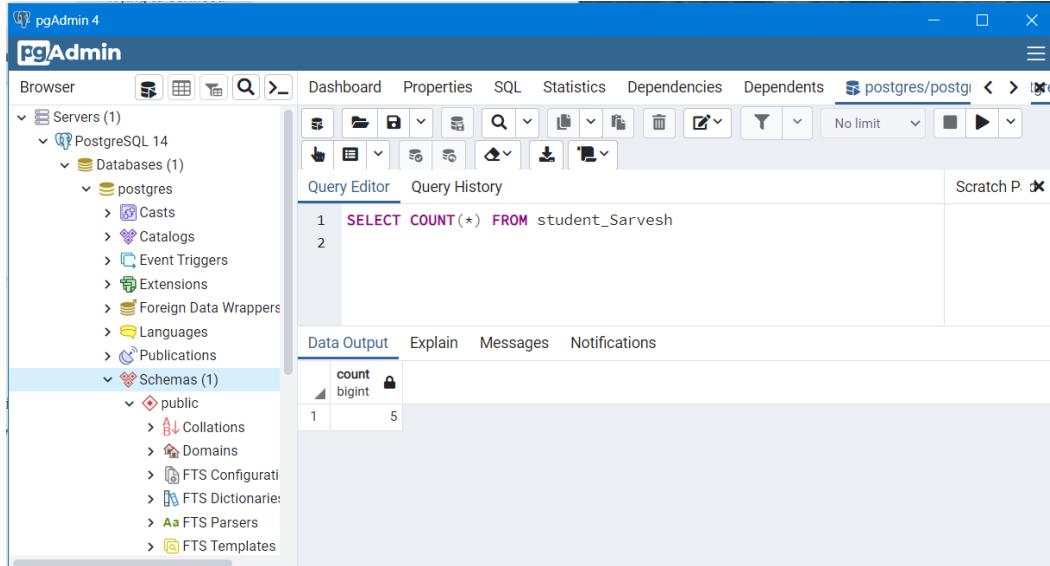
The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Servers (1) > PostgreSQL 14 > Databases (1) > postgres > Schemas (1) > public', the 'privileges' icon is highlighted. The main window displays a query editor with the following SQL command:

```
1 REVOKE SELECT, INSERT, UPDATE, DELETE ON student_Sarvesh FROM Sarvesh
```

The status bar at the bottom indicates 'Query returned successfully in 65 msec.'

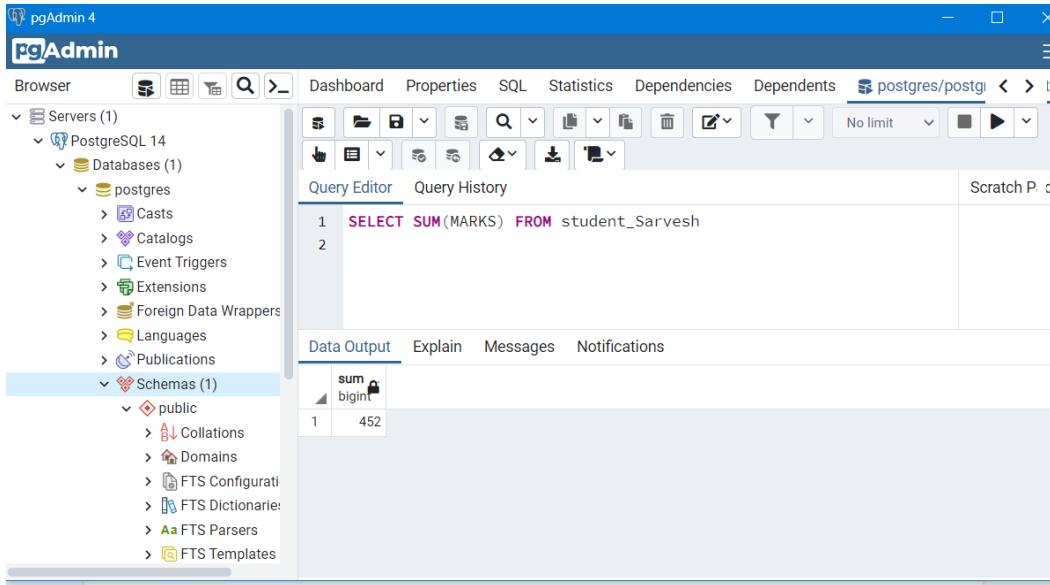
Built-in Functions of SQL : Built-in functions are predefined functions that give you convenient ways to manipulate your data. Examples are count, sum, avg, min, max, etc.

COUNT() : This function returns the number of rows that matches a specified criterion.



The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Servers (1) > PostgreSQL 14 > Databases (1) > postgres > Schemas (1) > public', the 'student_Sarvesh' schema is selected. The 'Query Editor' tab is active, displaying the query: 'SELECT COUNT(*) FROM student_Sarvesh'. The 'Data Output' tab shows the result: count | bigint | 1 | 5. The 'Explain' tab is also visible.

SUM() : This function returns the total sum of a numeric column.



The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Servers (1) > PostgreSQL 14 > Databases (1) > postgres > Schemas (1) > public', the 'student_Sarvesh' schema is selected. The 'Query Editor' tab is active, displaying the query: 'SELECT SUM(MARKS) FROM student_Sarvesh'. The 'Data Output' tab shows the result: sum | bigint | 1 | 452. The 'Explain' tab is also visible.

MAX() : This function returns the largest value of the selected column.

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Servers (1) > PostgreSQL 14 > Databases (1) > postgres > Schemas (1) > public', the 'Schemas (1)' node is selected. In the main pane, the 'Query Editor' tab is active with the following SQL query:

```
1 SELECT MAX(MARKS) FROM student_Sarvesh
```

The 'Data Output' tab shows the result of the query:

	max
1	92

MIN() : This function returns the smallest value of the selected column.

The screenshot shows the pgAdmin 4 interface. The left sidebar is identical to the previous one. In the main pane, the 'Query Editor' tab is active with the following SQL query:

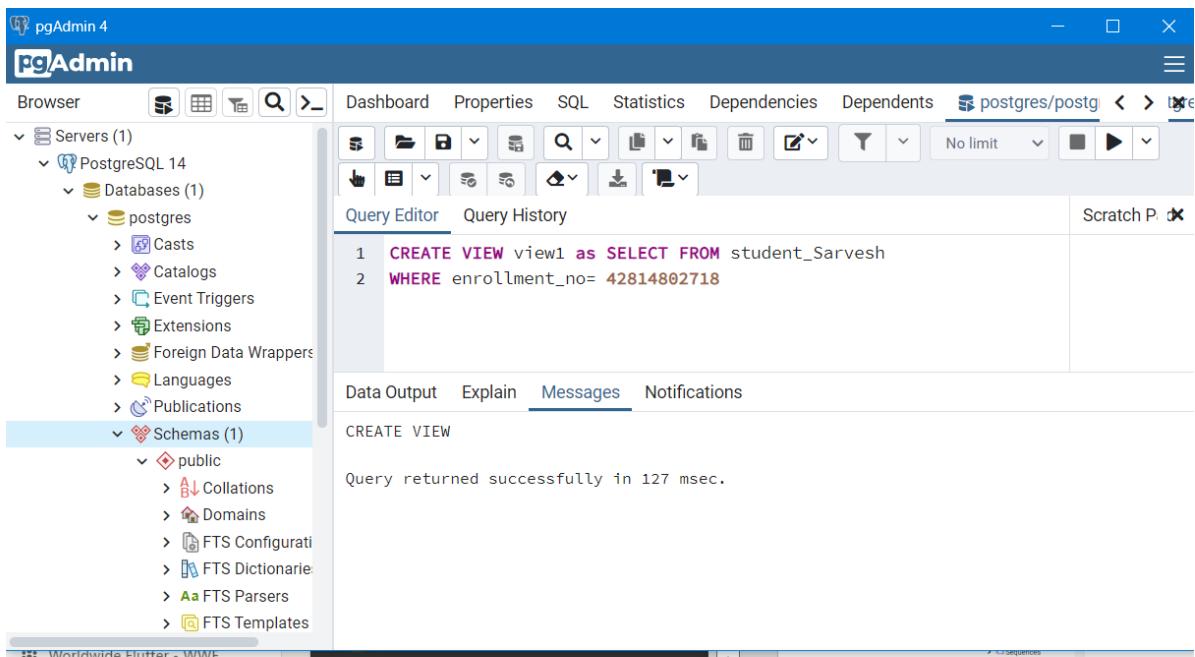
```
1 SELECT MIN(MARKS) FROM student_Sarvesh
```

The 'Data Output' tab shows the result of the query:

	min
1	89

INDEXES and VIEWS :

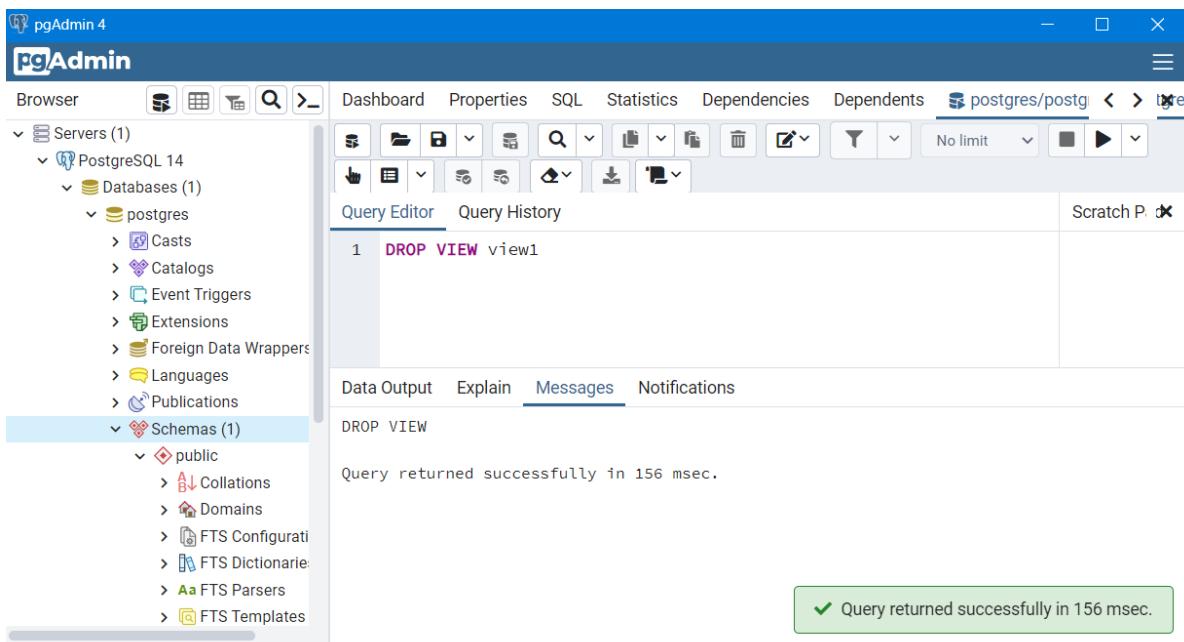
A **VIEW** is simply any SELECT query that has been given a name and saved in the database. For this reason, a view is sometimes called a named query or a stored query.



The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure under 'Servers (1)'. In the 'Databases (1)' section, 'postgres' is selected. Under 'Schemas (1)', 'public' is selected. The 'Query Editor' tab is active, showing the SQL code for creating a view:

```
1 CREATE VIEW view1 AS SELECT * FROM student_Sarvesh
2 WHERE enrollment_no = 42814802718
```

The 'Messages' tab shows the result of the query: "Query returned successfully in 127 msec."



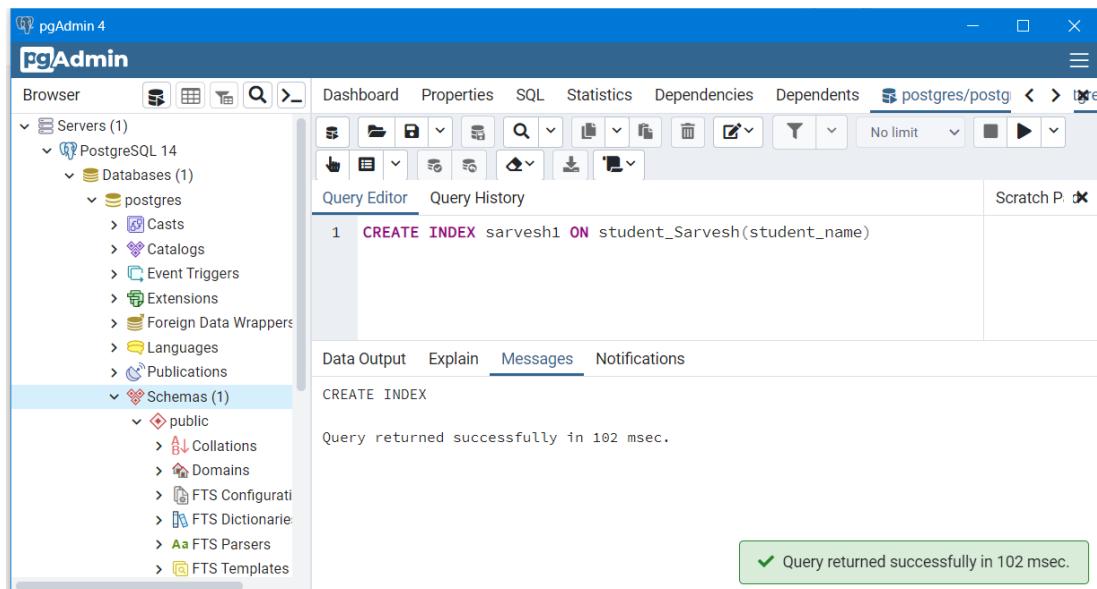
The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure under 'Servers (1)'. In the 'Databases (1)' section, 'postgres' is selected. Under 'Schemas (1)', 'public' is selected. The 'Query Editor' tab is active, showing the SQL code for dropping a view:

```
1 DROP VIEW view1
```

The 'Messages' tab shows the result of the query: "Query returned successfully in 156 msec." A green message box at the bottom right indicates "Query returned successfully in 156 msec."

INDEX is a data structure that the database uses to find records within a table more quickly. Indexes are built on one or more columns of a table; each index maintains a list of values within that field that are sorted in ascending or descending order. Rather than sorting records on the

field or fields during query execution, the system can simply access the rows in order of the index.



The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Servers > PostgreSQL 14 > Databases > postgres > Schemas > public', the 'CREATE INDEX' query is being run in the Query Editor. The query is:

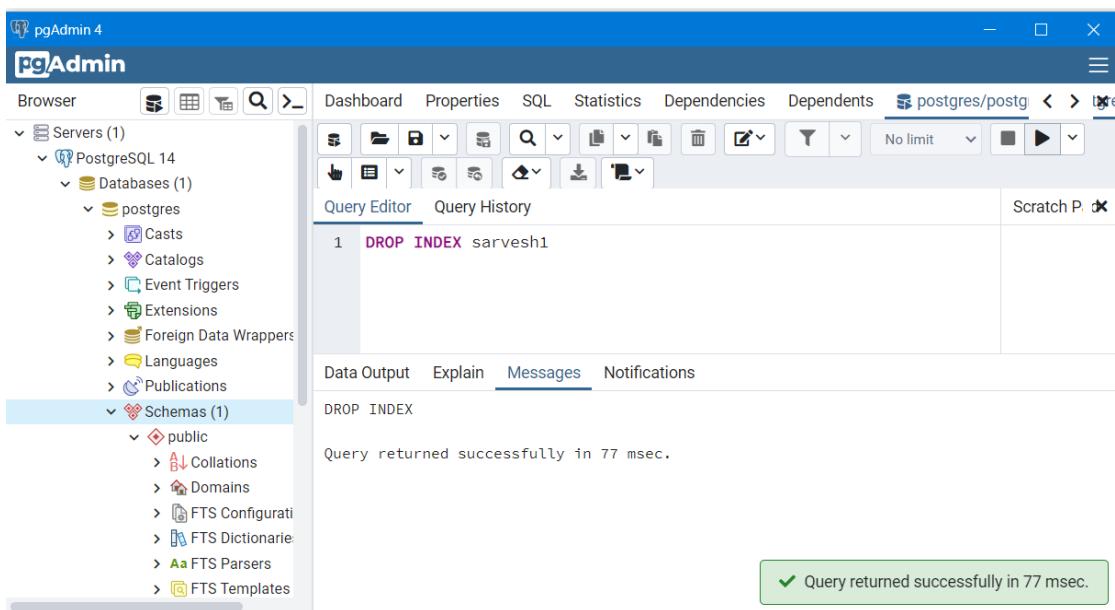
```
1 CREATE INDEX sarvesh1 ON student_Sarvesh(student_name)
```

The Messages tab shows the result:

CREATE INDEX

Query returned successfully in 102 msec.

A green message box at the bottom right indicates: ✓ Query returned successfully in 102 msec.



The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Servers > PostgreSQL 14 > Databases > postgres > Schemas > public', the 'DROP INDEX' query is being run in the Query Editor. The query is:

```
1 DROP INDEX sarvesh1
```

The Messages tab shows the result:

DROP INDEX

Query returned successfully in 77 msec.

A green message box at the bottom right indicates: ✓ Query returned successfully in 77 msec.

Experiment 3

Aim :Write SQL queries on Nested queries and Join.

Theory:

1) Nested Queries:

There are mainly two types of nested queries:

- **Independent Nested Queries:** In independent nested queries, query execution starts from innermost query to outermost queries. The execution of the inner query is independent of the outer query, but the result of the inner query is used in execution of the outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.
- **Co-related Nested Queries:** In correlated nested queries, the output of the inner query depends on the row which is being currently executed in the outer query.

Finding COURSE_ID for COURSE='ADBMS' or 'STQA'

Select COURSE_ID from STUDENT_COURSE

where COURSE = 'ADBMS' or COURSE = 'STQA'

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure under 'Servers (1)'. The 'Browser' section shows 'PostgreSQL 14' with 'Databases (1)' containing 'postgres'. Inside 'postgres', there are various objects like Casts, Catalogs, Event Triggers, Extensions, Foreign Data Wrappers, Languages, Publications, and Schemas (1). The 'public' schema contains Collations, Domains, FTS Configuration, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, and Functions. The 'Query Editor' tab is active, displaying the following SQL code:

```
1 SELECT course_id FROM student_course
2 WHERE course='ADBMS' or course='STQA'
3
```

The 'Data Output' tab shows the results of the query:

course_id
1
2
1
2

- IN:

SELECT enrollment_no FROM student_course

WHERE course_id IN

(SELECT course_id FROM student_course WHERE course='ADBMS' or course='STQA');

```

File Object Tools Help
QL 14
ases (1)
stgres
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Procedures
Sequences
Tables (1)
Trigger Functions
Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres@PostgreSQL 14*
Query Editor Query History Scratch P.
1 SELECT enrollment_no FROM student_course
2 WHERE course_id IN
3 (SELECT course_id FROM student_course WHERE course='ADBMS' or course='STQA');
4
Data Output Explain Messages Notifications
enrollment_no [PK] bigint
1 42814802718
2 42414802718
3 42114802718
4 20514802718

```

Note: If we want to find out names of STUDENTS who have either enrolled in ‘ADBMS’ or ‘STQA’, it can be done as:

SELECT student_name from student_Sarvesh where enrollment_no IN

(Select enrollment_no from student_course where course_id IN

(SELECT course_id from student_course where course='ADBMS' or C_NAME='STQA'));

```

1 SELECT student_name from student_Sarvesh where enrollment_no IN
2 (Select enrollment_no from student_course where course_id IN
3 (SELECT course_id from student_course where course='ADBMS' or course='STQA')));

```

student_name
Sarvesh Tiwari
Pavitra Wala
Himanshu Dahiya
Ishaan Garg

- **NOT IN:** If we want to find out **enrollment_nos** of **STUDENTs** who have neither enrolled in ‘ADBMS’ nor in ‘STQA’, it can be done as:

SELECT student_name from student_Sarvesh where enrollment_no NOT IN

(Select enrollment_no from student_course where course_id IN

(SELECT course_id from student_course where course='ADBMS' or course='STQA'));

```

1 SELECT student_name from student_Sarvesh where enrollment_no NOT IN
2 (Select enrollment_no from student_course where course_id IN
3 (SELECT course_id from student_course where course='ADBMS' or course='STQA')));

```

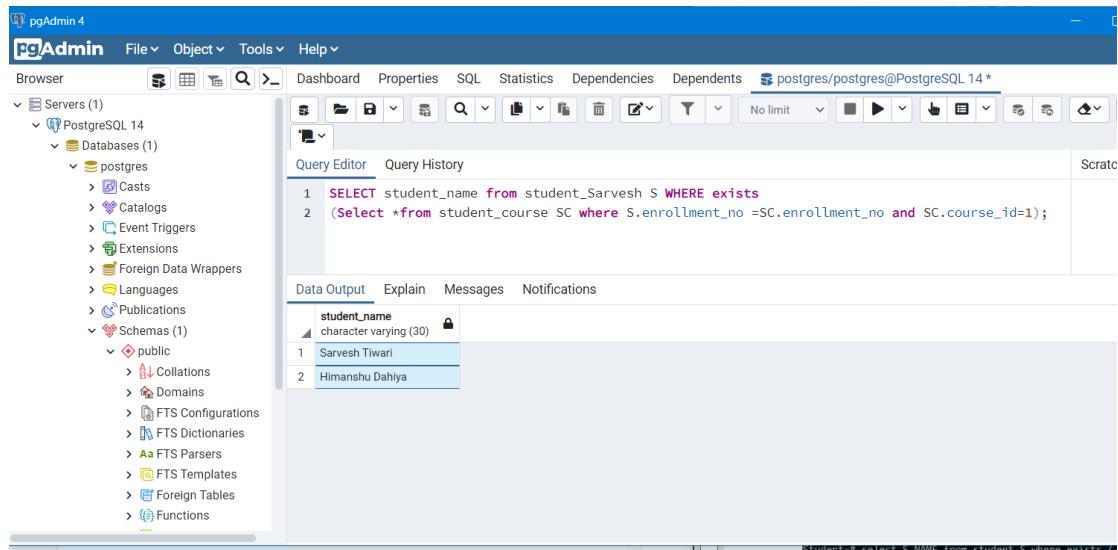
student_name
Tanya Batra

- **EXISTS:**

If we want to find out **STUDENT_NAME** of STUDENTS who are enrolled in **COURSE_ID** '1', it can be done with the help of co-related nested query as:

SELECT student_name from student_Sarvesh S WHERE exists

*(Select *from student_course SC where S.enrollment_no =SC.enrollment_no and SC.course_id='C1');*



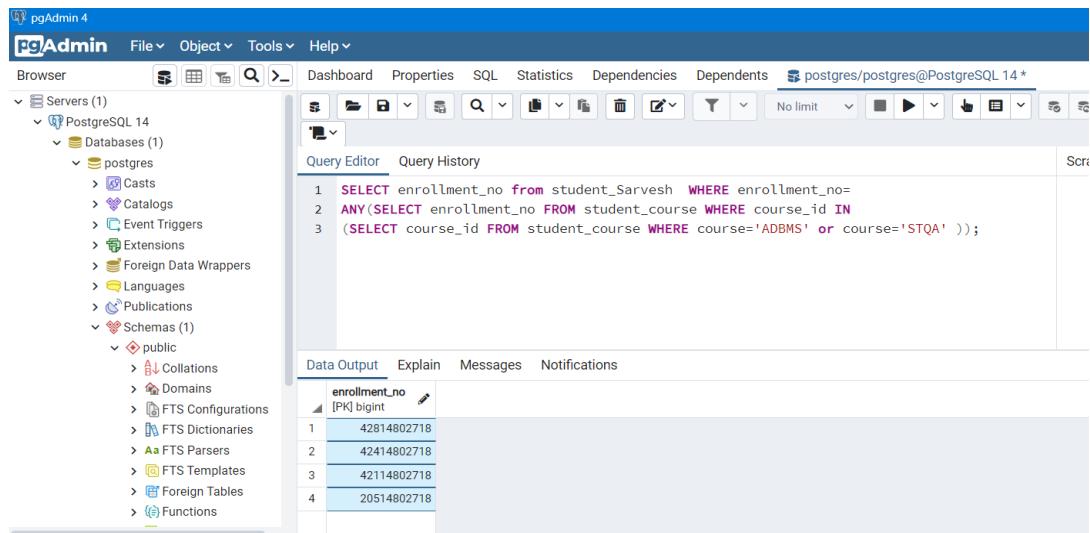
```

pgAdmin 4
File Object Tools Help
Browser Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres@PostgreSQL 14*
Servers (1)
  PostgreSQL 14
    Databases (1)
      postgres
        Casts
        Catalogs
        Event Triggers
        Extensions
        Foreign Data Wrappers
        Languages
        Publications
        Schemas (1)
          public
            Collations
            Domains
            FTS Configurations
            FTS Dictionaries
            FTS Parsers
            FTS Templates
            Foreign Tables
            Functions
Query Editor Query History
1 SELECT student_name from student_Sarvesh S WHERE exists
2 (Select *from student_course SC where S.enrollment_no =SC.enrollment_no and SC.course_id=1);

Data Output Explain Messages Notifications
student_name
character varying(30)
1 Sarvesh Tiwari
2 Himanshu Dahiya

```

- **ANY:**



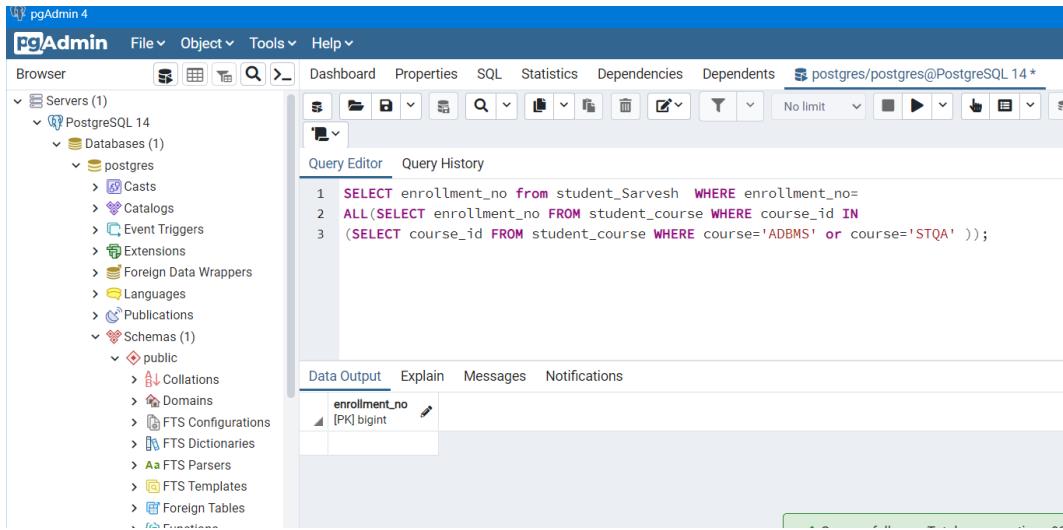
```

pgAdmin 4
File Object Tools Help
Browser Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres@PostgreSQL 14*
Servers (1)
  PostgreSQL 14
    Databases (1)
      postgres
        Casts
        Catalogs
        Event Triggers
        Extensions
        Foreign Data Wrappers
        Languages
        Publications
        Schemas (1)
          public
            Collations
            Domains
            FTS Configurations
            FTS Dictionaries
            FTS Parsers
            FTS Templates
            Foreign Tables
            Functions
Query Editor Query History
1 SELECT enrollment_no from student_Sarvesh WHERE enrollment_no=
2 ANY(SELECT enrollment_no FROM student_course WHERE course_id IN
3 (SELECT course_id FROM student_course WHERE course='ADBMS' or course='STQA' ));

Data Output Explain Messages Notifications
enrollment_no
[PK] bigint
1 42814802718
2 42414802718
3 42114802718
4 20514802718

```

- **ALL:**



2) JOIN: A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

- CROSS JOIN
- INNER JOIN
- OUTER JOIN
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN

CROSS JOIN: The CROSS JOIN is used to generate a paired combination of each row of the first table with each row of the second table. This join type is also known as cartesian join.

INNER JOIN: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

LEFT OUTER JOIN: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN

RIGHT OUTER JOIN: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. The rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

FULL OUTER JOIN: FULL JOIN creates the result-set by combining the result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain *NULL* values.

- **CROSS JOIN**

The screenshot shows the pgAdmin 4 interface with a query editor window. The query is:

```
1 SELECT * FROM student_sarvesh CROSS JOIN student_course;
```

The results are displayed in a table with two sets of columns. The first set of columns (from the left) corresponds to the `student_sarvesh` table, and the second set corresponds to the `student_course` table. The table has 20 rows, each containing a combination of one row from each table.

	student_name	enrollment_no	marks	area	branch	student_name	enrollment_no	course_id	course
	character varying (30)	bigint	integer	character varying (50)	character varying (30)	character varying (30)	bigint	integer	character varying (30)
1	Sanvesh Tiwari	42814802718	90	New Delhi	CSE	Sarvesh Tiwari	42814802718	1	ADMS
2	Pavitra Wallia	42414802718	92	New Delhi	CSE	Sarvesh Tiwari	42814802718	1	ADMS
3	Tanya Batra	42614802718	91	New Delhi	CSE	Sarvesh Tiwari	42814802718	1	ADMS
4	Himanshu Dahiya	42114802718	89	Faridabad	CSE	Sarvesh Tiwari	42814802718	1	ADMS
5	Ishaan Garg	20514802718	90	Chandigarh	CSE	Sarvesh Tiwari	42814802718	1	ADMS
6	Sanvesh Tiwari	42814802718	90	New Delhi	CSE	Pavitra Wallia	42414802718	2	STQA
7	Pavitra Wallia	42414802718	92	New Delhi	CSE	Pavitra Wallia	42414802718	2	STQA
8	Tanya Batra	42614802718	91	New Delhi	CSE	Pavitra Wallia	42414802718	2	STQA
9	Himanshu Dahiya	42114802718	89	Faridabad	CSE	Pavitra Wallia	42414802718	2	STQA
10	Ishaan Garg	20514802718	90	Chandigarh	CSE	Pavitra Wallia	42414802718	2	STQA
11	Sanvesh Tiwari	42814802718	90	New Delhi	CSE	Tanya Batra	42614802718	3	DMBI
12	Pavitra Wallia	42414802718	92	New Delhi	CSE	Tanya Batra	42614802718	3	DMBI
13	Tanya Batra	42614802718	91	New Delhi	CSE	Tanya Batra	42614802718	3	DMBI
14	Himanshu Dahiya	42114802718	89	Faridabad	CSE	Tanya Batra	42614802718	3	DMBI
15	Ishaan Garg	20514802718	90	Chandigarh	CSE	Tanya Batra	42614802718	3	DMBI
16	Sanvesh Tiwari	42814802718	90	New Delhi	CSE	Himanshu	42114802718	1	ADMS
17	Pavitra Wallia	42414802718	92	New Delhi	CSE	Himanshu	42114802718	1	ADMS
18	Tanya Batra	42614802718	91	New Delhi	CSE	Himanshu	42114802718	1	ADMS
19	Himanshu Dahiya	42114802718	89	Faridabad	CSE	Himanshu	42114802718	1	ADMS
20	Ishaan Garg	20514802718	90	Chandigarh	CSE	Himanshu	42114802718	1	ADMS

- **INNER JOIN**

pgAdmin 4

File Object Tools Help

Browser Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres@PostgreSQL 14*

Servers (1) PostgreSQL 14 Databases (1) postres

Databases (1) postres

Casts Catalogs Event Triggers Extensions Foreign Data Languages Publications Schemas (1) public

Collations Domains FTS Configuration FTS Dictionaries FTS Parameters FTS Terms Foreign Functions

Query Editor Query History

```
1 SELECT S.student_name, course_id FROM student_sarvesh S
2 INNER JOIN student_course SC on S.enrollment_no=SC.enrollment_no;
```

Data Output Explain Messages Notifications

course_id	integer
1	1
2	2
3	3
4	1
5	2

• LEFT OUTER JOIN

pgAdmin 4

File Object Tools Help

Browser Dashboard Properties SQL Statistics Dependencies Dependents postgres/postgres@PostgreSQL 14*

Tables (2)

student_course

Columns (4)

- student_name
- enrollment_no
- course_id
- course

Constraints Indexes RLS Policies Rules Triggers

student_sarvesh

Columns (5)

- student_name
- enrollment_no

Query Editor Query History

```
1 SELECT S.student_name, branch FROM student_sarvesh S
2 LEFT OUTER JOIN student_course SC on S.enrollment_no=SC.enrollment_no;
```

Data Output Explain Messages Notifications

student_name	branch
Sarvesh Tiwari	CSE
Pavitra Walia	CSE
Tanya Batra	CSE
Himanshu Dahiya	CSE
Ishaan Garg	CSE

● RIGHT OUTER JOIN

The screenshot shows the pgAdmin 4 interface with the following details:

- Servers:** PostgreSQL 14 - postgres
- Databases:** postgres
- Query Editor:**

```
1 SELECT * FROM student_sarvesh S
2 RIGHT OUTER JOIN student_course SC on S.enrollment_no=SC.enrollment_no;
```
- Data Output:** Results of the query, showing 6 rows.

student_name	enrollment_no	marks	area	branch	student_name	enrollment_no	course_id	course
1 Sarvesh Tiwari	42814802718	90	New Delhi	CSE	Sarvesh Tiwari	42814802718	1	ADBMS
2 Pavitra Walla	42414802718	92	New Delhi	CSE	Pavitra Walla	42414802718	2	STQA
3 Tanya Batra	42614802718	91	New Delhi	CSE	Tanya Batra	42614802718	3	DMBI
4 Himanshu Dahiya	42114802718	89	Faridabad	CSE	Himanshu	42114802718	1	ADBMS
5 Ishaan Garg	20514802718	90	Chandigarh	CSE	Ishaan Garg	20514802718	2	STQA
6 [null]	[null]	[null]	[null]	[null]	Shubham Singla	20214802718	3	DMBI

● FULL OUTER JOIN

The screenshot shows the pgAdmin 4 interface with the following details:

- Servers:** PostgreSQL 14 - postgres
- Databases:** postgres
- Tables:** student_course
- Query Editor:**

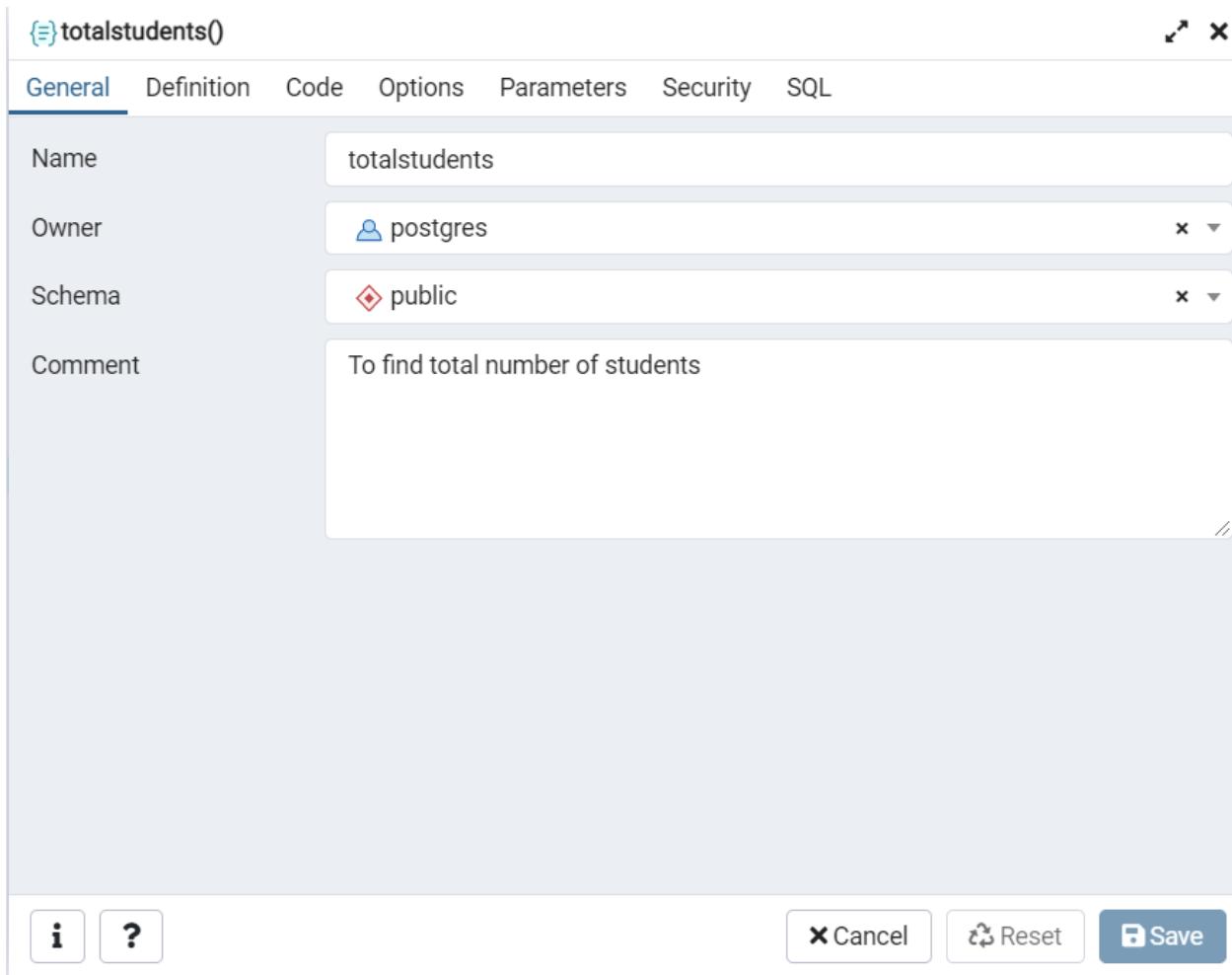
```
1 SELECT S.student_name,branch FROM student_sarvesh S
2 FULL OUTER JOIN student_course SC on S.enrollment_no=SC.enrollment_no;
```
- Data Output:** Results of the query, showing 6 rows.

student_name	branch
1 Sarvesh Tiwari	CSE
2 Pavitra Walla	CSE
3 Tanya Batra	CSE
4 Himanshu Dahiya	CSE
5 Ishaan Garg	CSE
6 [null]	[null]

Experiment 4

Aim :PL/SQL Programs

1. Write a program to calculate total students enrolled areawise.
- WE WILL CREATE A FUNCTION **TOTALSTUDENTS** . THIS FUNCTION WILL CALCULATE TOTAL NO OF STUDENTS



- THEN WE WILL SET THE RETURN TYPE OF THE FUNCTION AS INTEGER AND THEN WE WILL SET THE LANGUAGE

{totalstudents()}

General Definition Code Options Parameters Security SQL

Return type integer

Language plpgsql

Arguments

Data type	Mode	Argument name	Default
-----------	------	---------------	---------

i **?** **Cancel** **Reset** **Save**

- WE WRITE THE CODE SO THAT THIS FUNCTION RETURNS THE TOTAL NO OF STUDENTS

{totalstudents()}

General Definition Code Options Parameters Security SQL

```
1 begin
2   RETURN count(enrollment_no) from student_sarvesh;
3 end;
4
```

- THIS IS THE FUNCTION IN SQL.

Create - Function

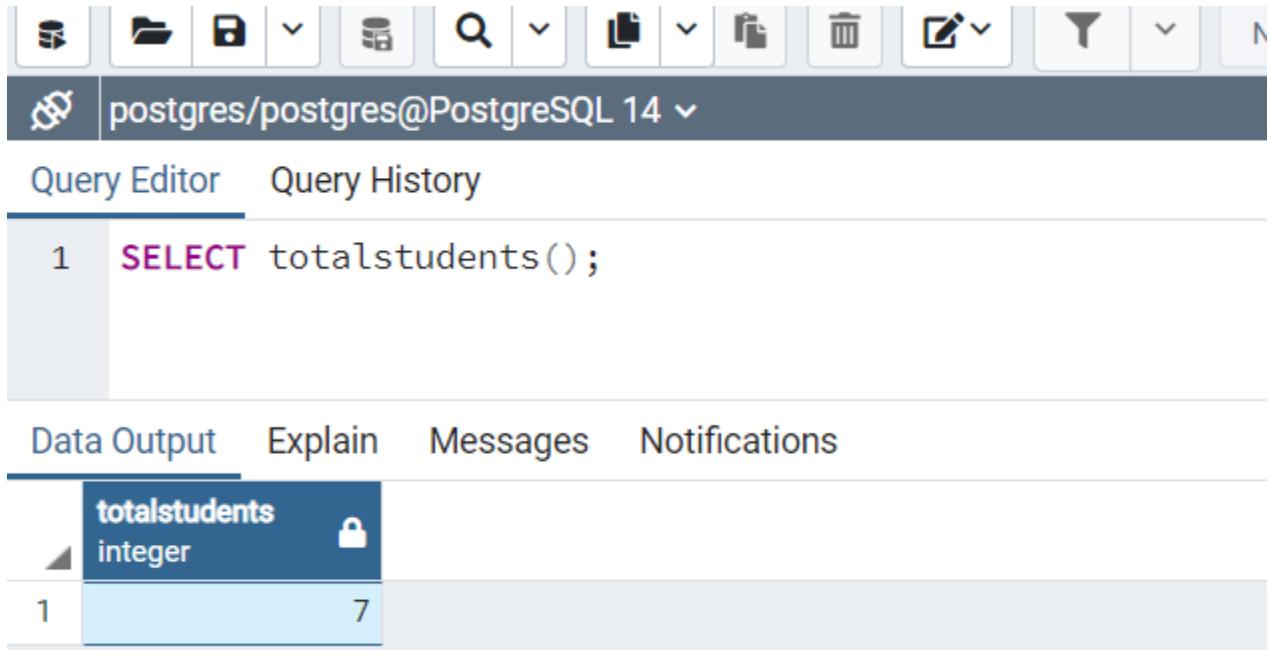
General Definition Code Options Parameters Security **SQL**

```

1 CREATE FUNCTION public.totalstudents()
2     RETURNS integer
3     LANGUAGE 'plpgsql'
4
5 AS $BODY$
6 begin
7     RETURN count(enrollment_no) FROM student_sarvesh;
8 end;
9 $BODY$;
10
11 ALTER FUNCTION public.totalstudents()
12     OWNER TO postgres;
13
14 COMMENT ON FUNCTION public.totalstudents()
15     IS 'To find total number of students';

```

- WE RUN THE FUNCTION USING SELECT FUNCTION NAME().



The screenshot shows the pgAdmin 4 interface. At the top, there's a toolbar with various icons. Below it is a connection bar showing 'postgres/postgres@PostgreSQL 14'. The main area has tabs for 'Query Editor' (which is active) and 'Query History'. In the Query Editor, a single line of SQL is written: 'SELECT totalstudents();'. Below the editor, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. Under 'Data Output', there's a table with one row. The table has two columns: 'totalstudents' and 'integer'. The value '7' is displayed in the 'integer' column. The entire interface is set against a light gray background.

totalstudents	integer
1	7

- Write a program to calculate average marks obtained by the students residing in area that starts with "R"
- We will make a function named **avgmarks**.

avgmarks()

General Definition Code Options Parameters Security SQL

Name	avgmarks
Owner	postgres
Schema	public
Comment	To calculate avg marks of students residing in an area that starts with "R"

- Now we set the return type and language plpgsql

avgmarks()

General **Definition** Code Options Parameters Security SQL

Return type	integer
Language	plpgsql

Arguments

Data type	Mode	Argument name	Default

- We write the code for the function avgmarks.

{avgmarks()}

General	Definition	Code	Options	Parameters	Security	SQL
---------	------------	-------------	---------	------------	----------	-----

```

1 begin
2     RETURN AVG(marks) FROM student_sarvesh where area like 'N%';
3 end;
4

```

- This is the function in SQL

{Create - Function}

General	Definition	Code	Options	Parameters	Security	SQL
---------	------------	------	---------	------------	----------	------------

```

1 CREATE FUNCTION public.avgmarks()
2     RETURNS integer
3     LANGUAGE 'plpgsql'
4
5 AS $BODY$
6 begin
7     RETURN AVG(marks) FROM student_sarvesh where area like 'N%';
8 end;
9 $BODY$;
10
11 ALTER FUNCTION public.avgmarks()
12     OWNER TO postgres;
13
14 COMMENT ON FUNCTION public.avgmarks()
15     IS 'To calculate avg marks of students residing in an area that starts with

```

- Now in query editor , we will run the function we write select function name()

The screenshot shows a PostgreSQL query editor interface. At the top, there is a toolbar with various icons for file operations like save, open, and search. Below the toolbar, there are two tabs: "Query Editor" (which is selected) and "Query History". In the main area, a single query is listed:

```
1 SELECT avgmarks();
```

Below the query, there are four navigation tabs: "Data Output" (selected), "Explain", "Messages", and "Notifications". Under "Data Output", a table is displayed with one row of data:

	avgmarks
1	91

Experiment 5

Aim : Write the cursor to increase the marks of students by 10%.

Theory:

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:Implicit cursors and Explicit cursors.

Code:

```
CREATE OR REPLACE FUNCTION increasemarks()
RETURNS SETOF varchar AS
$func$
DECLARE
rec record;
BEGIN
FOR rec IN
SELECT *
FROM student_sarvesh
LOOP
update student_sarvesh
set marks=marks*1.10;
END LOOP;
END
$LANGUAGE plpgsql VOLATILE;
select increasemarks();
```

```
select * from student_sarvesh;
```

Output:

- Before increasing the marks of students by 10%.

The screenshot shows a database interface with a toolbar at the top featuring 'Data Output', 'Explain', 'Messages', and 'Notifications'. Below the toolbar is a table with five rows of data. The table has six columns with the following headers and data:

	student_name character varying (30)	enrollment_no bigint	marks integer	area character varying (50)	branch character varying (30)
1	Sarvesh Tiwari	42814802718	90	New Delhi	CSE
2	Pavitra Walia	42414802718	92	New Delhi	CSE
3	Tanya Batra	42614802718	91	New Delhi	CSE
4	Himanshu Dahiya	42114802718	89	Faridabad	CSE
5	Ishaan Garg	20514802718	90	Chandigarh	CSE

- After increasing the marks of students by 10%.

postgres/postgres@PostgreSQL 14 ▾

Query Editor Query History

```

1 CREATE OR REPLACE FUNCTION increasemarks()
2   RETURNS SETOF varchar AS
3 $func$ 
4   DECLARE
5     rec record;
6   BEGIN
7     FOR rec IN
8       SELECT *
9         FROM student_sarvesh
10    LOOP
11      update student_sarvesh
12        set marks=marks*1.10;
13    END LOOP;
14  END
15 $func$ LANGUAGE plpgsql VOLATILE;
16 select increasemarks();
17 select * from student_sarvesh;
```

Data Output Explain Messages Notifications

	student_name character varying (30)	enrollment_no [PK] bigint	marks integer	area character varying (50)	branch character varying (30)
1	Sarvesh Tiwari	42814802718	213	New Delhi	CSE
2	Pavitra Walia	42414802718	216	New Delhi	CSE
3	Tanya Batra	42614802718	215	New Delhi	CSE
4	Himanshu Dahiya	42114802718	210	Faridabad	CSE
5	Ishaan Garg	20514802718	213	Chandigarh	CSE

Experiment 6

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

Theory:

An exception occurs when the PL/SQL engine encounters an instruction which it cannot execute due to an error that occurs at run-time. These errors will not be captured at the time of compilation and hence these need to be handled only at the run-time.

Syntax:

```
CREATE [ PROCEDURE | FUNCTION ]
AS
BEGIN
<Execution block>
RAISE <exception_name>
EXCEPTION
WHEN <exception_name> THEN
<Handler>
END;
```

Program:

- A. Execution of instruction without any error.

```
do $$

declare
    rec record;
    s_name text = 'Sarvesh Tiwari';
begin
    select enrollment_no, student_sarvesh.student_name
into strict rec
    from student_sarvesh
    where student_sarvesh.student_name = s_name;
    -- catch exception
exception
    when no_data_found then
```

```

        raise exception 'student with Name "%" is not
assigned', s_name;
end;
$$
language plpgsql;
```

Output:

The screenshot shows a PostgreSQL query editor interface. At the top, it says "postgres/postgres@PostgreSQL 14". Below that is a toolbar with "Query Editor" and "Query History" tabs, where "Query Editor" is selected. The main area contains a numbered PL/pgSQL block:

```

1 do $$ 
2 declare
3     rec record;
4     s_name text = 'Sarvesh Tiwari';
5 begin
6     select enrollment_no, student_sarvesh.student_name into strict rec
7     from student_sarvesh
8     where student_sarvesh.student_name = s_name;
9     -- catch exception
10    exception
11        when no_data_found then
12            raise exception 'student with Name "%" is not assigned', s_name;
13    end;
14 $$
15 language plpgsql;
```

Below the code, there are tabs for "Data Output", "Explain", "Messages" (which is underlined), and "Notifications". Under "Messages", the output is:

DO

Query returned successfully in 65 msec.

B. Execution of instruction with an exception.

```

do $$ 
declare
    rec record;
```

```

        s_name text = 'Pavitra Goel';
begin
    select enrollment_no, student_sarvesh.student_name
into strict rec
    from student_sarvesh
    where student_sarvesh.student_name = s_name;
    -- catch exception
exception
    when no_data_found then
        raise exception 'student with Name "%" is not
assigned', s_name;
end;
$$
language plpgsql;

```

Output:

The screenshot shows a PostgreSQL Query Editor interface. The title bar says "postgres/postgres@PostgreSQL 14". The main area displays a PL/pgSQL function definition with line numbers 1 through 16. Lines 11 and 12 are highlighted in blue, indicating the error occurred at these specific lines. The code defines a function that selects a student record based on name and raises an exception if no record is found.

```

1 do $$ 
2 declare
3     rec record;
4     s_name text = 'Pavitra Goel';
5 begin
6     select enrollment_no, student_sarvesh.student_name into strict rec
7     from student_sarvesh
8     where student_sarvesh.student_name = s_name;
9     -- catch exception
10    exception
11    when no_data_found then
12        raise exception 'student with Name "%" is not assigned', s_name;
13    end;
14    $$
15 language plpgsql;
16

```

Below the code, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Messages" tab is selected, showing the following error message:

ERROR: student with Name "Pavitra Goel" is not assigned
CONTEXT: PL/pgSQL function inline_code_block line 12 at RAISE
SQL state: P0001

Experiment 7

Aim : Procedure & Function

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

Theory:

PostgreSQL allows us to extend the database functionality with user-defined functions by using various procedural languages, which are often referred to as stored procedures.

With stored procedures you can create your own custom functions and reuse them in applications or as part of other database's workflows.

Syntax:

```
create [or replace] procedure
procedure_name(parameter_list)
language plpgsql
as $$

declare
-- variable declaration
begin
-- stored procedure body
end; $$
```

Code:

```
CREATE OR REPLACE PROCEDURE
    public.cse_avg_marks(IN student_branch text, INOUT
avg_marks real)
LANGUAGE 'plpgsql'
AS $BODY$

begin
    select avg(marks) into avg_marks
    from student_sarvesh
    where branch = student_branch;
end;
```

\$BODY\$;

Output:

The screenshot shows the pgAdmin 4 interface. At the top, there's a toolbar with various icons. Below it is a connection bar showing 'postgres/postgres@PostgreSQL 14'. The main area is a 'Query Editor' tab, which is active. The code in the editor is:

```
1 CREATE OR REPLACE PROCEDURE
2     public.cse_avg_marks(IN student_branch text, INOUT avg_marks real)
3 LANGUAGE 'plpgsql'
4 AS $BODY$
5 begin
6     select avg(marks) into avg_marks
7     from student_sarvesh
8     where branch = student_branch;
9 end;
$BODY$;
11
12 --calling the Procedure 'avg_mark'
13 CALL cse_avg_marks ('CSE',0)
```

Below the code, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. Under 'Data Output', there's a table with one row:

avg_marks	real
1	88.14286

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

Code:

```
CREATE OR REPLACE PROCEDURE
    public.total_students_branchwise(IN student_branch
text, INOUT total_students real)
LANGUAGE 'plpgsql'
AS $BODY$
begin
    select COUNT(enrollment_no) into total_students
    from student_sarvesh
    where branch = student_branch;
```

```

end;
$BODY$;

--calling the Procedure 'avg_mark'
CALL total_students_branchwise ('CSE',0)

```

Output:

- All entries in the table.

postgres/postgres@PostgreSQL 14 ▾

Query Editor Query History

1 `SELECT * FROM student_sarvesh;`

Data Output Explain Messages Notifications

	<code>student_name</code> character varying (30)	<code>enrollment_no</code> [PK] bigint	<code>marks</code> integer	<code>area</code> character varying (50)	<code>branch</code> character varying (30)
1	Sarvesh Tiwari	42814802718	90	New Delhi	CSE
2	Pavitra Walia	42414802718	92	New Delhi	CSE
3	Tanya Batra	42614802718	91	New Delhi	CSE
4	Himanshu Dahiya	42114802718	89	Faridabad	CSE
5	Ishaan Garg	20514802718	90	Chandigarh	CSE
6	Shubham	12414802718	80	Rajasthan	CSE
7	Jatin	41914802718	85	Ranchi	CSE
8	Lakshay	21414802718	93	Mumbai	EEE
9	Mrigank	31414802718	99	Pune	IT

- Total number of students in “CSE” branch



postgres/postgres@PostgreSQL 14 ▾

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

```
1 CALL total_students_branchwise ('CSE',0)
```

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

	total_students	real	lock
1		7	

- Total number of students in “IT” branch.



postgres/postgres@PostgreSQL 14 ▾

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

```
1 CALL total_students_branchwise ('IT',0)
```

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

	total_students	real	lock
1		1	

Experiment 8

Aim : Trigger

- a. Create a trigger on the table after inserting a new student into the table.

Theory:

A trigger is a set of actions that are run automatically when a specified change operation (SQL INSERT, UPDATE, DELETE or TRUNCATE statement) is performed on a specified table. Triggers are useful for tasks such as enforcing business rules, validating input data, and keeping an audit trail.

Advantages of Triggers

These are the following advantages of Triggers:

- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

The syntax for creating a trigger is –

```
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;
```

Implementation :

a. Trigger on the table after inserting a new student into the table.

Output:

```
postgres=# create or replace function insertion() returns trigger as $body$  
postgres$# begin  
postgres$# insert into student(student_name,roll_no,marks,area,branch) values (new.student_name,new.roll_no,new.marks,new.area,new.branch);  
postgres$# return new;  
postgres$# end;  
postgres$# $body$ language plpgsql;  
CREATE FUNCTION  
  
postgres=# create trigger insertoperation  
postgres-# after update on student  
postgres-# for each row  
postgres-# execute procedure insertion();  
CREATE TRIGGER  
  
postgres=# insert into student values('Hina',21,45,'Delhi','CSE');  
INSERT 0 1  
postgres=
```

b. Row trigger to insert the existing values of the student table into a new table when the marks of the student are updated.

Output:

```
postgres=# create or replace function update_marks() returns trigger as $body$  
postgres$# begin  
postgres$# insert into student(student_name,roll_no,marks,area,branch) values(new.student_name,new.roll_no,new.marks,new.area,new.branch);  
postgres$# return new;  
postgres$# end;  
postgres$# $body$ language plpgsql;  
CREATE FUNCTION  
  
postgres=# create trigger markschange  
postgres-# after update on student  
postgres-# for each row  
postgres-# execute procedure update_marks();  
CREATE TRIGGER
```

```
postgres=# update student set marks=marks*2;  
UPDATE 4  
postgres=# select * from student;  
student_name | roll_no | marks | area | branch  
-----+-----+-----+-----+-----  
Hina | 21 | 90 | Delhi | CSE  
Rahul | 1 | 62 | Delhi | CSE  
Sarvesh | 4 | 210 | Haryana | EEE  
Tanya | 3 | 256 | Punjab | ECE  
(4 rows)
```

THANK YOU !