

*Lab ﬁlE*

ADVANCED DBMS LAB

**(ETCS-457)**

Submitted to: Name: **Sarvesh Nath Tiwari**

**Ms. Savita Sharma** Roll No.: 42814802718

Assistant Professor Semester: 7th

CSE Dept., MAIT Group: 7C-7



Maharaja Agrasen Institute of Technology, PSP Area, Sector – 22, Rohini New Delhi – 110085

|  |  |  |  |
| --- | --- | --- | --- |
| **S No.** | **List of experiments** | **Date** | **Remarks** |
| **1.** | 1. Learn to install PostgreSql. 2. 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   1. DDL (eg create, alter, drop, rename, truncate), 2. DML (eg. Insert, update, delete etc.), 3. DCL (eg. Grant, revoke etc.) 4. Built-in Functions (eg. Sum, min, max, avg, count, lower, upper, trim, len etc.) 5. Indexes and views: Create and Drop |  |  |
| **3.** | Write SQL queries on Nested queries and Join. |  |  |
| **4.** | **PL/SQL**   1. Write a program to calculate total students enrolled areawise. 2. 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**   1. Write the procedure to get the average marks of students for branch “CSE”. 2. Write a function that accepts the branch and returns the total no. of   students of the branch. |  |  |
| **8.** | **Trigger**   1. Create a trigger on the table after inserting a new student into the table. 2. 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)

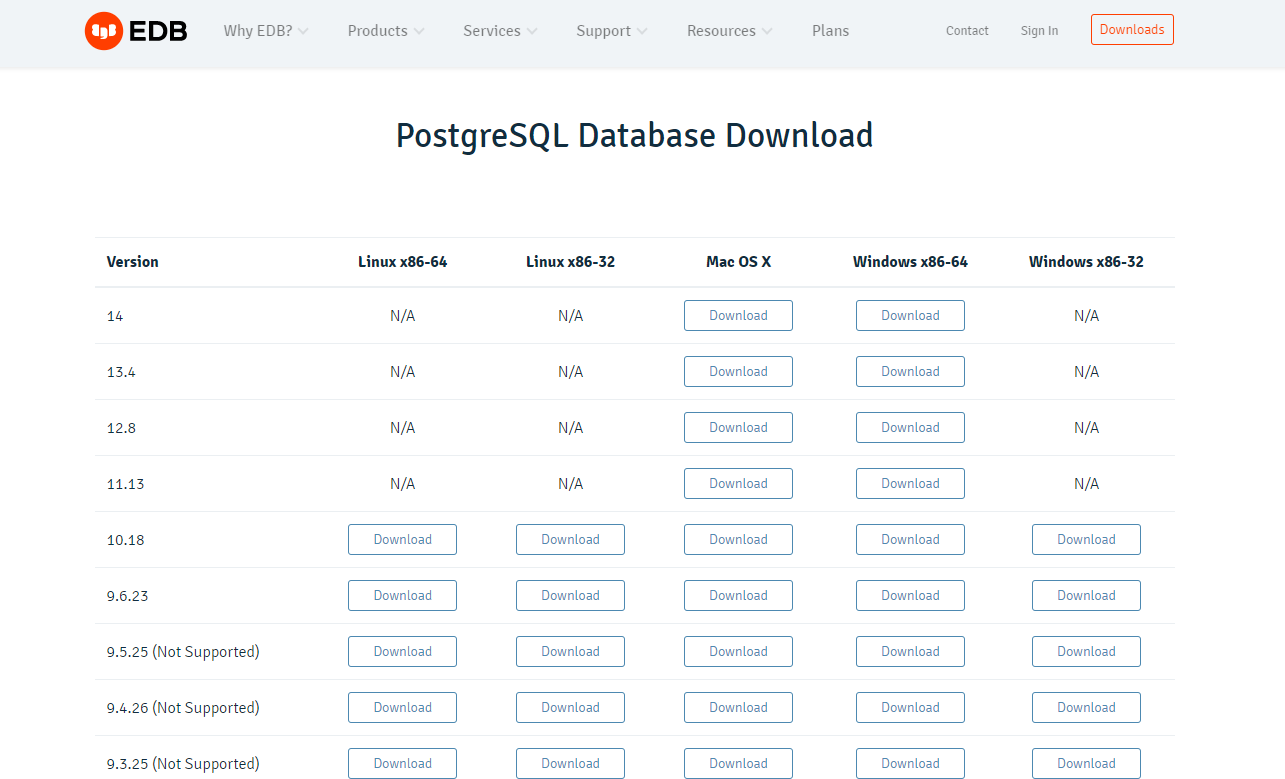
**Aim :** Learn to install PostgreSql.

**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.

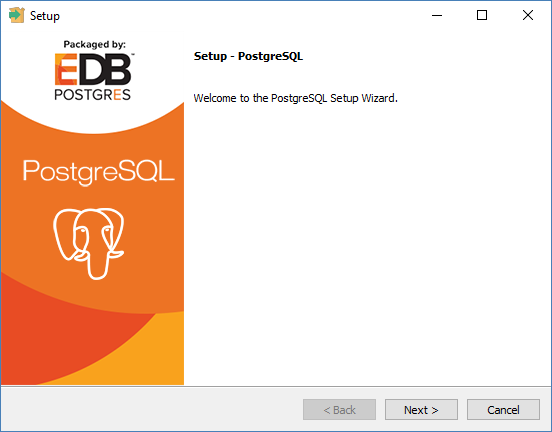
## Installation of PostgreSQL:

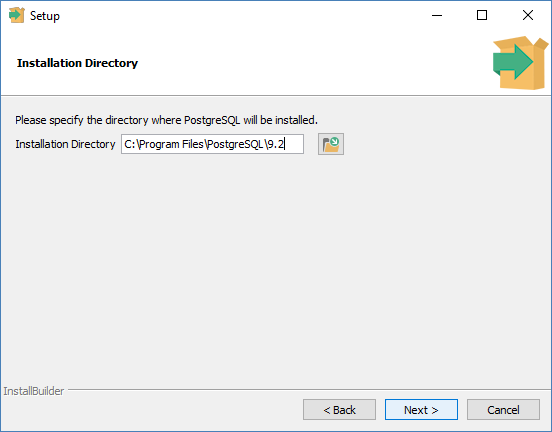
* We will download PostgreSQL installer for Windows
* First, we need to go to the download page of PostgreSQL installers on the EnterpriseDB.
* Then we will click the download link as shown below figure.
* It will take a few minutes to complete the download.



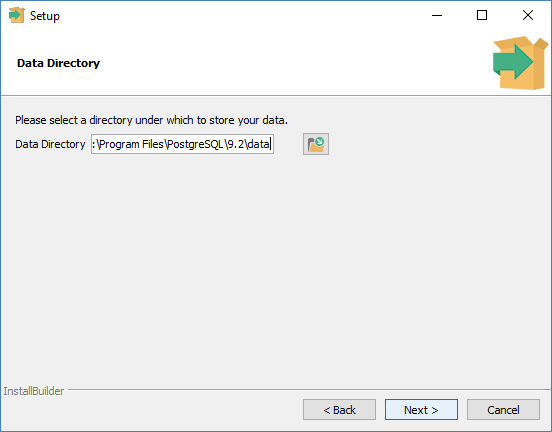
The further installation steps are:

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.

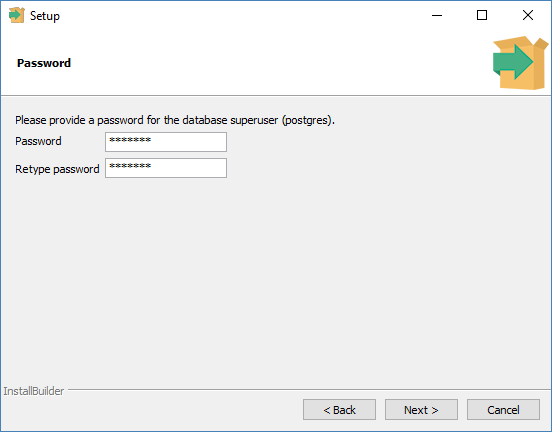


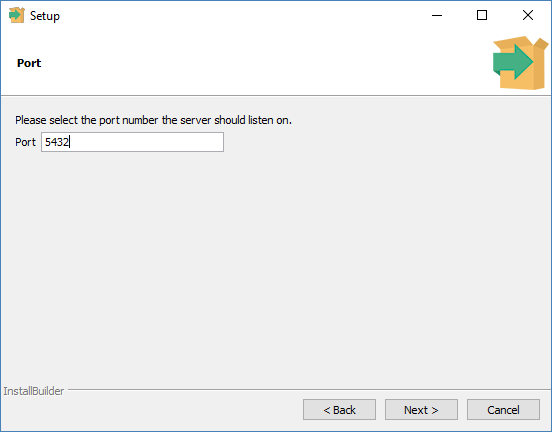
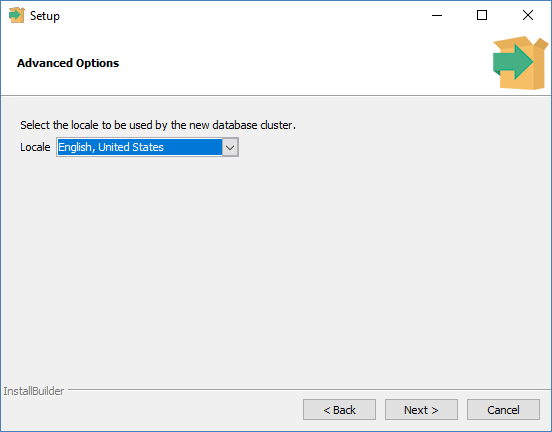


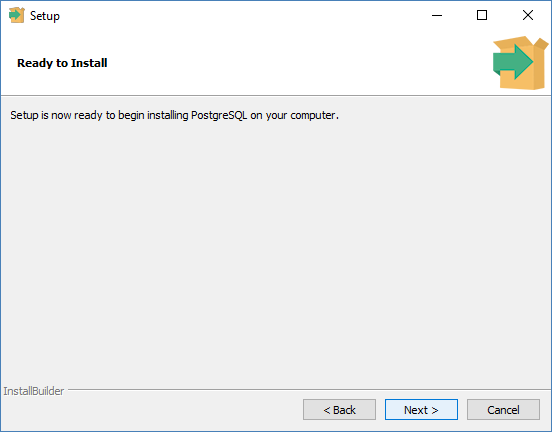
1. The next step of the installation process would be to select the directory where data would be stored, by default it is stored under "data" directory.

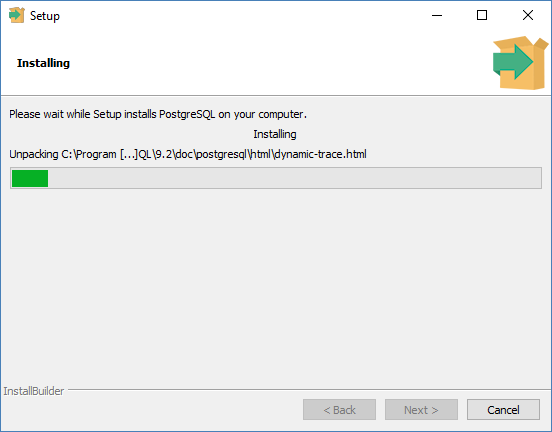


1. In the next step, setup asks for password, so you can use your favorite password



1. In the next step, keep the port as default.
2. In the next step, when asked for "Locale", "English, United States" has been selected.



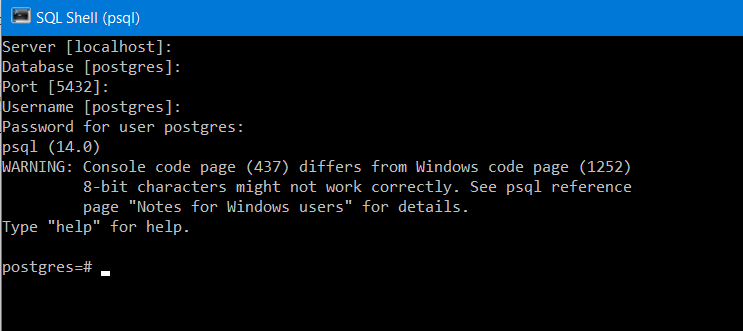


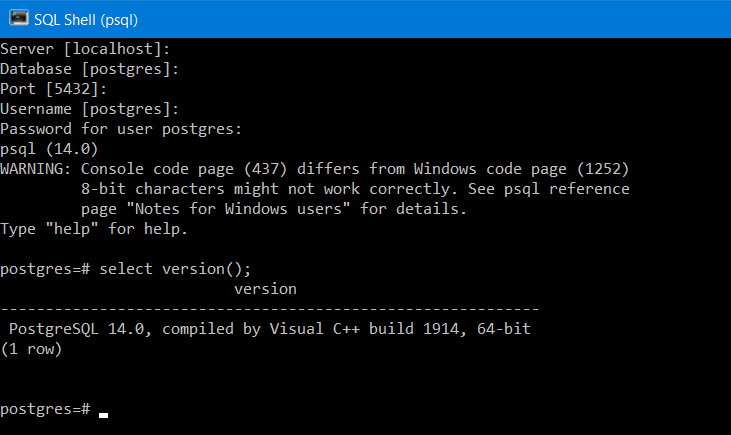
1. It takes a while to install PostgreSQL on your system. On completion of the installation process, you will get the following screen. Uncheck the checkbox and click on the Finish button.



## 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.



1. Enter the command SELECT version () and you will see the following output.

# 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.

## 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

## MySQL

* + Open-source
  + Compatible with a wide range of engines and interfaces; one Huof 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

## 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

## 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

## 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

My-SQL 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** |
| *Open Source* | Completely Open source | Open source, but owned by  Oracle and offers commercial versions |
| *ACID*  *Compliance* | Complete ACID Compliance | Some versions are compliant |
| *SQL Compliance* | Almost fully compliant | Some versions are compliant |
| *Concurrency Support* | MVCC implementation supports multiple requests without read locks | Support in some versions. |
| *Security* | Secure from ground up with SSL support | SSL support in some versions |
| *NoSQL/JSON*  *Support* | Multiple supported features | JSON data support only |
| *Access Methods* | Supports all standards | Supports all standards |
| *Replication* | 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 |
| *Materialized Views* | Supported | Not supported |
| *Temporary Tables* | Supported | Supported |
| *GeoSpatial Data* | Supported | Supported |
| *Programming Languages* | Supported | Not supported |
| *Extensible Type System* | 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.

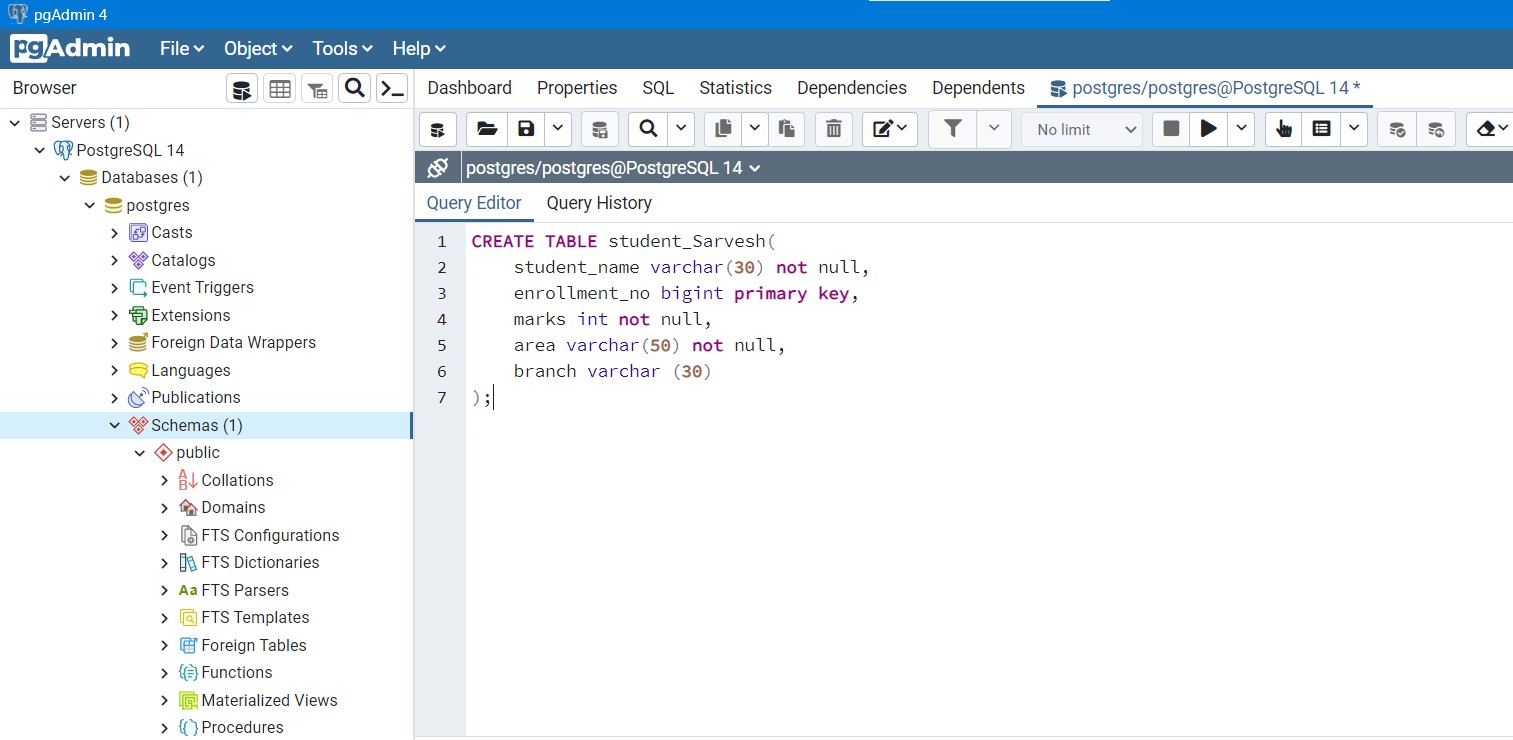
Let us build the table “Student”:

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

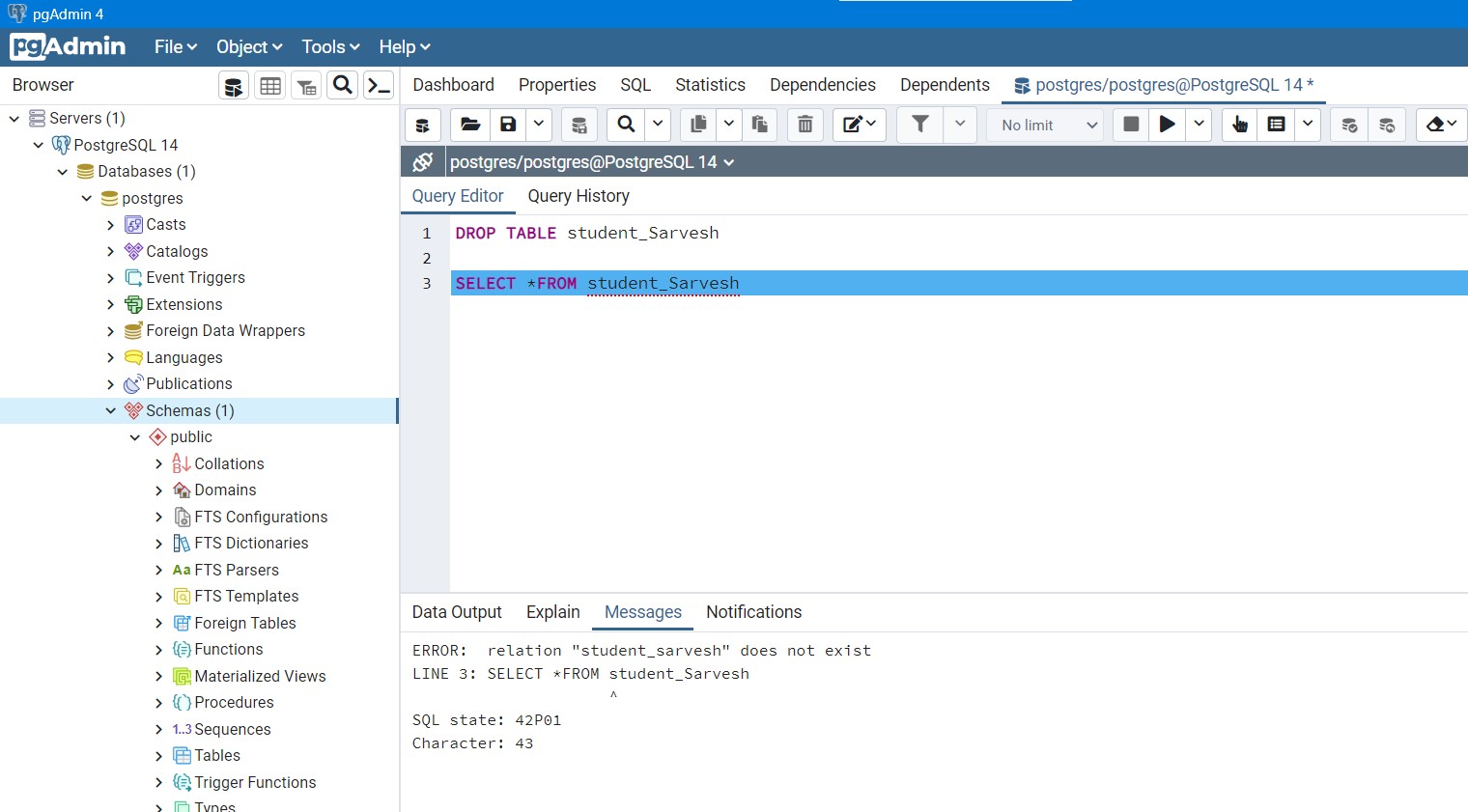
create table student\_Sarvesh( student\_name varchar(30) not null, enrollment\_no bigint primary key, marks int not null,

area varchar(50) not null, branch varchar (30)

);

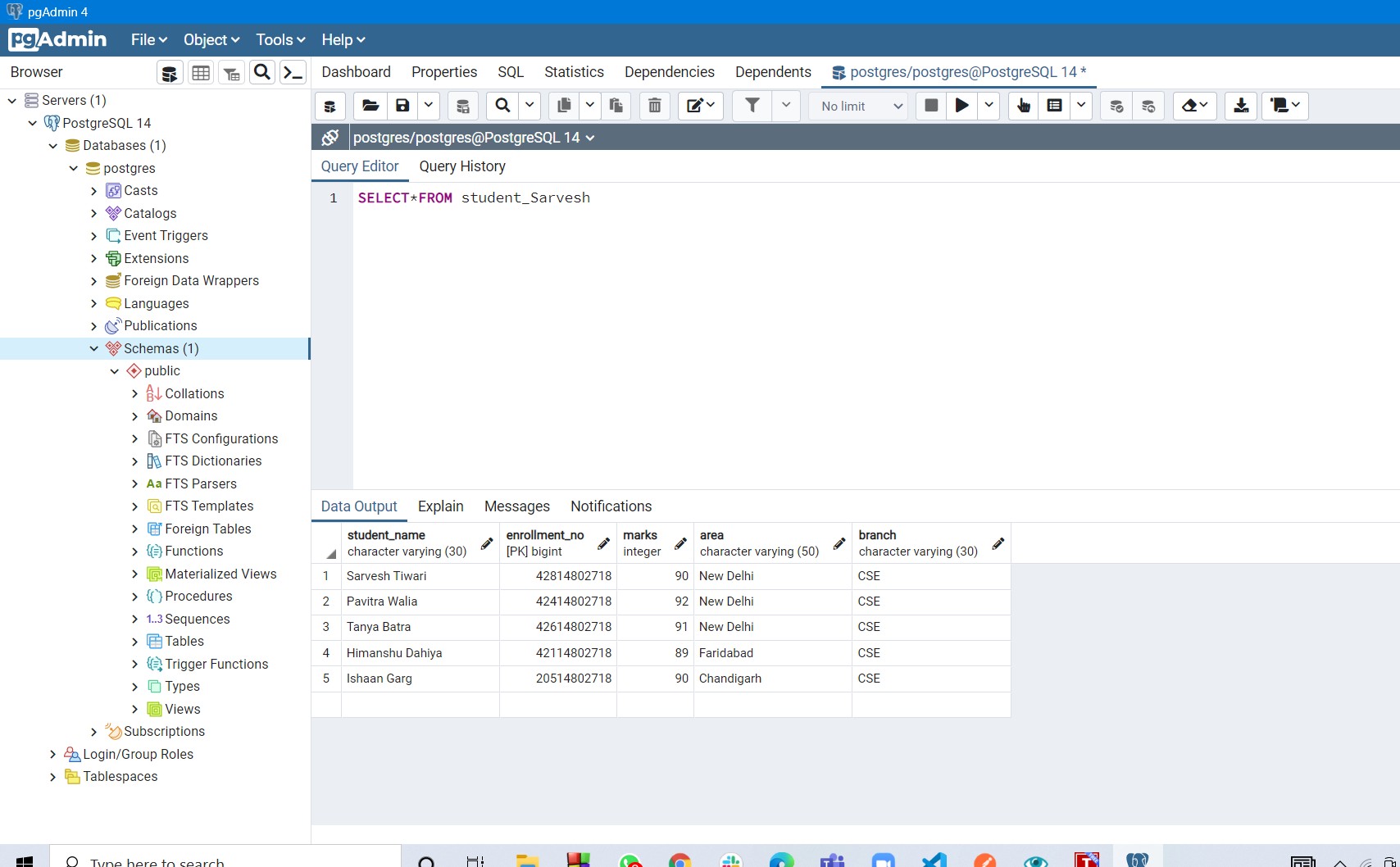


**DROP-** The DROP command is a type of SQL DDL command that is used to delete an existing database or an object within a database.

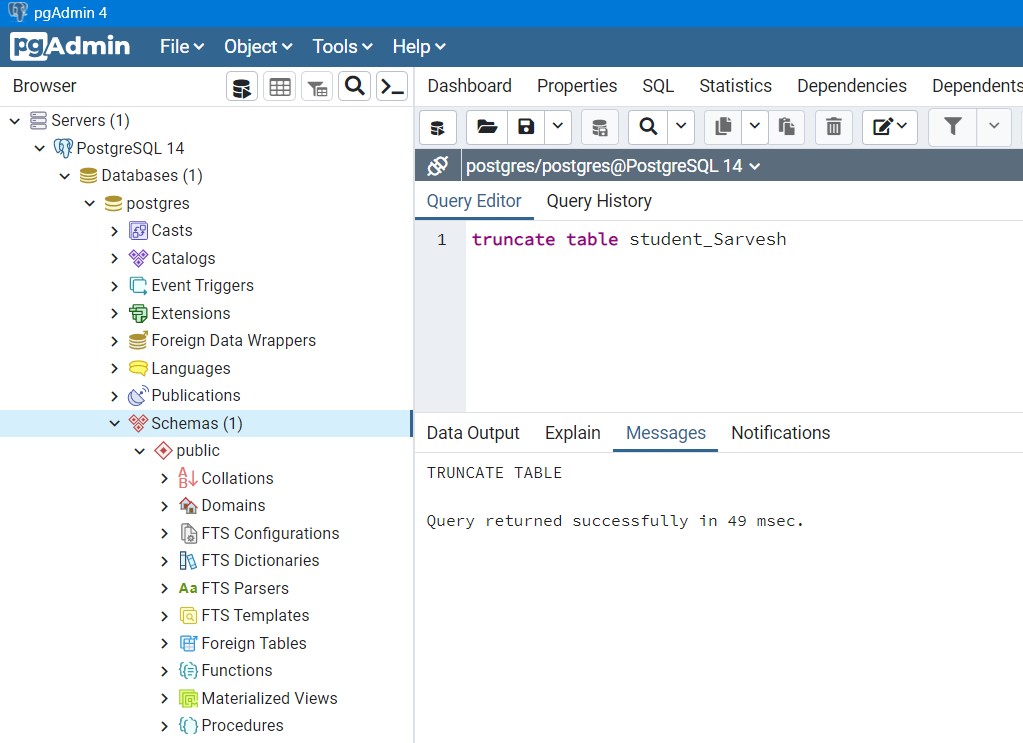


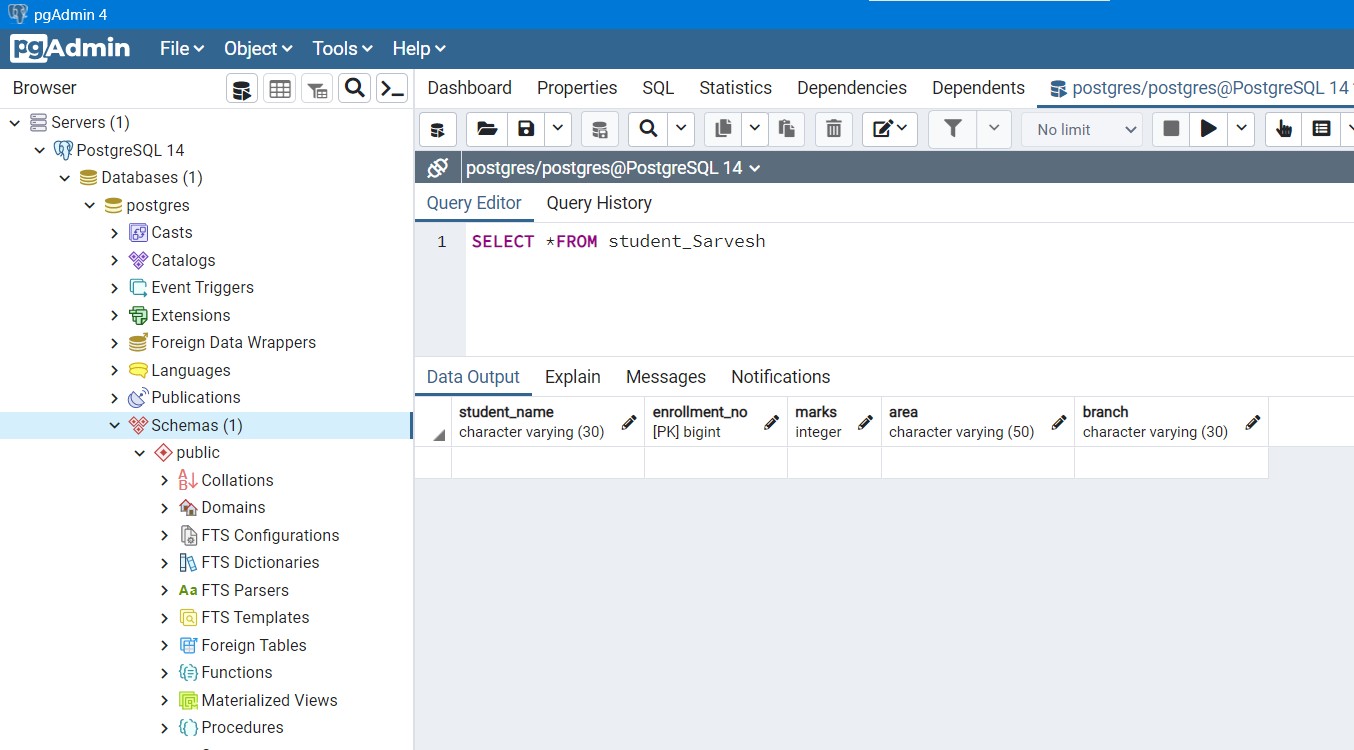
**TRUNCATE :** The TRUNCATE command in SQL DDL is used to remove all the records from a table. Let’s insert a few records in the Books table:

Before TRUNCATE:



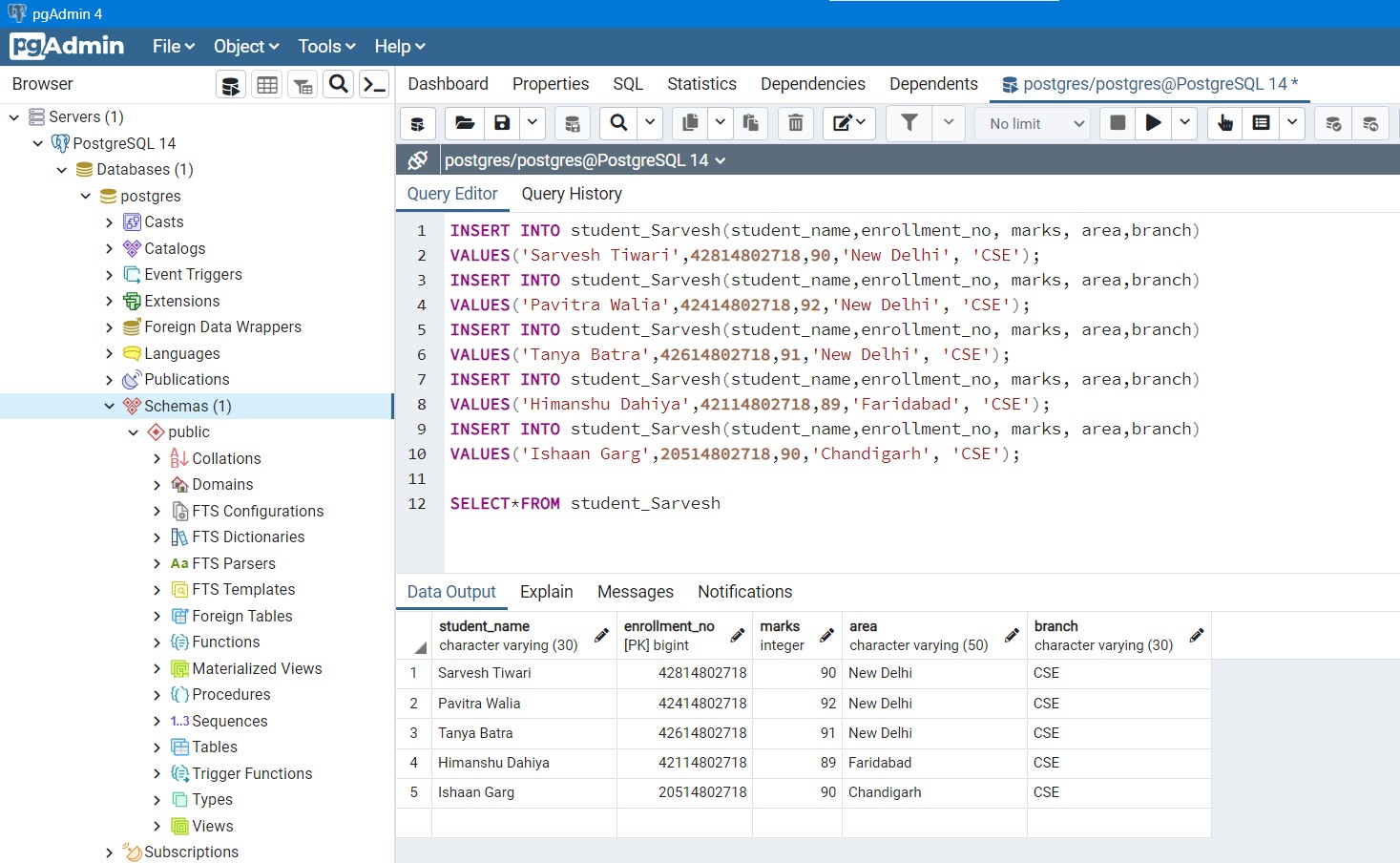
After TRUNCATE:





* **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.

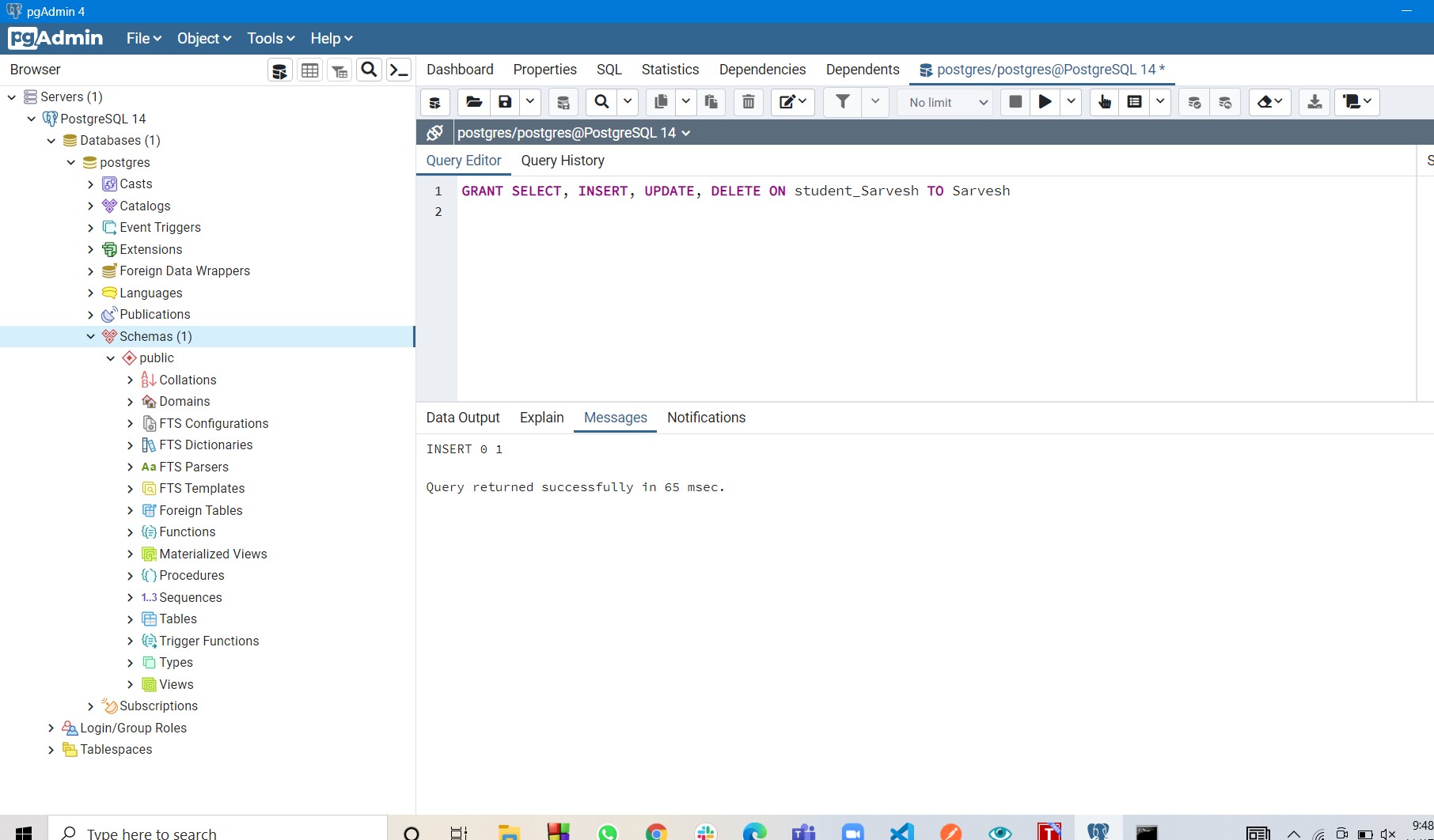


**DELETE:** It is used to remove one or more rows from a table.

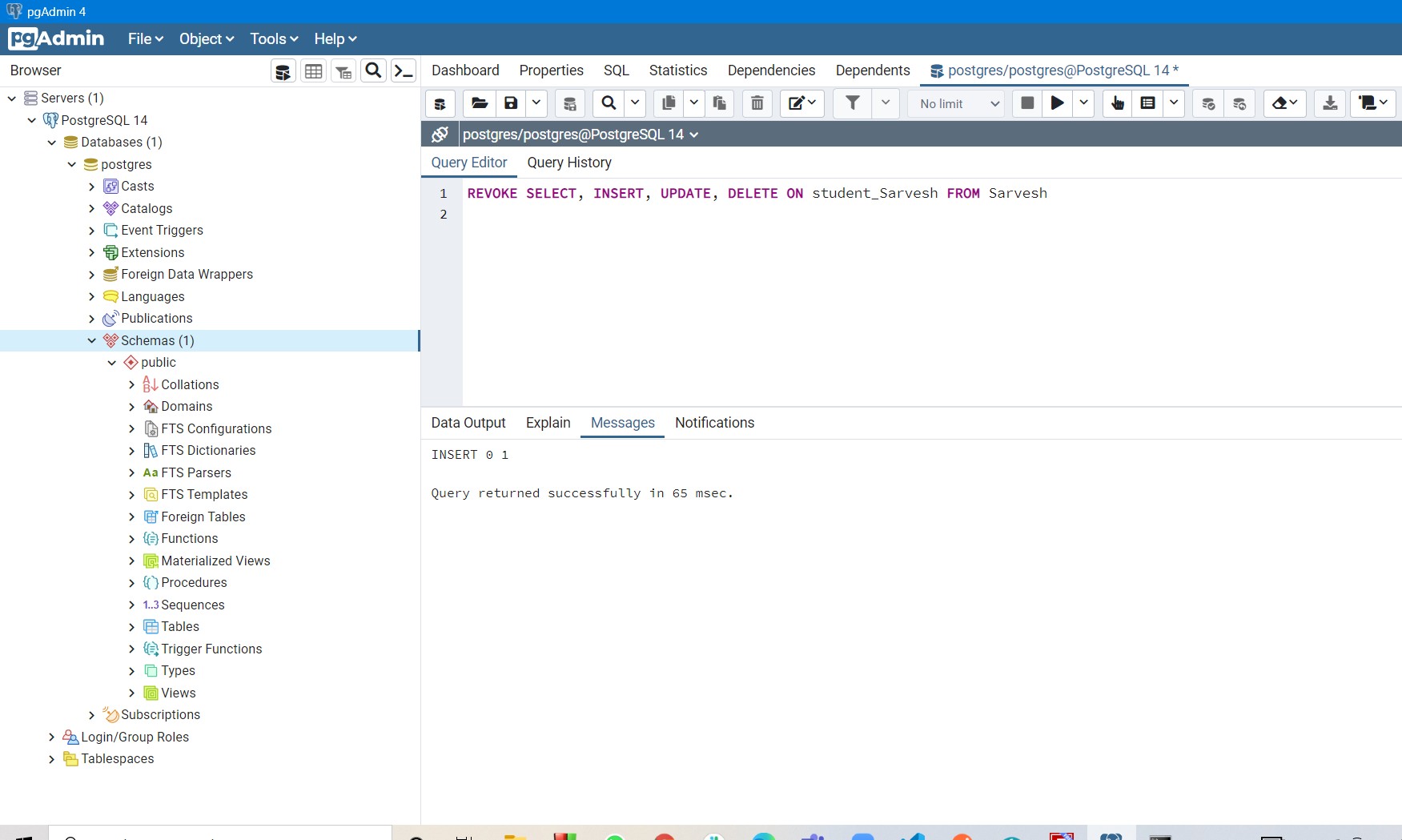


* + **Data Control Language(DCL)**: It includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

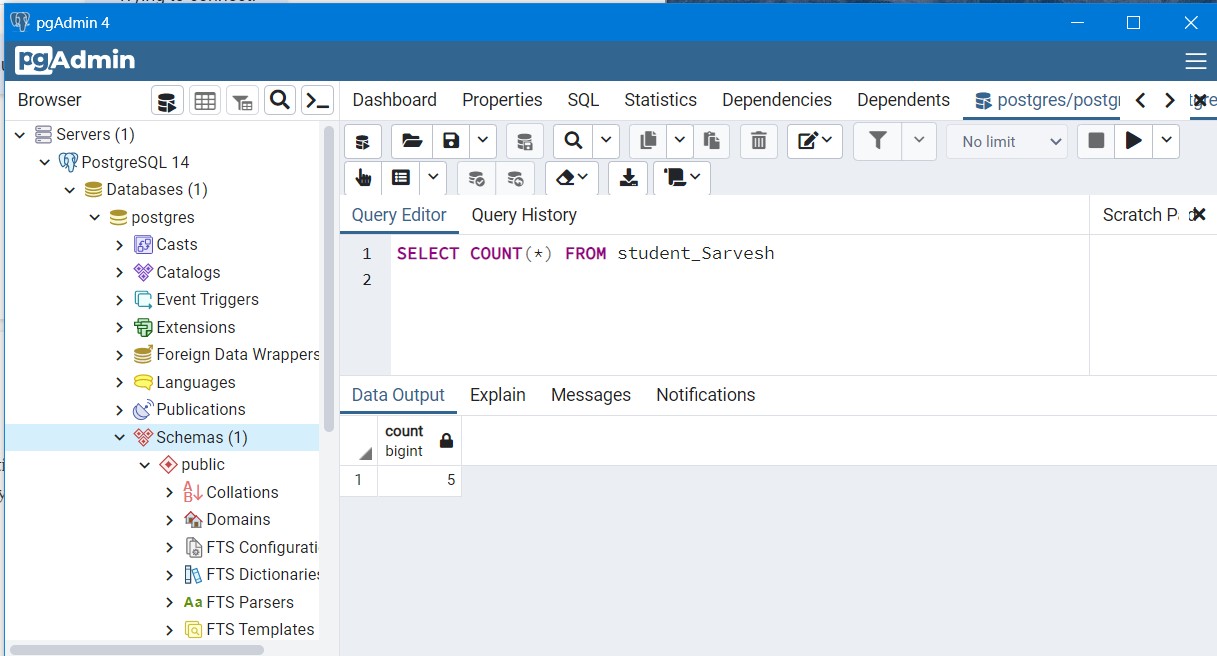
**GRANT**: It is used to give user access privileges to a database.



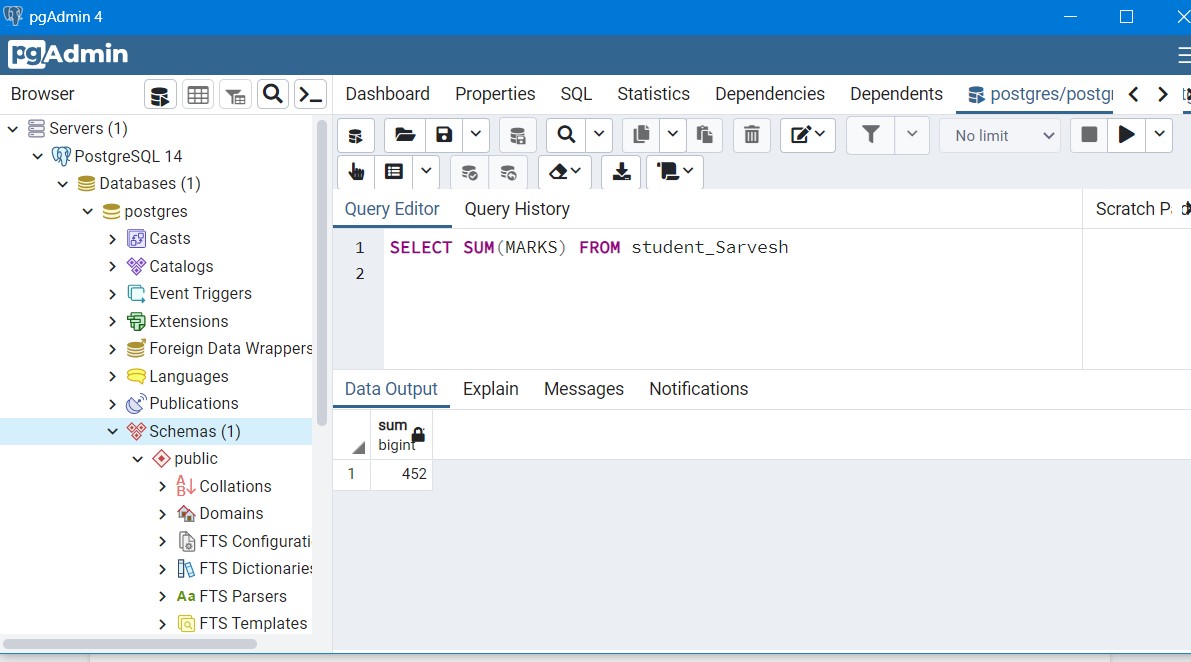
**REVOKE:** It is used to take back permissions from the user.



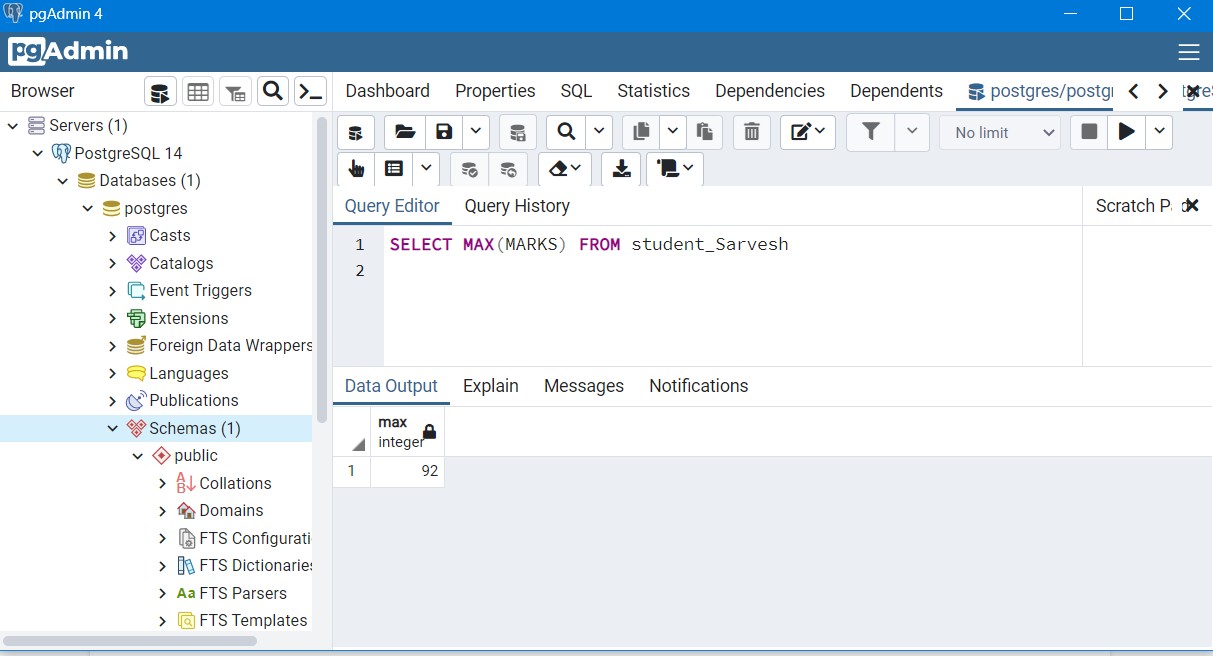
**Built-in Functions of SQL** : Built-in functions are predefined functions that give you convenient ways to manipulate your data. Examples are count, sum, avg, min, max, etc.

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

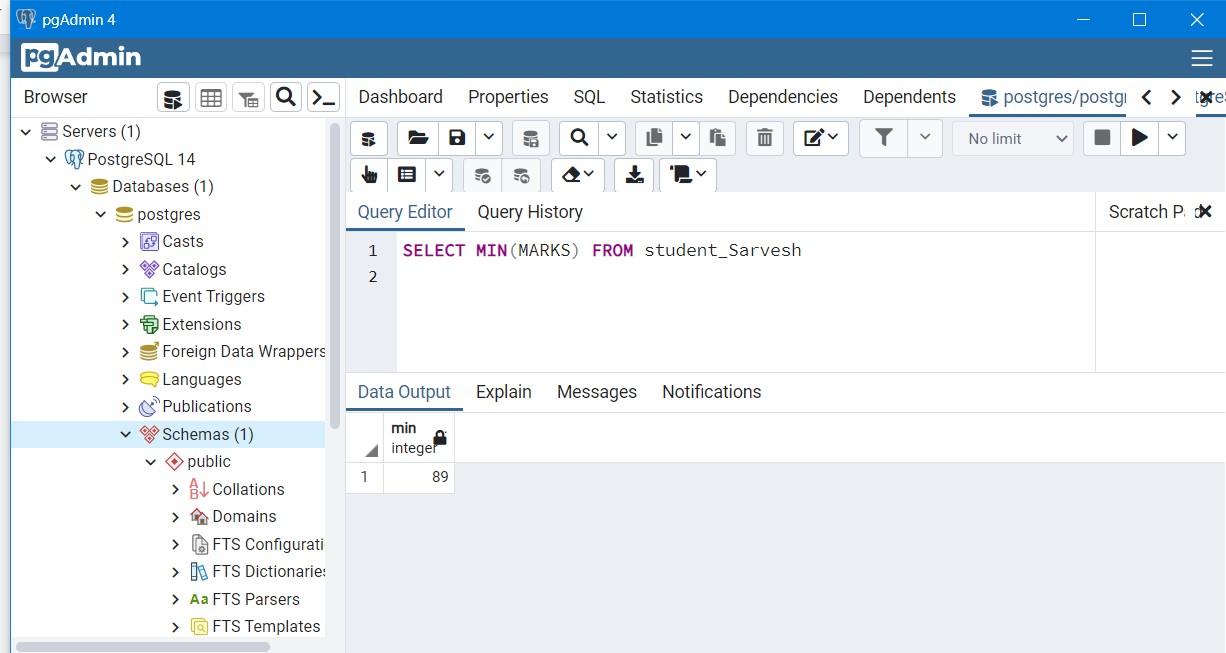
**SUM() :** This function returns the total sum of a numeric column.



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

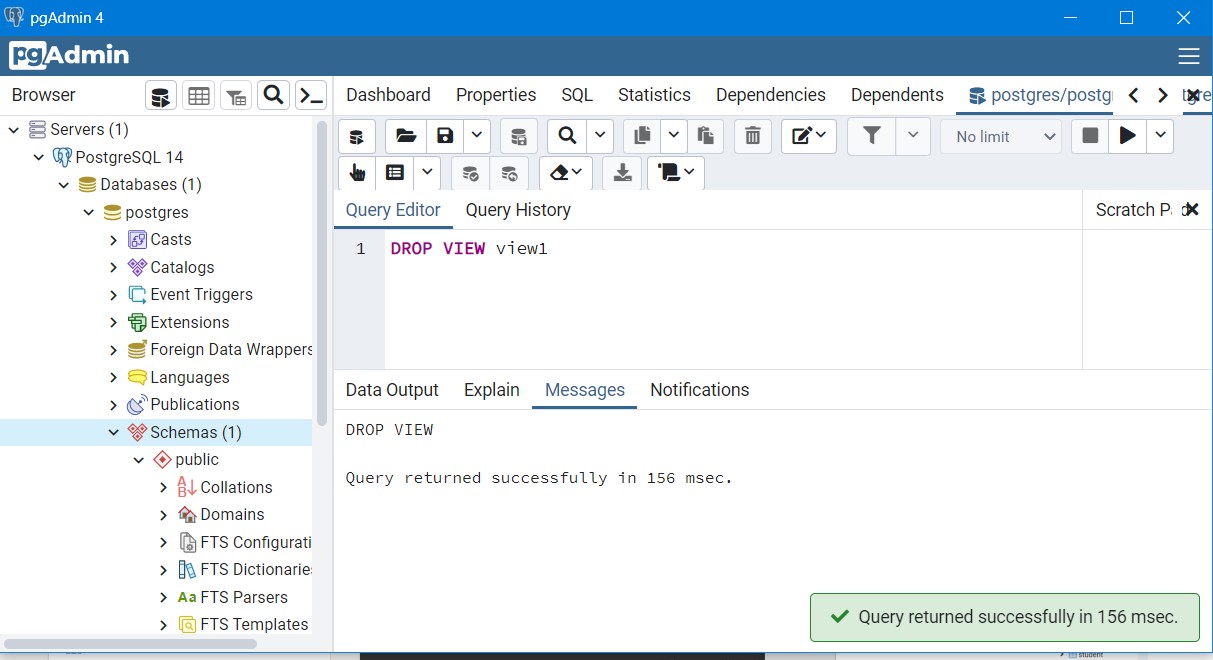


**MIN() :** This function returns the smallest value of the selected column.



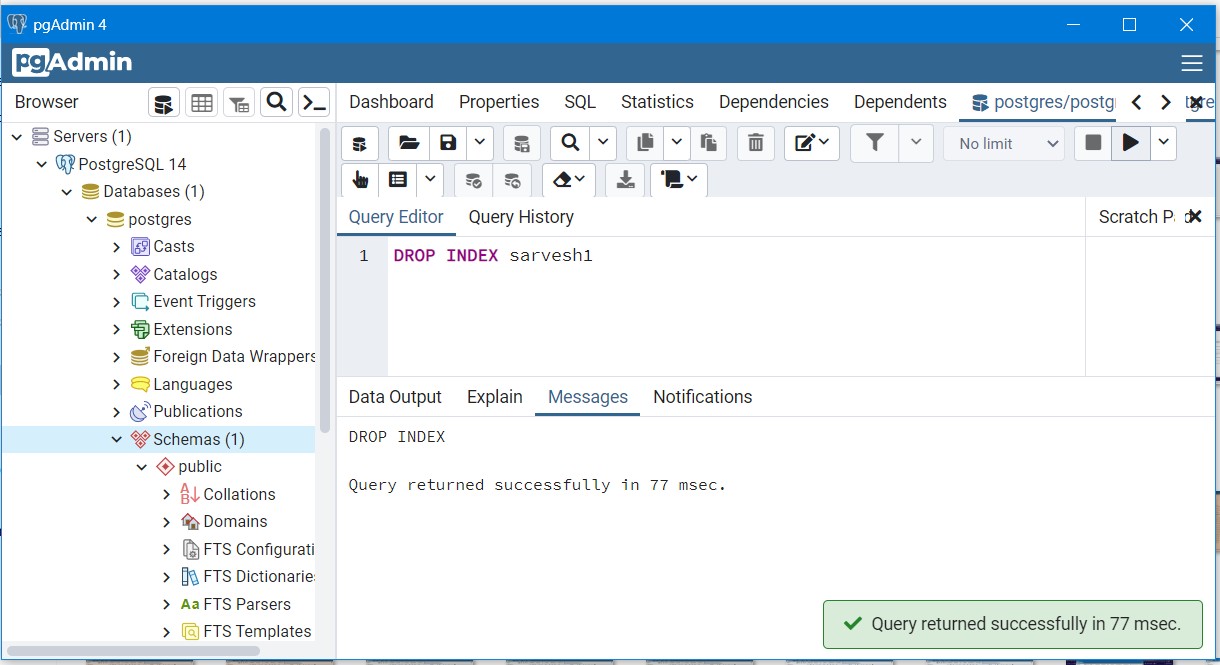
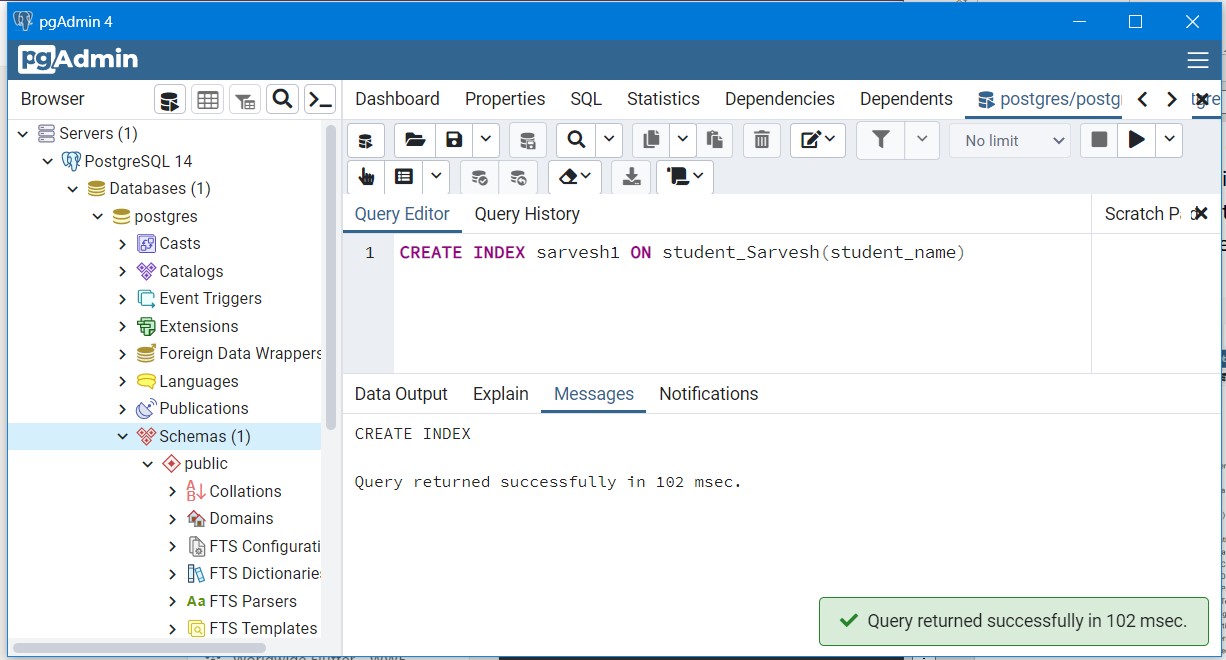
**INDEXES and VIEWS :**

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.



**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.



# 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’

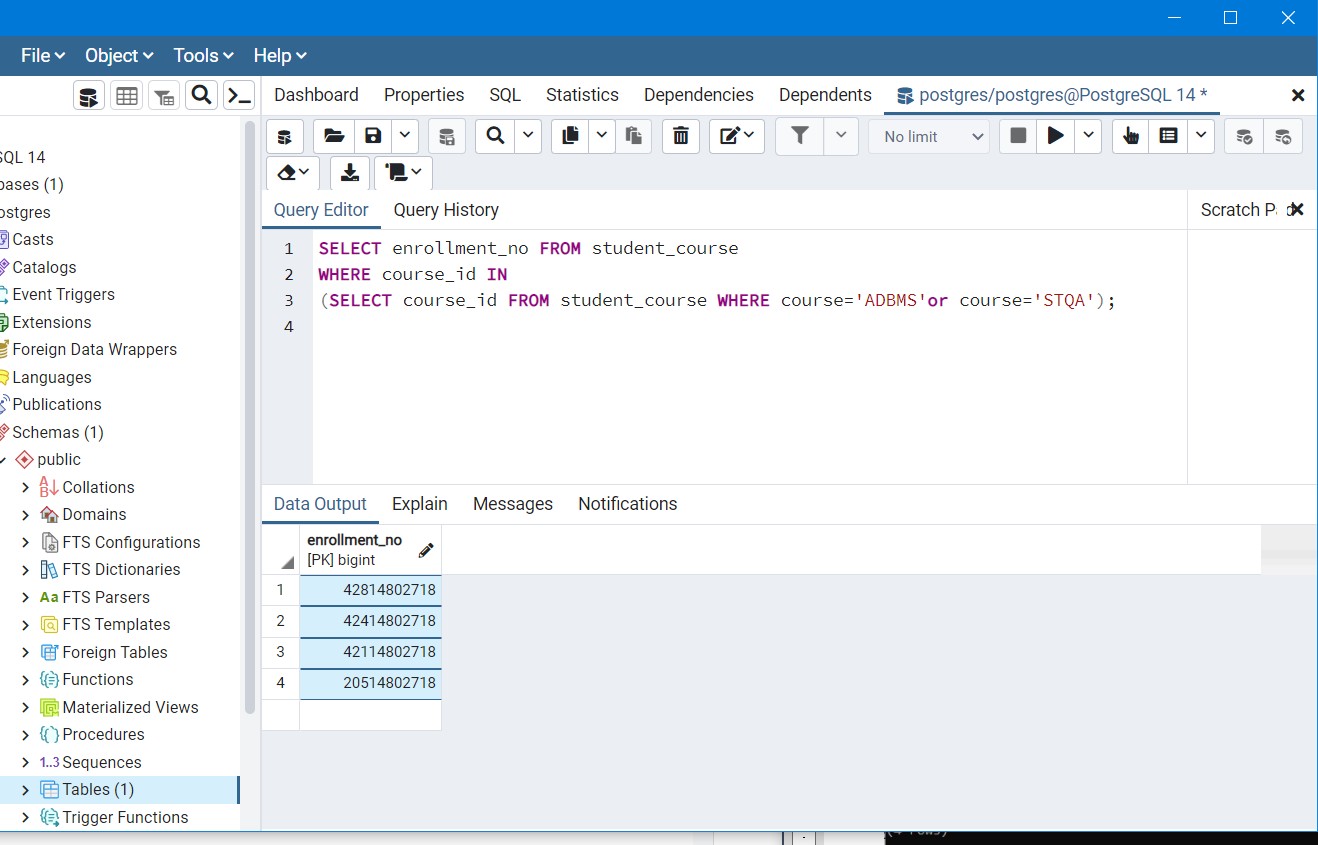
*Select COURSE\_ID from STUDENT\_COURSE where COURSE = ‘ADBMS’ or COURSE = ‘STQA’*



## IN:

*SELECT enrollment\_no FROM student\_course WHERE course\_id IN*

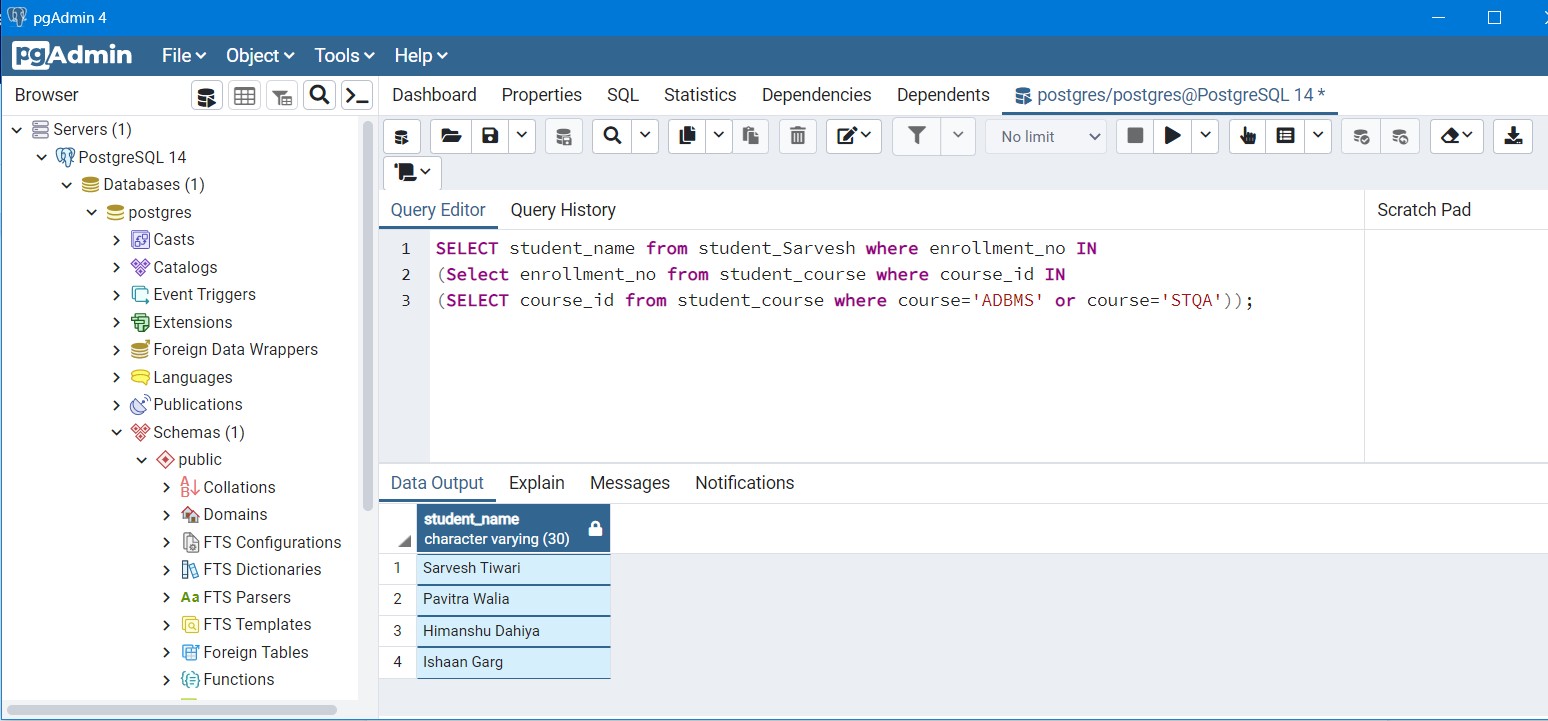
*(SELECT course\_id FROM student\_course WHERE course='ADBMS'or course='STQA');*



**Note:** If we want to find out names of **STUDENT**s 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*

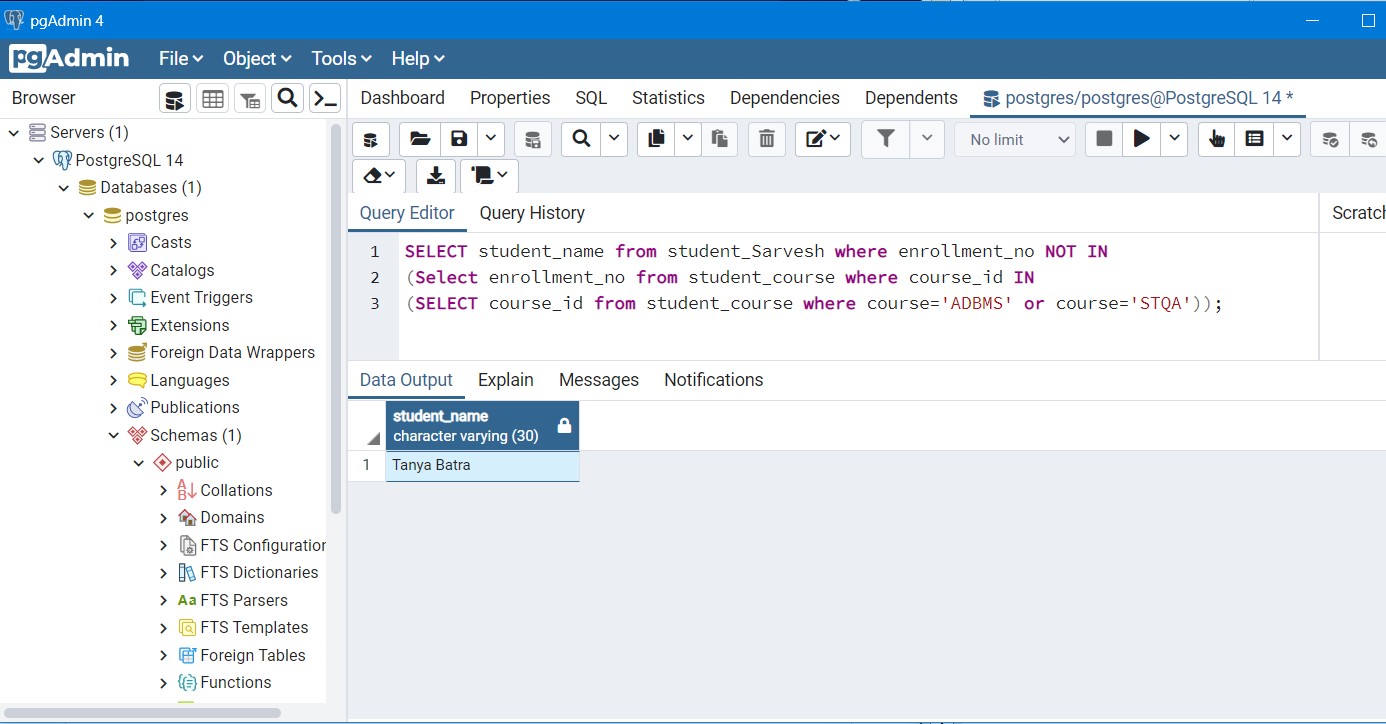
*(SELECT course\_id from student\_course where course='ADBMS' or C\_NAME='STQA'));*



* **NOT IN:** If we want to find out **enrollment\_no**s of **STUDENT**s 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'));*



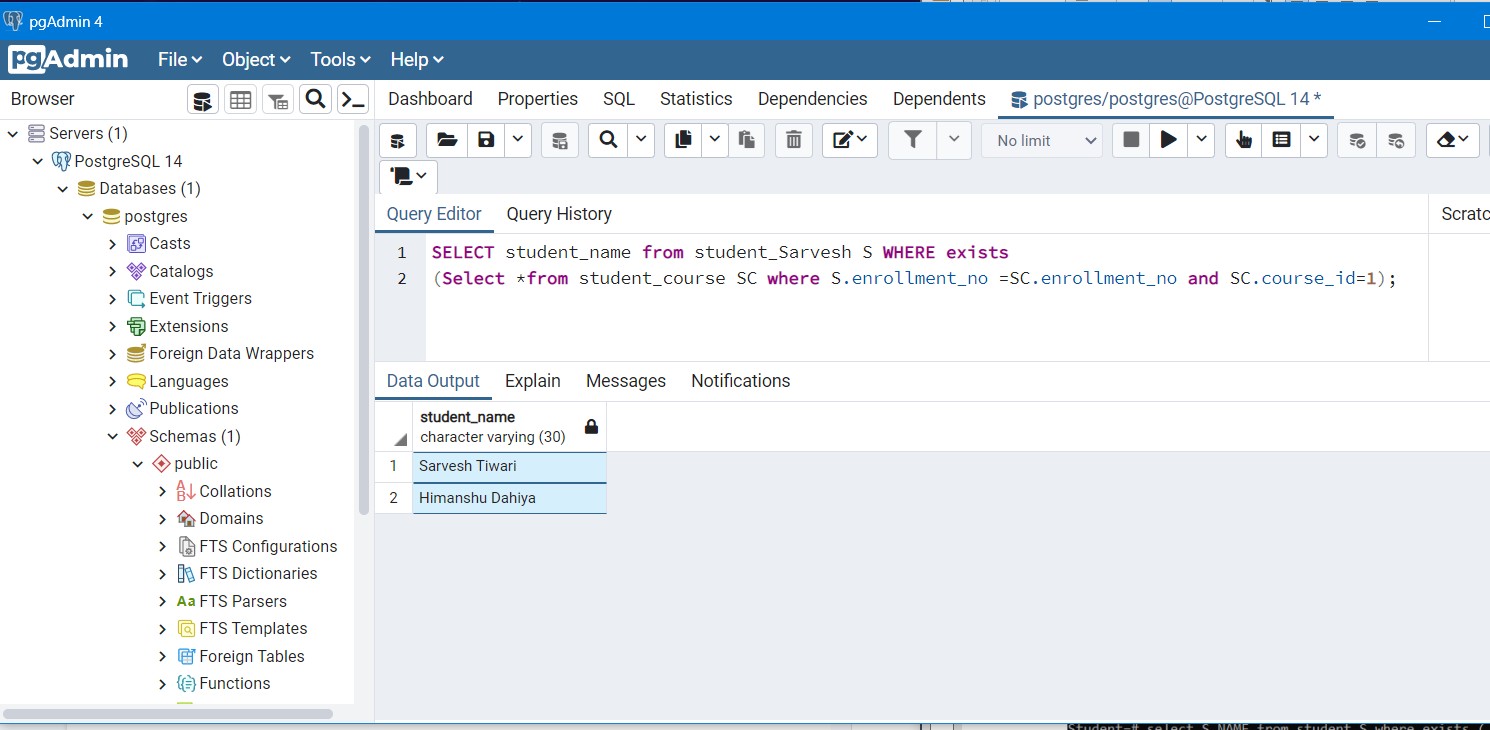
## EXISTS:

If we want to find out **STUDENT\_NAME** of **STUDENT**s who are enrolled in **COURSE\_ID**

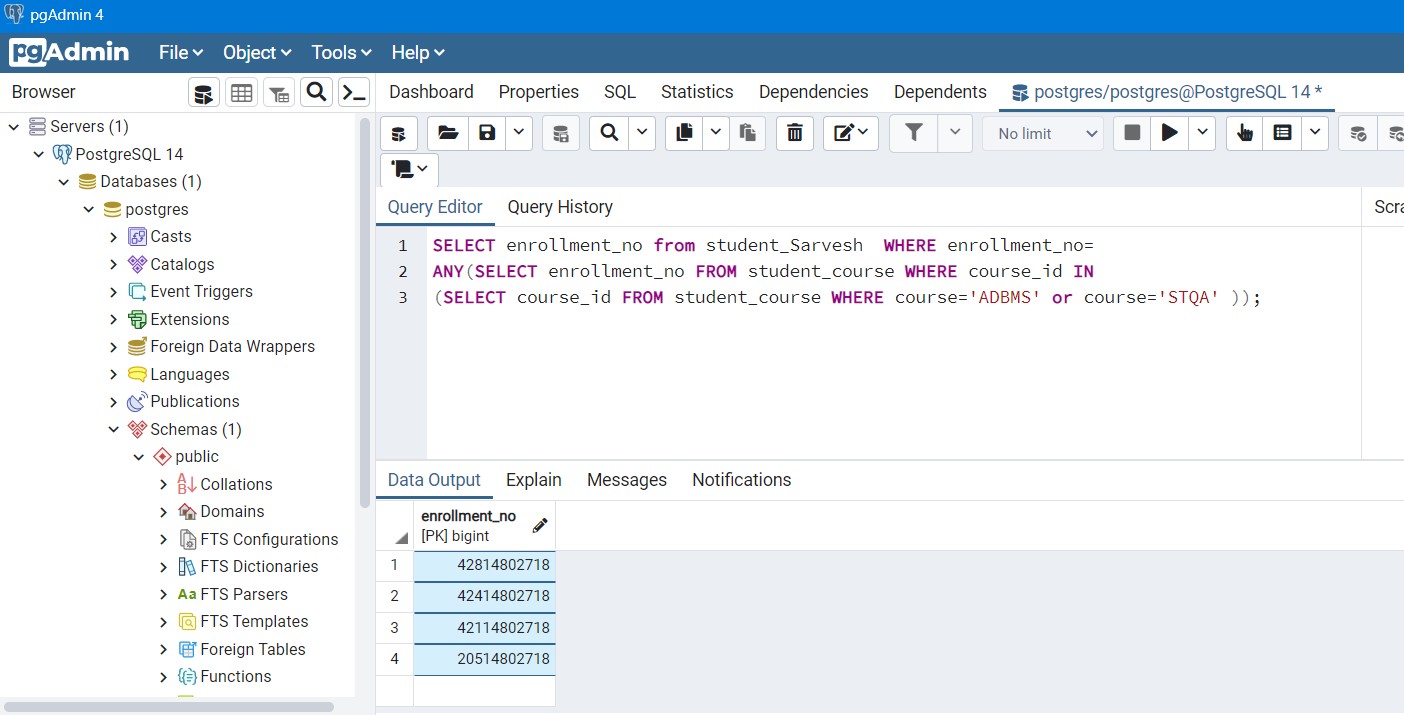
‘1’, it can be done with the help of co-related nested query as:

*SELECT student\_name from student\_Sarvesh S WHERE exists*

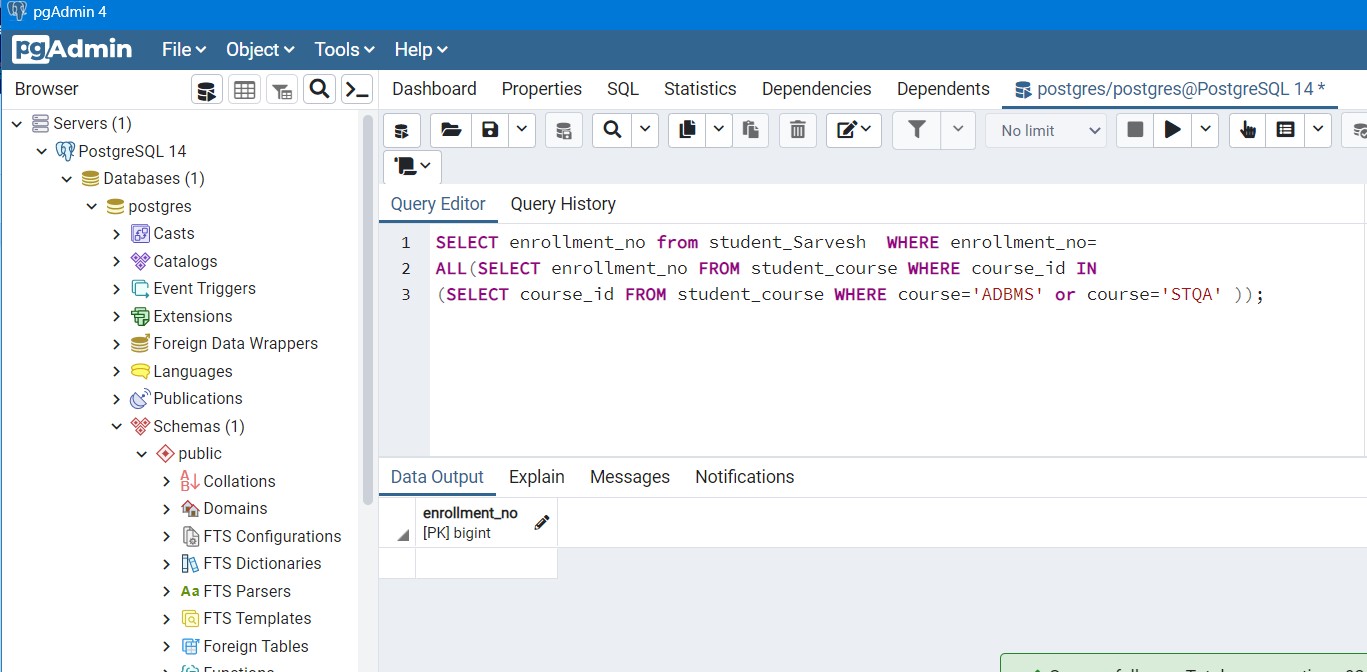
*(Select \*from student\_course SC where S.enrollment\_no =SC.enrollment\_no and SC.course\_id='C1');*



## ANY:



* **ALL:**



## 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.

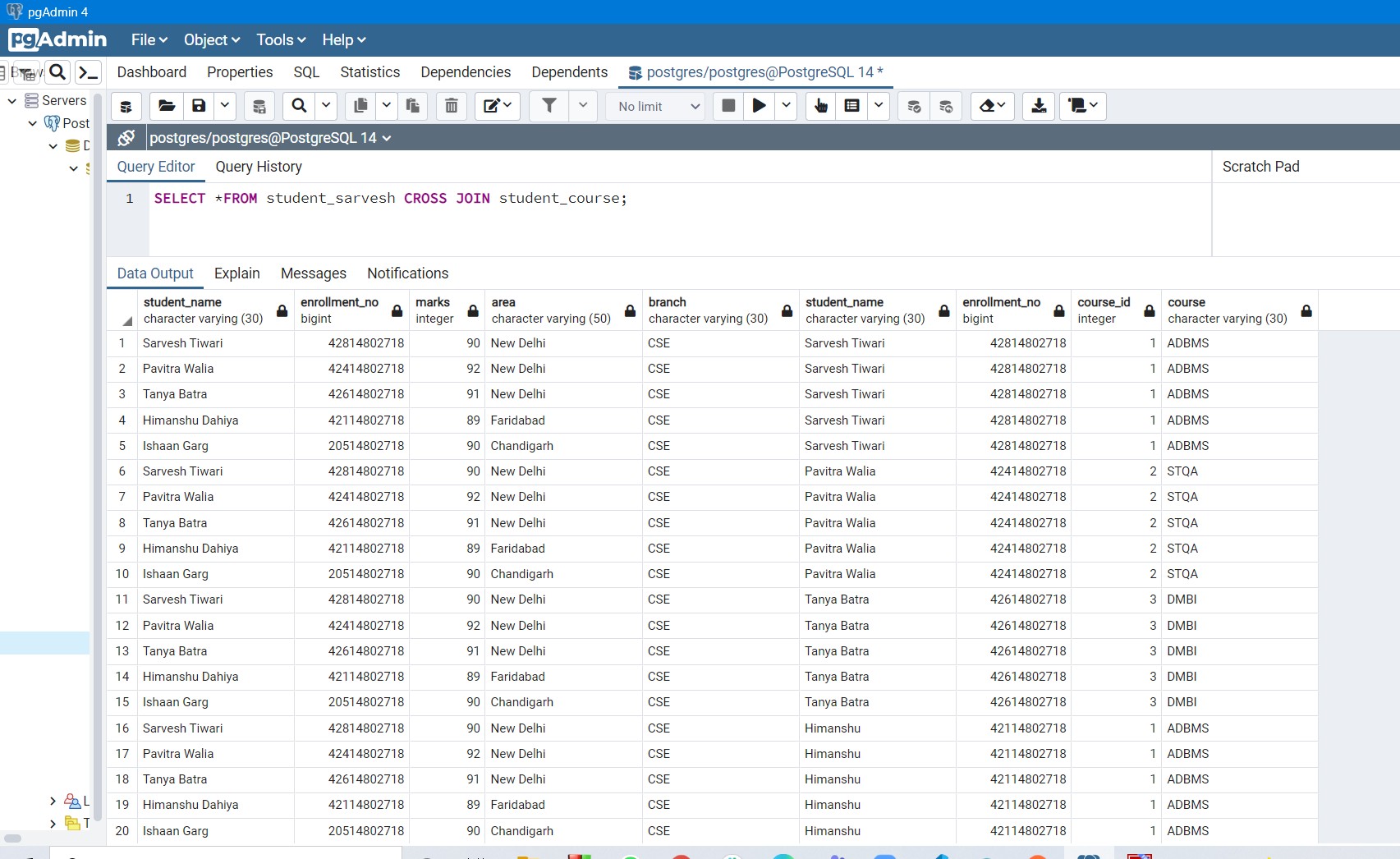
**LEFT OUTER JOIN**: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN

**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*.

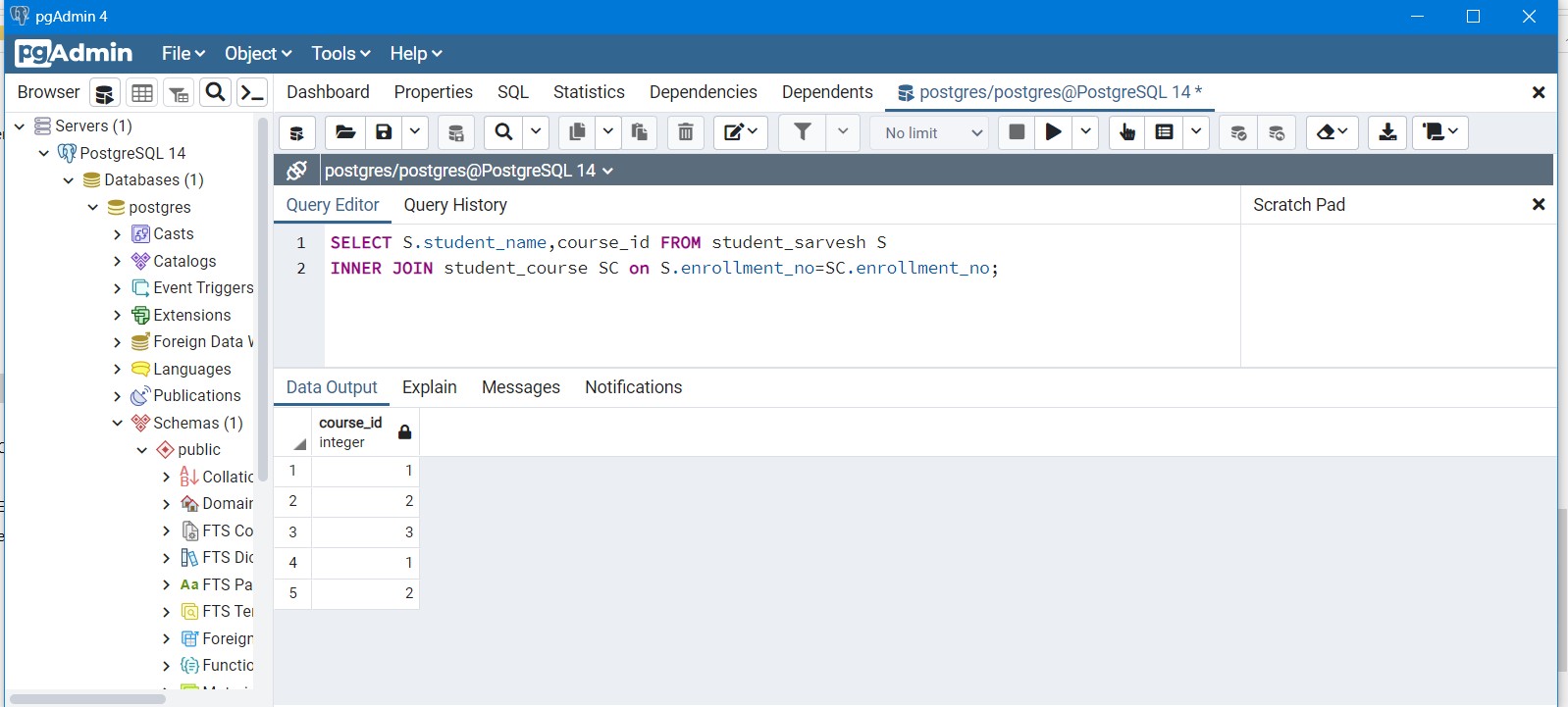
RIGHT JOIN is also known as RIGHT OUTER JOIN.

**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.

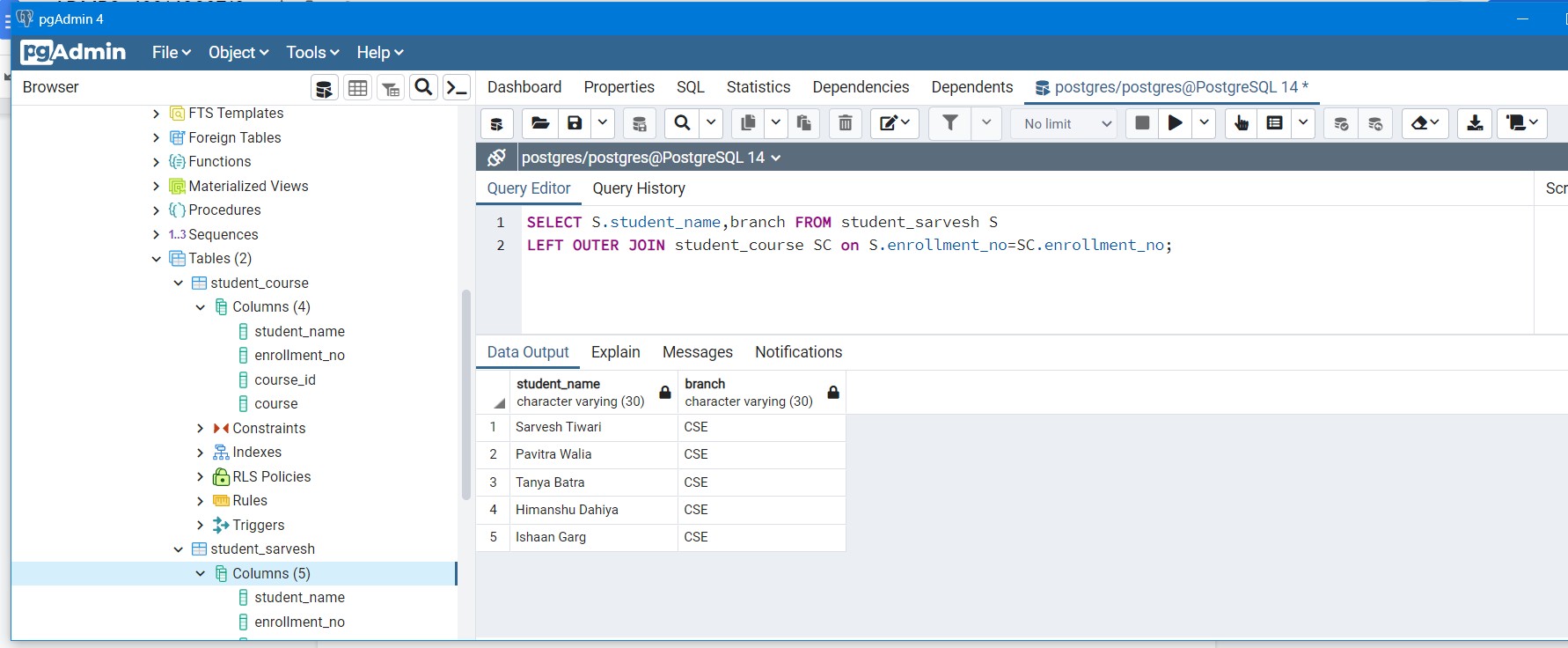
## CROSS JOIN



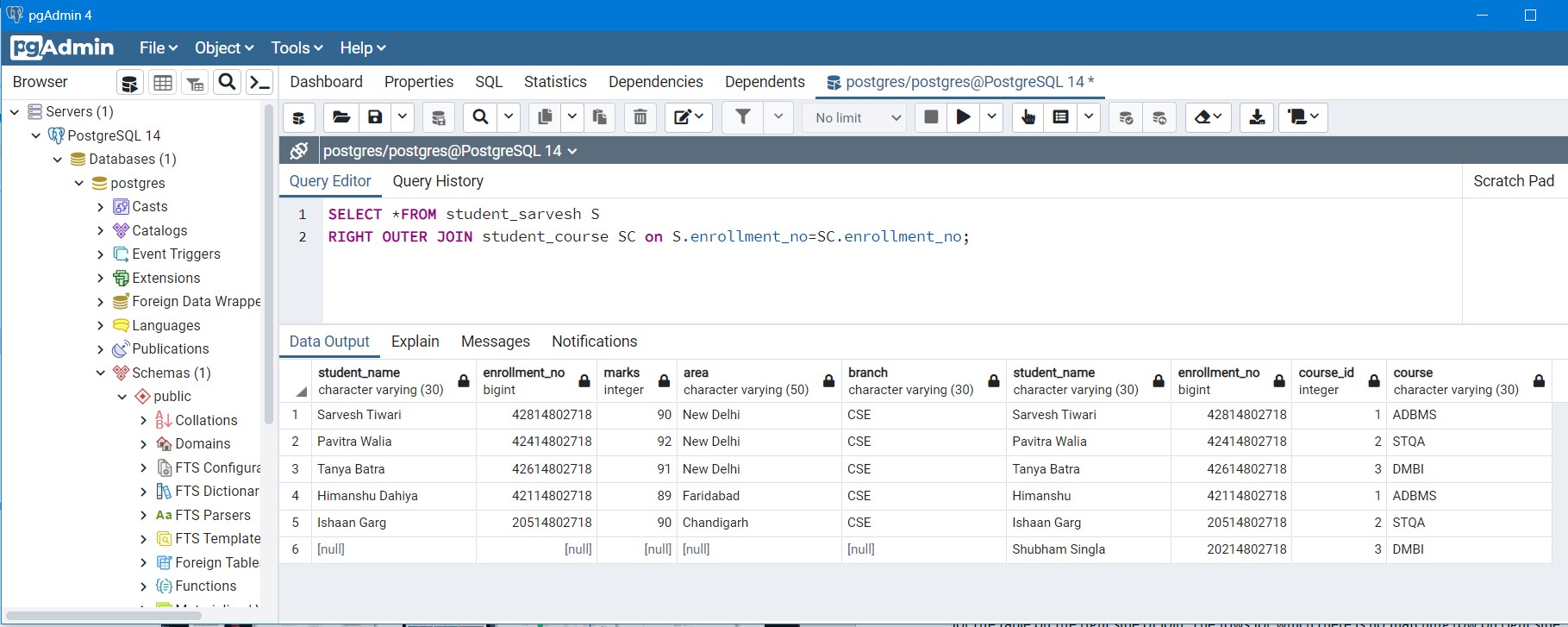
* **INNER JOIN**



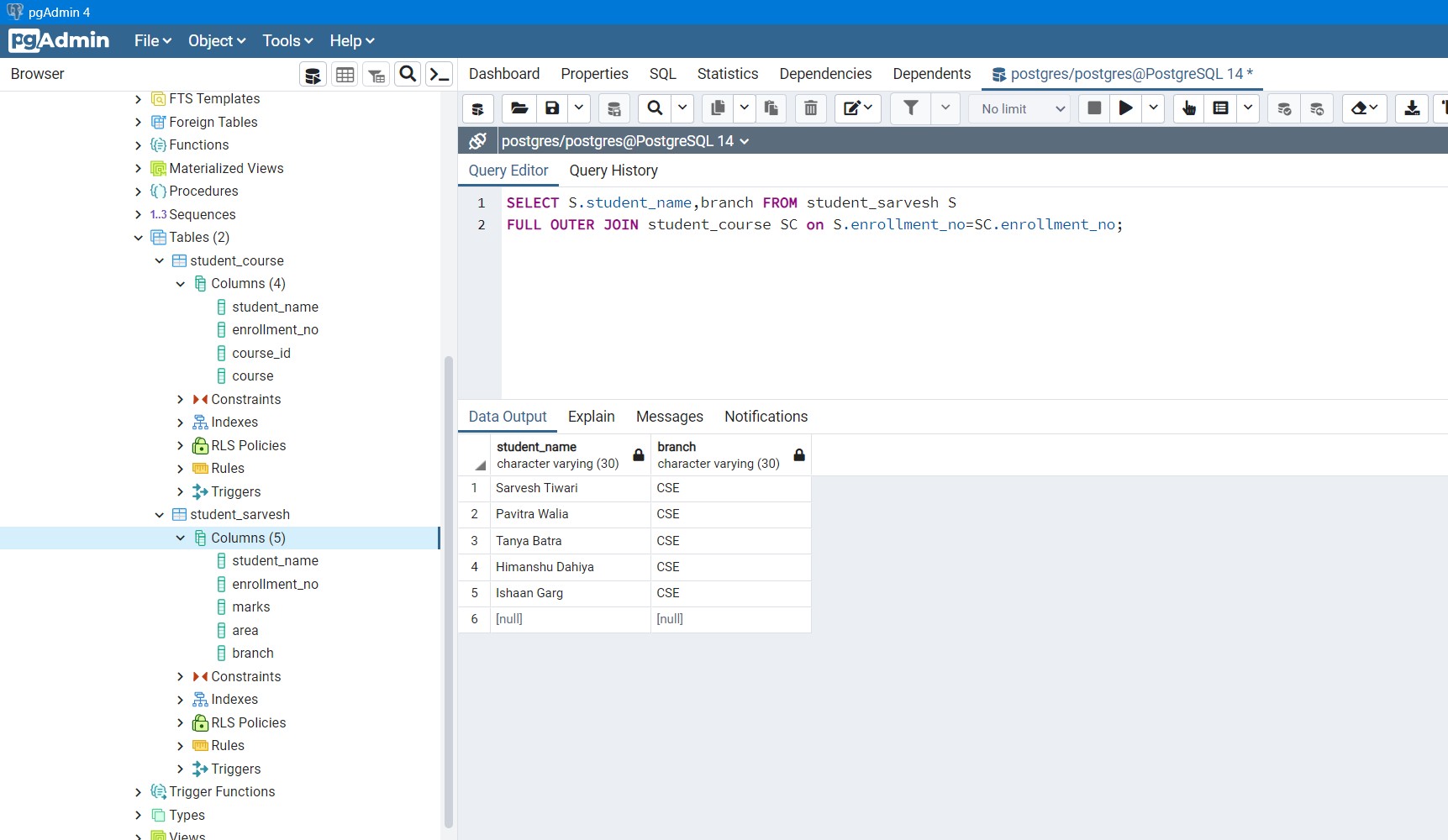
## LEFT OUTER JOIN



* **RIGHT OUTER JOIN**



* **FULL OUTER JOIN**

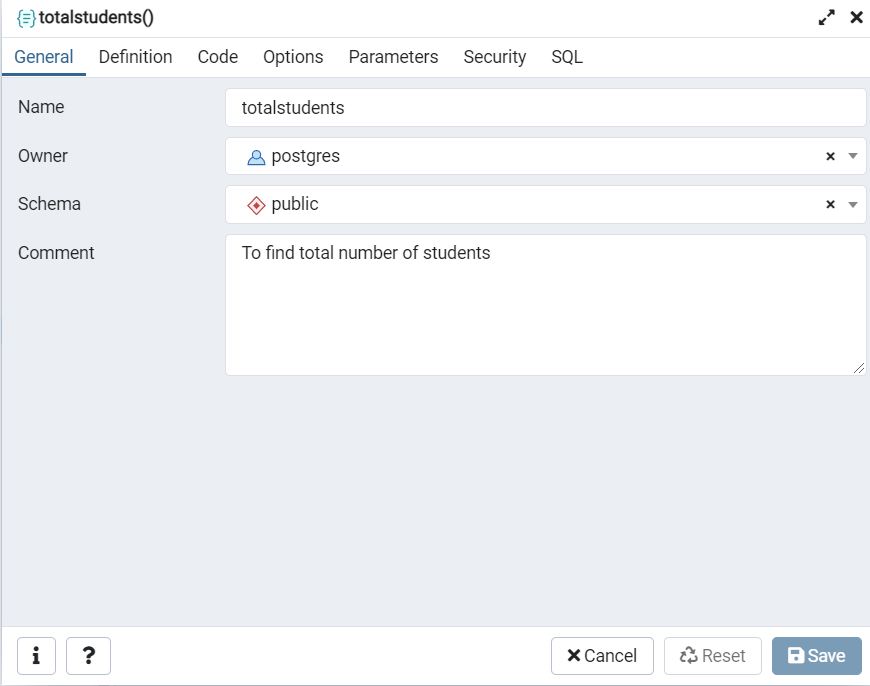


# Experiment 4

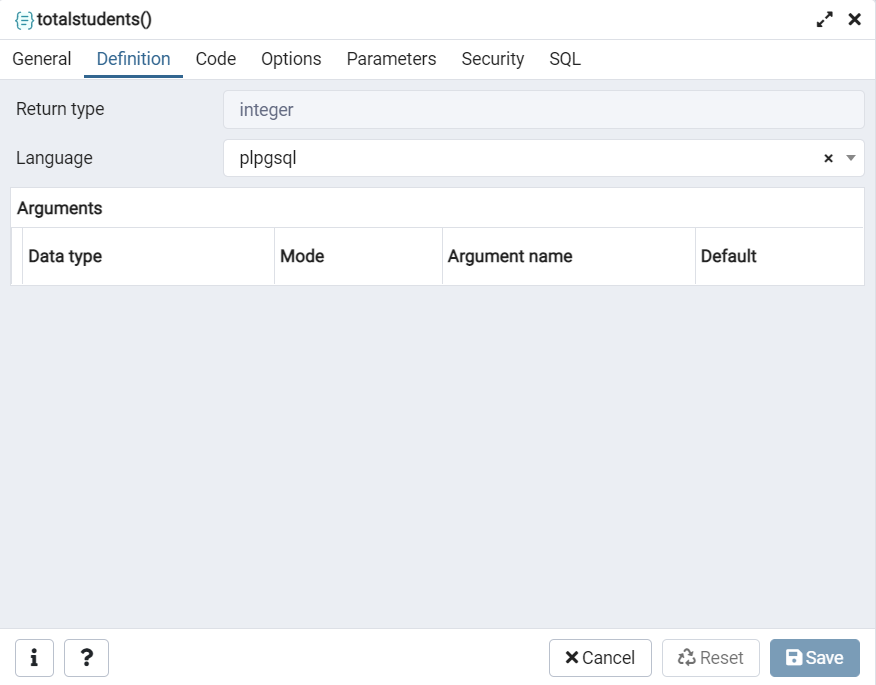
**Aim :**PL/SQL Programs

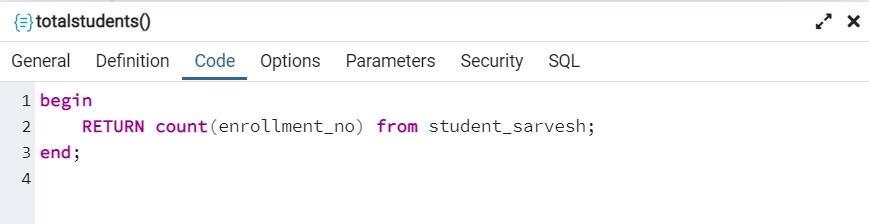
1. Write a program to calculate total students enrolled areawise.

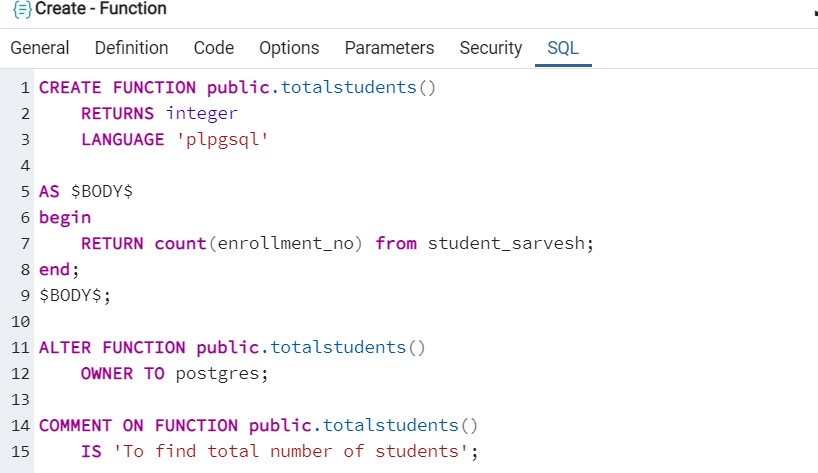
* WE WILL CREATE A FUNCTION **TOTALSTUDENTS** . THIS FUNCTION WILL CALCULATE TOTAL NO OF STUDENTS



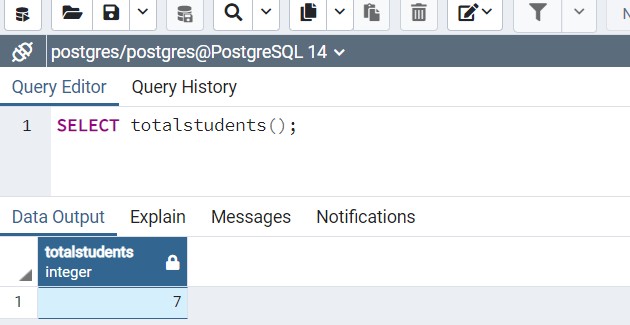
* THEN WE WILL SET THE RETURN TYPE OF THE FUNCTION AS INTEGER AND THEN WE WILL SET THE LANGUAGE



* WE WRITE THE CODE SO THAT THIS FUNCTION RETURNS THE TOTAL NO OF STUDENTS
* THIS IS THE FUNCTION IN SQL.

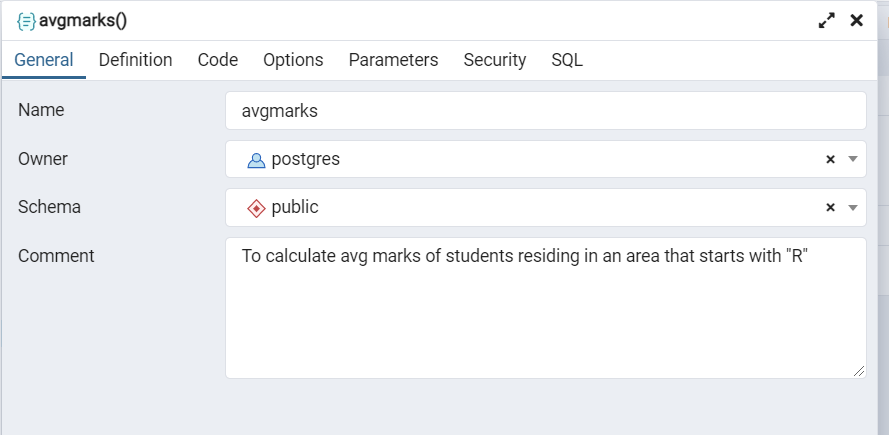


* WE RUN THE FUNCTION USING SELECT FUNCTION NAME().

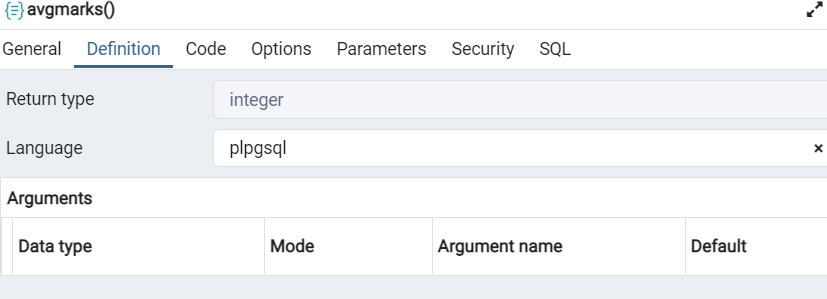


1. Write a program to calculate average marks obtained by the students residing in area that starts with “R”

* We will make a function named **avgmarks**.



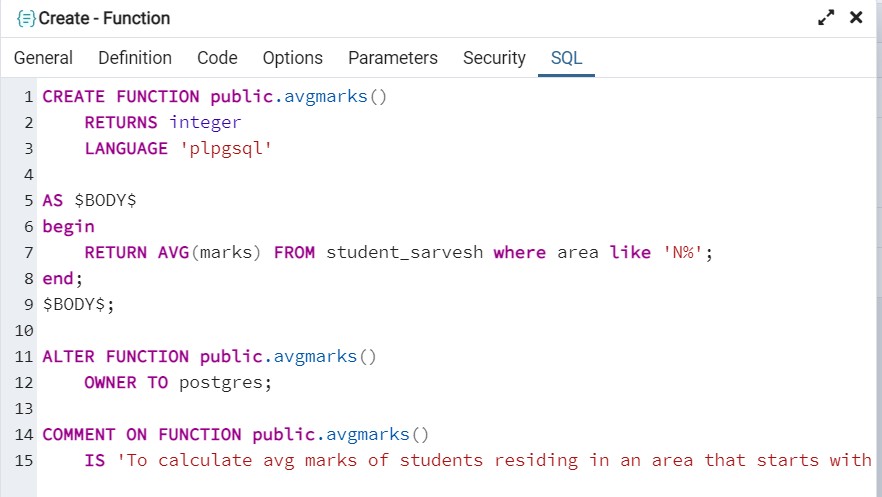
* Now we set the return type and language plpgsql



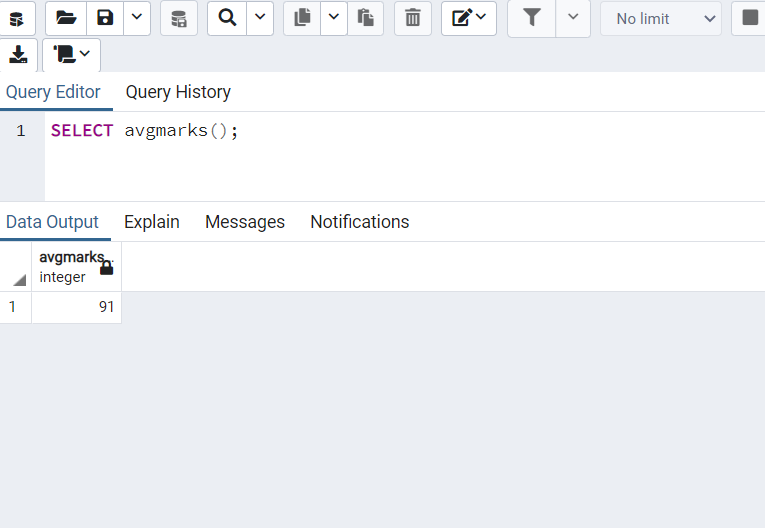
* We write the code for the function avgmarks.



* This is the function in SQL



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



# 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;

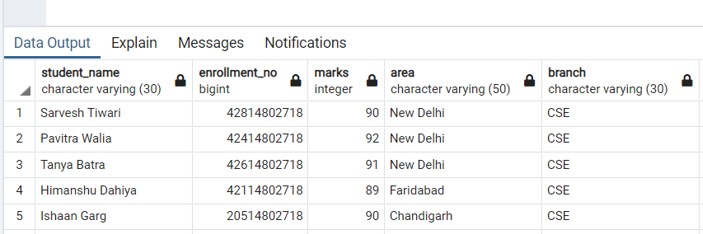
END

$func$ LANGUAGE plpgsql VOLATILE; select increasemarks();

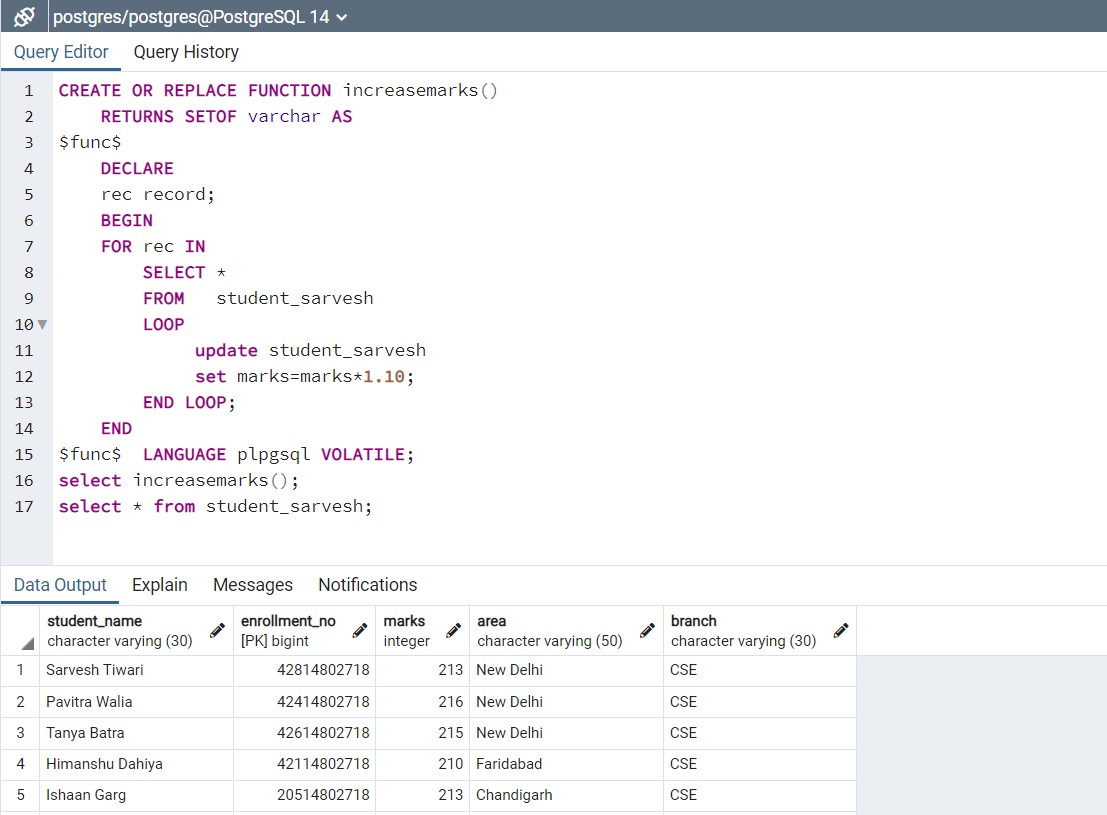
select \* from student\_sarvesh;

## Output:

* + Before increasing the marks of students by 10%.



* After increasing the marks of students by 10%.



# 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:

* 1. 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:

* 1. 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

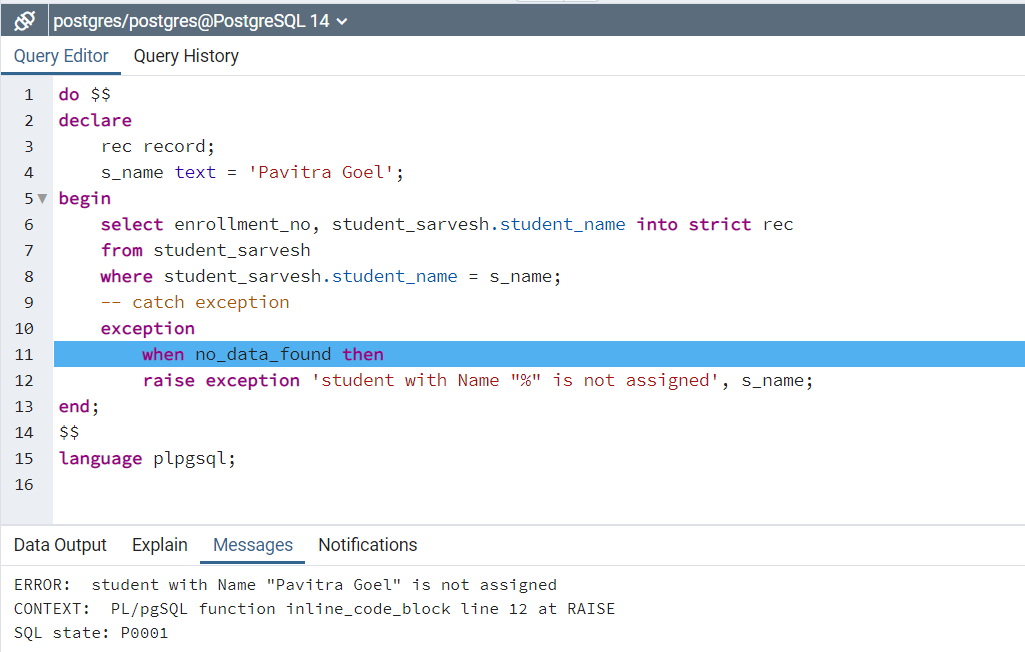
raise exception 'student with Name "%" is not assigned', s\_name;

end;

$$

language plpgsql;

## Output:

.

# Experiment 7

## Aim : Procedure & Function

1. 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

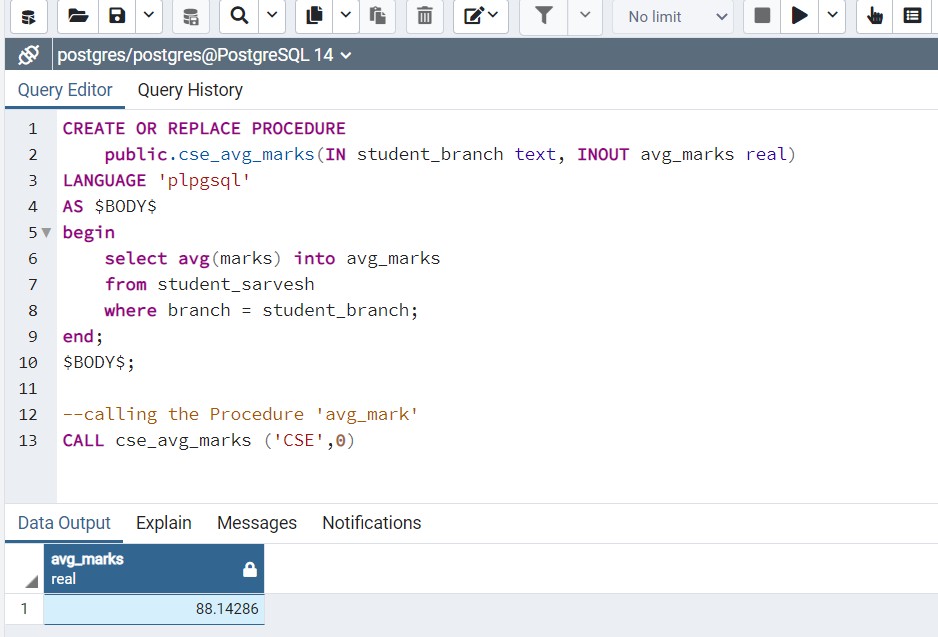
select avg(marks) into avg\_marks from student\_sarvesh

where branch = student\_branch;

end;

$BODY$;

## Output:



1. 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;

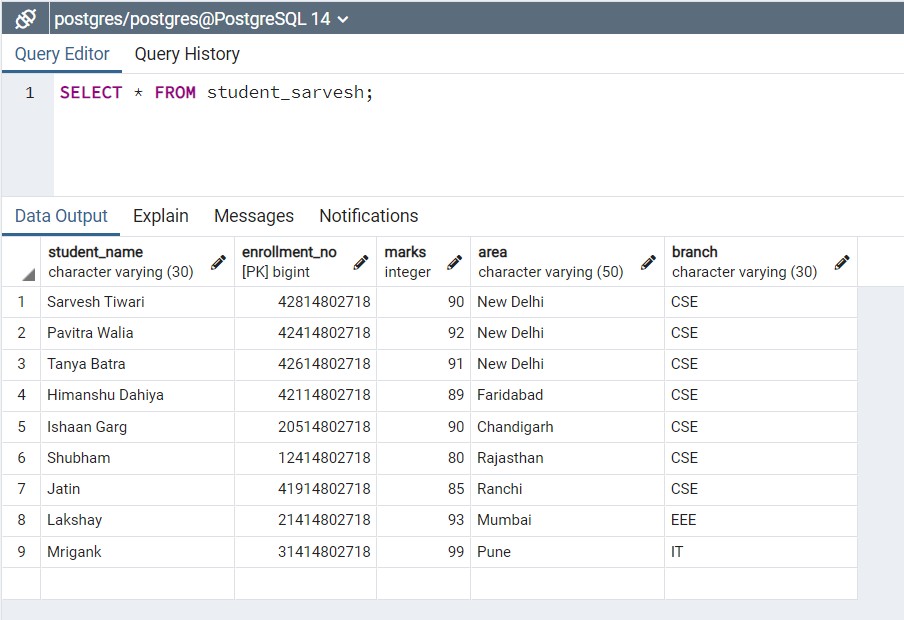
end;

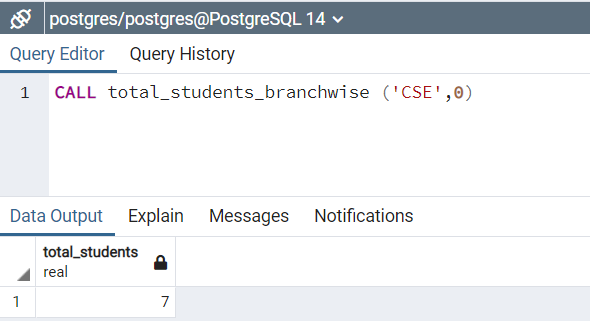
$BODY$;

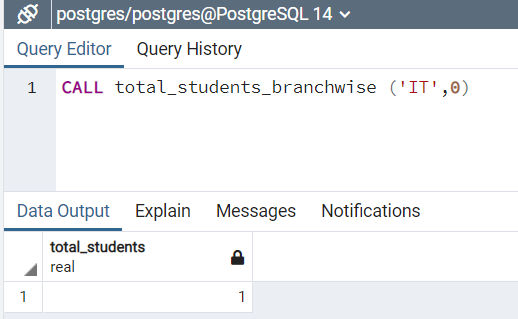
--calling the Procedure 'avg\_mark'

CALL total\_students\_branchwise ('CSE',0)

## Output:

* All entries in the table.
* Total number of students in “CSE” branch



* Total number of students in “IT” branch.

# Experiment 8

## Aim : Trigger

* 1. 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

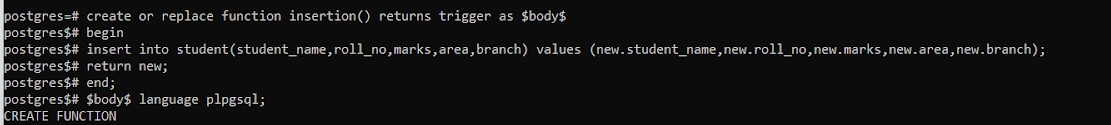
Declaration-statements BEGIN

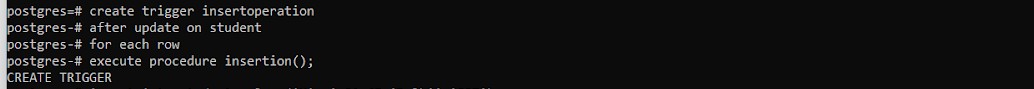
Executable-statements EXCEPTION

Exception-handling-statements END;

## Implementation :

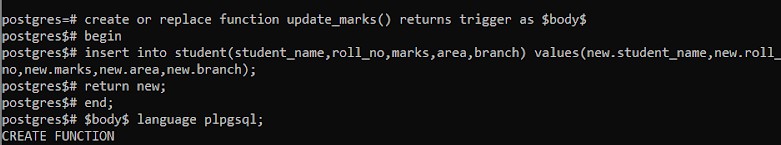
1. **Trigger on the table after inserting a new student into the table. Output:**



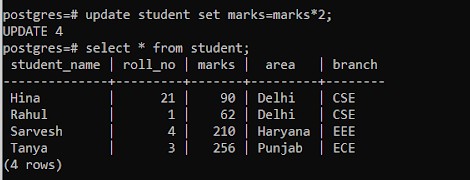




## Row trigger to insert the existing values of the student table into a new table when the marks of the student are updated.

**Output:**





**THANK YOU !**