**ADBMS LAB**

**(ETCS – 457)**

**Faculty name:** Mrs. ZAMEER FATIMA **Student name:** Ayush Pandey

**Roll No.:** 45014802718

**Semester:** 7th Semester

**Group:** 7-C-8



Maharaja Agrasen Institute of Technology

PSP Area, Sector – 22, Rohini, New Delhi – 110085



**MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

**VISION**

To nurture young minds in a learning environment of high academic value and imbibe spiritual and ethical values with technological and management competence.

**MISSION**

**The Institute shall endeavor to incorporate the following basic missions in the teaching methodology:**

**Engineering Hardware – Software Symbiosis**

Practical exercises in all Engineering and Management disciplines shall be carried out by Hardware equipment as well as the related software enabling deeper understanding of basic concepts and encouraging inquisitive nature.

**Life – Long Learning**

The Institute strives to match technological advancements and encourage students to keep updating their knowledge for enhancing their skills and inculcating their habit of continuous learning.

**Liberalization and Globalization**

The Institute endeavors to enhance technical and management skills of students so that they are intellectually capable and competent professionals with Industrial Aptitude to face the challenges of globalization.

**Diversification**

The Engineering, Technology and Management disciplines have diverse fields of studies with different attributes. The aim is to create a synergy of the above attributes by encouraging analytical thinking.

**Entrepreneurship**

The Institute strives to develop potential Engineers and Managers by enhancing their skills and research capabilities so that they become successful entrepreneurs and responsible citizens.



**MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY**

**COMPUTER SCIENCE AND ENGINEERING DEPARTMENT**

**VISION**

To produce “Critical Thinkers of Innovative Technology”.

**MISSION**

To foster an open, multidisciplinary and highly collaborative research environment for producing world-class engineers capable of providing innovative solutions to real life problems and fulfil societal needs.

**PRACTICAL RECORD**

**PAPER CODE : ETCS-457**

**Name of the student : Ayush Pandey**

**University Roll No. : 45014802718**

**Branch : CSE**

**Group : 7C-8**

**PRACTICAL DETAILS**

1. Experiments according to ADBMS lab syllabus prescribed by GGSIPU

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Exp. No.** | **Experiment Name** | **Marks** | | | | | **Total Marks** | **Signature with Date** |
| **R1** | **R2** | **R3** | **R4** | **R5** |  |  |
| 1.(a) | WAP to show and install PostgreSql. |  |  |  |  |  |  |  |
| 1.(b) | Compare different Databases such as PostgreSQL, Oracle, MySql and Maria DB. |  |  |  |  |  |  |  |
| 2. | Consider Schema: Student (student\_name, enrollment\_no, marks, area, branch).  Write SQL queries on  •DDL (create, alter, drop, rename, truncate),  •DML (Insert, update, delete etc.),  •DCL (Grant, revoke etc.)  •Built-in Functions (sum, min, max, avg, count, lower, upper, trim, len etc.)  •Indexes and views: Create and Drop |  |  |  |  |  |  |  |
| 3. | Write SQL queries on Nested queries and Join. |  |  |  |  |  |  |  |
| 4. | Write a program  a. To calculate total students enrolled area-wise.  b. To calculate average marks obtained by students residing in area that starts with "R" |  |  |  |  |  |  |  |
| 5. | Write the cursor to increase the marks of student by 10%. |  |  |  |  |  |  |  |
| 6. | Write a program to create exceptions if enrollment no. is not issued to student by the university and raise the exception explicitly by using raise command. |  |  |  |  |  |  |  |
| 7. | Procedure & Function  a. Write the procedure to get the average marks of students for branch “CSE”.  b. Write a function that accepts branch and returns the total no. of students of the branch. |  |  |  |  |  |  |  |
| 8. | Trigger  a. Create a trigger on table after inserting a new student into table.  b. Write a row trigger to insert the existing values of the student table in to a new table when the marks of student is updated. |  |  |  |  |  |  |  |

**Date: 16/09/2021**

**Experiment-1(a)**

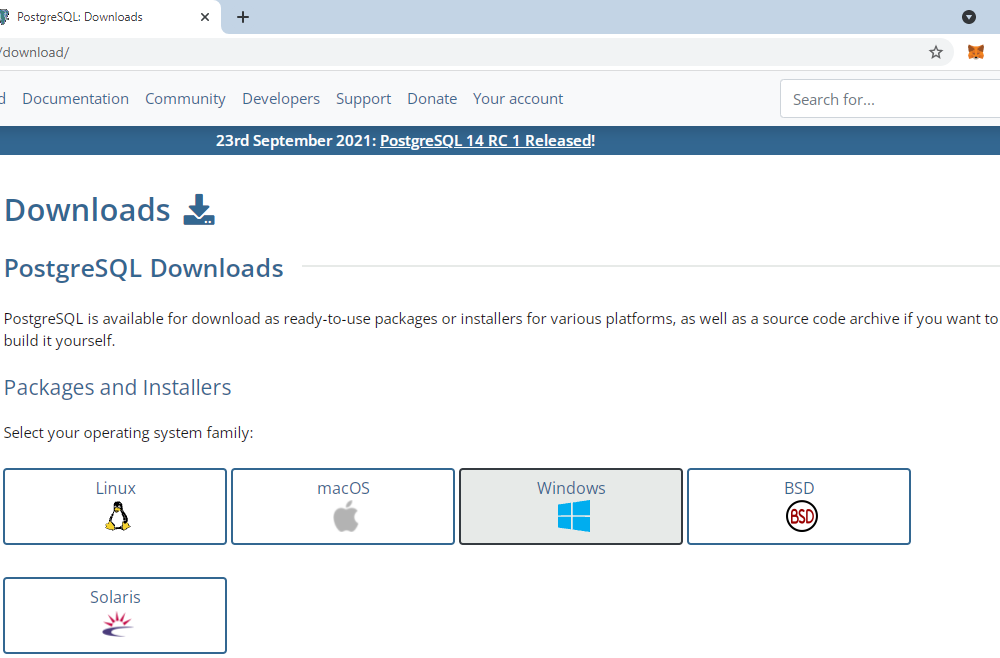
**Aim:** WAP to show and install PostgreSql.

**Theory:**

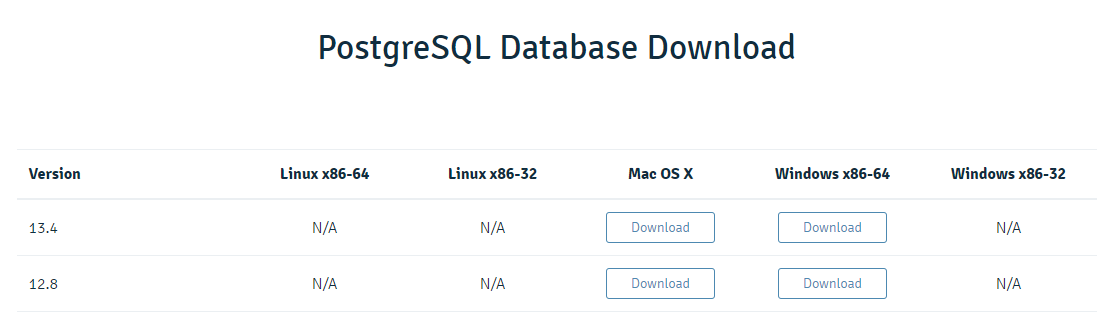
**PostgreSQL**, also known as Postgres, is a free and open-source relational database management system emphasizing extensibility and SQL compliance. It was originally named POSTGRES, referring to its origins as a successor to the Ingres database developed at the University of California, Berkeley.

**Steps to Install:-**

1. Download Postgres Installer here. Postgres Installer is available for PostgreSQL 9.5, 9.6, 10, 11, and 12(beta).

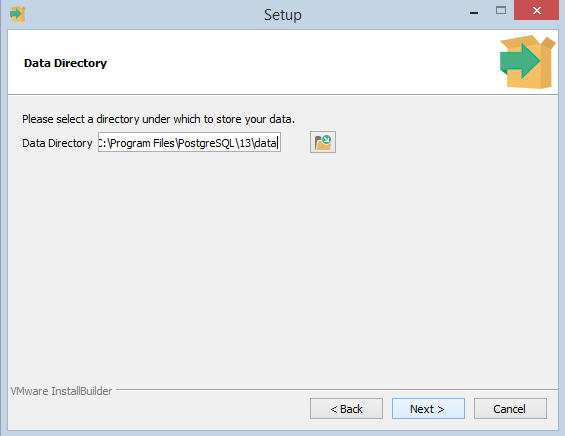


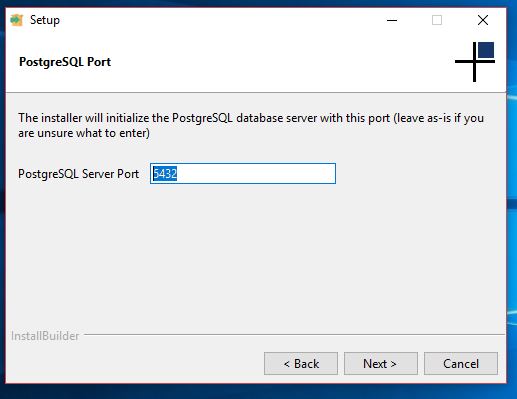
1. Click on the executable file to run the installer. Select your preferred language and the version.



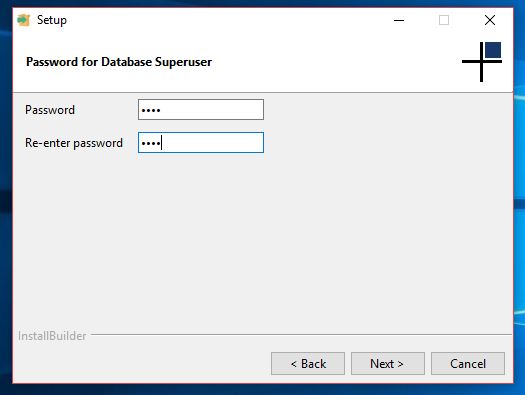


1. Specify directory where you want to install PostgreSQL. Specify PostgreSQL server port. You can leave this as default if you’re unsure what to enter.

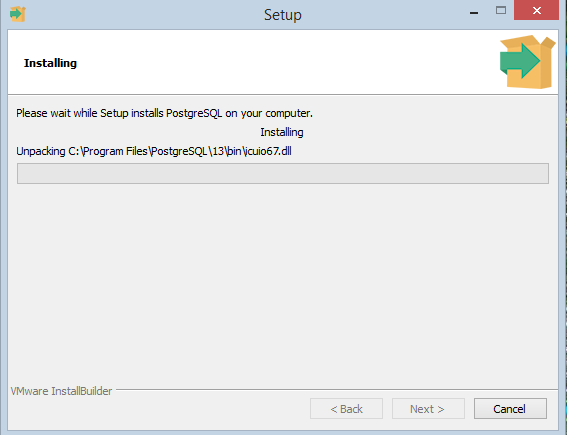


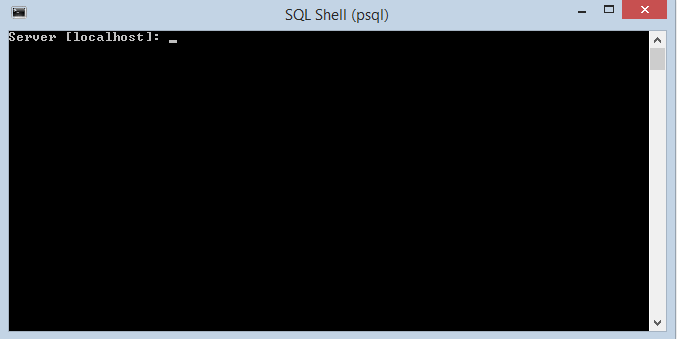


1. Specify data directory to initialize PostgreSQL database. Create a PostgreSQL user password.



1. Click next to begin PostgreSQL installation. Wait for the installation to finish. Launch the kernel at completion.





**Date: 16/09/2021**

**Experiment-1(b)**

**Aim:** Comparison of different databases such as Postgre, Oracle, IBM DB2, MYSql and Maria database.

**Difference between MariaDB and PostgreSQL:**

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

**Difference between IBM DB2 and PostgreSQL:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **IBM DB2** | **PostgreSQL** |
| 1 | IBM DB2 is a relational database model. | PostgreSQL is a object-relational database model |
| 2 | IBM DB2 was developed by IBM in 1983. | PostgreSQL was developed by PostgreSQL Global Development group in 1989. |
| 3 | In IBM DB2 partitioning is done by sharding.. | In IBM DB2 partitioning is done by use of list, hash and range. |
| 4 | It has a commercial license | It is a open source software. |
| 5 | It is written in C and C++ languages | It is written in C++ language |
| 6 | It is a family of database management products given by IBM. | It is a advanced relational DBMS that is a extended form of SQL. |
| 7 | It has less availability as compared to PostgreSQL. | It has more availability as compared to IBM DB2. |

**Difference between Oracle and PostgreSQL:**

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

**Difference between MySQL and PostgreSQL:**

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

**Date: 23/09/2021**

**Experiment-2**

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

Write SQL queries on

• DDL (create, alter, drop, rename, truncate),

• DML (Insert, update, delete etc.),

• DCL (Grant, revoke etc.)

• Built-in Functions (sum, min, max, avg, count, lower, upper, trim, len etc.)

• Indexes and views: Create and Drop

**Theory:**

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

Here are some commands that come under DDL:

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

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

Here are some commands that come under DML:

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

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

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

Here are some commands that come under DCL:

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

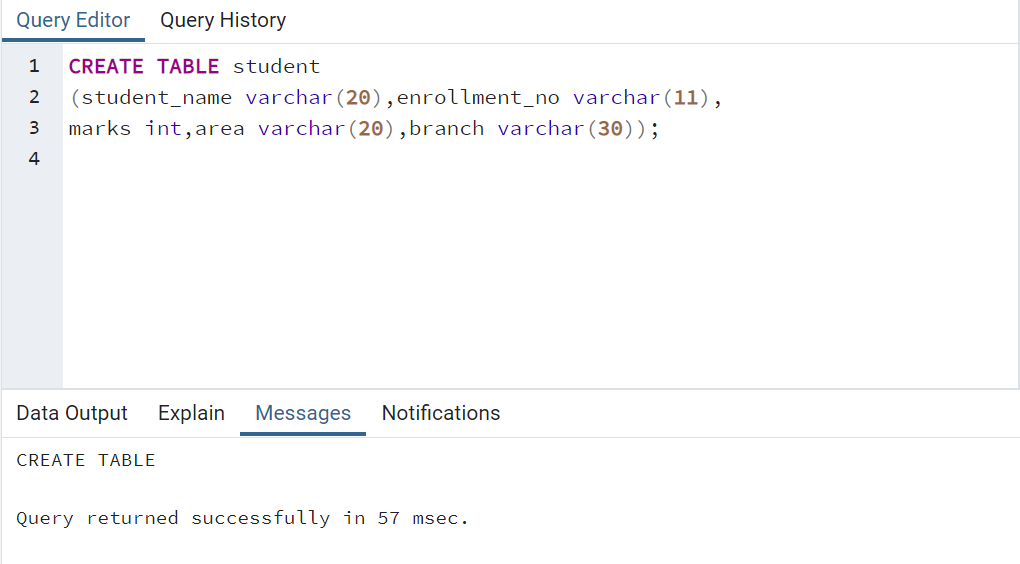
**4. Built In Functions**

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

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

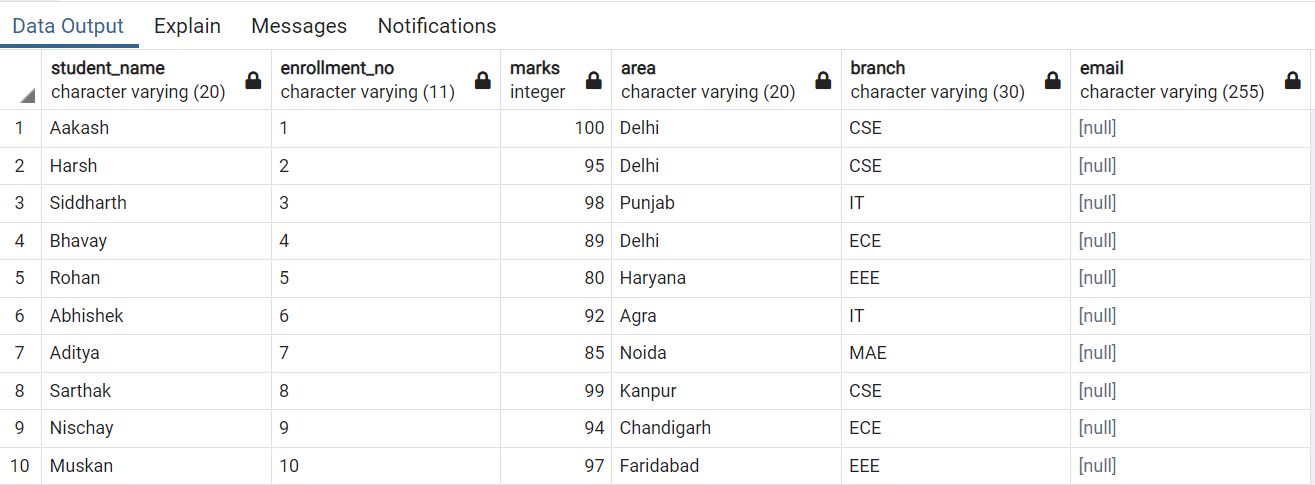
* Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables or another view.
* A view is deleted using the DROP VIEW command.
* **DDL COMMANDS:**

**CREATE TABLE:**

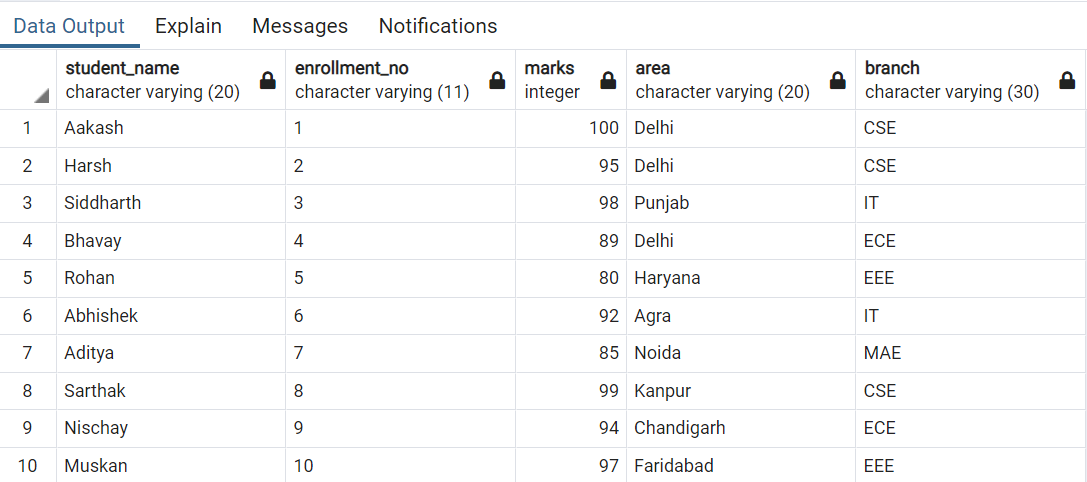


**ALTER TABLE:**

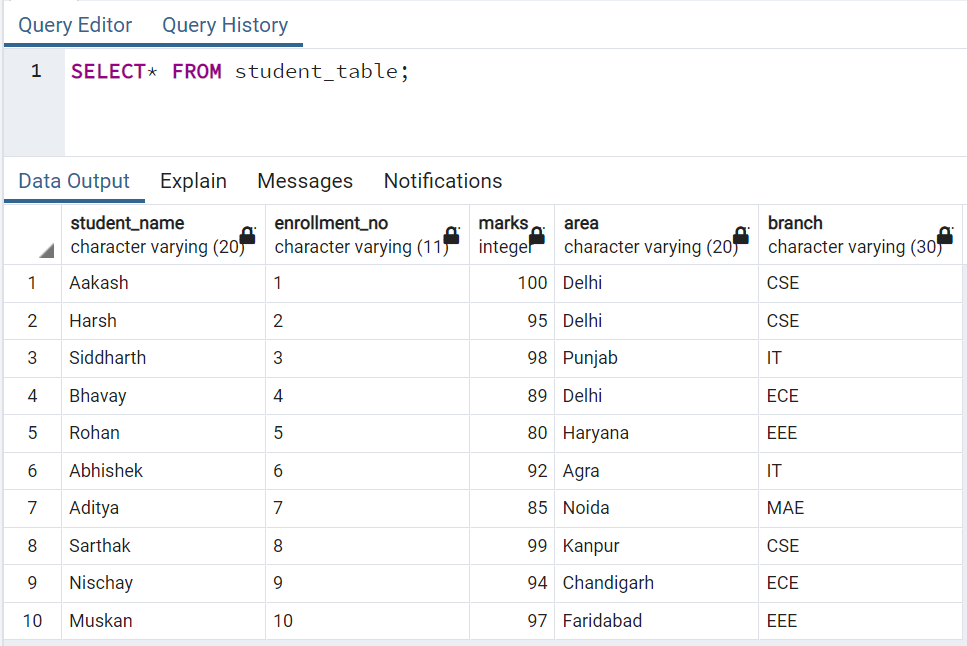
Query: ALTER TABLE student ADD email varchar(255);



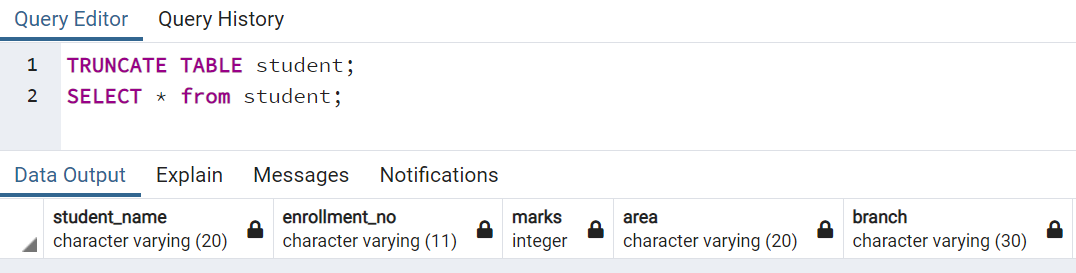
Query: ALTER TABLE student DROP COLUMN email;



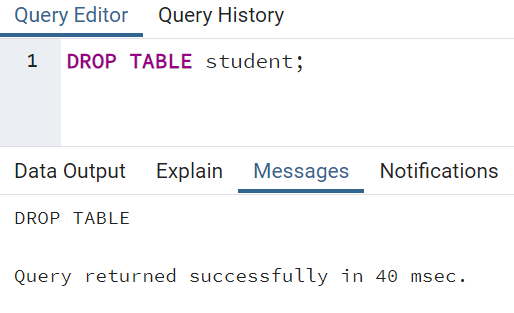
Query: ALTER TABLE student RENAME TO student\_table; **(RENAME TABLE)**



**TRUNCATE TABLE:**



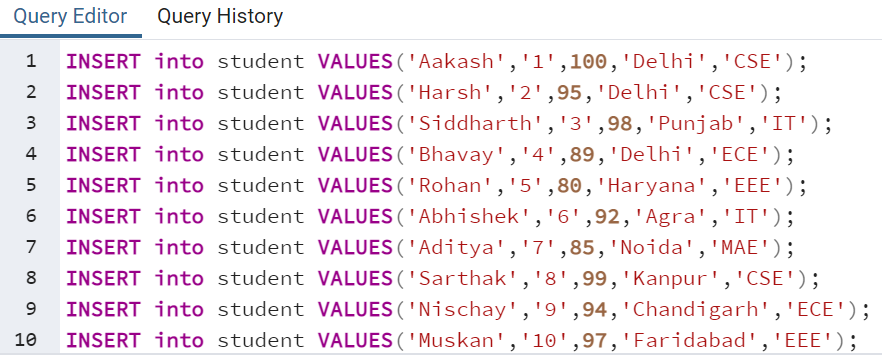
**DROP TABLE:**

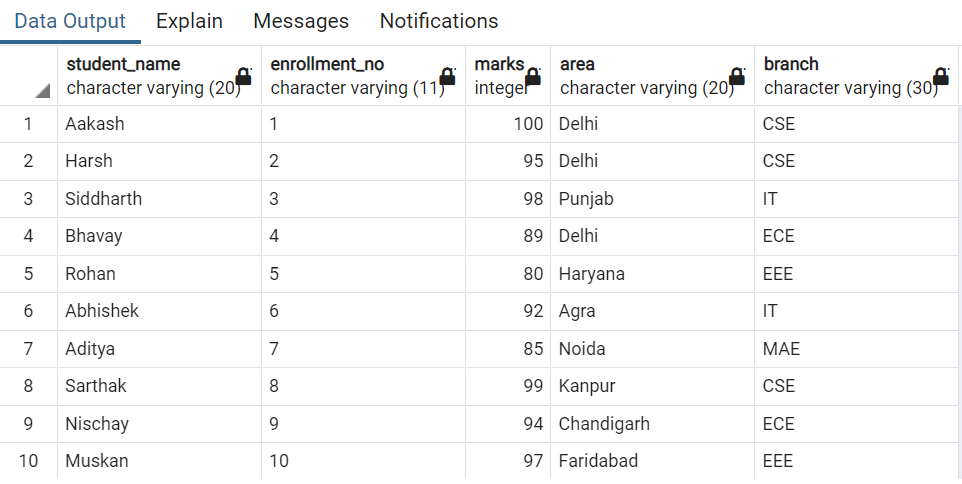


* **DML COMMANDS:**

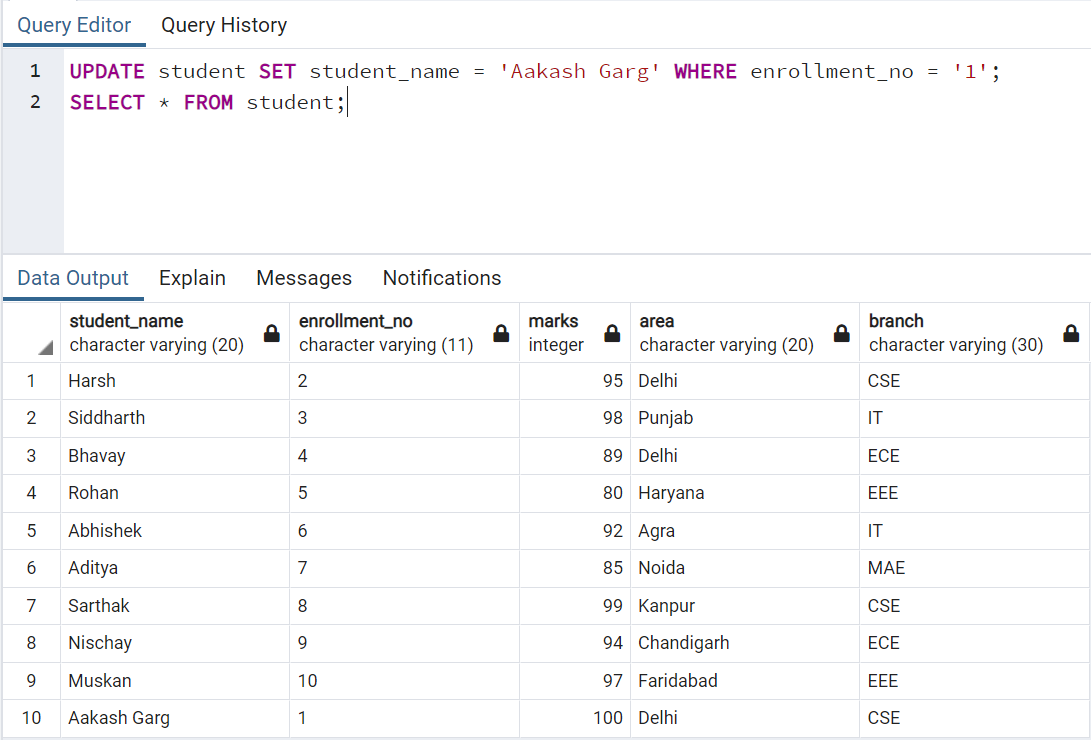
**INSERT INTO TABLE**

**Query:**

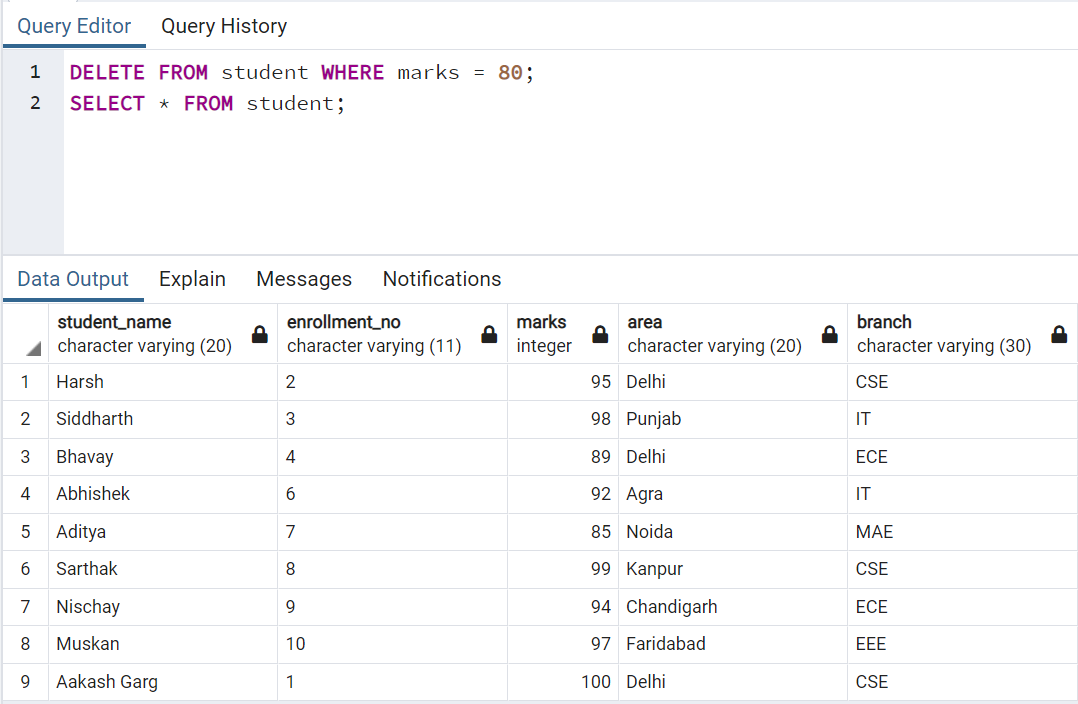




**UPDATE TABLE:**



**DELETE TABLE:**



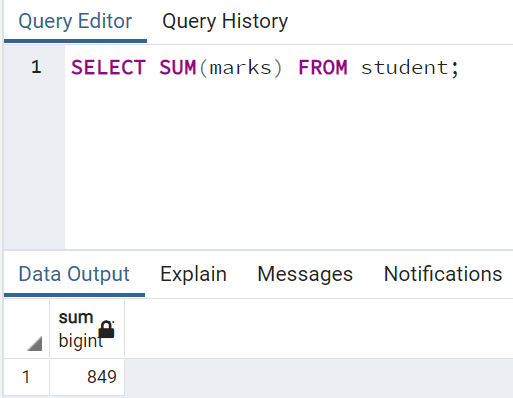
* **DCL COMMANDS:**

**GRANT COMMAND:** SYNTAX**:** grant privilege\_name on object\_nameto {user\_name | public | role\_name}

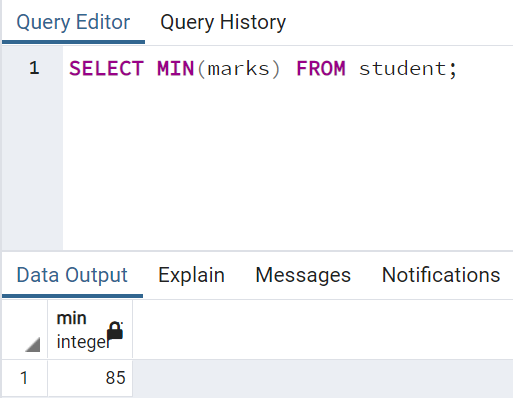
**REVOKE COMMAND:** SYNTAX: revoke privilege\_name on object\_name from {user\_name | public | role\_name}

* **BUILT IN FUNCTIONS:**

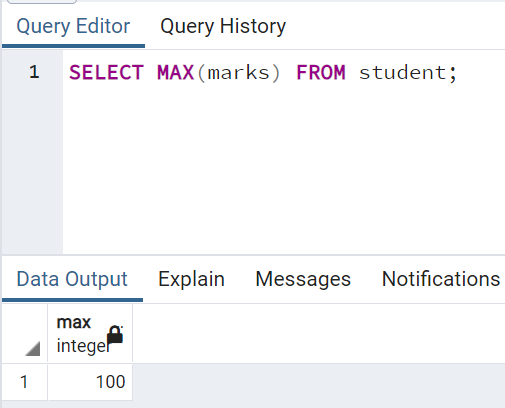
**SUM:**



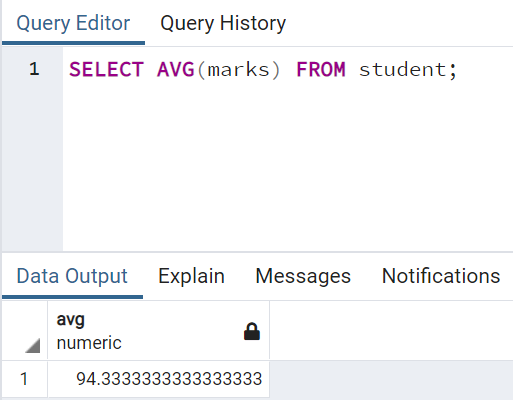
**MIN:**



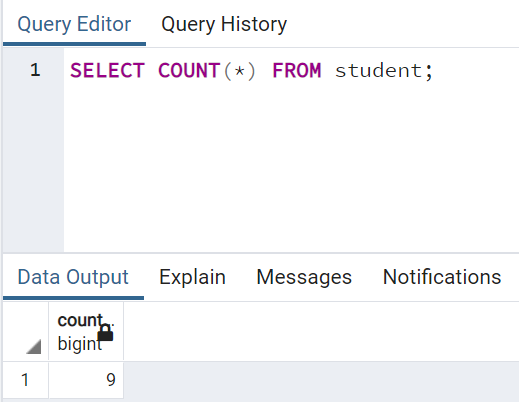
**MAX:**



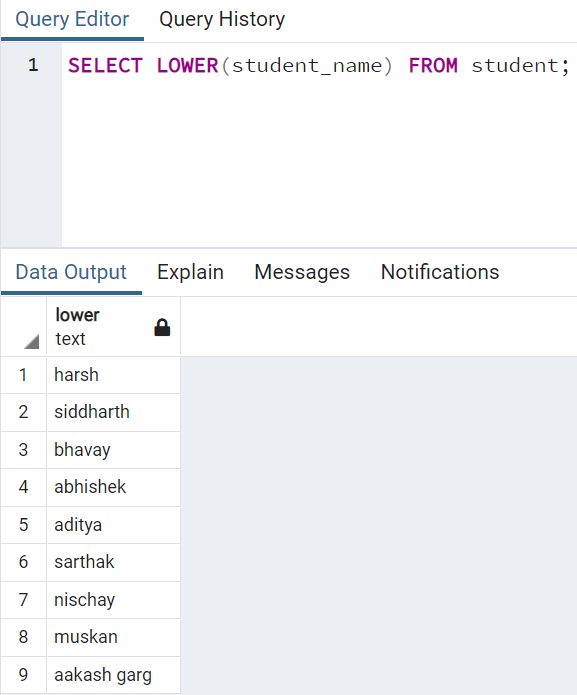
**AVG:**



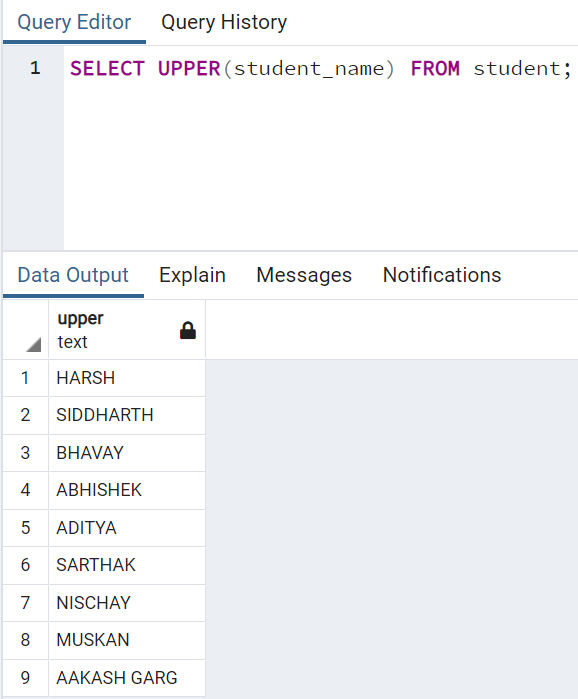
**COUNT:**



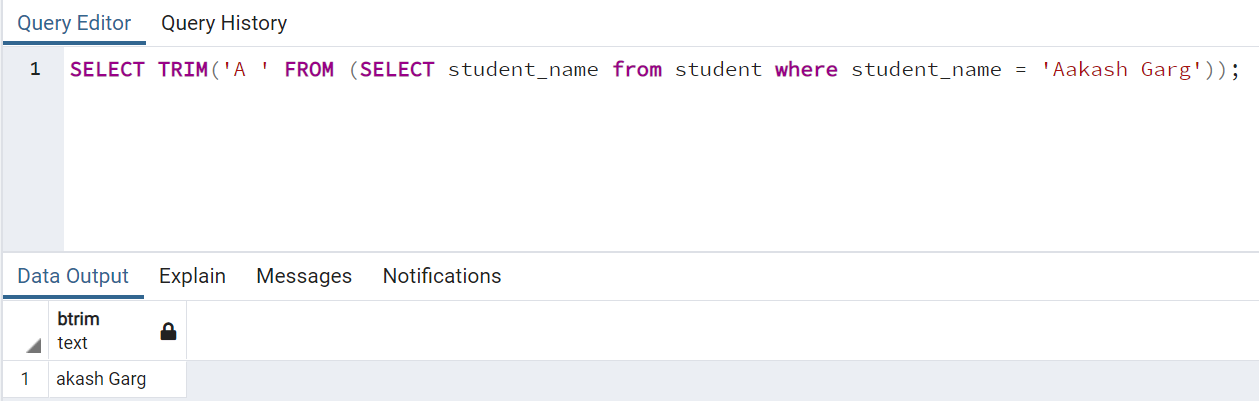
**LOWER:**



**UPPER:**

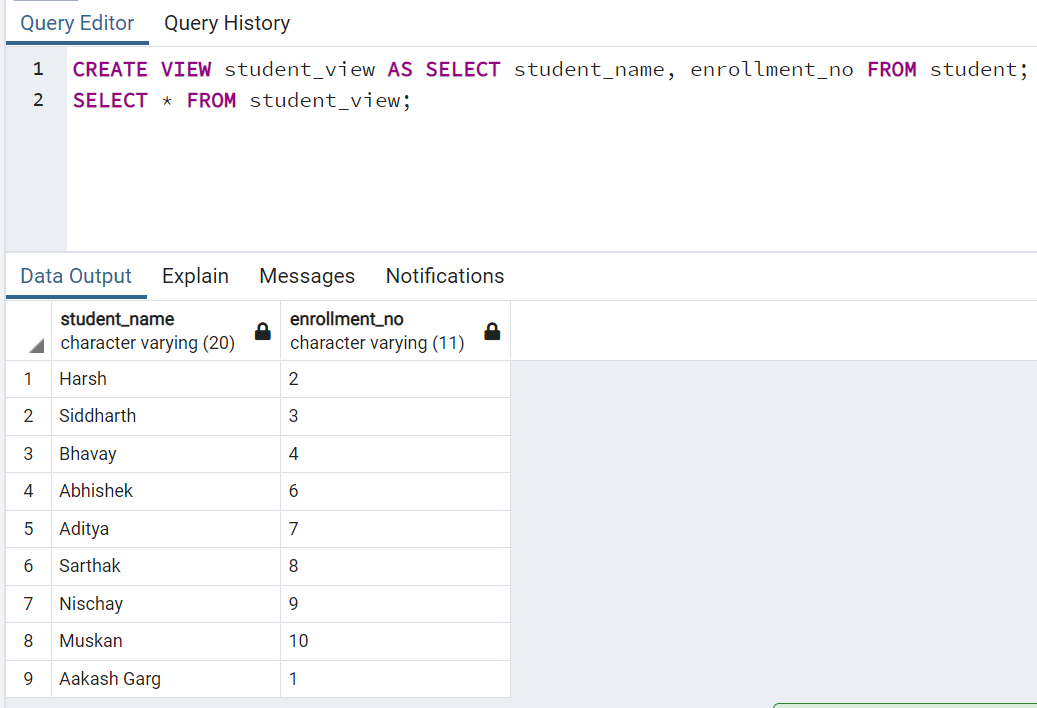


**TRIM:**

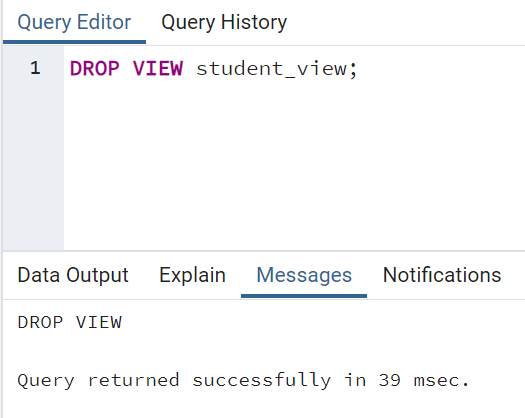


* **VIEWS:**

**CREATE VIEW:**



**DROP VIEW:**



**Date: 30/09/2021**

**Experiment-3**

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

**Theory:**

**Nested Queries:**

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

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

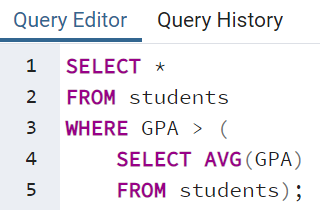
Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

**Joins:**

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

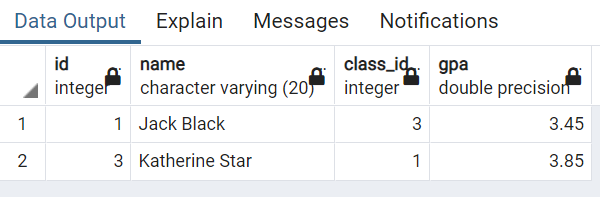
PostgreSQL supports inner join, left join, right join, full outer join, cross join, natural join, and a special kind of join called self-join.

* **Nested Queries**



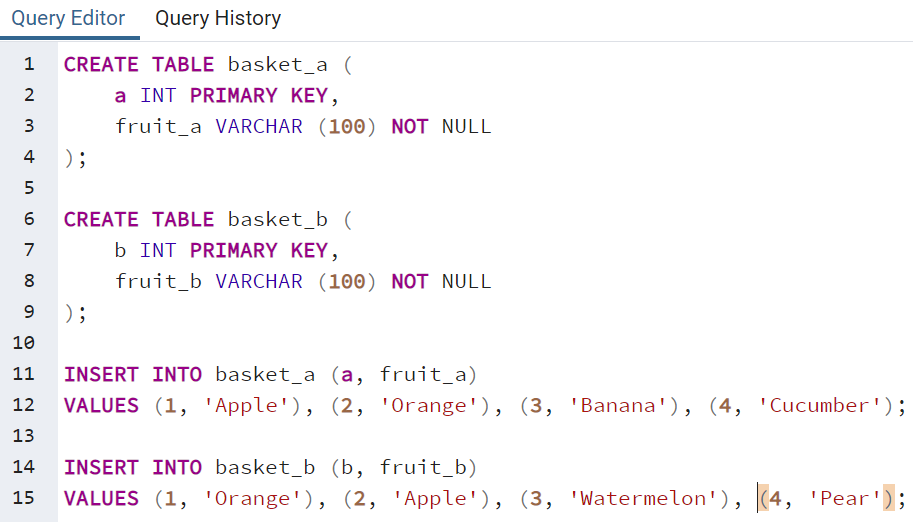
Our subquery here returns a single value (i.e. a table with a single column and a single row). This is important for the comparison operator to work. With the average GPA score returned by the inner query, the outer query can select the students who satisfy our filter condition (i.e. a GPA score above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

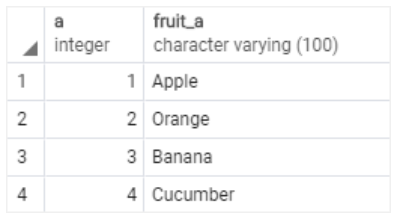
And here is the result:



* **Joins**

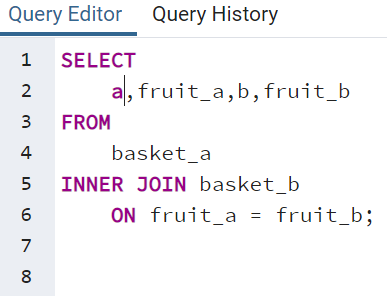
Suppose we have two tables called basket\_a and basket\_b that store fruits:

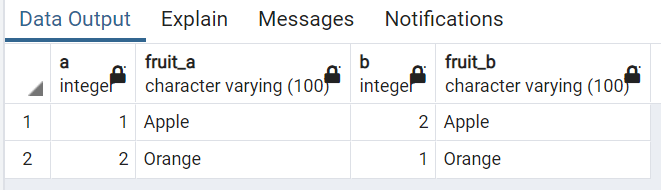




**PostgreSQL inner join**

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

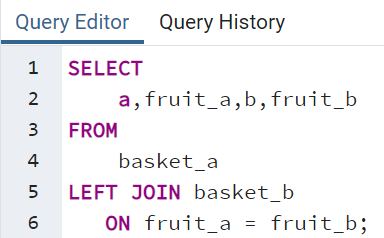


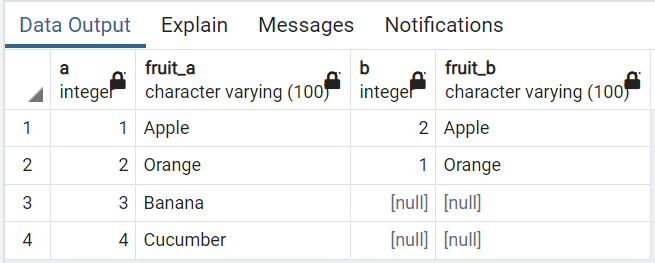


The inner join examines each row in the first table (basket\_a). It compares the value in the fruit\_a column with the value in the fruit\_b column of each row in the second table (basket\_b). If these values are equal, the inner join creates a new row that contains columns from both tables and adds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

## PostgreSQL left join

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





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

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

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

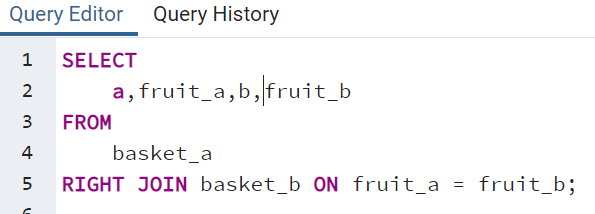
## PostgreSQL right join

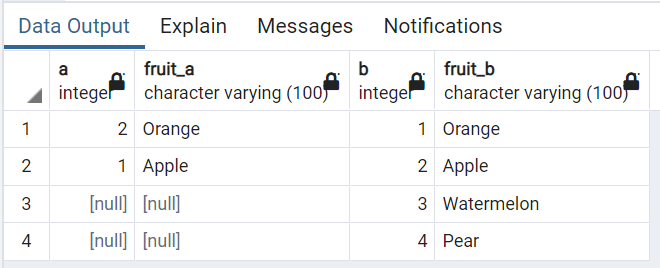
The right-join is a reversed version of the left join. The right join starts selecting data from the right table. It compares each value in the fruit\_b column of every row in the right table with each value in the fruit\_a column of every row in the fruit\_a table.

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

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

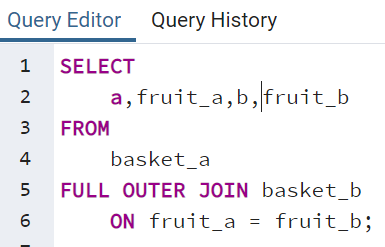
The following statement uses the right join to join the basket\_a table with the basket\_b table:

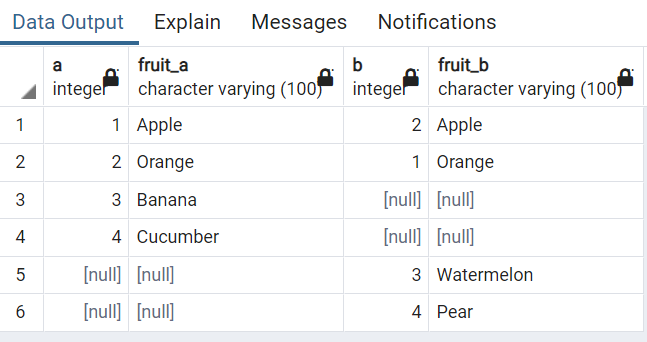




## PostgreSQL full outer join

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





**Date: 07/10/2021**

**Experiment-4**

**Aim:** Write a program

* 1. To calculate total students enrolled area-wise.
  2. To calculate average marks obtained by students residing in area that starts with "R"

**Theory:**

**Count Function:**

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

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

**Like Function:**

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

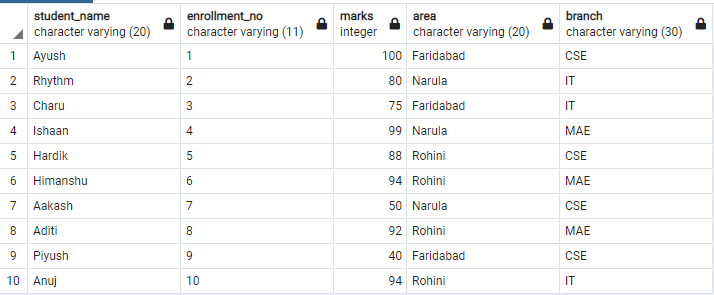
There are two wildcards used in conjunction with the LIKE operator −

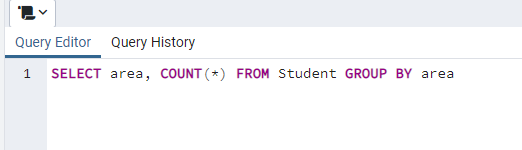
* The percent sign (%)
* The underscore (\_)

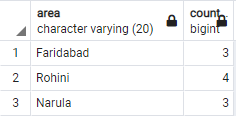
The percent sign represents zero, one, or multiple numbers or characters. The underscore represents a single number or character. These symbols can be used in combinations.

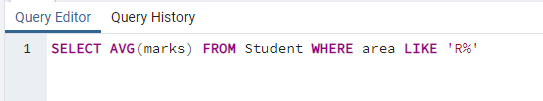
If either of these two signs is not used in conjunction with the LIKE clause, then the LIKE acts like the equals operator.

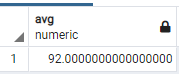
**Main Table:**

****









**Date: 14/10/2021**

**Experiment-5**

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

**Theory:**

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

**Implicit Cursors:**

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

**Explicit Cursors:**

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

**How to create explicit cursor**

**Declare Cursor Object.**

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

**Open Cursor Connection.**

Syntax: OPEN cursor\_connectionx:

**Fetch Data from cursor.**

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

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

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

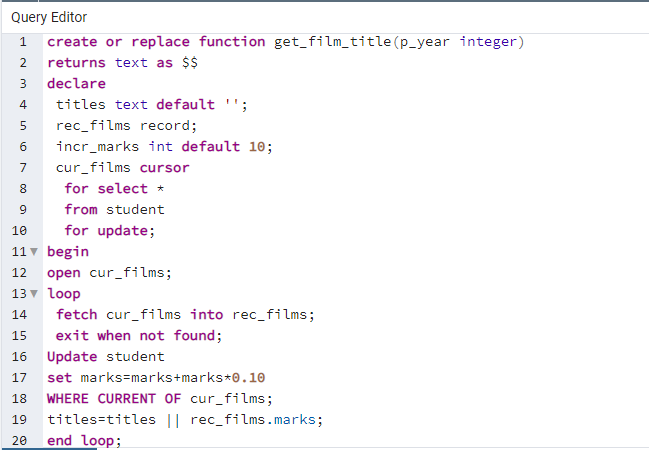
**Close cursor connection.**

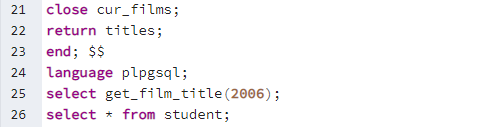
Syntax: CLOSE cursor\_name

**Deallocate cursor memory.**

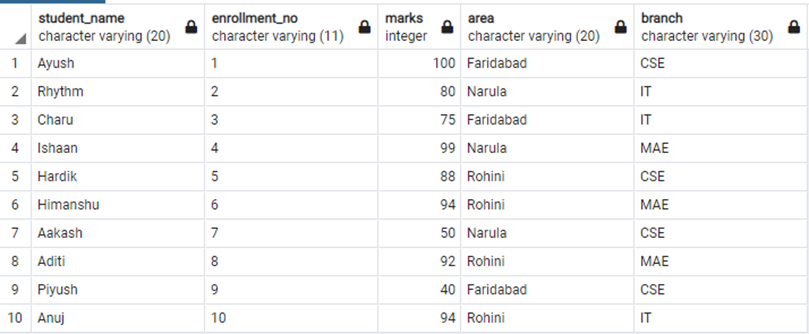
Syntax: DEALLOCATE cursor\_name

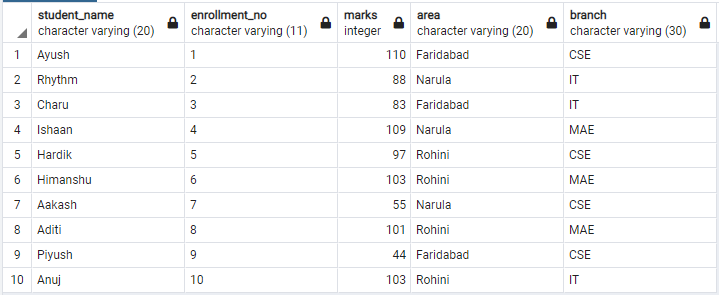
**CODE:**





**OUTPUT:**

****

****

**Date: 21/10/2021**

**Experiment-6**

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

**Theory:**

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

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

<<label>>

declare

begin

statements;

exception

when condition [or condition...] then

handle\_exception;

[when condition [or condition...] then

handle\_exception;]

[when others then

handle\_other\_exceptions;

]

end;

How it works:

1. First, when an error occurs between the begin and exception, PL/pgSQL stops the execution and passes the control to the exception list.

2. Second, PL/pgSQL searches for the first condition that matches the occurring error.

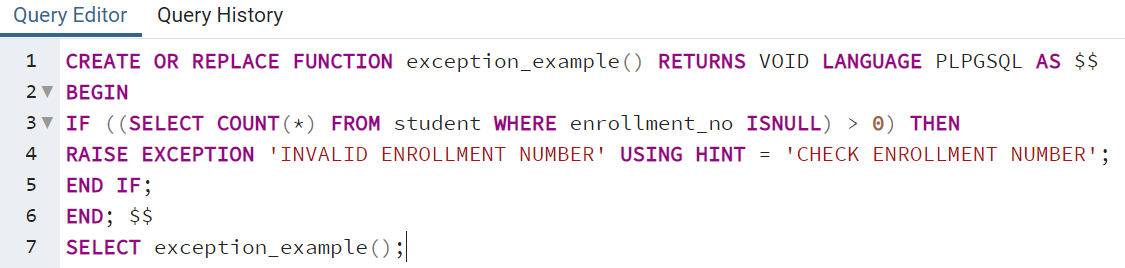
3. Third, if there is a match, the corresponding handle\_exception statements will execute. PL/pgSQL passes the control to the statement after the end keyword.

4. Finally, if no match found, the error propagates out and can be caught by the exception clause of the enclosing block. In case there is no enclosing block with the exception clause, PL/pgSQL will abort the processing.

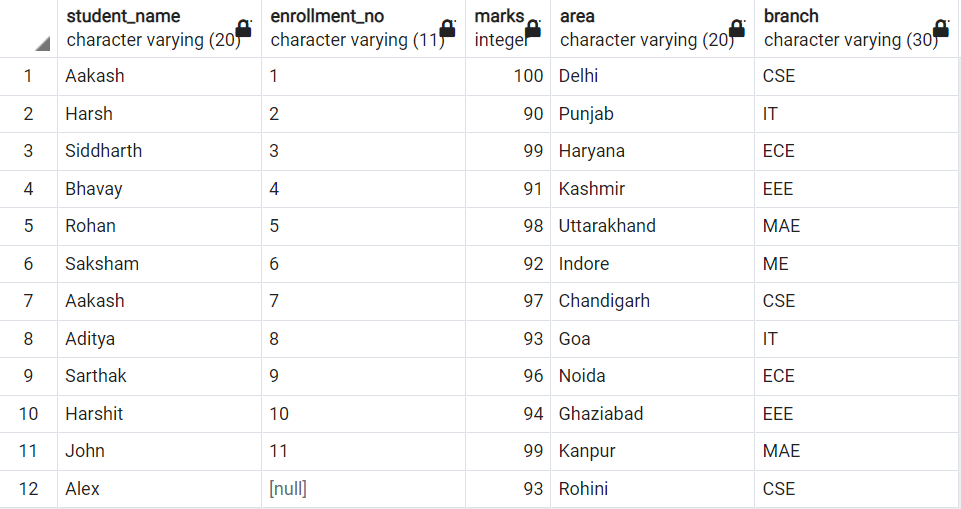
The condition names can be no\_data\_found in case of a select statement return no rows or too\_many\_rows if the select statement returns more than one row.

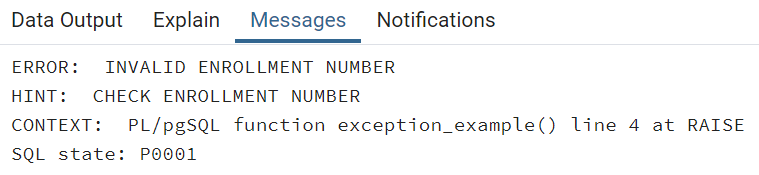
It’s also possible to specify the error condition by SQLSTATE code. For example, P0002 for no\_data\_found and P0003 for too\_many\_rows.

**CODE:**



**OUTPUT:**





**Date: 21/10/2021**

**Experiment-7**

**Aim:**  Procedure & Function

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

**Theory:**

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

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

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

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

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

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

**Syntax for Creating a PROCEDURE**

CREATE [OR REPLACE] PROCEDURE procedure\_name

[(parameter\_name [IN | OUT | IN OUT] type [,...])]

{IS | AS}

BEGIN

<procedure\_body>

END procedure\_name;

**Creating a FUNCTION**

A function is created using the CREATE FUNCTION statement. The simplified syntax for the

CREATE OR REPLACE PROCEDURE statement is as follows –

CREATE [OR REPLACE] FUNCTION

function\_name [(parameter\_name [IN | OUT | IN OUT] type [, ...])]

RETURN return\_datatype

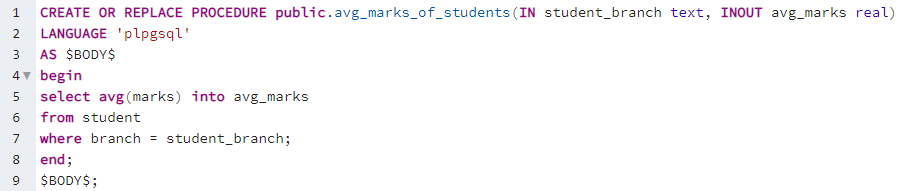
{IS | AS} BEGIN

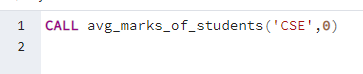
<function\_body>

END [function\_name];

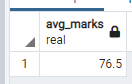
**CODE:**

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

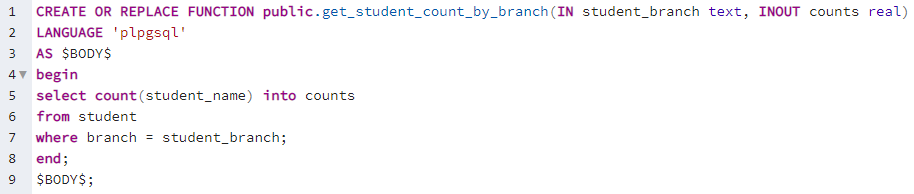
****

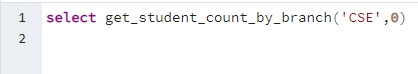
****

**OUTPUT:**

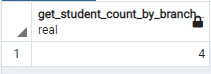
****

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

****

****

**OUTPUT:**

****

**Date: 28/10/2021**

**Experiment-8**

**Aim:**  Trigger

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

**Theory:**

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

**Syntax for creating a trigger**

CREATE [OR REPLACE] TRIGGER trigger\_name

{BEFORE | AFTER | INSTEAD OF}

{INSERT [OR]| UPDATE [OR]| DELETE}

[OF col\_name]

ON table\_name

[REFERENCING OLD AS o NEW AS n]

[FOR EACH ROW]

WHEN (condition)

DECLARE

Declaration-statements

BEGIN

Executable-statements

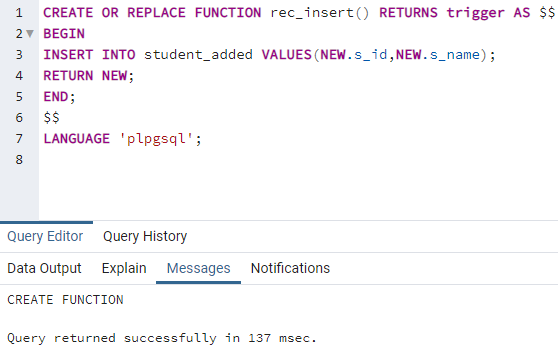
EXCEPTION

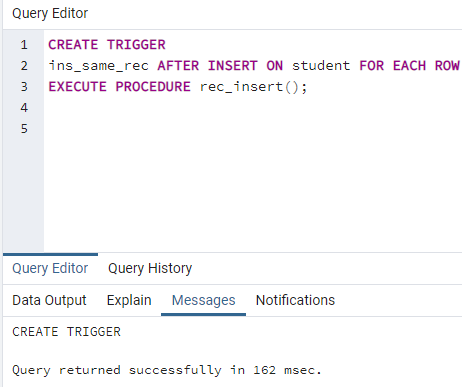
Exception-handling-statements

END;

**CODE:**

1. **Create a trigger on table after inserting a new student into table.**

****

****

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

