Experiment - 1

Advanced Database Management System Lab

Aim

Write a brief note on PostGreSQl. List the features of PostGreSQL and its installation steps.

Syeda Reeha Quasar
14114802719
7C7

EXPERIMENT – 1

Aim:

Write a brief note on PostGreSQl. List the features of PostGreSQL and its installation steps.

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.
A Brief History of PostgreSQL

PostgreSQL, originally called Postgres, was created at UCB by a computer science professor named Michael Stonebraker. Stonebraker started Postgres in 1986 as a follow-up project to its predecessor, Ingres, now owned by Computer Associates.

- **1977-1985** – A project called INGRES was developed.
  - Proof-of-concept for relational databases
  - Established the company Ingres in 1980
  - Bought by Computer Associates in 1994
- **1986-1994** – POSTGRES
  - Development of the concepts in INGRES with a focus on object orientation and the query language - Quel
  - The code base of INGRES was not used as a basis for POSTGRES
  - Commercialized as Illustra (bought by Informix, bought by IBM)
- **1994-1995** – Postgres95
  - Support for SQL was added in 1994
  - Released as Postgres95 in 1995
  - Re-released as PostgreSQL 6.0 in 1996
  - Establishment of the PostgreSQL Global Development Team

Key Features of PostgreSQL

PostgreSQL runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. It supports text, images, sounds, and video, and includes programming interfaces for C / C++, Java, Perl, Python, Ruby, Tcl and Open Database Connectivity (ODBC).

PostgreSQL supports a large part of the SQL standard and offers many modern features including the following –

- Complex SQL queries

- SQL Sub-selects
- Foreign keys
- Trigger
- Views
- Transactions
- Multiversion concurrency control (MVCC)
- Streaming Replication (as of 9.0)
- Hot Standby (as of 9.0)

You can check official documentation of PostgreSQL to understand the above-mentioned features. PostgreSQL can be extended by the user in many ways. For example by adding new –

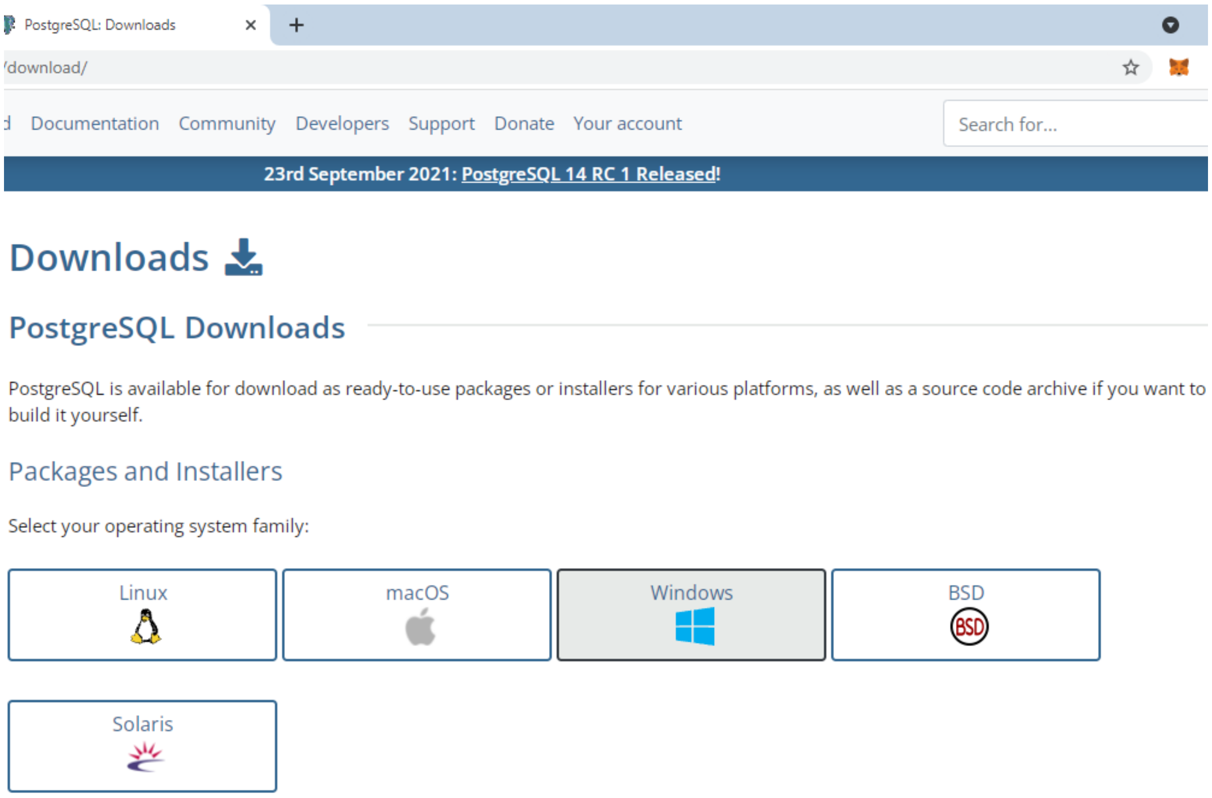
- Data types
- Functions
- Operators
- Aggregate functions
- Index methods

Procedural Languages Support

PostgreSQL supports four standard procedural languages, which allows the users to write their own code in any of the languages and it can be executed by PostgreSQL database server. These procedural languages are - PL/pgSQL, PL/Tcl, PL/Perl and PL/Python. Besides, other non-standard procedural languages like PL/PHP, PL/V8, PL/Ruby, PL/Java, etc., are also supported.

Steps to Install: -

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


PostgreSQL: Downloads

/download/

Documentation Community Developers Support Donate Your account

Search for...

23rd September 2021: PostgreSQL 14 RC 1 Released!

Downloads

PostgreSQL Downloads

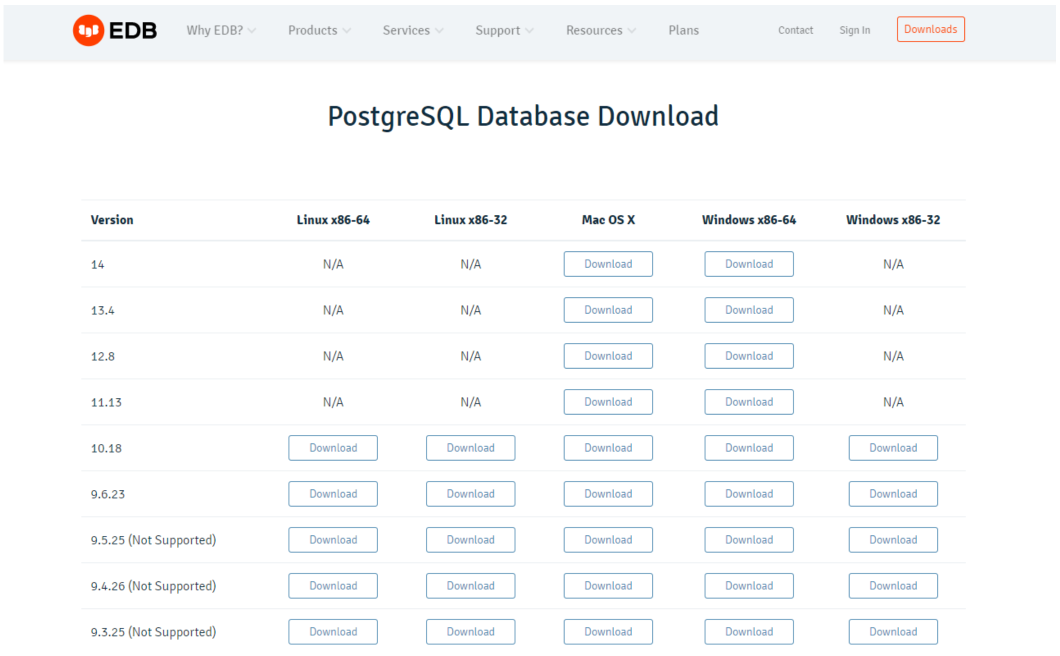
PostgreSQL is available for download as ready-to-use packages or installers for various platforms, as well as a source code archive if you want to build it yourself.

Packages and Installers

Select your operating system family:

Linux | macOS | Windows | BSD | Solaris

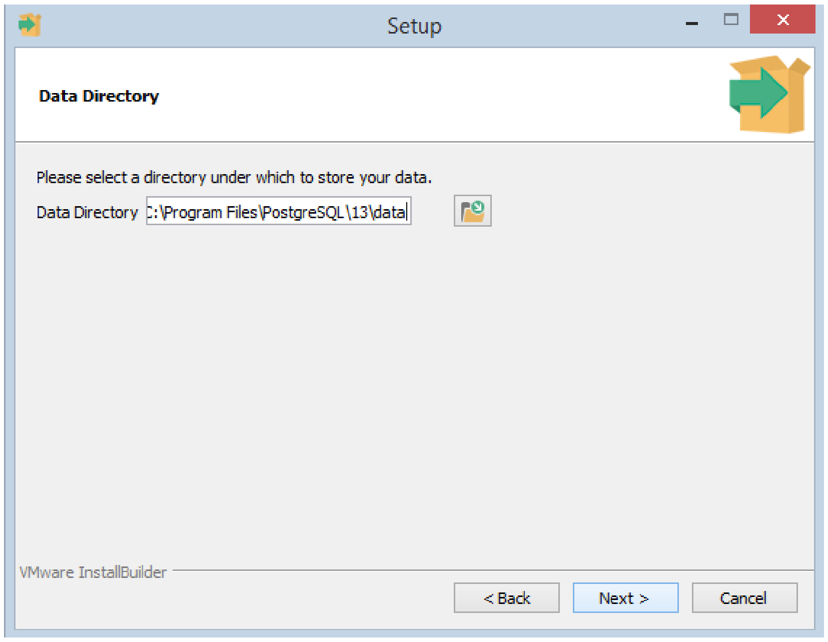
2. Click on the executable file to run the installer. Select your preferred language and the version.


EDB Why EDB? Products Services Support Resources Plans Contact Sign In Downloads

PostgreSQL Database Download

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	Download	Download	Download	Download
9.6.23	Download	Download	Download	Download	Download
9.5.25 (Not Supported)	Download	Download	Download	Download	Download
9.4.26 (Not Supported)	Download	Download	Download	Download	Download
9.3.25 (Not Supported)	Download	Download	Download	Download	Download

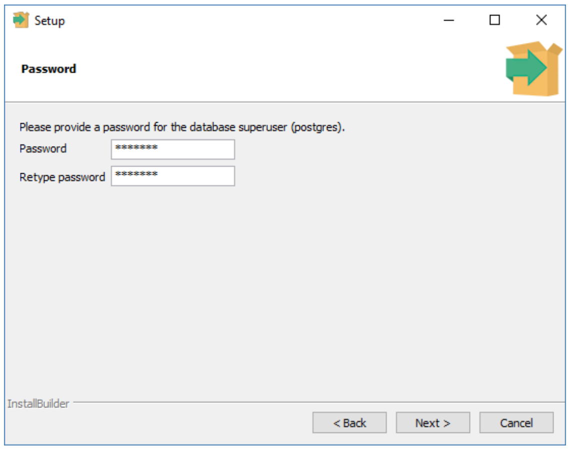
3. Specify directory where you want to install PostgreSQL. Specify PostgreSQL server port. You can leave this as default if you're unsure what to enter.



Please select a directory under which to store your data.

Data Directory C:\Program Files\PostgreSQL\13\data

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

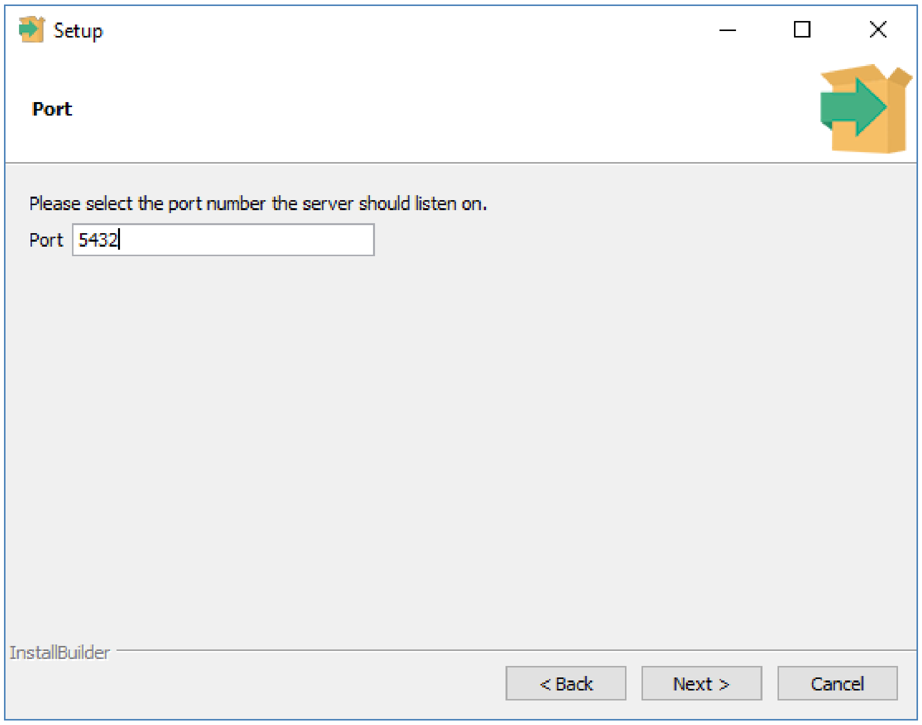


Please provide a password for the database superuser (postgres).

Password *******

Retype password *******

In the next step, keep the port as default.



Setup

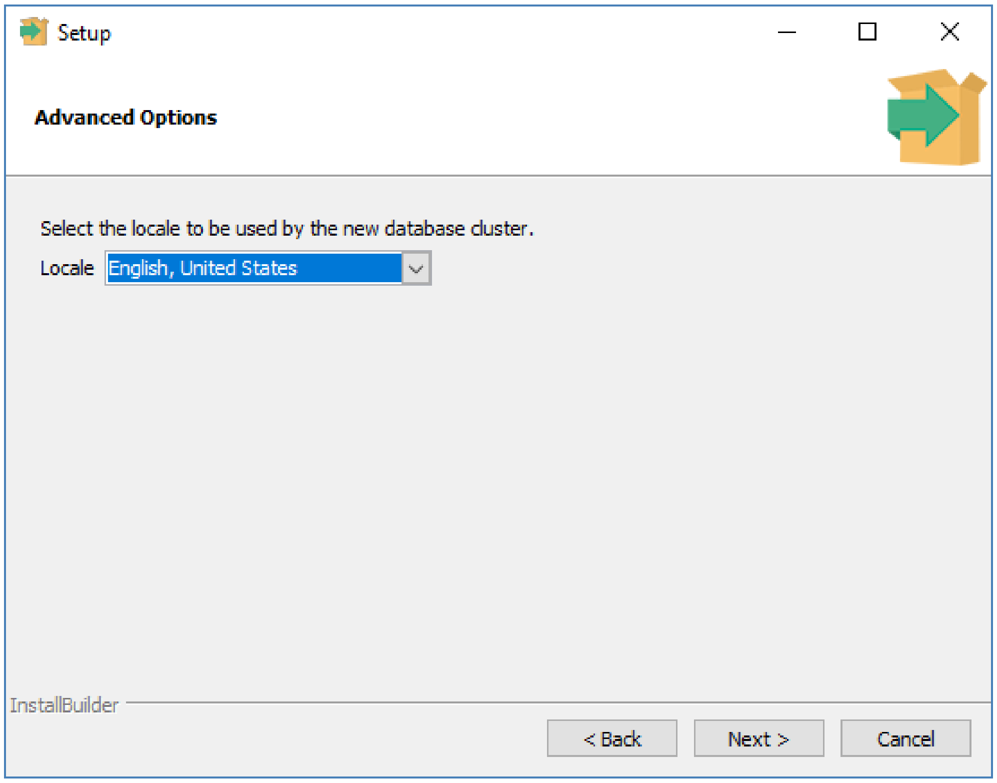
Port

Please select the port number the server should listen on.

Port 5432

InstallBuilder

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



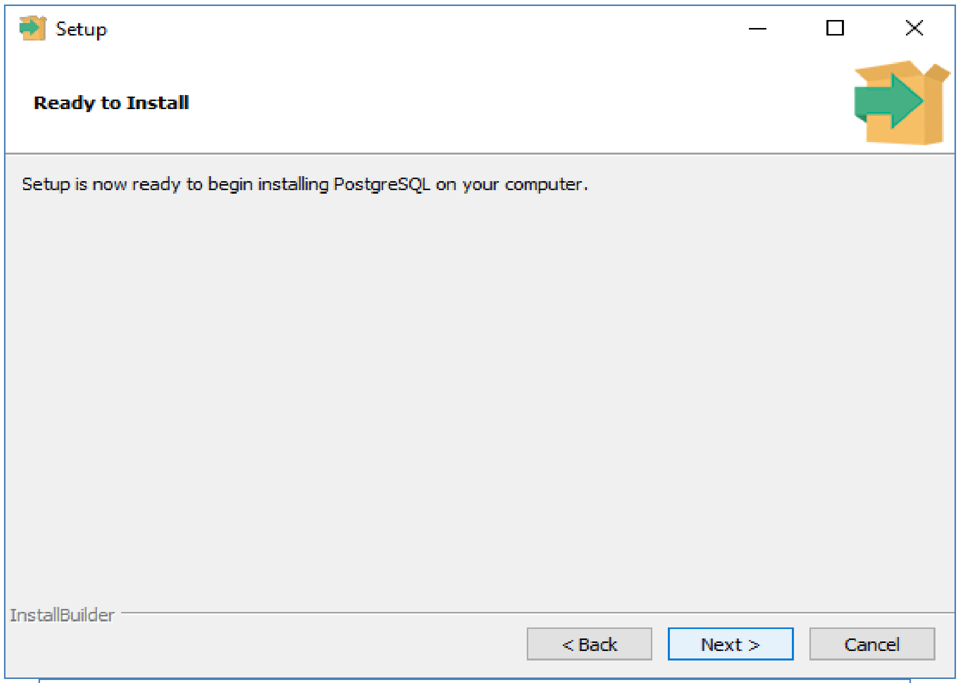
Setup

Advanced Options

Select the locale to be used by the new database cluster.

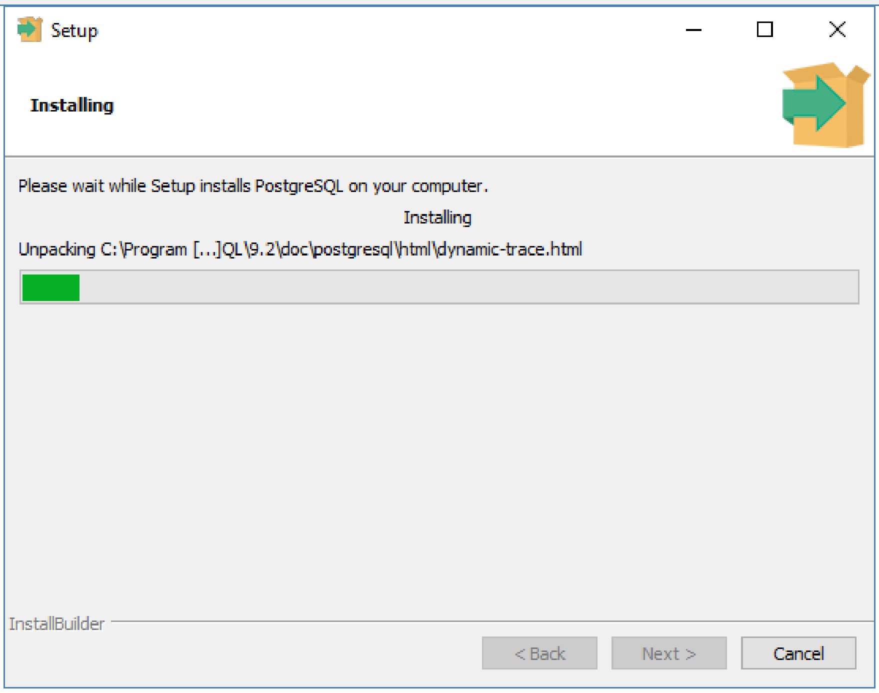
Locale English, United States

InstallBuilder


Setup

Ready to Install

Setup is now ready to begin installing PostgreSQL on your computer.


Setup

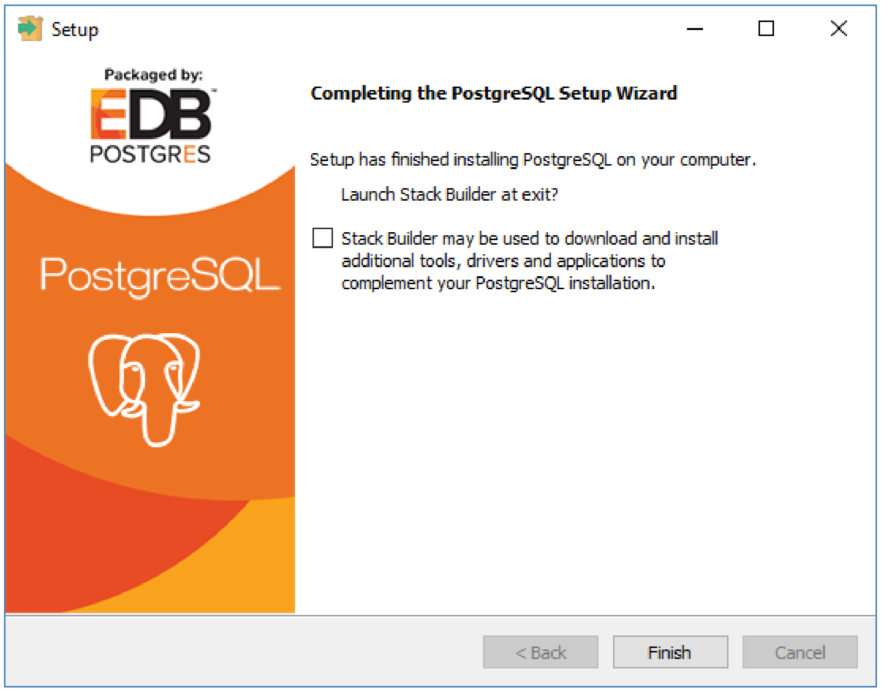
Installing

Please wait while Setup installs PostgreSQL on your computer.

Installing

Unpacking C:\Program [...]QL\9.2\doc\postgresql\html\dynamic-trace.html

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.



Completing the PostgreSQL Setup Wizard

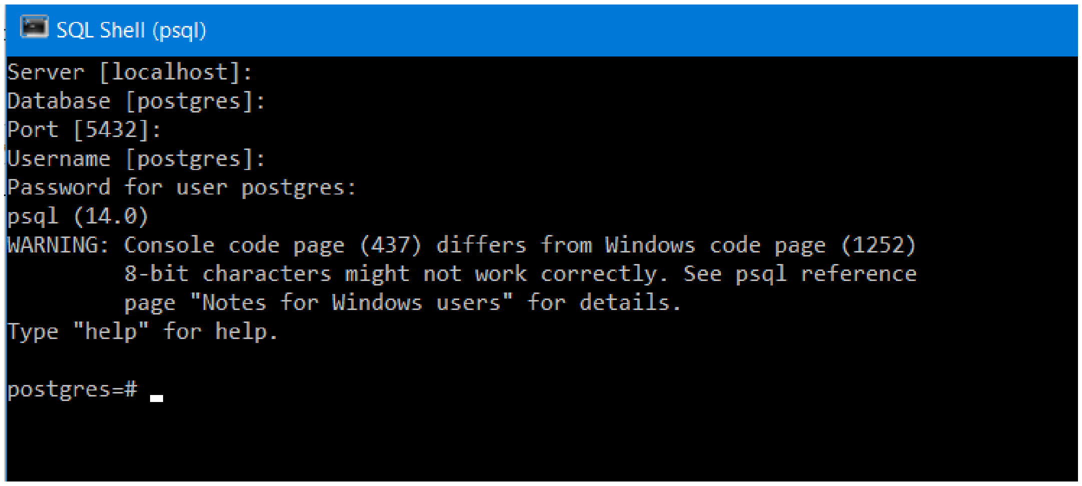
Setup has finished installing PostgreSQL on your computer.

Launch Stack Builder at exit?

☐ Stack Builder may be used to download and install additional tools, drivers and applications to complement your PostgreSQL installation.

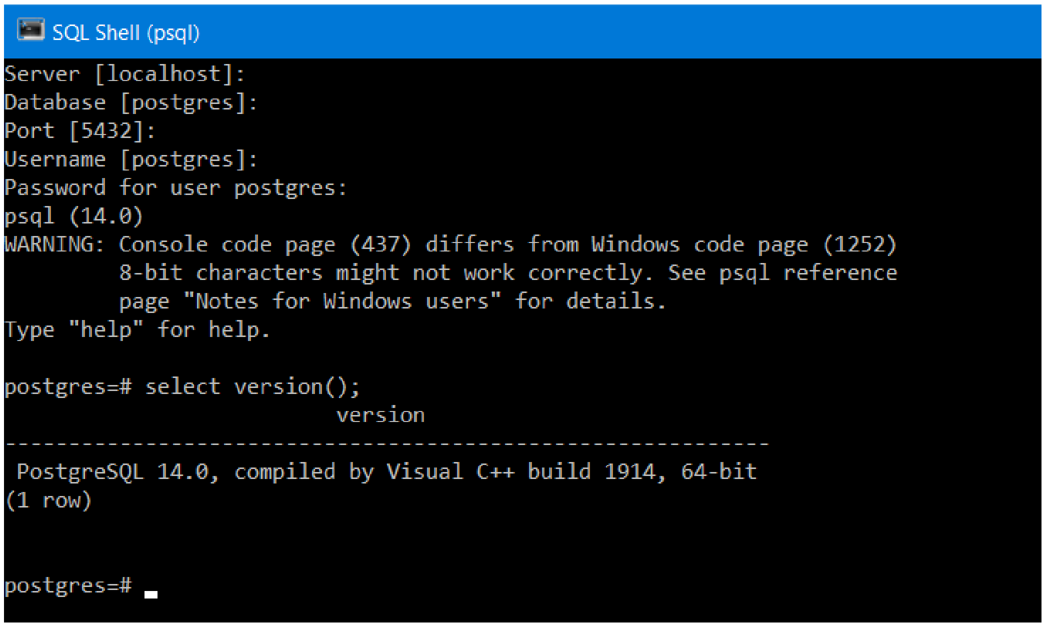
Verify the Installation of PostgreSQL:

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.

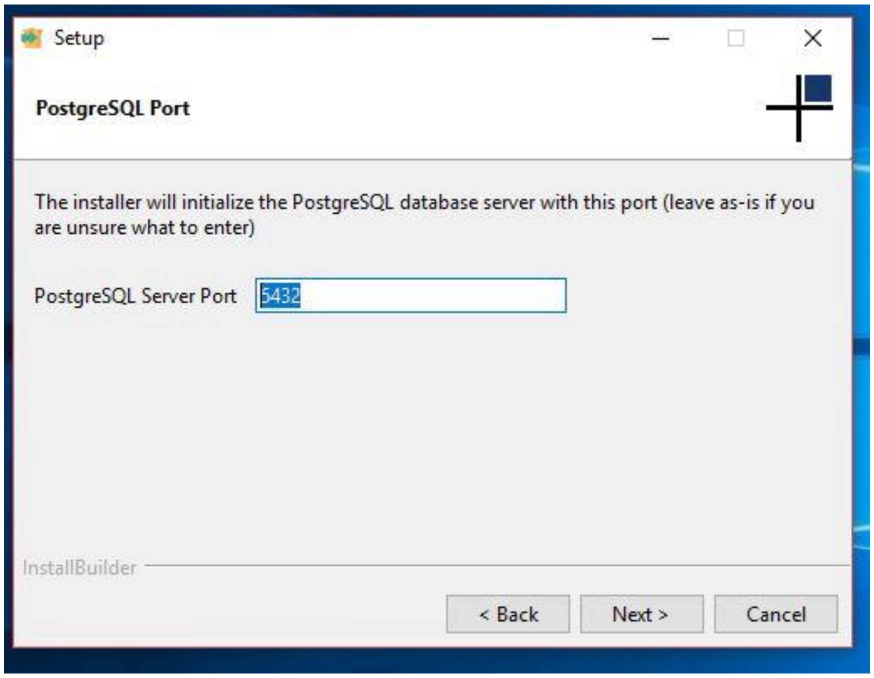


```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (14.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users"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

postgres=#
```

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



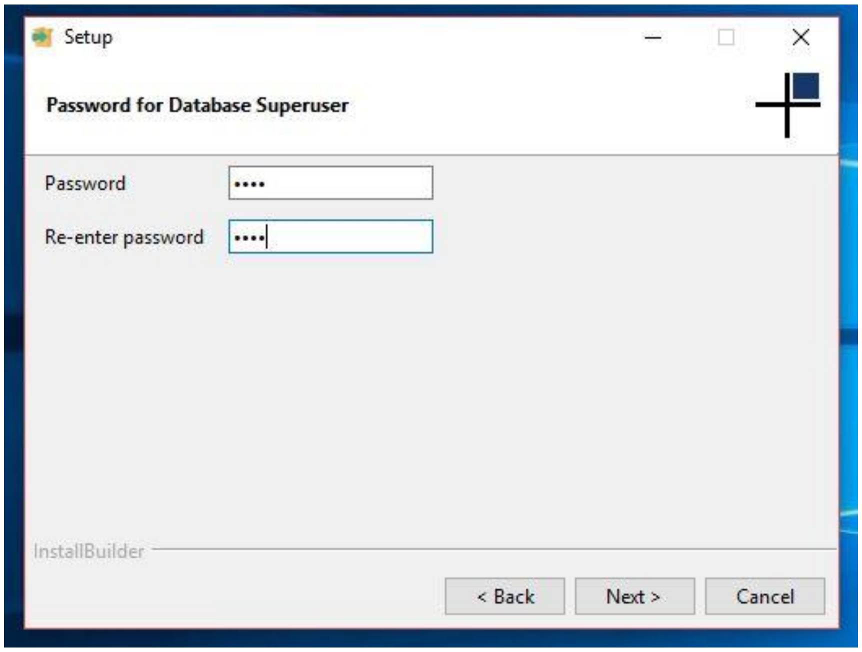
```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (14.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users"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

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


postgres=#
```



The installer will initialize the PostgreSQL database server with this port (leave as-is if you are unsure what to enter)

PostgreSQL Server Port: 5432

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

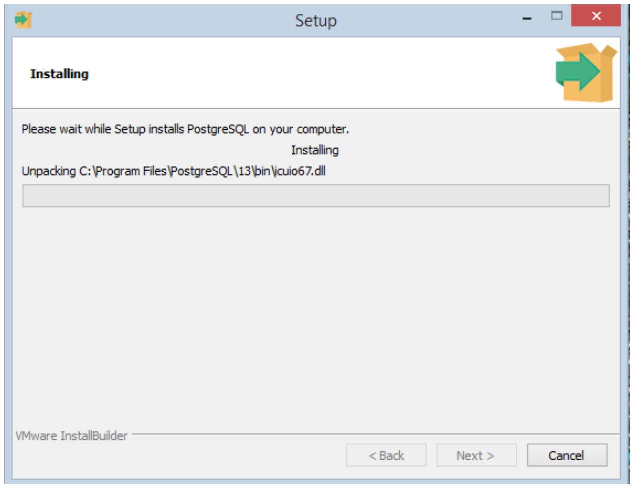

Setup

Password for Database Superuser

Password ••••

Re-enter password ••••

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

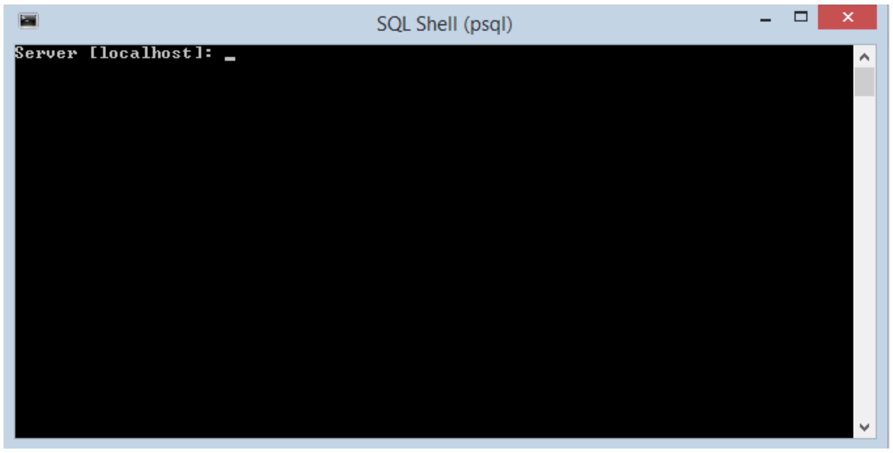

Setup

Installing

Please wait while Setup installs PostgreSQL on your computer.

Installing

Unpacking C:\Program Files\PostgreSQL\13\bin\icuio67.dll


SQL Shell (psql)

```
Server [localhost]: _
```

Experiment - 2

Advanced Database Management System Lab

Aim

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

Syeda Reeha Quasar
14114802719
7C7

EXPERIMENT – 2

Aim:

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

Theory:

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 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

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
- 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

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.

Experiment - 3

Advanced Database Management System Lab

Aim

Writing DDL, DML, DCL and some built in SQL queries.

Syeda Reeha Quasar
14114802719
7C7

EXPERIMENT – 3

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.

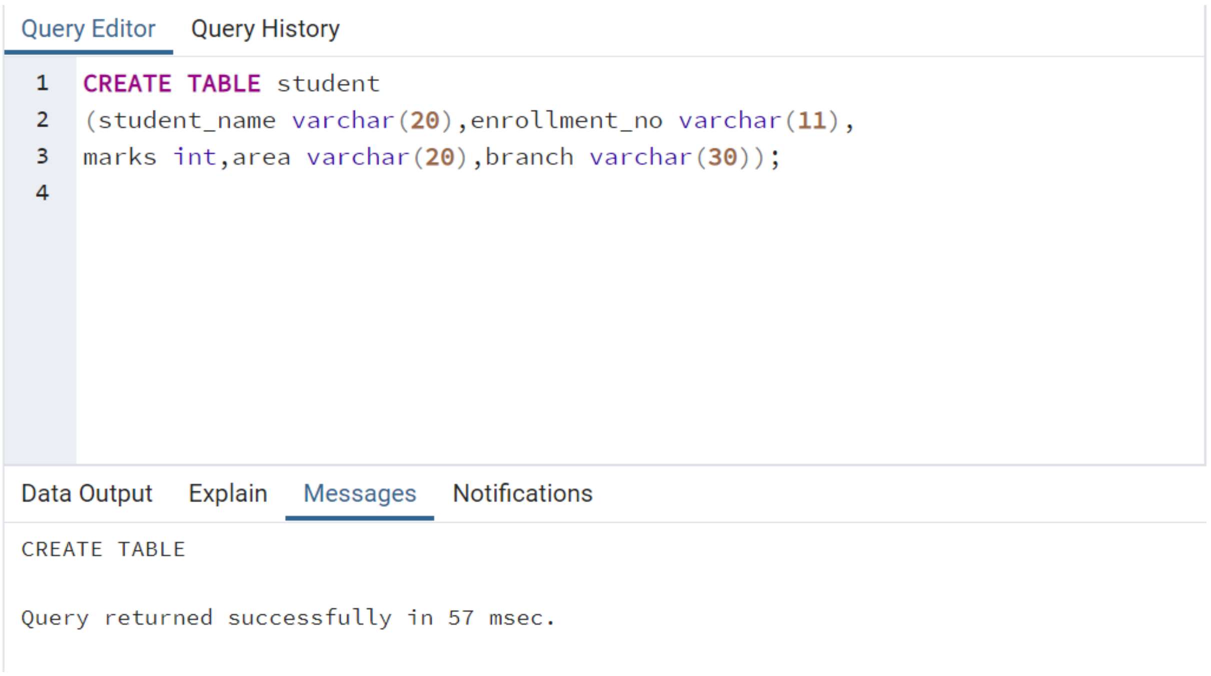
- Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables or another view.

- A view is deleted using the DROP VIEW command.

- **DDL COMMANDS:**

```
-- Table: public.Student

-- DROP TABLE IF EXISTS public."Student";

CREATE TABLE IF NOT EXISTS public."Student"
(
    student_name character varying COLLATE pg_catalog."default",
    enrolment_no bigint NOT NULL,
    marks bigint,
    area character varying COLLATE pg_catalog."default",
    branch character varying COLLATE pg_catalog."default",
    CONSTRAINT "Student_pkey" PRIMARY KEY (enrolment_no)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public."Student"
    OWNER to admin;
•
```

CREATE TABLE:

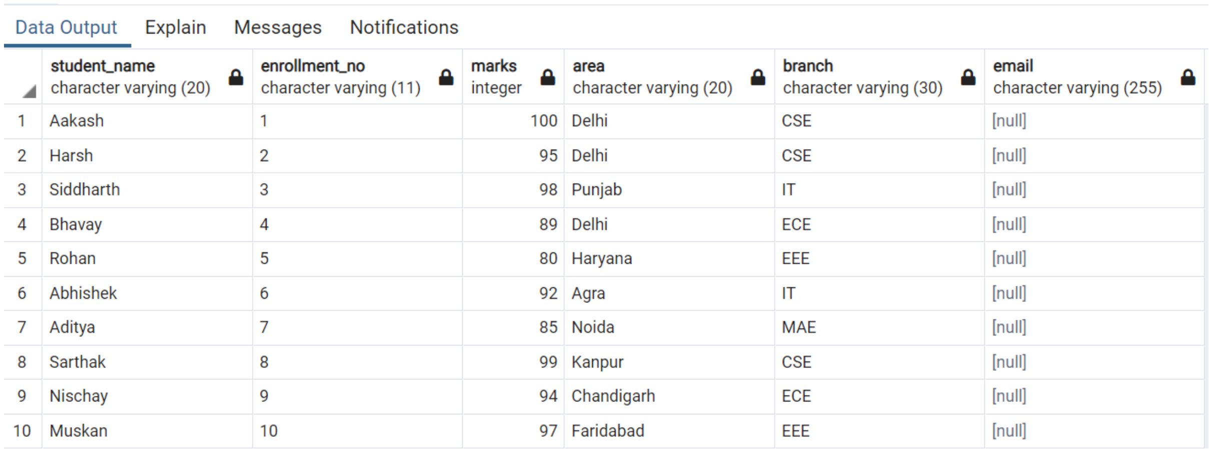


```
CREATE TABLE student
(student_name varchar(20),enrollment_no varchar(11),
marks int,area varchar(20),branch varchar(30));
```

```
CREATE TABLE

Query returned successfully in 57 msec.
```

ALTER TABLE:

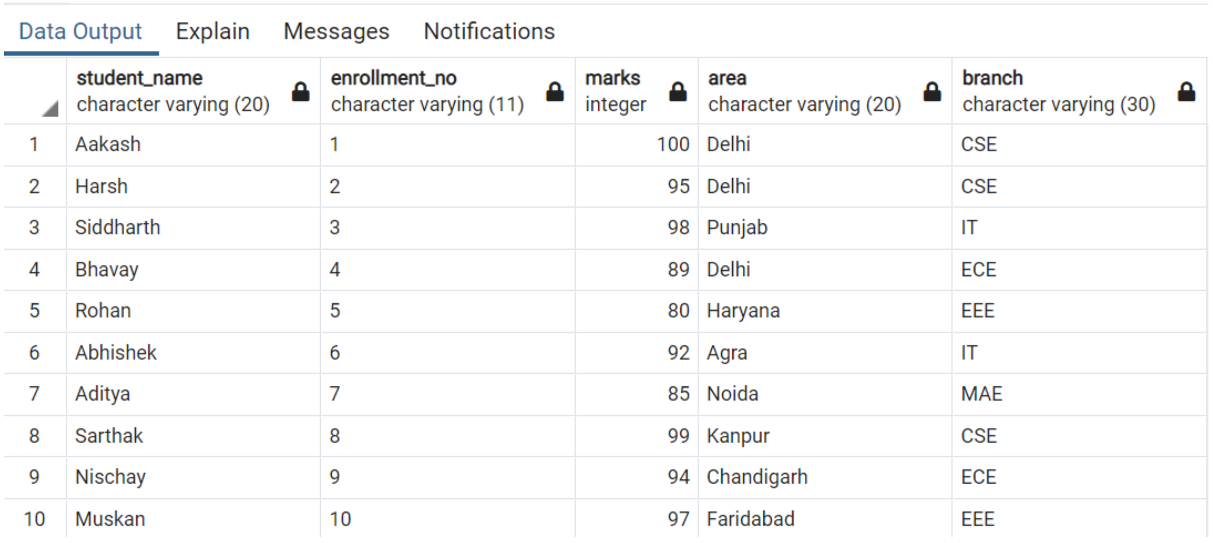
Query: ALTER TABLE student ADD email varchar(255);



Data Output Explain Messages Notifications

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

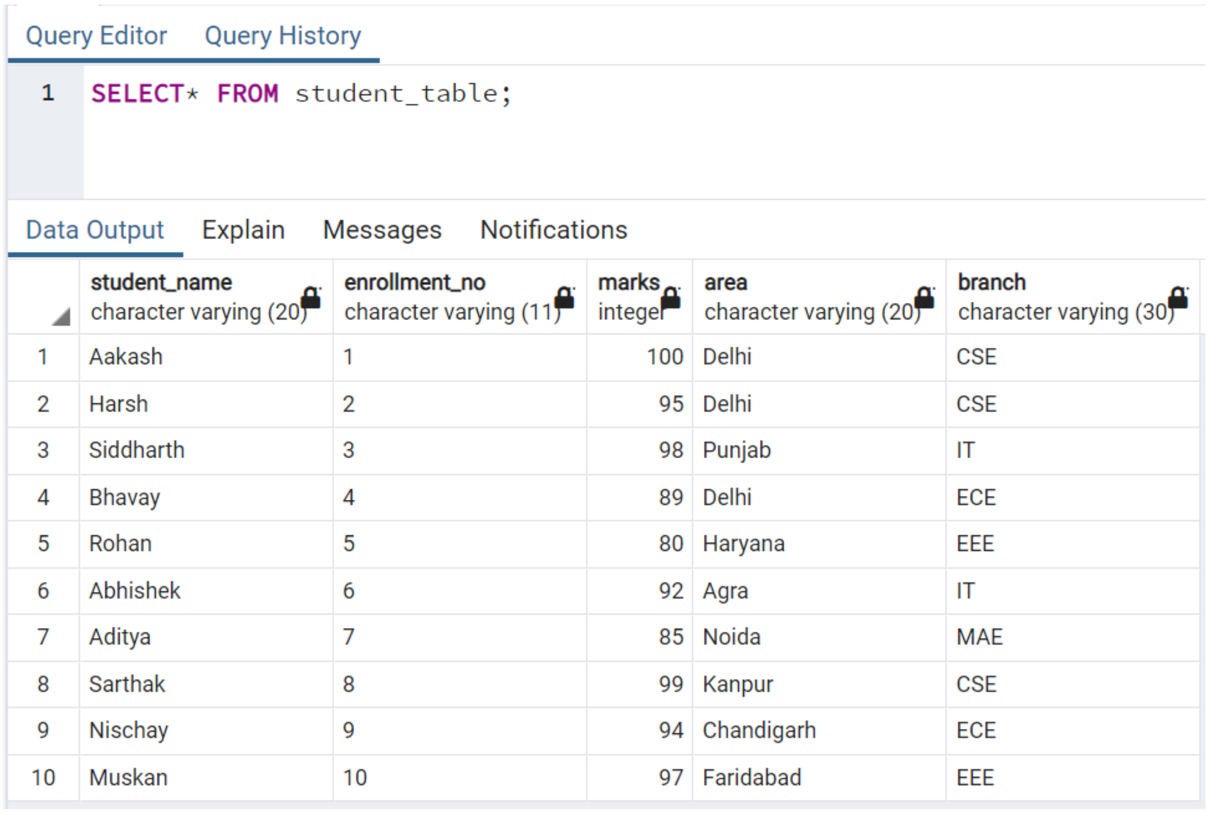
Query: ALTER TABLE student DROP COLUMN email;



Data Output Explain Messages Notifications

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

Query: ALTER TABLE student RENAME TO student_table; **(RENAME TABLE)**



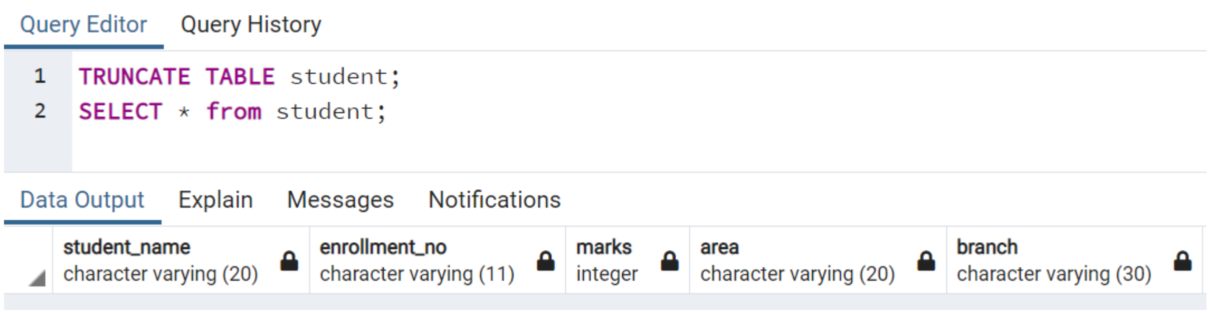
Query Editor   Query History

```
SELECT* FROM student_table;
```

Data Output   Explain   Messages   Notifications

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

TRUNCATE TABLE:



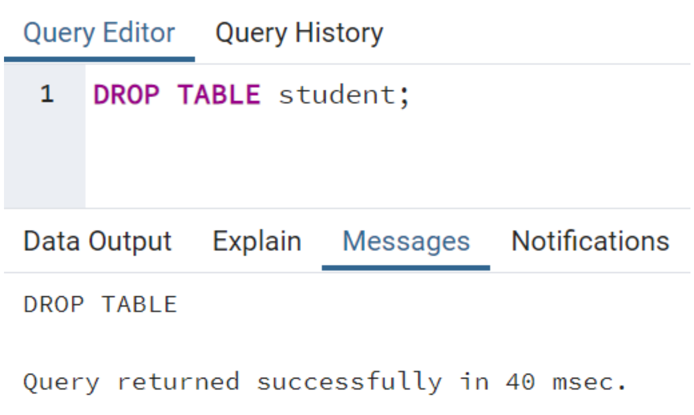
Query Editor   Query History

```
TRUNCATE TABLE student;
SELECT * from student;
```

Data Output   Explain   Messages   Notifications

student_name character varying (20)	enrollment_no character varying (11)	marks integer	area character varying (20)	branch character varying (30)

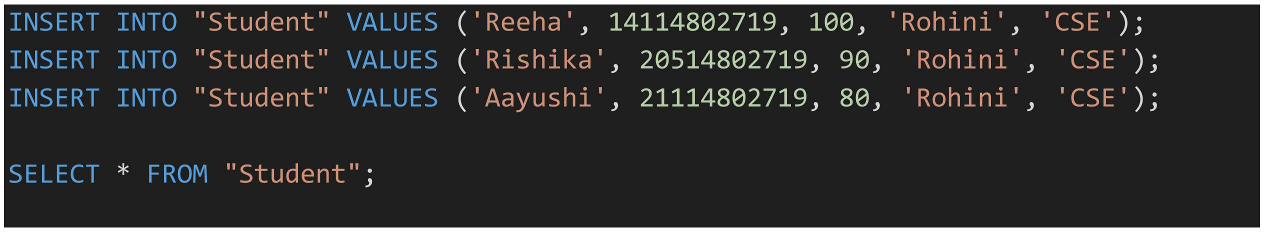
DROP TABLE:



```sql
DROP TABLE student;
```

```
DROP TABLE

Query returned successfully in 40 msec.
```

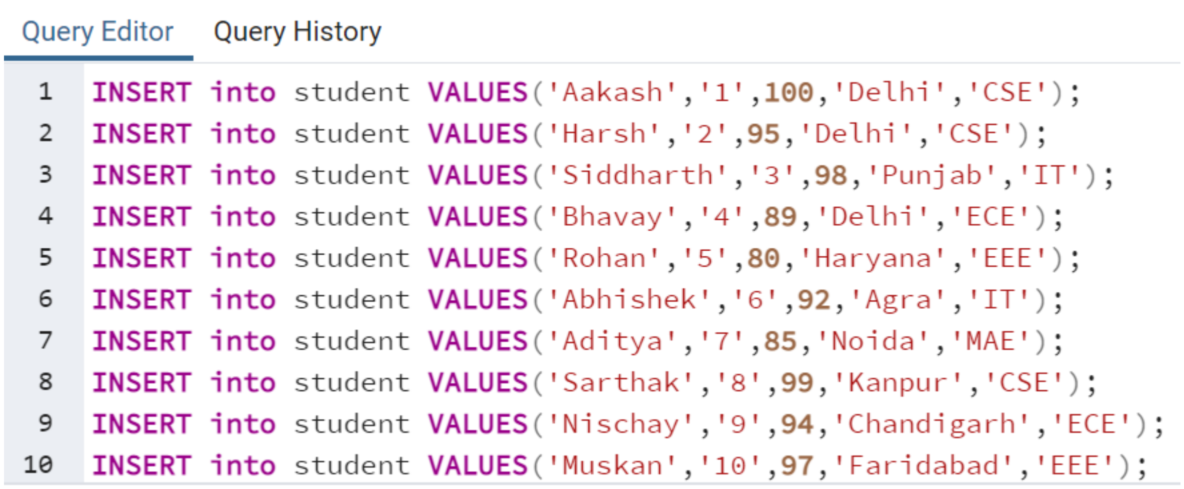
- **DML COMMANDS:**



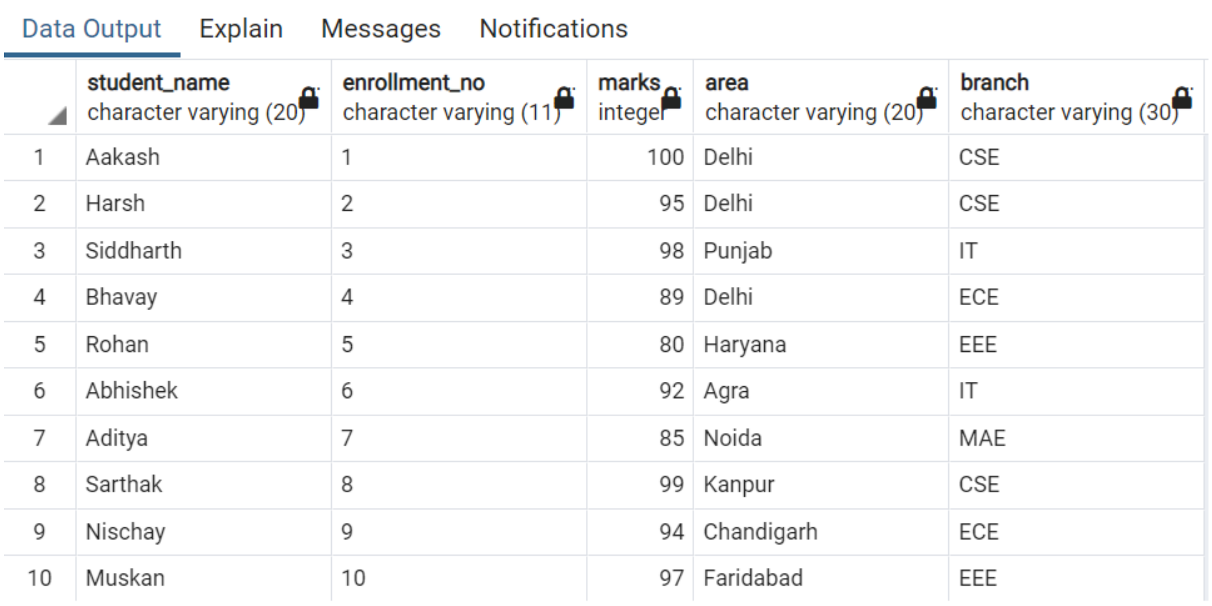
```sql
INSERT INTO "Student" VALUES ('Reeha', 14114802719, 100, 'Rohini', 'CSE');
INSERT INTO "Student" VALUES ('Rishika', 20514802719, 90, 'Rohini', 'CSE');
INSERT INTO "Student" VALUES ('Aayushi', 21114802719, 80, 'Rohini', 'CSE');

SELECT * FROM "Student";
```

INSERT INTO TABLE

Query:

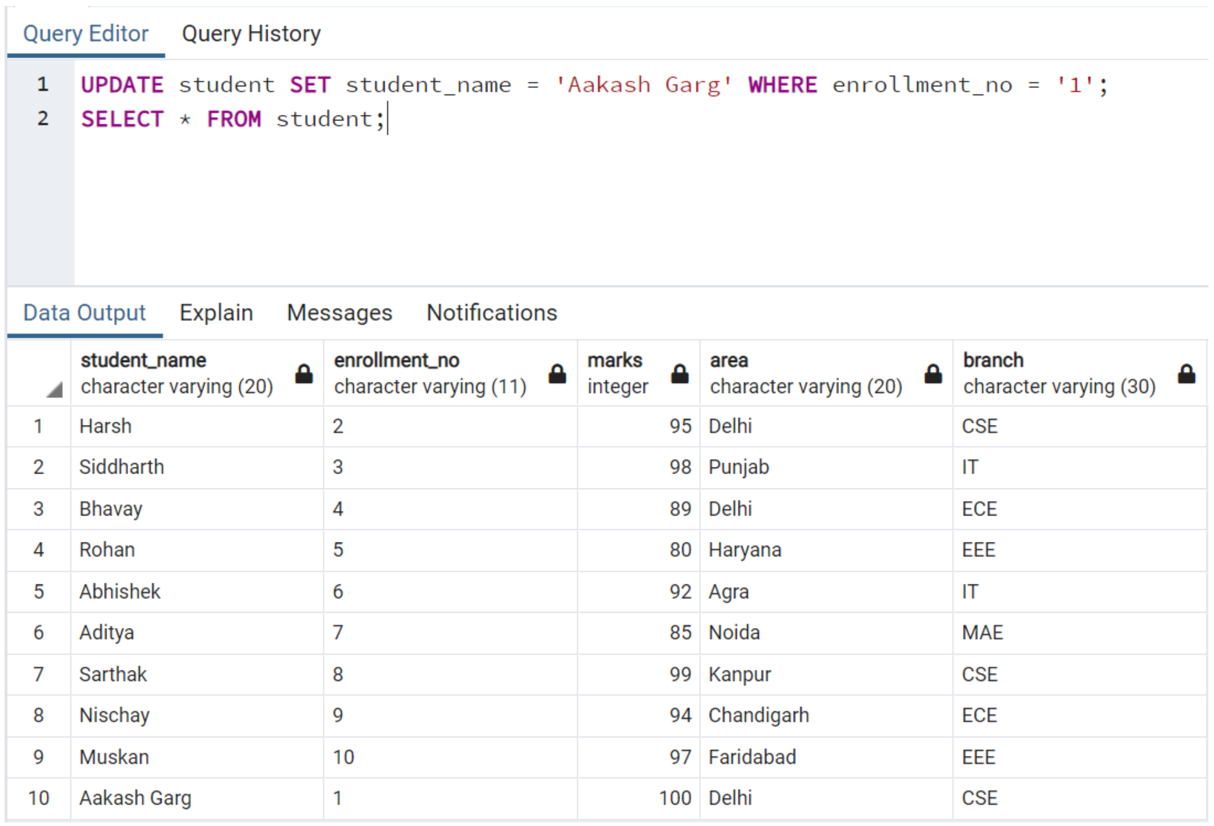


```sql
INSERT into student VALUES('Aakash','1',100,'Delhi','CSE');
INSERT into student VALUES('Harsh','2',95,'Delhi','CSE');
INSERT into student VALUES('Siddharth','3',98,'Punjab','IT');
INSERT into student VALUES('Bhavay','4',89,'Delhi','ECE');
INSERT into student VALUES('Rohan','5',80,'Haryana','EEE');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



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

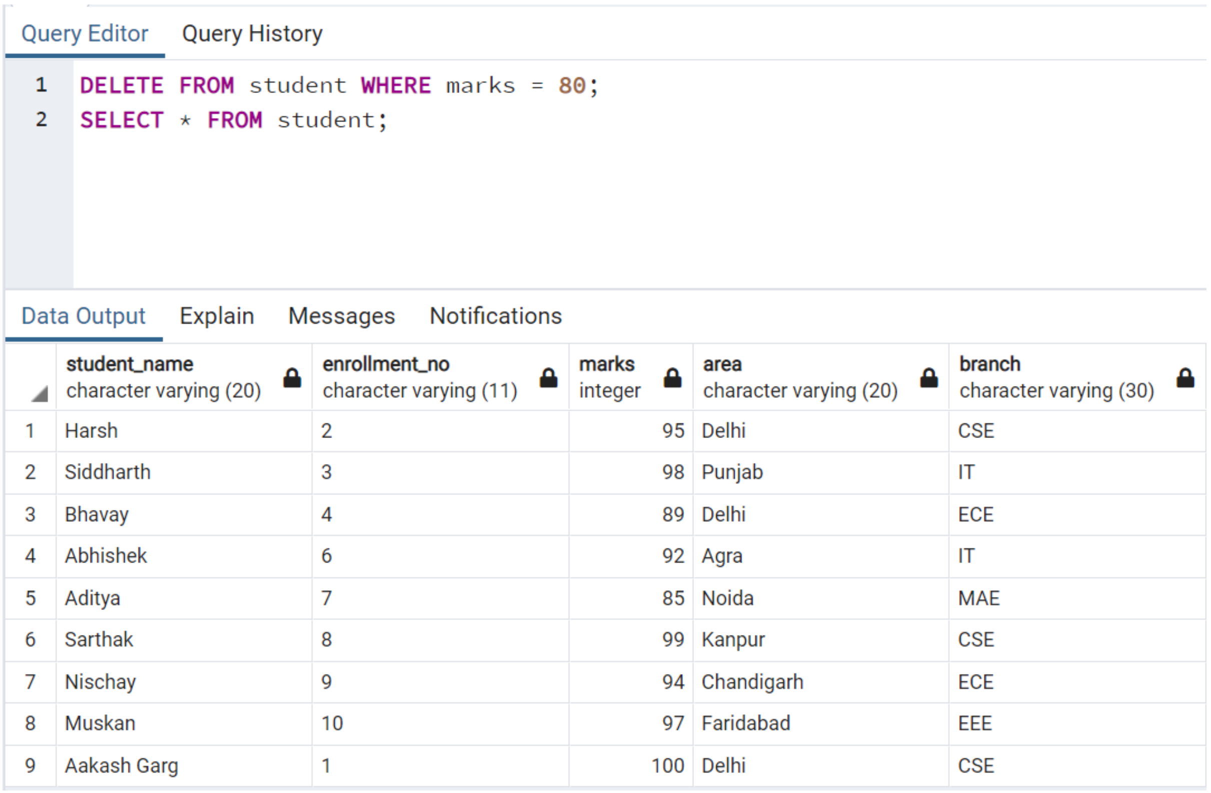
UPDATE TABLE:



```sql
UPDATE student SET student_name = 'Aakash Garg' WHERE enrollment_no = '1';
SELECT * FROM student;
```

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

DELETE TABLE:


Query Editor Query History

```sql
DELETE FROM student WHERE marks = 80;
SELECT * FROM student;
```

Data Output Explain Messages Notifications

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

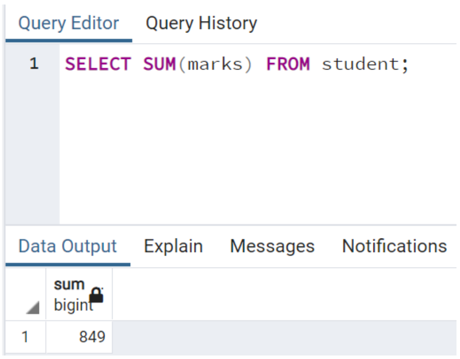
- **DCL COMMANDS:**

GRANT COMMAND: SYNTAX**:** grant privilege_name on object_name to {user_name | public | role_name}

REVOKE COMMAND: SYNTAX: revoke privilege_name on object_name from {user_name | public | role_name}

- **BUILT IN FUNCTIONS:**

SUM:

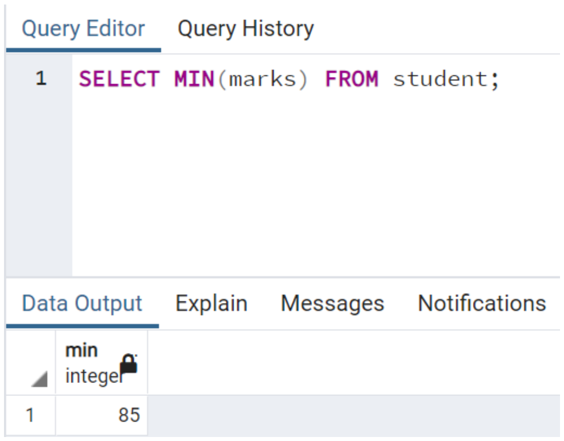

Query Editor Query History

```sql
SELECT SUM(marks) FROM student;
```

Data Output Explain Messages Notifications

	sum bigint
1	849

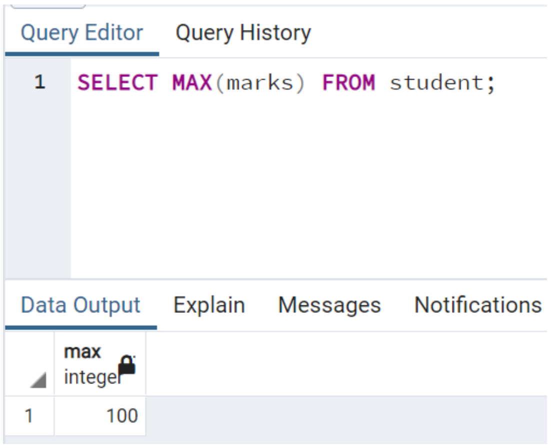
MIN:



```
SELECT MIN(marks) FROM student;
```

```
   min
   integer
1       85
```

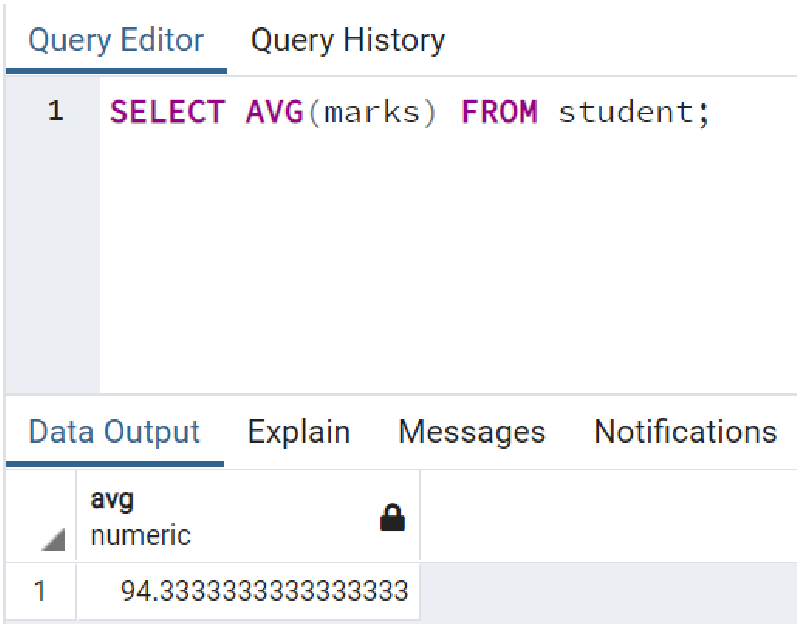
MAX:



```
SELECT MAX(marks) FROM student;
```

```
   max
   integer
1      100
```

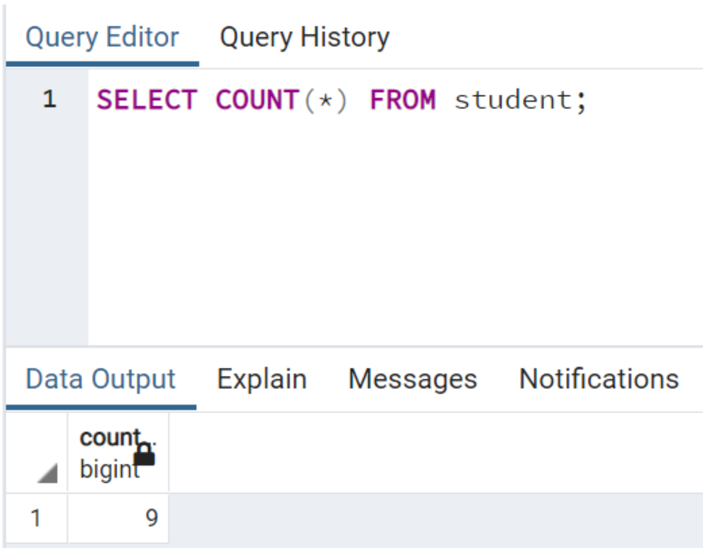
AVG:



```
SELECT AVG(marks) FROM student;
```

```
   avg
   numeric
1    94.3333333333333333
```

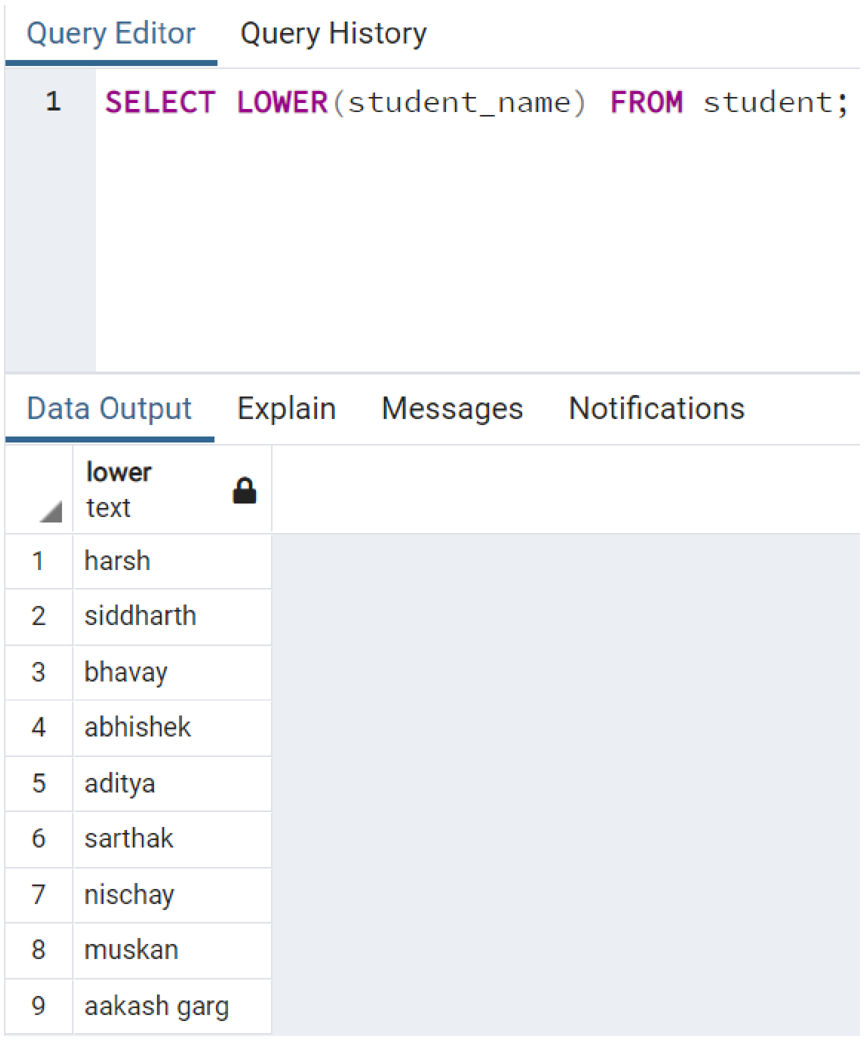
COUNT:



```sql
SELECT COUNT(*) FROM student;
```

```
   count
   bigint
1       9
```

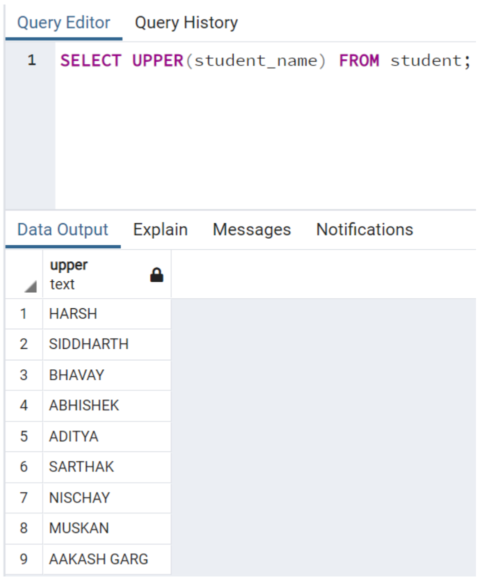
LOWER:



```sql
SELECT LOWER(student_name) FROM student;
```

```
   lower
   text
1  harsh
2  siddharth
3  bhavay
4  abhishek
5  aditya
6  sarthak
7  nischay
8  muskan
9  aakash garg
```

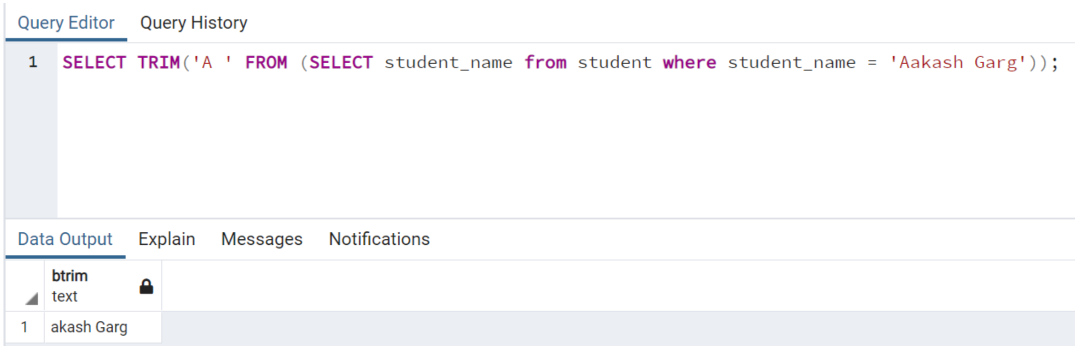
UPPER:



```
SELECT UPPER(student_name) FROM student;
```

	upper text
1	HARSH
2	SIDDHARTH
3	BHAVAY
4	ABHISHEK
5	ADITYA
6	SARTHAK
7	NISCHAY
8	MUSKAN
9	AAKASH GARG

TRIM:

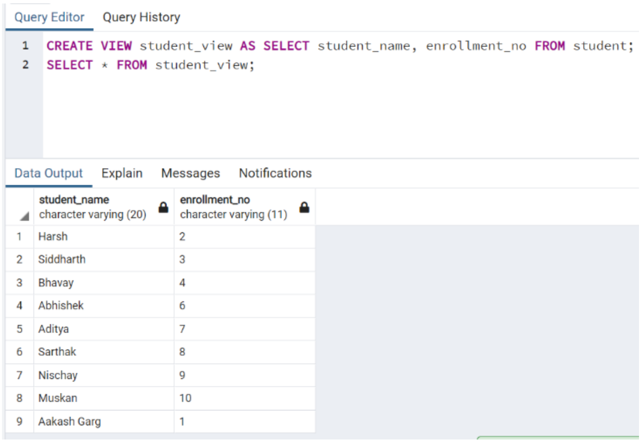


```
SELECT TRIM('A ' FROM (SELECT student_name from student where student_name = 'Aakash Garg'));
```

	btrim text
1	akash Garg

- **VIEWS:**

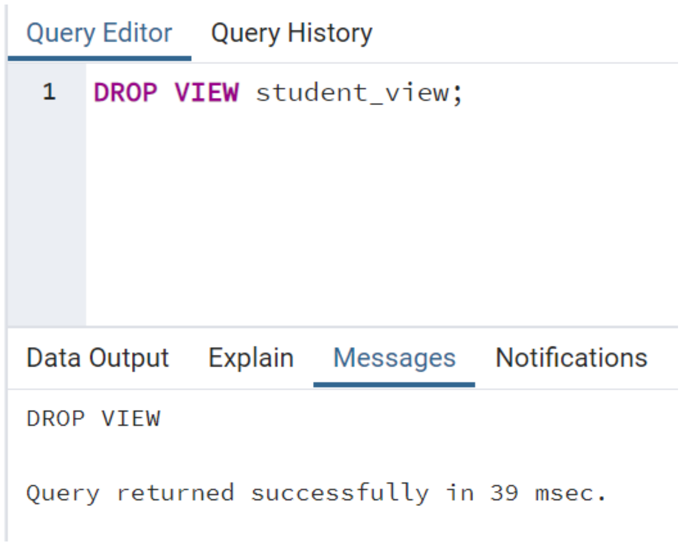
CREATE VIEW:



```
CREATE VIEW student_view AS SELECT student_name, enrollment_no FROM student;
SELECT * FROM student_view;
```

	student_name character varying (20)	enrollment_no character varying (11)
1	Harsh	2
2	Siddharth	3
3	Bhavay	4
4	Abhishek	6
5	Aditya	7
6	Sarthak	8
7	Nischay	9
8	Muskan	10
9	Aakash Garg	1

DROP VIEW:



```
DROP VIEW student_view;
```

```
DROP VIEW

Query returned successfully in 39 msec.
```

Experiment - 4

Advanced Database Management System Lab

Aim

Write SQL queries on Nested queries and Join.

Syeda Reeha Quasar
14114802719
7C7

Experiment - 5

Advanced Database Management System Lab

Aim

Write a program

a. To calculate total students enrolled area-wise.

b. To calculate average marks obtained by students residing in area that starts with "R"

Syeda Reeha Quasar
14114802719
7C7

Experiment - 6

Advanced Database Management System Lab

Aim

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

Syeda Reeha Quasar
14114802719
7C7

Experiment - 7

Advanced Database Management System Lab

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.

Syeda Reeha Quasar
14114802719
7C7

Experiment - 8

Advanced Database Management System Lab

Aim

Procedure & Function

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

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

Syeda Reeha Quasar
14114802719
7C7

Experiment - 9

Advanced Database Management System Lab

Aim

Trigger

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

Syeda Reeha Quasar
14114802719
7C7

Case Study Experiment - 10

Advanced Database Management System Lab

Aim

Design Database - Inventory Database(represent for purchase/order details).

Syeda Reeha Quasar
14114802719
7C7

EXPERIMENT – 4

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.

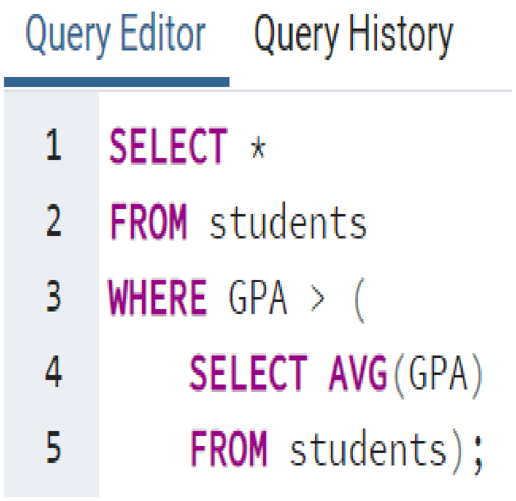
Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

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.

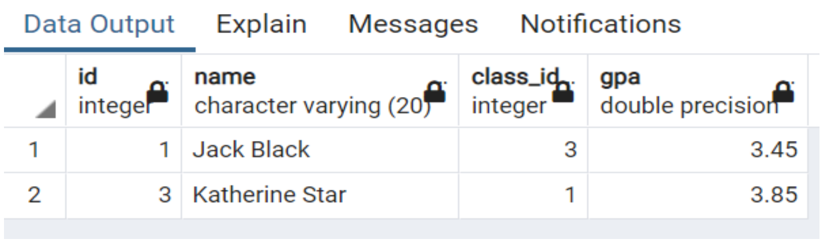
- **Nested Queries**



```
SELECT *
FROM students
WHERE GPA > (
    SELECT AVG(GPA)
    FROM students);
```

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

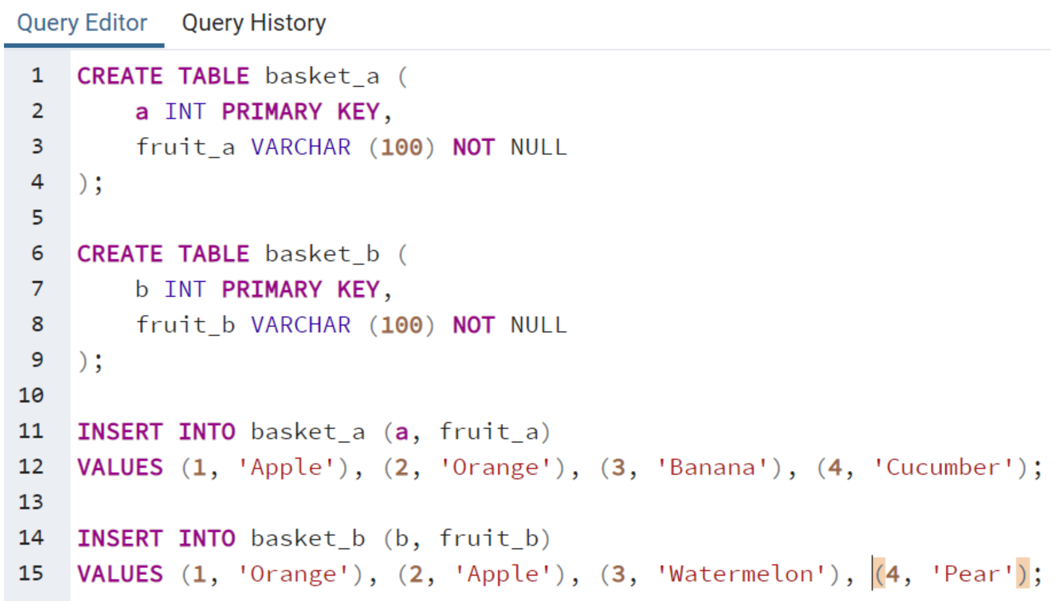
And here is the result:



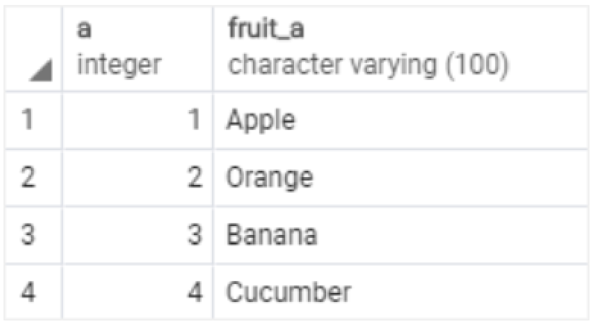
	id integer	name character varying (20)	class_id integer	gpa double precision
1	1	Jack Black	3	3.45
2	3	Katherine Star	1	3.85

- **Joins**

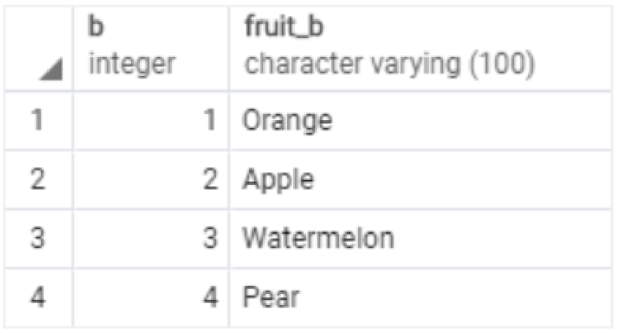
Suppose we have two tables called basket_a and basket_b that store fruits:



```
CREATE TABLE basket_a (
    a INT PRIMARY KEY,
    fruit_a VARCHAR (100) NOT NULL
);

CREATE TABLE basket_b (
    b INT PRIMARY KEY,
    fruit_b VARCHAR (100) NOT NULL
);

INSERT INTO basket_a (a, fruit_a)
VALUES (1, 'Apple'), (2, 'Orange'), (3, 'Banana'), (4, 'Cucumber');

INSERT INTO basket_b (b, fruit_b)
VALUES (1, 'Orange'), (2, 'Apple'), (3, 'Watermelon'), (4, 'Pear');
```



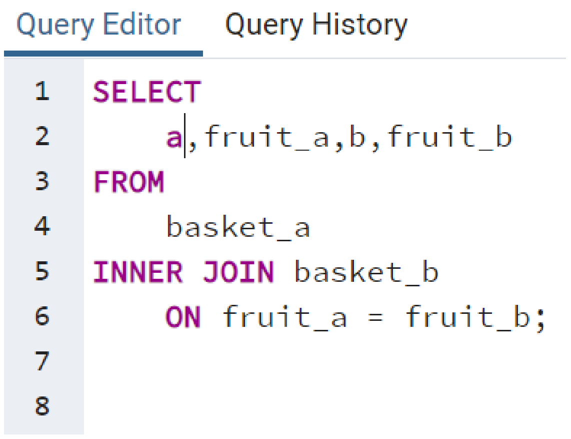
	a integer	fruit_a character varying (100)
1	1	Apple
2	2	Orange
3	3	Banana
4	4	Cucumber



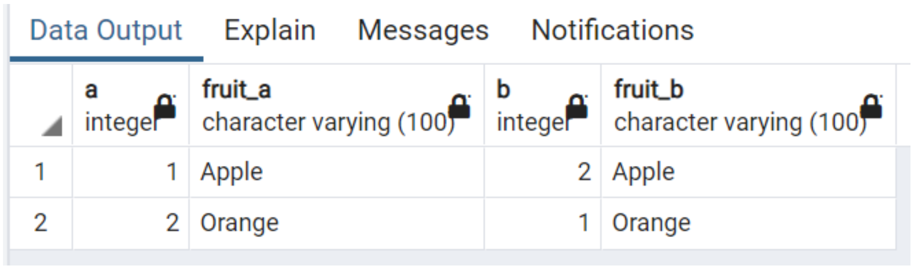
	b integer	fruit_b character varying (100)
1	1	Orange
2	2	Apple
3	3	Watermelon
4	4	Pear

PostgreSQL inner join

The following statement joins the first table (basket_a) with the second table (basket_b) by matching the values in the fruit_a and fruit_b columns:



```
SELECT
    a,fruit_a,b,fruit_b
FROM
    basket_a
INNER JOIN basket_b
    ON fruit_a = fruit_b;
```

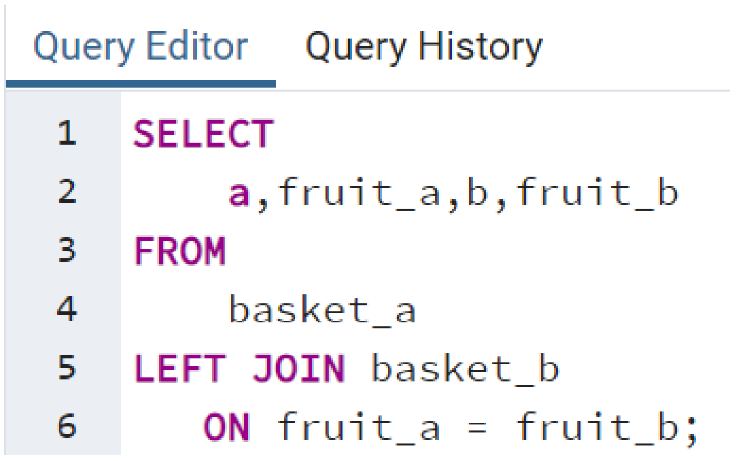

Data Output | Explain | Messages | Notifications

	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange

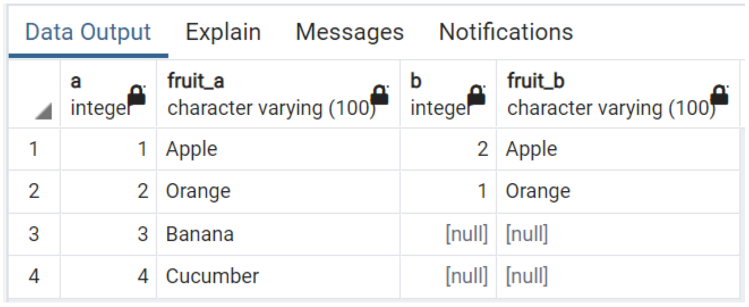
The inner join examines each row in the first table (basket_a). It compares the value in the fruit_a column with the value in the fruit_b column of each row in the second table (basket_b). If these values are equal, the inner join creates a new row that contains columns from both tables and adds this 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.


Query Editor | Query History

```
SELECT
    a,fruit_a,b,fruit_b
FROM
    basket_a
LEFT JOIN basket_b
   ON fruit_a = fruit_b;
```


Data Output | Explain | Messages | Notifications

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

The left join starts selecting data from the left table. It compares values in the fruit_a column with the values in the fruit_b column in the basket_b table.

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

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

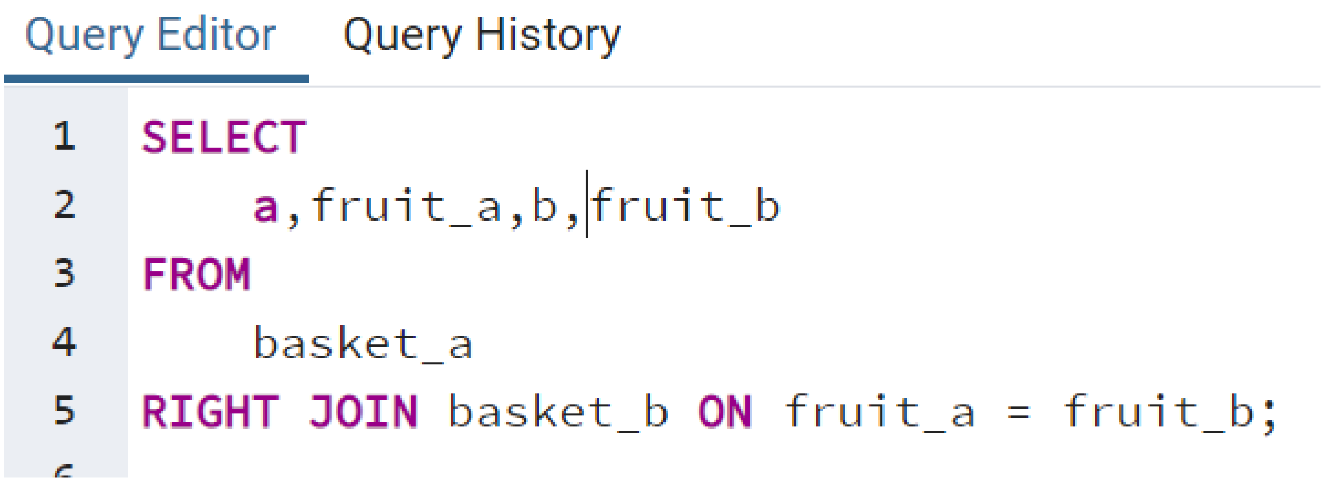
PostgreSQL right join

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

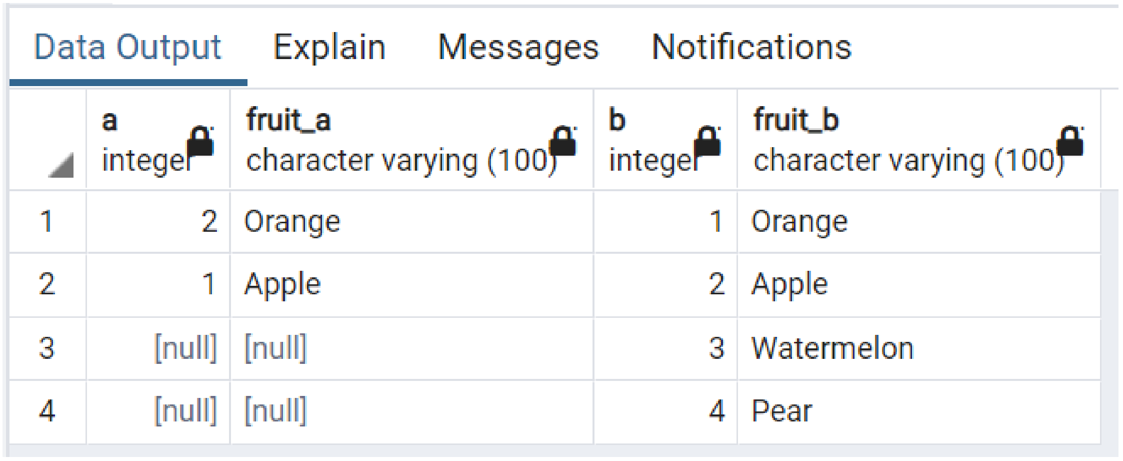
If these values are equal, the right join creates a new row that contains columns from both tables.

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

The following statement uses the right join to join the basket_a table with the basket_b table:



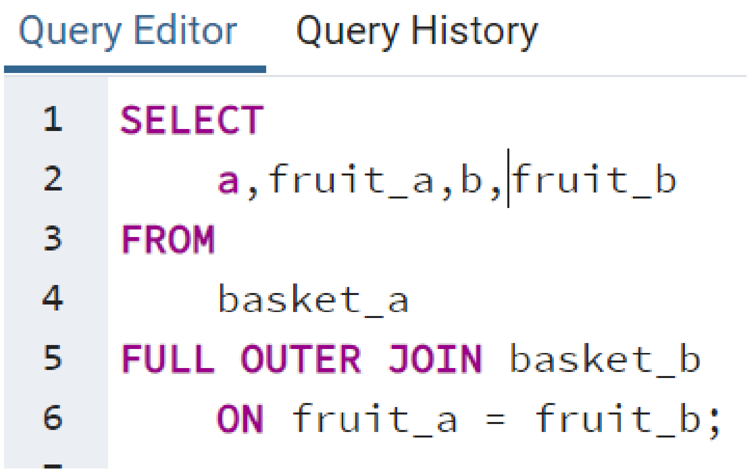
```
SELECT
    a,fruit_a,b,fruit_b
FROM
    basket_a
RIGHT JOIN basket_b ON fruit_a = fruit_b;
```



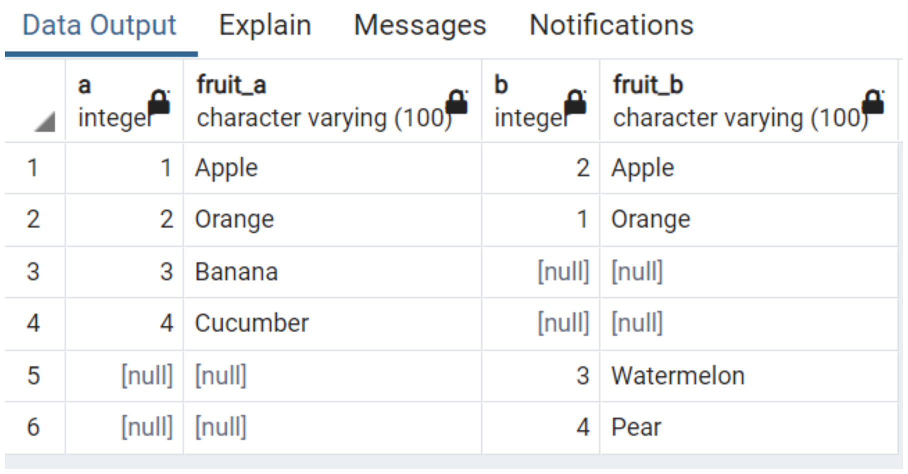
	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	2	Orange	1	Orange
2	1	Apple	2	Apple
3	[null]	[null]	3	Watermelon
4	[null]	[null]	4	Pear

PostgreSQL full outer join

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



```
SELECT
    a,fruit_a,b,fruit_b
FROM
    basket_a
FULL OUTER JOIN basket_b
    ON fruit_a = fruit_b;
```



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

EXPERIMENT – 5

Aim:

Write a program

a. To calculate total students enrolled area-wise.

b. To calculate average marks obtained by students residing in area that starts with "R"

Theory:

Count Function:

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

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

Like Function:

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

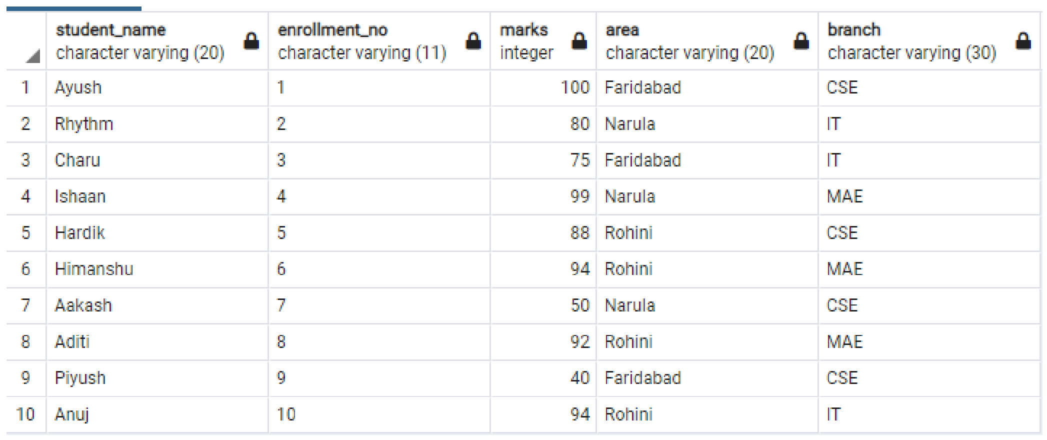
There are two wildcards used in conjunction with the LIKE operator –

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

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

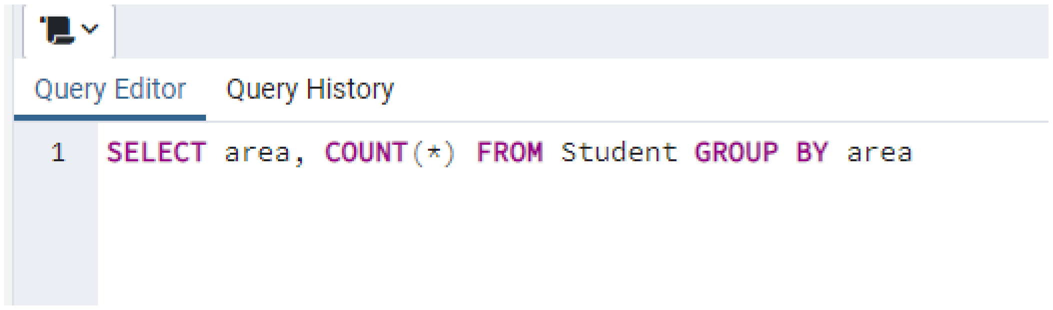
If either of these two signs is not used in conjunction with the LIKE clause, then the LIKE acts like the equals operator.

Main Table:



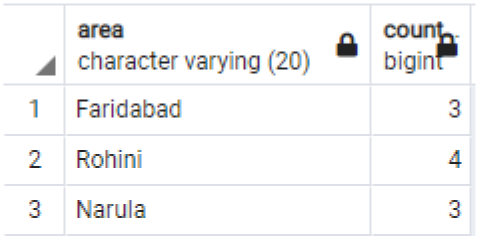
	student_name character varying (20)	enrollment_no character varying (11)	marks integer	area character varying (20)	branch character varying (30)
1	Ayush	1	100	Faridabad	CSE
2	Rhythm	2	80	Narula	IT
3	Charu	3	75	Faridabad	IT
4	Ishaan	4	99	Narula	MAE
5	Hardik	5	88	Rohini	CSE
6	Himanshu	6	94	Rohini	MAE
7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

a.)



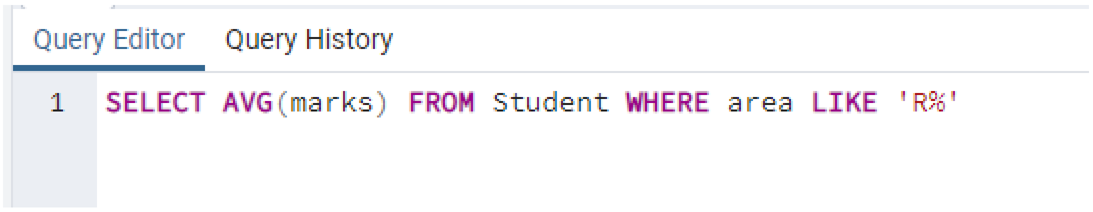
Query Editor Query History

```
SELECT area, COUNT(*) FROM Student GROUP BY area
```



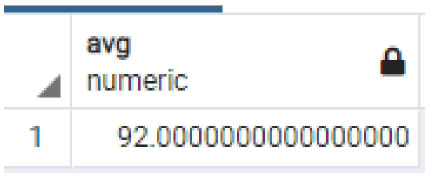
	area character varying (20)	count bigint
1	Faridabad	3
2	Rohini	4
3	Narula	3

b.)



Query Editor Query History

```
SELECT AVG(marks) FROM Student WHERE area LIKE 'R%'
```



	avg numeric
1	92.0000000000000000

EXPERIMENT – 6

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

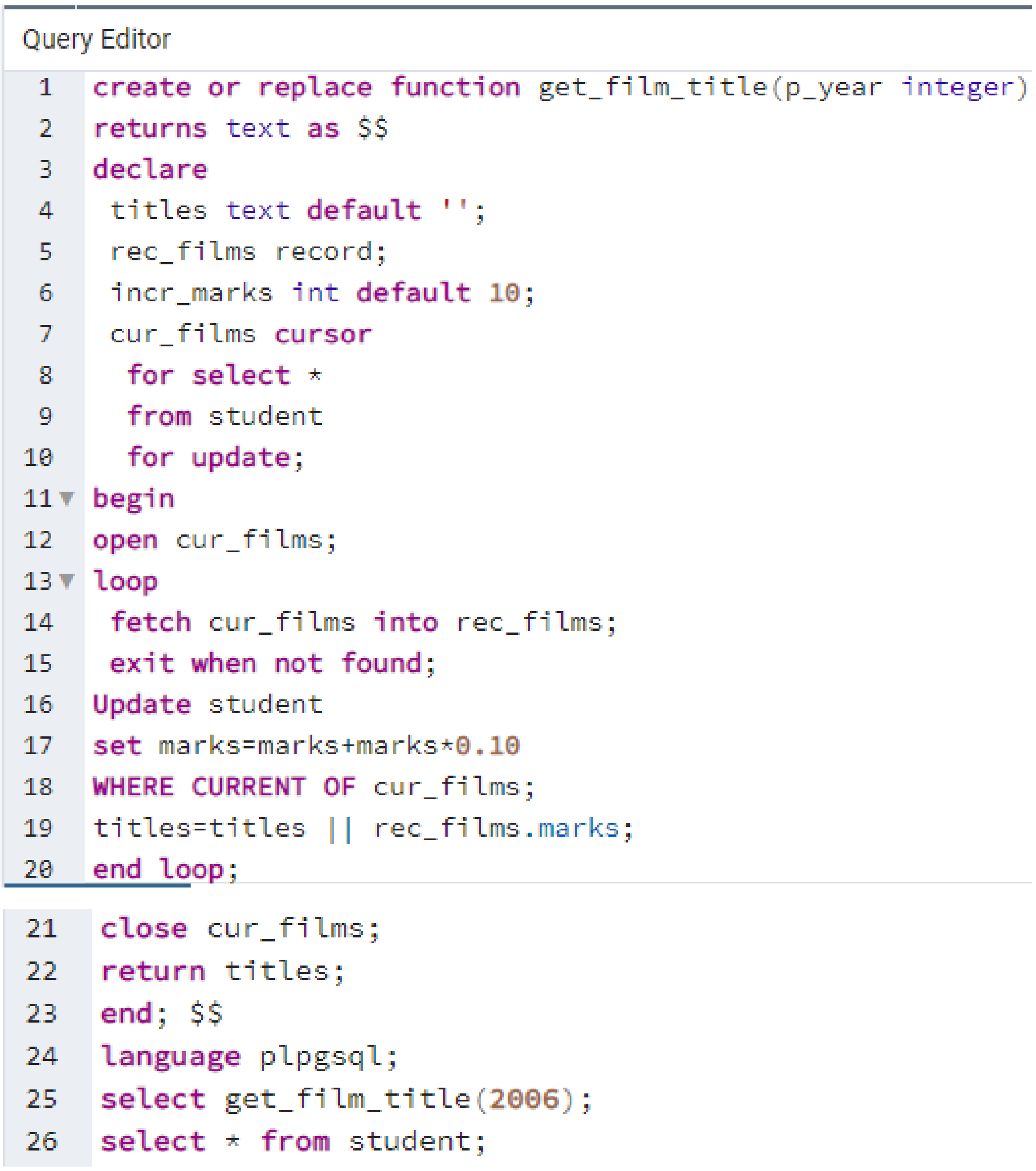
Close cursor connection.

Syntax: CLOSE cursor_name

Deallocate cursor memory.

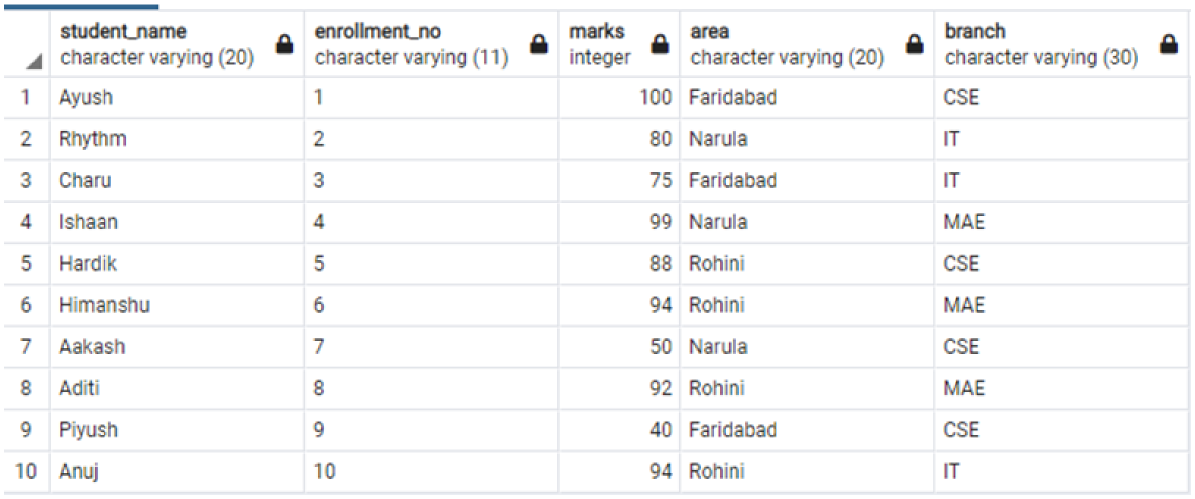
Syntax: DEALLOCATE cursor_name

CODE:

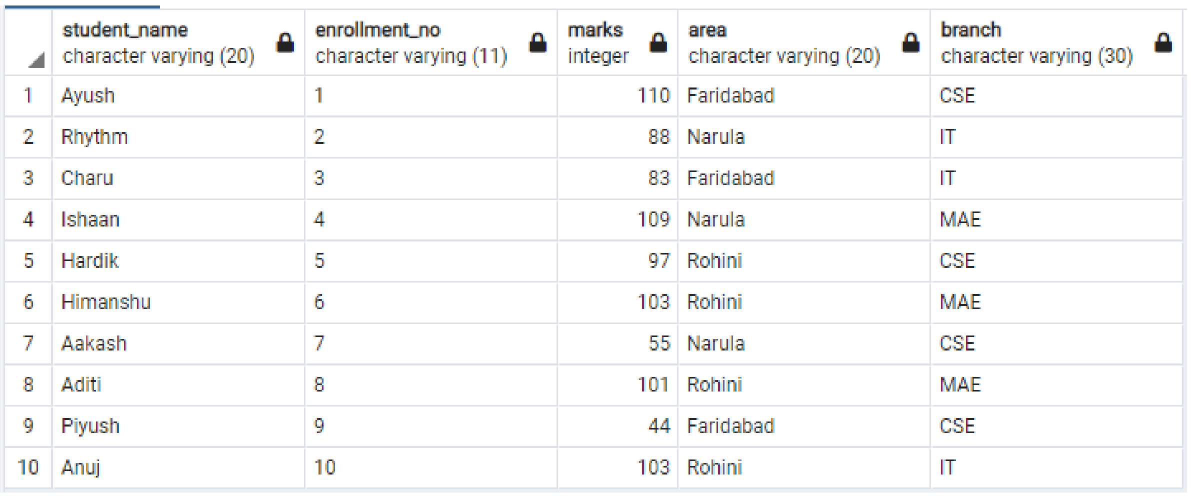


```
create or replace function get_film_title(p_year integer)
returns text as $$
declare
 titles text default '';
 rec_films record;
 incr_marks int default 10;
 cur_films cursor
  for select *
  from student
  for update;
begin
open cur_films;
loop
 fetch cur_films into rec_films;
 exit when not found;
Update student
set marks=marks+marks*0.10
WHERE CURRENT OF cur_films;
titles=titles || rec_films.marks;
end loop;
close cur_films;
return titles;
end; $$
language plpgsql;
select get_film_title(2006);
select * from student;
```

OUTPUT:



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



	student_name character varying (20)	enrollment_no character varying (11)	marks integer	area character varying (20)	branch character varying (30)
1	Ayush	1	110	Faridabad	CSE
2	Rhythm	2	88	Narula	IT
3	Charu	3	83	Faridabad	IT
4	Ishaan	4	109	Narula	MAE
5	Hardik	5	97	Rohini	CSE
6	Himanshu	6	103	Rohini	MAE
7	Aakash	7	55	Narula	CSE
8	Aditi	8	101	Rohini	MAE
9	Piyush	9	44	Faridabad	CSE
10	Anuj	10	103	Rohini	IT

EXPERIMENT – 7

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.

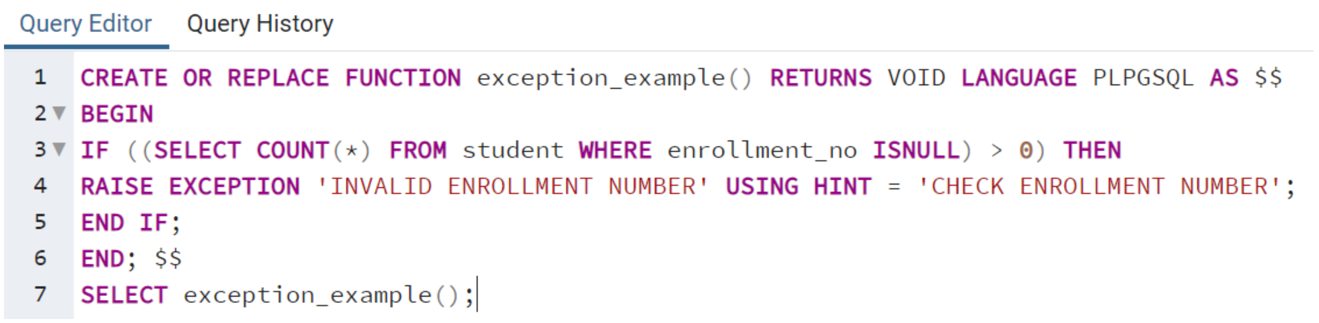
3. Third, if there is a match, the corresponding handle_exception statements will execute. PL/pgSQL passes the control to the statement after the end keyword.

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.

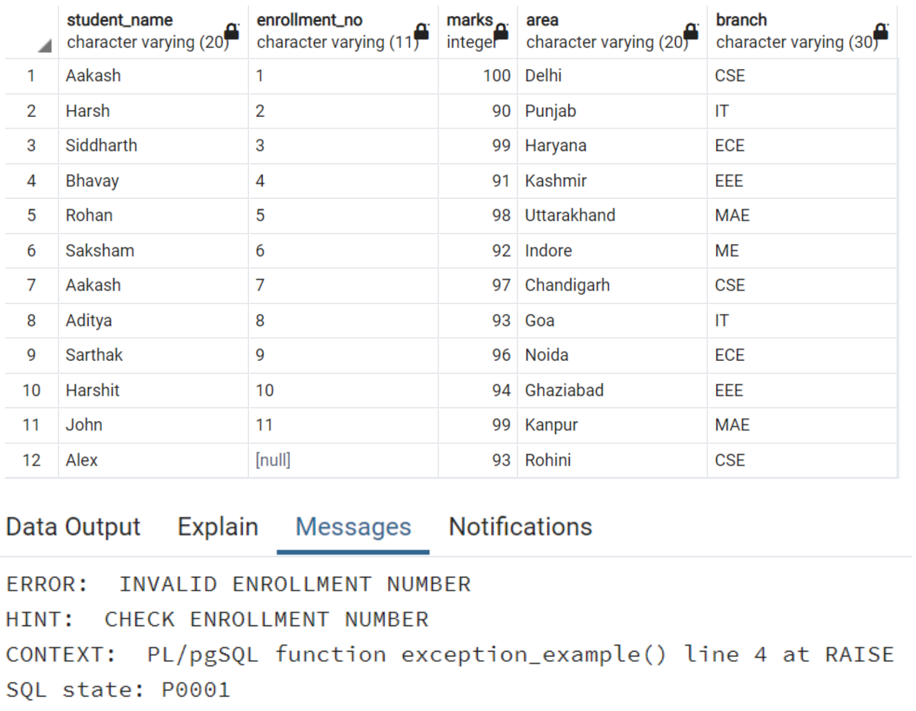
It's also possible to specify the error condition by SQLSTATE code. For example, P0002 for no_data_found and P0003 for too_many_rows.

CODE:



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

OUTPUT:



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

```
ERROR:  INVALID ENROLLMENT NUMBER
HINT:  CHECK ENROLLMENT NUMBER
CONTEXT:  PL/pgSQL function exception_example() line 4 at RAISE
SQL state: P0001
```

EXPERIMENT – 8

Aim:

Procedure & Function

a. Write the procedure to get the average marks of students for branch "CSE".

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

Theory:

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

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

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

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

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

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

Syntax for Creating a PROCEDURE

```
CREATE [OR REPLACE] PROCEDURE procedure_name

[(parameter_name [IN | OUT | IN OUT] type [,...])]

{IS | AS}

BEGIN

<procedure_body>

END procedure_name;
```

Creating a FUNCTION

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

CREATE OR REPLACE PROCEDURE statement is as follows –

CREATE [OR REPLACE] FUNCTION

function_name [(parameter_name [IN | OUT | IN OUT] type [, ...])]

RETURN return_datatype

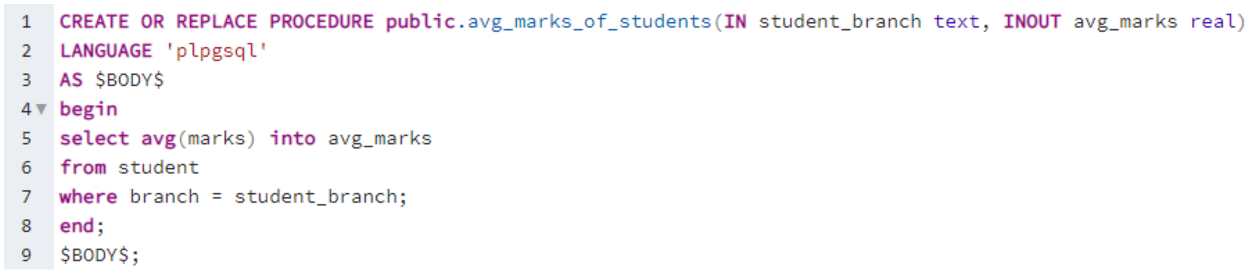
{IS | AS} BEGIN

<function_body>

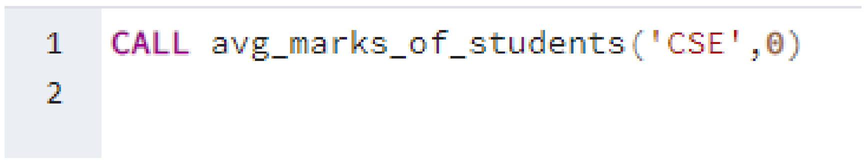
END [function_name];

CODE:

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

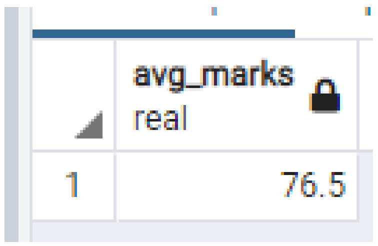


```sql
CREATE OR REPLACE PROCEDURE public.avg_marks_of_students(IN student_branch text, INOUT avg_marks real)
LANGUAGE 'plpgsql'
AS $BODY$
begin
select avg(marks) into avg_marks
from student
where branch = student_branch;
end;
$BODY$;
```



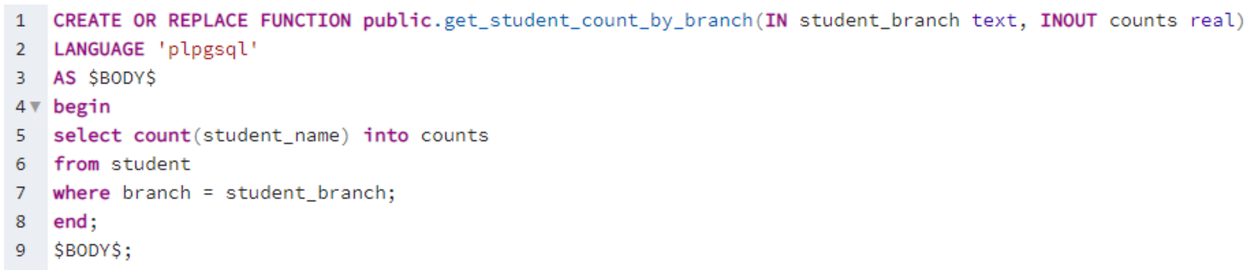
```sql
CALL avg_marks_of_students('CSE',0)
```

OUTPUT:

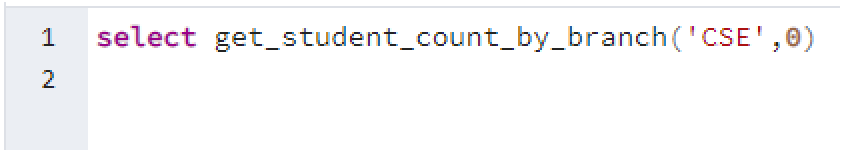


```
   avg_marks
   real
1       76.5
```

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

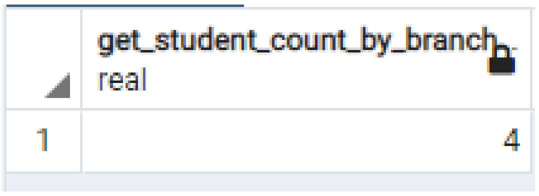


```sql
CREATE OR REPLACE FUNCTION public.get_student_count_by_branch(IN student_branch text, INOUT counts real)
LANGUAGE 'plpgsql'
AS $BODY$
begin
select count(student_name) into counts
from student
where branch = student_branch;
end;
$BODY$;
```



```sql
select get_student_count_by_branch('CSE',0)
```

OUTPUT:



	get_student_count_by_branch real
1	4

EXPERIMENT – 9

Aim:

Trigger

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

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

Theory:

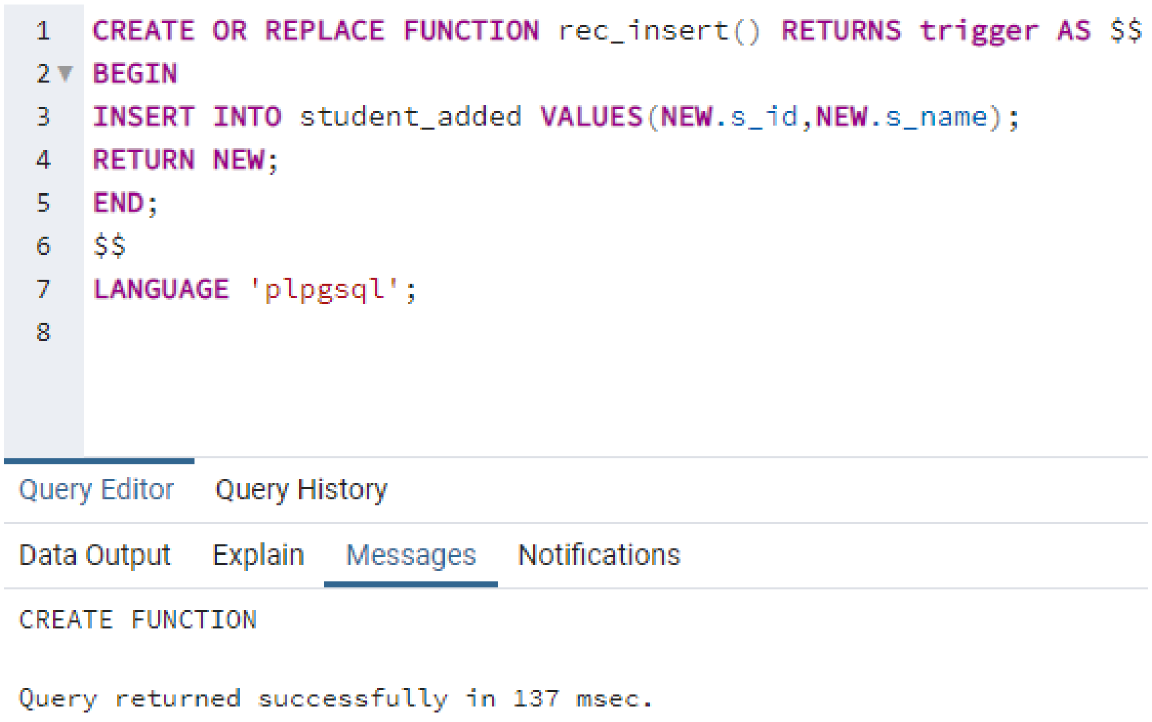
A trigger is a named database object that is associated with a table, and it activates when a particular event (e.g. an insert, update or delete) occurs for the table/views. The statement CREATE TRIGGER creates a new trigger in PostgreSQL.

Syntax for creating a trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name

{BEFORE | AFTER | INSTEAD OF}

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

[OF col_name]

ON table_name

[REFERENCING OLD AS o NEW AS n]

[FOR EACH ROW]

WHEN (condition)

DECLARE

Declaration-statements

BEGIN

Executable-statements

EXCEPTION

Exception-handling-statements

END;
```

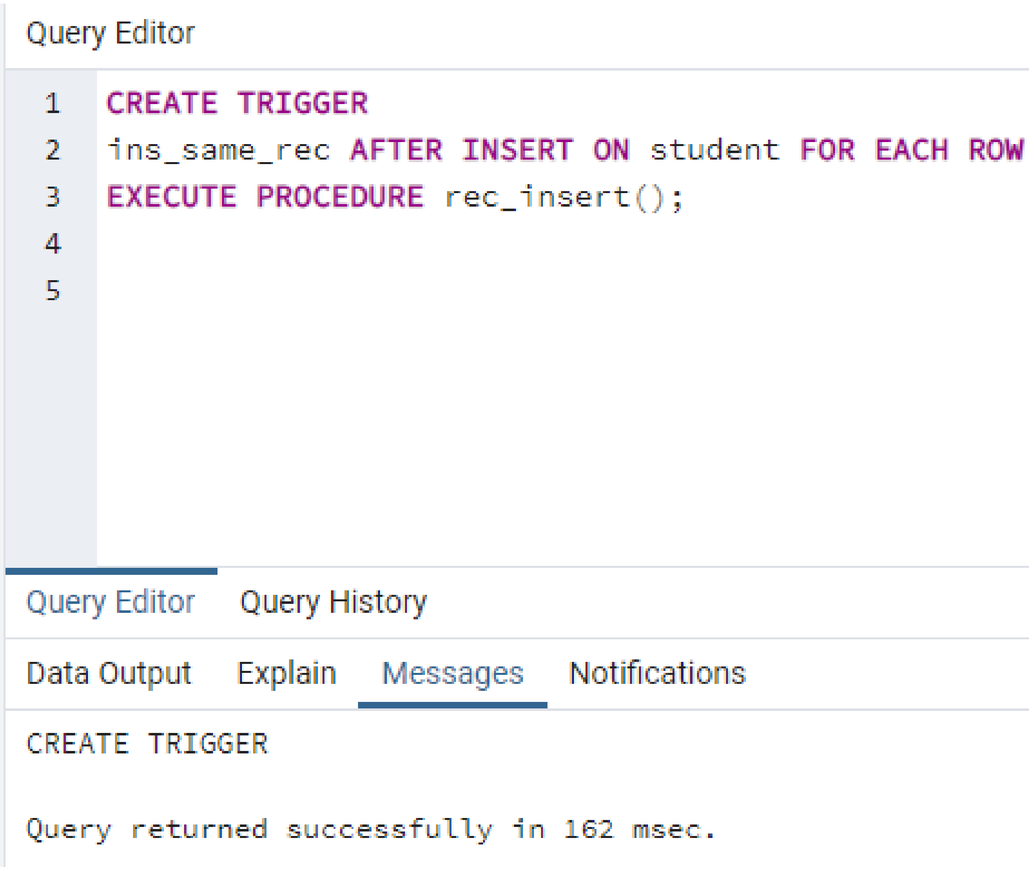
CODE:

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



```
CREATE OR REPLACE FUNCTION rec_insert() RETURNS trigger AS $$
BEGIN
INSERT INTO student_added VALUES(NEW.s_id,NEW.s_name);
RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

```

```
CREATE FUNCTION

Query returned successfully in 137 msec.
```



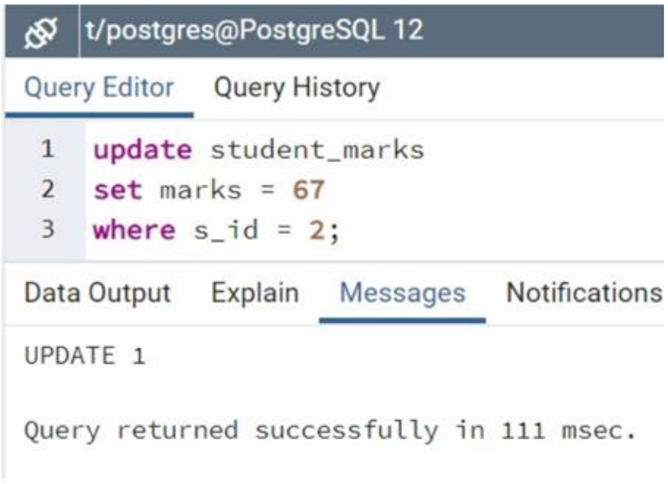
```
CREATE TRIGGER
ins_same_rec AFTER INSERT ON student FOR EACH ROW
EXECUTE PROCEDURE rec_insert();


```

```
CREATE TRIGGER

Query returned successfully in 162 msec.
```

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

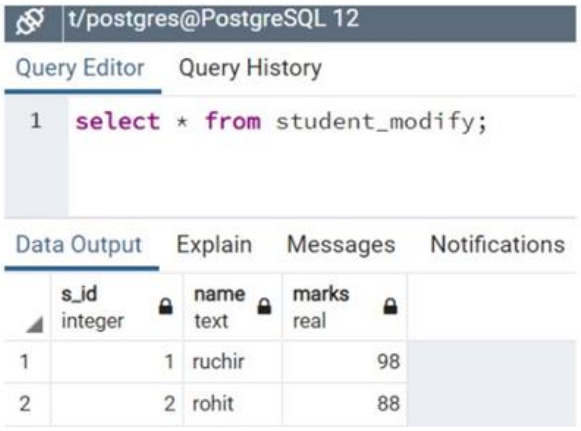
```
CREATE TRIGGER upd_same_rec
AFTER UPDATE
ON student_marks
FOR EACH ROW
EXECUTE PROCEDURE rec_update();

```

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



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

```
UPDATE 1

Query returned successfully in 111 msec.
```



```
select * from student_modify;
```

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

EXPERIMENT – 10

Aim:

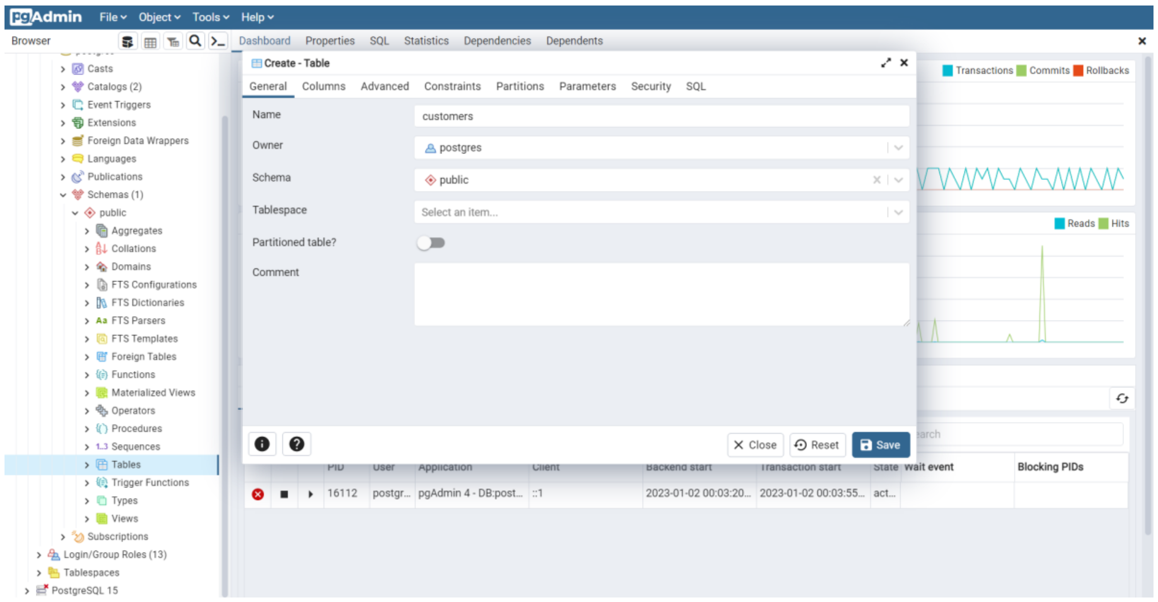
Design Database - Inventory Database(represent for purchase/order details).

Theory:

Steps to design a database schema of the Inventory Management System to manage the customers, items, and purchase orders.

Table Name: Customers

Description: Used to store customers information having data ID, first name, last name, and email.

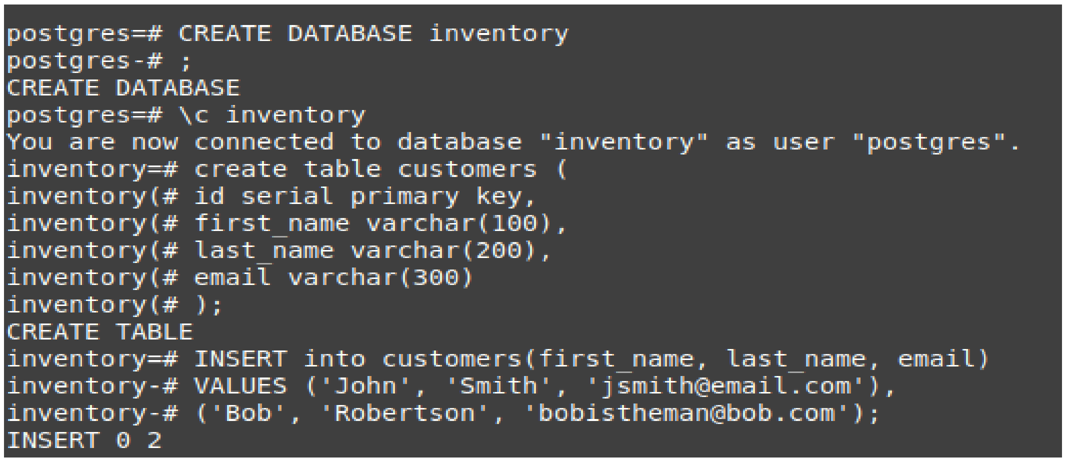


Create - Table

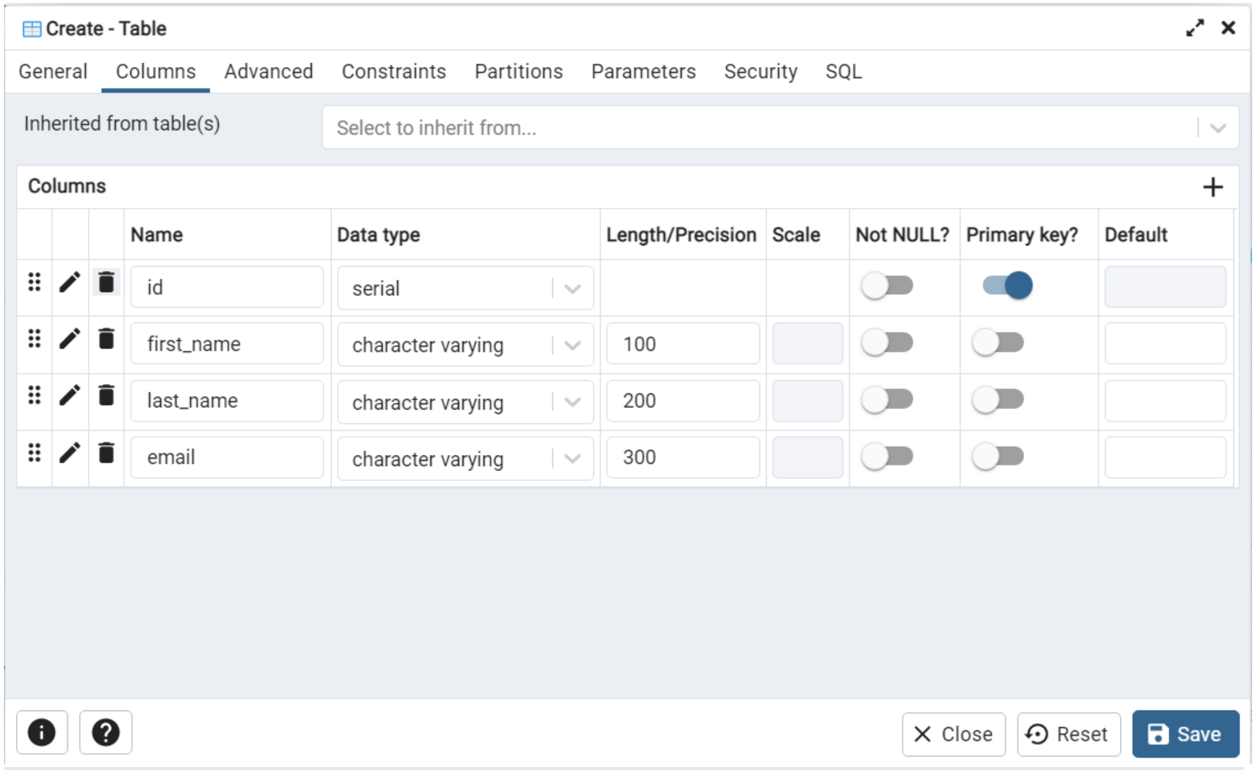
General | Columns | Advanced | Constraints | Partitions | Parameters | Security | SQL

Name	customers
Owner	postgres
Schema	public
Tablespace	Select an item...
Partitioned table?	
Comment	

PID	User	Application	Client	Backend start	Transaction start	State	Wait event	Blocking PIDs
16112	postgr...	pgAdmin 4 - DB:post...	::1	2023-01-02 00:03:20...	2023-01-02 00:03:55...	act...		



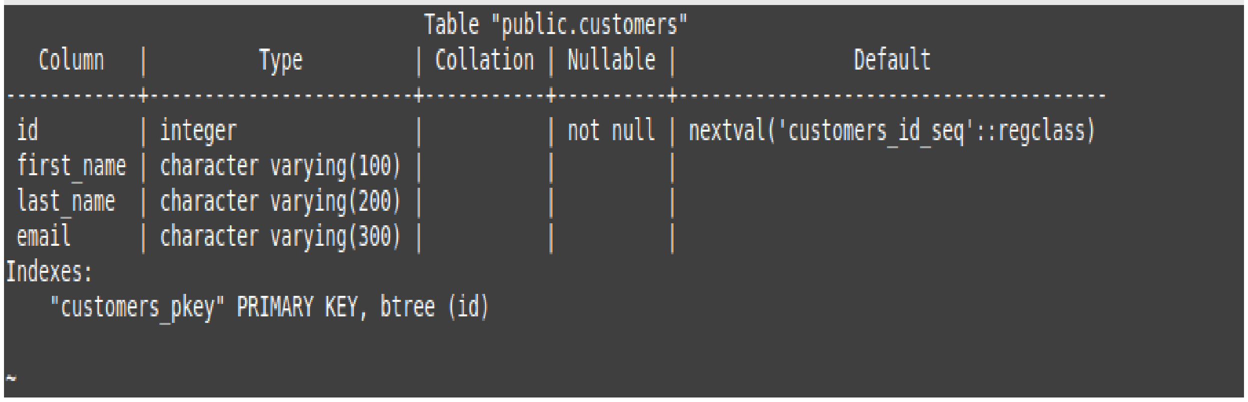
```
postgres=# CREATE DATABASE inventory
postgres-# ;
CREATE DATABASE
postgres=# \c inventory
You are now connected to database "inventory" as user "postgres".
inventory=# create table customers (
inventory(# id serial primary key,
inventory(# first_name varchar(100),
inventory(# last_name varchar(200),
inventory(# email varchar(300)
inventory(# );
CREATE TABLE
inventory=# INSERT into customers(first_name, last_name, email)
inventory-# VALUES ('John', 'Smith', 'jsmith@email.com'),
inventory-# ('Bob', 'Robertson', 'bobistheman@bob.com');
INSERT 0 2
```



Inherited from table(s): Select to inherit from...

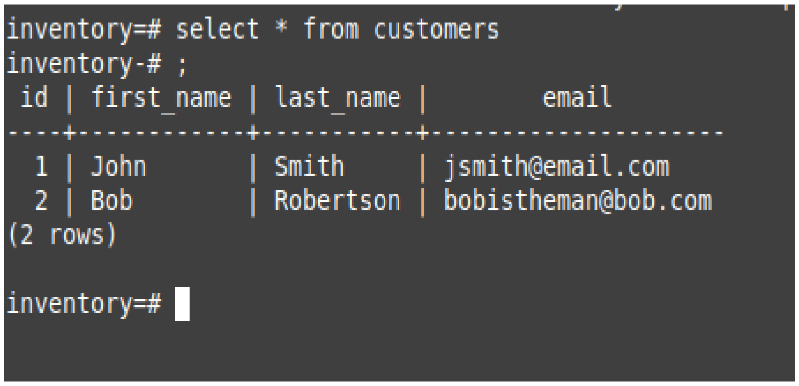
Columns

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
id	serial				✓	
first_name	character varying	100				
last_name	character varying	200				
email	character varying	300				

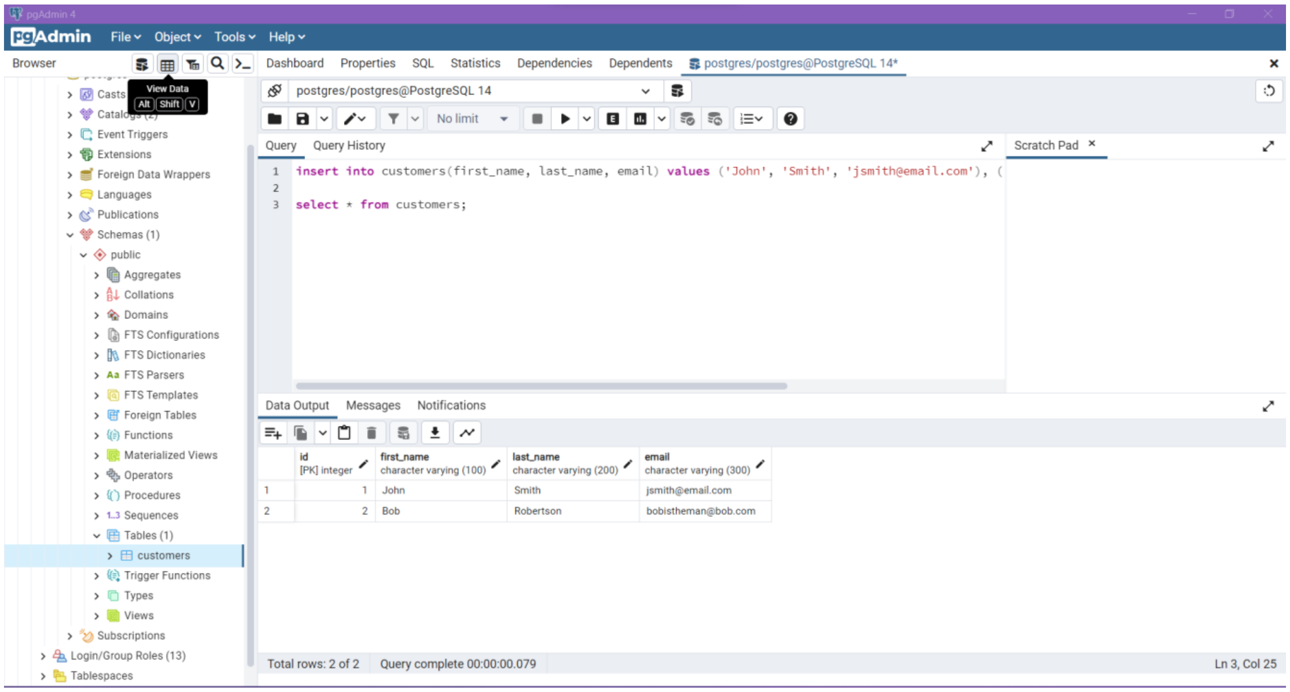


```
                              Table "public.customers"
  Column    |          Type          | Collation | Nullable |               Default
------------+------------------------+-----------+----------+--------------------------------------
 id         | integer                |           | not null | nextval('customers_id_seq'::regclass)
 first_name | character varying(100) |           |          |
 last_name  | character varying(200) |           |          |
 email      | character varying(300) |           |          |
Indexes:
    "customers_pkey" PRIMARY KEY, btree (id)


~
```

Data:



```
inventory=# select * from customers
inventory-# ;
 id | first_name | last_name |        email
----+------------+-----------+---------------------
  1 | John       | Smith     | jsmith@email.com
  2 | Bob        | Robertson | bobistheman@bob.com
(2 rows)

inventory=#
```



```
insert into customers(first_name, last_name, email) values ('John', 'Smith', 'jsmith@email.com'), (

select * from customers;
```

	id [PK] integer	first_name character varying (100)	last_name character varying (200)	email character varying (300)
1	1	John	Smith	jsmith@email.com
2	2	Bob	Robertson	bobistheman@bob.com

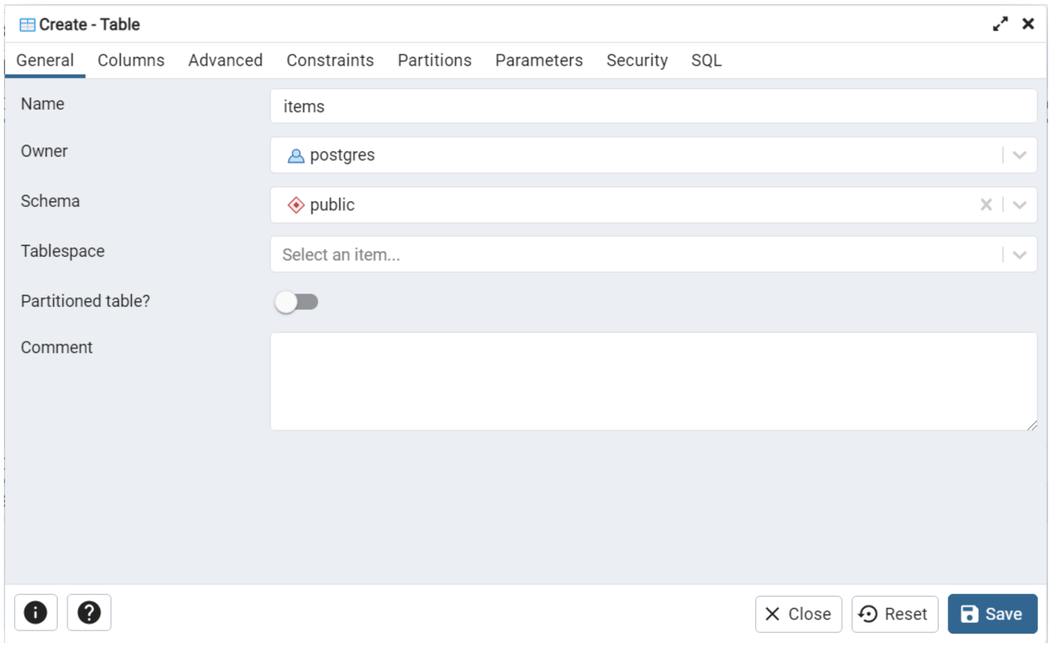
Total rows: 2 of 2   Query complete 00:00:00.079   Ln 3, Col 25

```
insert into customers(first_name, last_name, email) values ('John', 'Smith',
'jsmith@email.com'), ('Bob', 'Robertson', 'bobistheman@bob.com');
```

```
select * from customers;
```

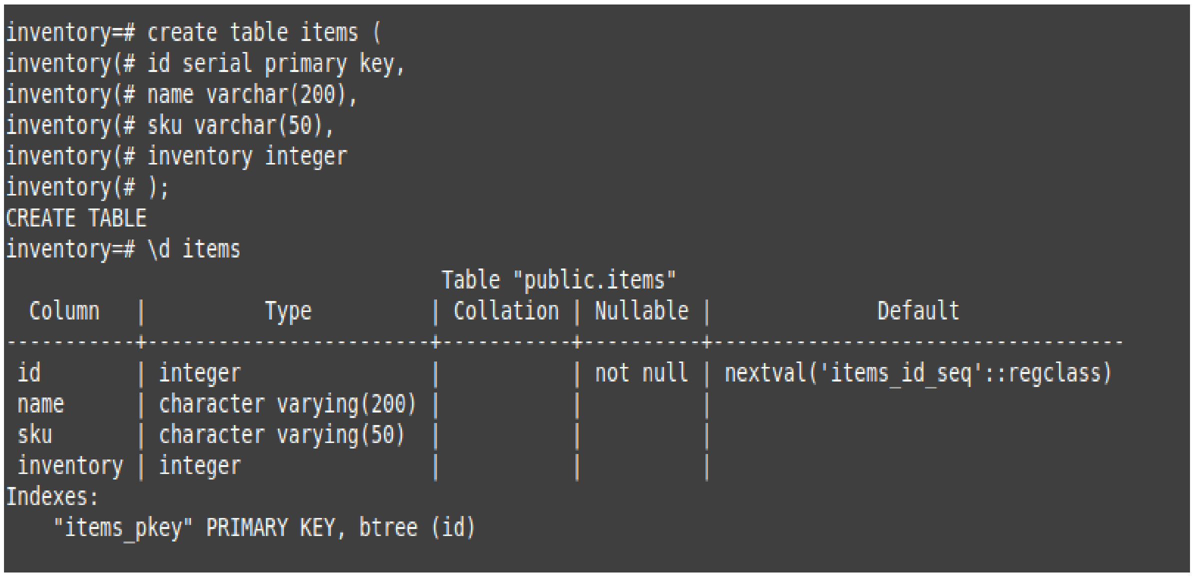
Table Name: items

Description: It will help in storing the items in the inventory having the values ID, its SKU (unit product id), and inventory(number of pieces).

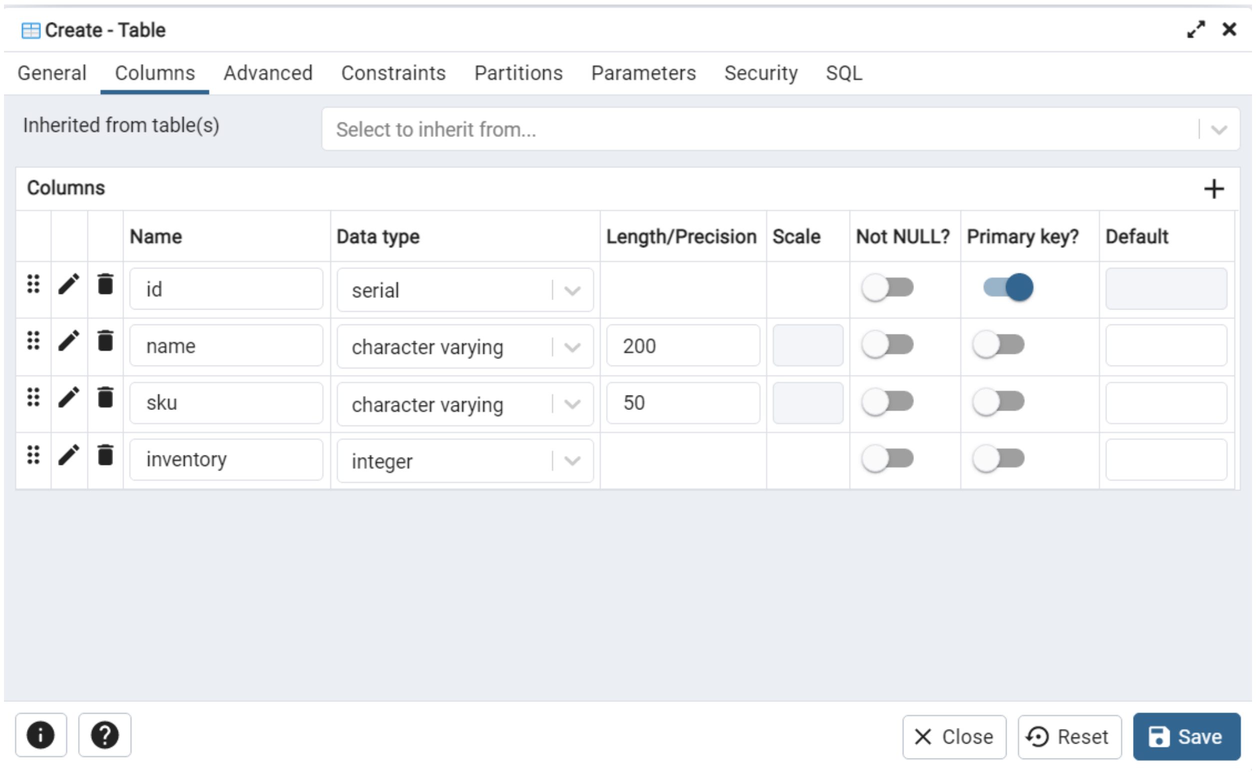


Create - Table

Name	items
Owner	postgres
Schema	public
Tablespace	Select an item...
Partitioned table?	
Comment	



```
inventory=# create table items (
inventory(# id serial primary key,
inventory(# name varchar(200),
inventory(# sku varchar(50),
inventory(# inventory integer
inventory(# );
CREATE TABLE
inventory=# \d items
                                    Table "public.items"
  Column   |          Type          | Collation | Nullable |              Default
-----------+------------------------+-----------+----------+-----------------------------------
 id        | integer                |           | not null | nextval('items_id_seq'::regclass)
 name      | character varying(200) |           |          |
 sku       | character varying(50)  |           |          |
 inventory | integer                |           |          |
Indexes:
    "items_pkey" PRIMARY KEY, btree (id)
```



Inherited from table(s): Select to inherit from...

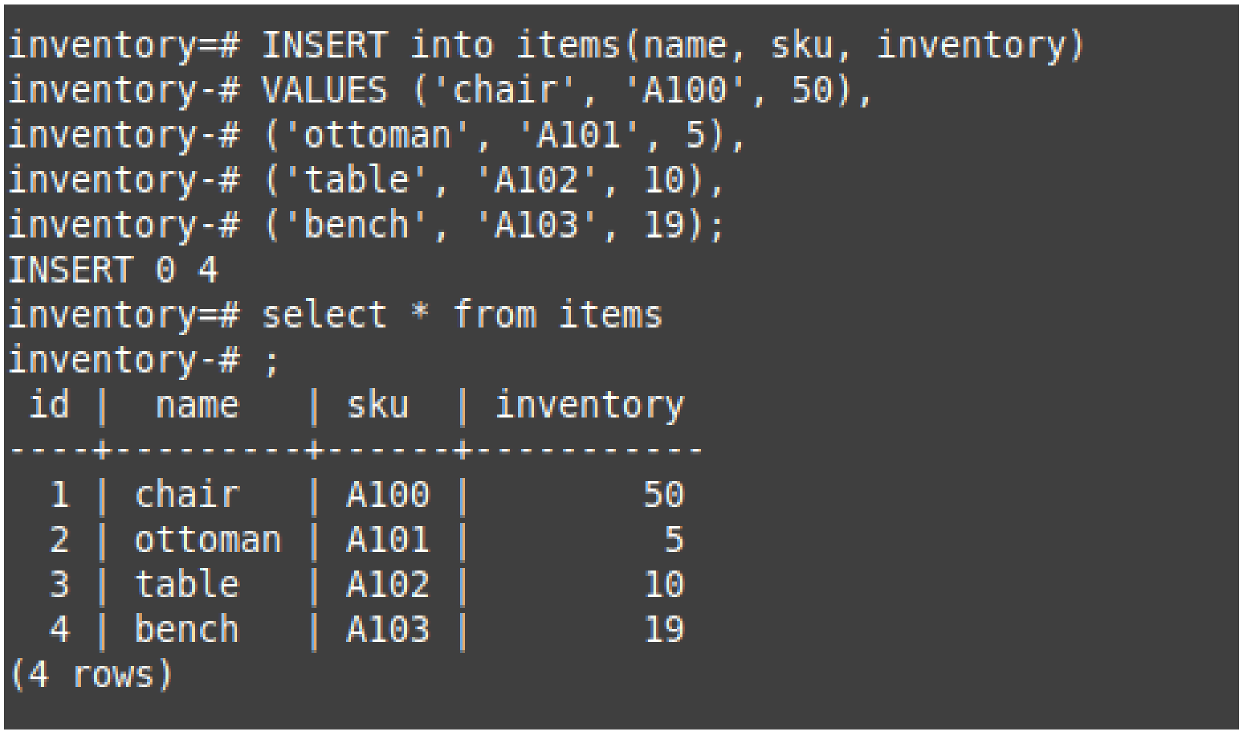
Columns

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
id	serial			off	on	
name	character varying	200		off	off	
sku	character varying	50		off	off	
inventory	integer			off	off	

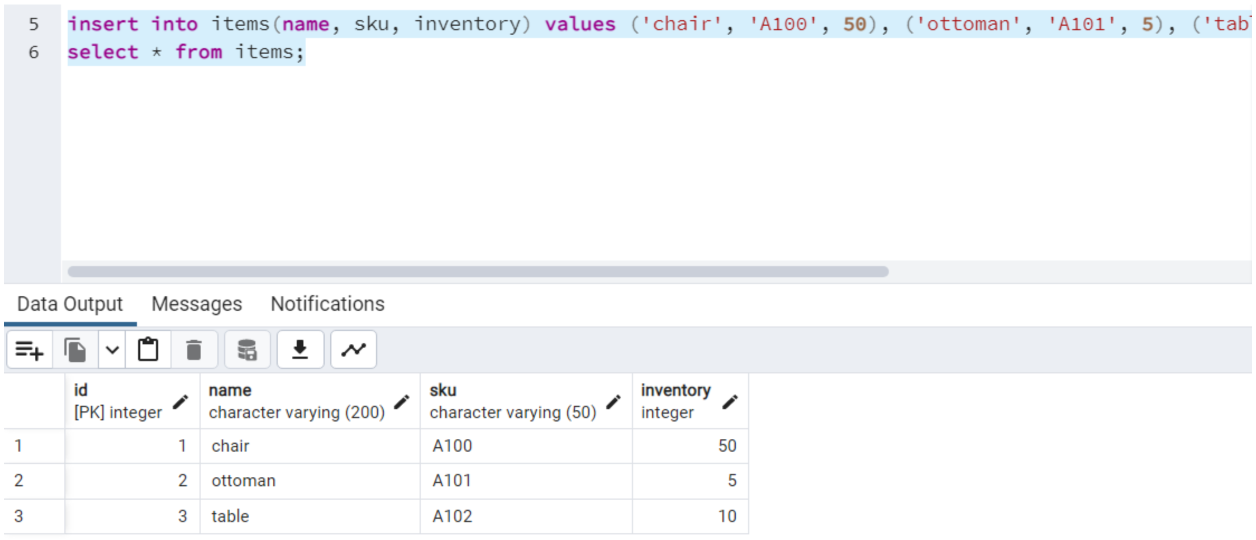
Data:

insert into items(name, sku, inventory) values ('chair', 'A100', 50), ('ottoman', 'A101', 5), ('table', 'A102', 10);

select * from items;



```
inventory=# INSERT into items(name, sku, inventory)
inventory-# VALUES ('chair', 'A100', 50),
inventory-# ('ottoman', 'A101', 5),
inventory-# ('table', 'A102', 10),
inventory-# ('bench', 'A103', 19);
INSERT 0 4
inventory=# select * from items
inventory-# ;
 id |   name   | sku  | inventory
----+----------+------+-----------
  1 | chair    | A100 |        50
  2 | ottoman  | A101 |         5
  3 | table    | A102 |        10
  4 | bench    | A103 |        19
(4 rows)
```

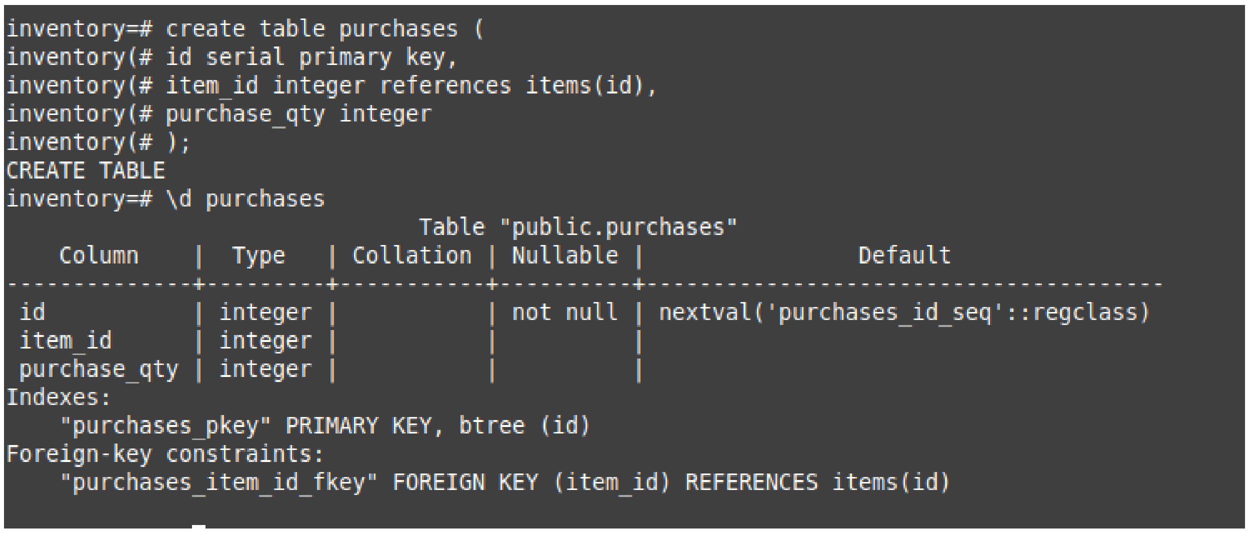


```
insert into items(name, sku, inventory) values ('chair', 'A100', 50), ('ottoman', 'A101', 5), ('tab
select * from items;
```

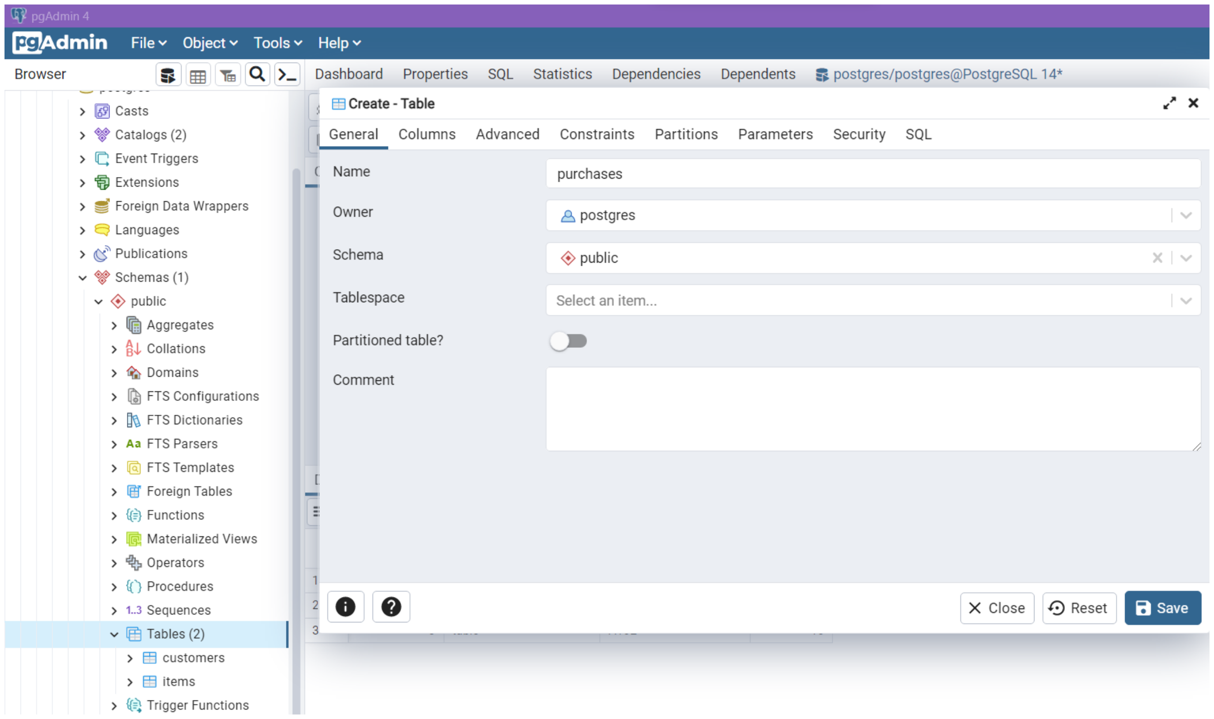
	id [PK] integer	name character varying (200)	sku character varying (50)	inventory integer
1	1	chair	A100	50
2	2	ottoman	A101	5
3	3	table	A102	10

Table Name: Purchases

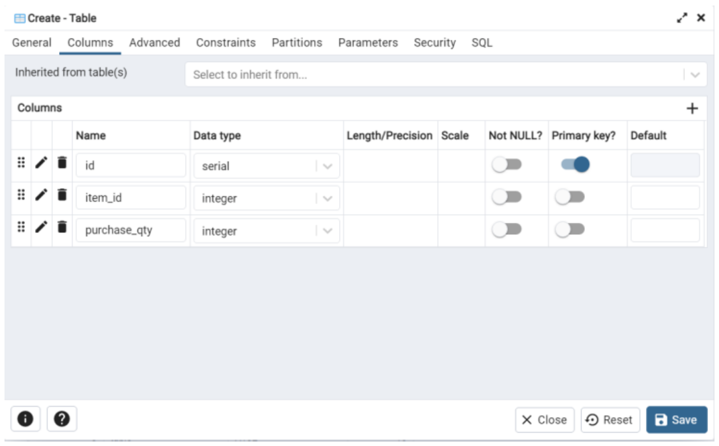
Description: It will help in storing the purchases made by the customers in the inventory having the values ID, item_id and purchase_quantity.



```
inventory=# create table purchases (
inventory(# id serial primary key,
inventory(# item_id integer references items(id),
inventory(# purchase_qty integer
inventory(# );
CREATE TABLE
inventory=# \d purchases
                               Table "public.purchases"
    Column    |  Type   | Collation | Nullable |               Default
--------------+---------+-----------+----------+---------------------------------------
 id           | integer |           | not null | nextval('purchases_id_seq'::regclass)
 item_id      | integer |           |          |
 purchase_qty | integer |           |          |
Indexes:
    "purchases_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "purchases_item_id_fkey" FOREIGN KEY (item_id) REFERENCES items(id)
```



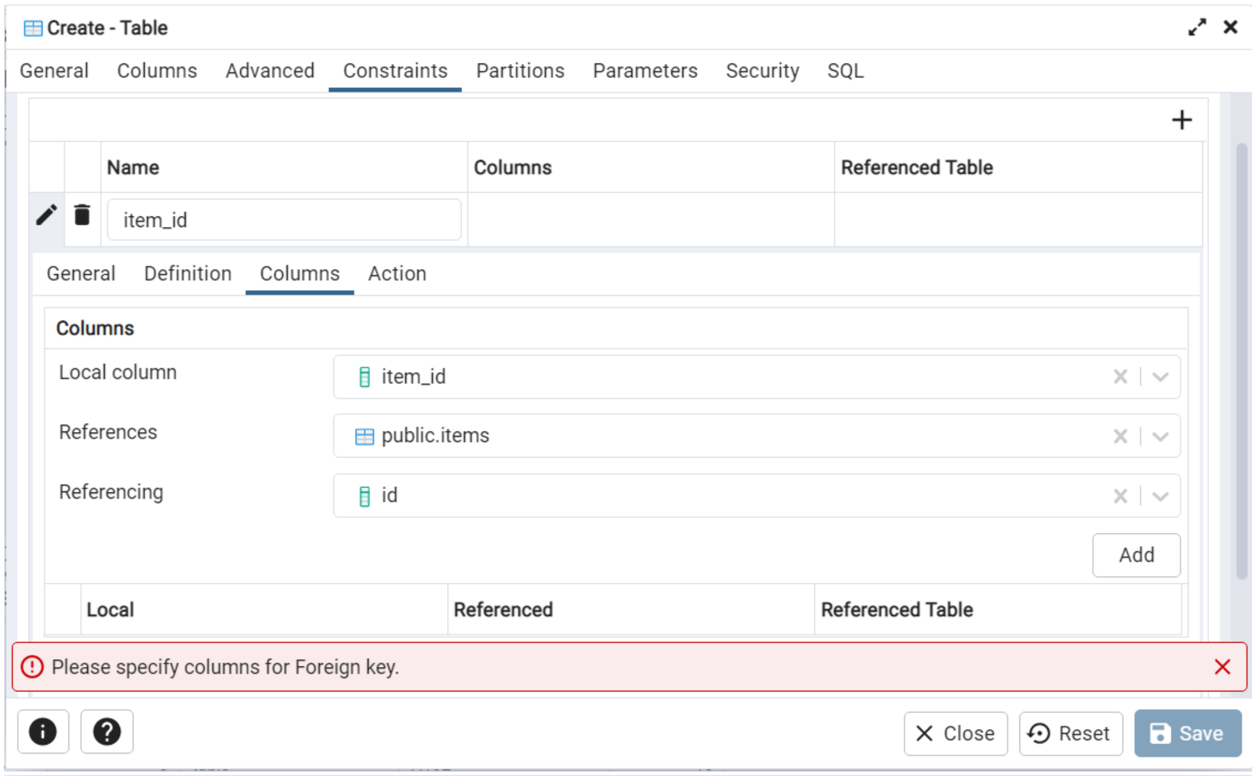
Name	purchases
Owner	postgres
Schema	public
Tablespace	Select an item...
Partitioned table?	off
Comment	



Inherited from table(s): Select to inherit from...

Columns

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
id	serial			off	on	
item_id	integer			off	off	
purchase_qty	integer			off	off	



Create - Table

General | Columns | Advanced | Constraints | Partitions | Parameters | Security | SQL

Name	Columns	Referenced Table
item_id		

General | Definition | Columns | Action

Columns

Local column: item_id

References: public.items

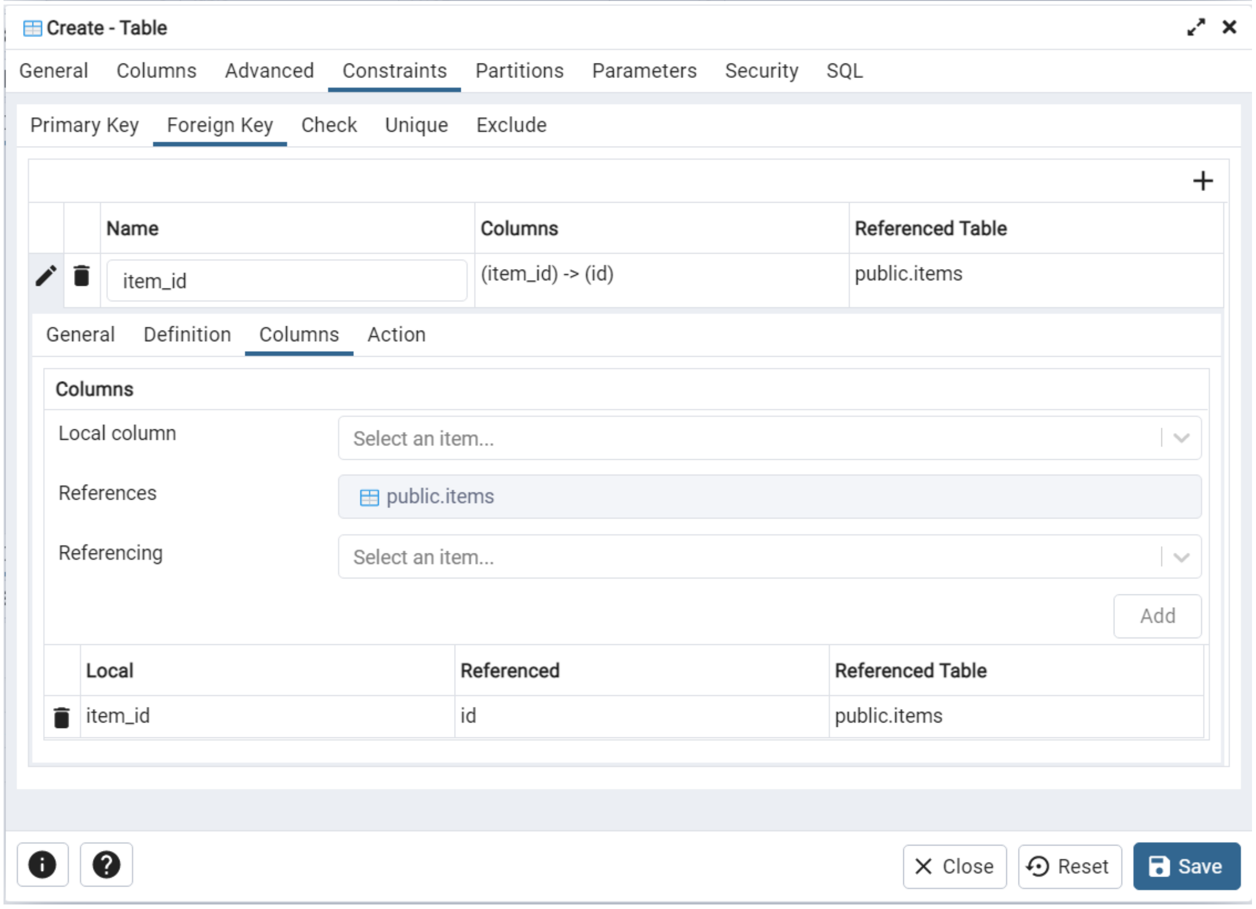
Referencing: id

Add

Local	Referenced	Referenced Table

Please specify columns for Foreign key.

Close | Reset | Save



Create - Table

General | Columns | Advanced | Constraints | Partitions | Parameters | Security | SQL

Primary Key | Foreign Key | Check | Unique | Exclude

Name	Columns	Referenced Table
item_id	(item_id) -> (id)	public.items

General | Definition | Columns | Action

Columns

Local column: Select an item...

References: public.items

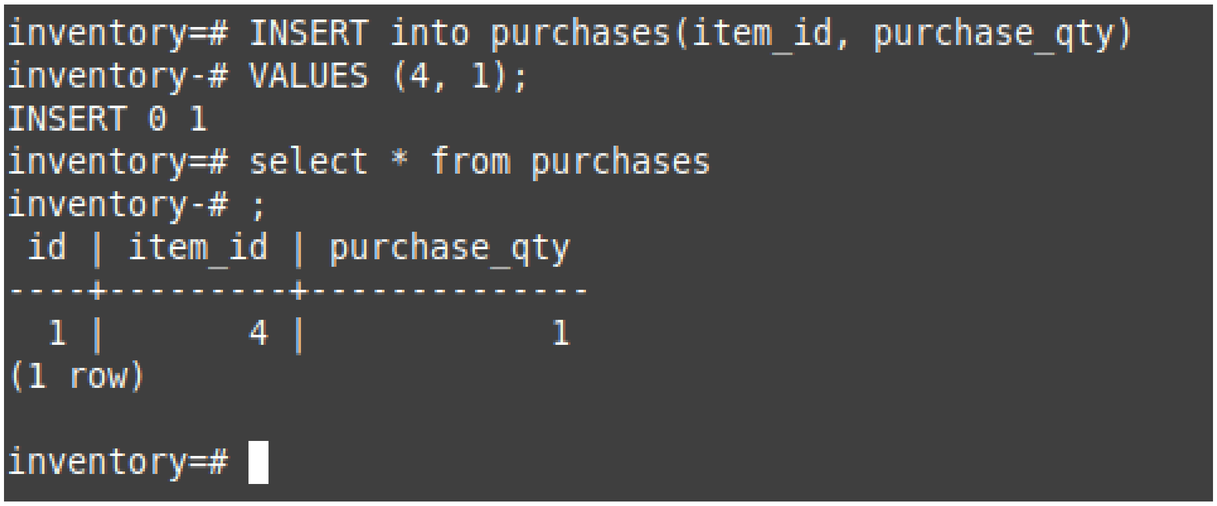
Referencing: Select an item...

Add

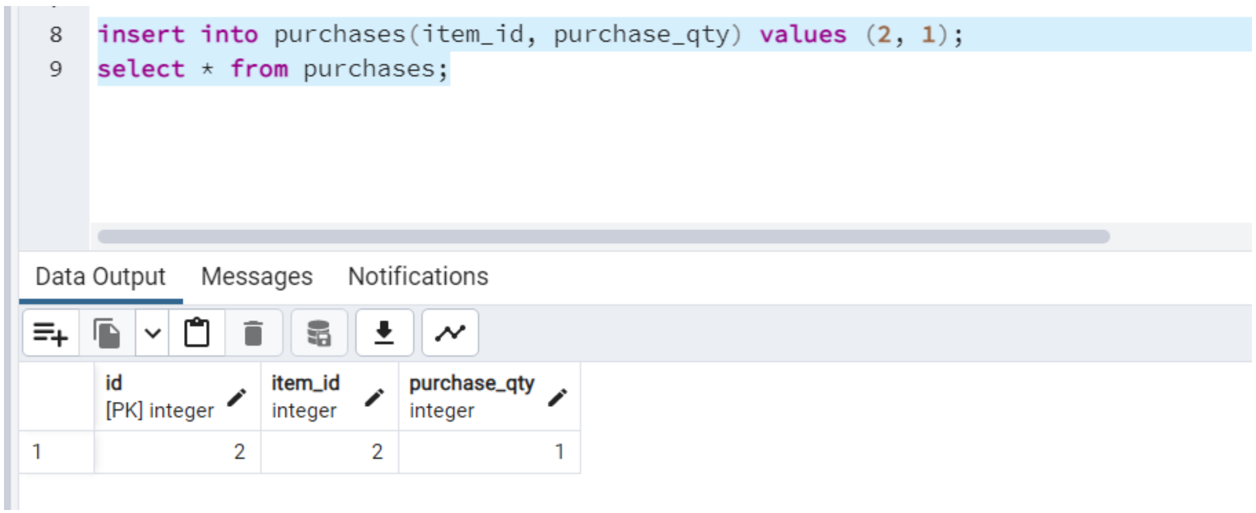
Local	Referenced	Referenced Table
item_id	id	public.items

Close | Reset | Save

Data:



```
inventory=# INSERT into purchases(item_id, purchase_qty)
inventory-# VALUES (4, 1);
INSERT 0 1
inventory=# select * from purchases
inventory-# ;
 id | item_id | purchase_qty
----+---------+--------------
  1 |       4 |            1
(1 row)

inventory=#
```



```
insert into purchases(item_id, purchase_qty) values (2, 1);
select * from purchases;
```

	id [PK] integer	item_id integer	purchase_qty integer
1	2	2	1

The customer database would handle the data the customers with their personal details, then the inventory items would be stored in the items table, and then the purchases would be able to take up the connection between the purchases made by the customers on the table and the items respectively.

Viva Questions

1. What is the process of splitting a large table into smaller pieces called in PostgreSQL?

It is called table partitioning.

2. What is a partitioned table in PostgreSQL?

The partitioned table is a logical structure. It is used to split a large table into smaller pieces, which are called partitions.

3. What purpose does pgAdmin in PostgreSQL server?

The pgAdmin in PostgreSQL is a data administration tool. It serves the purpose of retrieving, developing, testing, and maintaining databases.

4. How can you avoid unnecessary locking of a database?

We can use MVCC (Multi-version concurrency control) to avoid unnecessary locking of a database.

5. What is PL/Python?

PL/Python is a procedural language to which PostgreSQL provides support.

6. Which are the methods PostgreSQL provides to create a new database?

PostgreSQL provides the following methods to create a new database:

- Using CREATE DATABASE, an SQL command
- Using created a command-line executable

7. How do you delete the database in PostgreSQL?

We can delete the database by using any one of the below options:

- Using DROP DATABASE, an SQL command
- Using dropdb a command-line executable

Viva Questions

1. What does a schema contain?

A schema contains tables along with data types, views, indexes, operators, sequences, and functions.

2. What are the different operators in PostgreSQL?

The PostgreSQL operators include - Arithmetic operators, Comparison operators, Logical operators, and Bitwise operators.

3. What are database callback functions called? What is its purpose?

The database callback functions are called PostgreSQL Triggers. When a specified database event occurs, the PostgreSQL Triggers are performed or invoked automatically.

4. What indexes are used?

Indexes are used by the search engine to speed up data retrieval.

5. What does a Cluster index do?

Cluster index sorts table data rows based on their key values.

6. What are the benefits of specifying data types in columns while creating a table?

Some of these benefits include consistency, compactness, validation, and performance.

7. What do you need to do to update statistics in PostgreSQL?

To update statistics in PostgreSQL, we need to use a special function called a vacuum.

Viva Questions

1. What is the disadvantage of the DROP TABLE command in deleting complete data from an existing table?

Though the DROP TABLE command has the ability to delete complete data from an existing table, the disadvantage with it is - it removes complete table structure from the database. Due to this, we need to re-create a table to store data.

2. How can you delete complete data from an existing table?

We can delete complete data from an existing table using the PostgreSQL TRUNCATE TABLE command.

3. What are the different properties of a transaction in PostgreSQL? Which acronym is used to refer to them?

The properties of a transaction in PostgreSQL include Atomicity, Consistency, Isolation, and Durability. These are referred to by the acronym, namely ACID.

4. What purpose does the CTIDs field serve?

The CTIDs field identifies the specific physical rows in a table according to their block and offsets positions in that table.

5. Which are the commands used to control transactions in PostgreSQL?

The commands used to control transactions in PostgreSQL are BEGIN TRANSACTION, COMMIT, and ROLLBACK.

6. What are the main differences between SQL and PostgreSQL?

PostgreSQL is an advanced version of SQL.

- Unlike SQL, views in PostgreSQL are not updatable. Another difference is whereas SQL provides computed columns; the same cannot be expected from PostgreSQL.
- Unlike SQL, in PostgreSQL, you don't need to create a DLL to see the code what it is doing.
- PostgreSQL supports dynamic actions whereas SQL doesn't support them.

Viva Questions

1. How is security ensured in PostgreSQL?

PostgreSQL uses SSL connections to encrypt client or server communications so that security will be ensured.

2. What is the function of the Atomicity property in PostgreSQL?

Atomicity property ensures the successful completion of all the operations in a work unit.

3. What are the advantages of PostgreSQL?

Some of the advantages of PostgreSQL are open-source DBMS, community support, ACID compliance, diverse indexing techniques, full-text search, a variety of replication methods, and diversified extension functions, etc.

4. What does Write-Ahead Logging do?

The Write-Ahead Logging enhances database reliability by logging changes before any changes or updates are made to the database

5. What are some of the important data administration tools supported by PostgreSQL?

Some of the important data administration tools supported by PostgreSQL are Psql, Pgadmin, and Phppgadmin.

6. How can you store the binary data in PostgreSQL?

We can store the binary data in PostgreSQL either by using bytes or by using the large object feature.

7. What is a non-clustered index?

In a non-clustered index, the index rows order doesn't match the order in actual data.

Viva Questions

1. What is the purpose of table space in PostgreSQL?

It is a location in the disk. In this, PostgreSQL stores the data files, which contain indices and tables, etc.

2. Are there any disadvantages with PostgreSQL?

Yes. There are a few disadvantages. Some of these include the following:

- It is slower than MySQL on the performance front.
- It doesn't have the support of a good number of open source applications when compared to MySQL.
- Since it focuses more on compatibility, changes made to improve the speed need more work.

3. What does a token representation in a SQL Statement?

In a SQL Statement, a token represents an identifier, keyword, quoted identifier, special character symbol, or a constant.

4. What is the name of the process of splitting a large table into smaller pieces in PostgreSQL?

The name of the process of splitting a large table into smaller pieces in PostgreSQL is known as table partitioning.

5. What do you understand by a base directory in PostgreSQL?

The base directory in PostgreSQL is data_dir/base. It is a folder in PostgreSQL which contains all the sub-directories used by a database in clusters and stores all the data you have inserted in your databases.

Viva Questions

1. What are the different data types used in PostgreSQL?

Following is a list of the different data types supported by and used in PostgreSQL?

- Numeric types
- Character types
- Temporal types
- Boolean
- UUID
- Geometric primitives
- Arbitrary precision numeric
- XML
- Arrays etc.

2. What do you understand by string constants in PostgreSQL?

In the PostgreSQL database, a string constant is a sequence of some character bounded by single quotes ('). For example: 'This is an example of string Constant'.

3. What is the maximum size for a table in PostgreSQL?

PostgreSQL provides unlimited user database size, but it doesn't provide an unlimited size for tables. In PostgreSQL, the maximum size for a table is set to 32 TB.

4. What do you understand by a partitioned table in PostgreSQL?

In PostgreSQL, a partitioned table is a logical structure used to split a large table into smaller pieces. These small pieces of the tables are called partitions.

5. What does a schema contain in PostgreSQL?

In PostgreSQL, a schema contains tables and data types, views, indexes, operators, sequences, and functions.

Viva Questions

1. What is Multi-Version Concurrency Control in PostgreSQL? Why is it used?

Multi-Version Concurrency Control or MVCC is an advanced technique in PostgreSQL that improves database performance in a multi-user environment. It is mainly used to avoid unnecessary locking of the database by removing the time lag for the user to log into his database. This time lag occurs when someone else is accessing the content. In Multi-Version Concurrency Control or MVCC, all the transactions are kept as records. That's why PostgreSQL maintains data consistency, unlike most other database systems which use locks for concurrency control.

2. What is the key difference between multi-version and lock models?

The key difference between Multi-Version Concurrency Control and lock models is that in MVCC, the locks acquired for querying or reading the data doesn't conflict with locks acquired for writing data. In this case, reading never blocks writing, and writing never blocks reading. So, Multi-Version Concurrency Control has the upper hand compared to other lock models.

3. What is pgAdmin in the PostgreSQL server? Why is it used?

In the PostgreSQL server, PgAdmin is a free, open-source PostgreSQL database administration GUI or tool used in Microsoft Windows, Mac OS X, and Linux systems. PgAdmin is used to retrieve, develop, conduct quality testing, and maintain databases or other ongoing maintenances.

4. What are the Indices of PostgreSQL?

Indices of PostgreSQL are inbuilt functions or methods such as GIST Indices, hash table, and B-tree (Binary tree). The user uses these to scan the index in a backward manner. PostgreSQL also facilitates their users to define their indices of PostgreSQL.

5. What are the different types of operators used in PostgreSQL?

- Arithmetic operators
- Comparison operators
- Logical operators
- Bitwise operators

Viva Questions

1. How can you set up pgAdmin in PostgreSQL?

To set up pgAdmin in PostgreSQL, we should follow the steps given below:

- First, start and launch pgAdmin 4.
- Then, go to the "Dashboard" tab, click on the "Quick Link" section and then click on "Add new Server."
- After clicking on the "Add new Server", you have to select the "Connection" tab in the "Create-Server" window.
- Now, configure the connection by entering your server's IP address in the "Hostname/Address" field.
- At last, you have to specify the "Port" as "5432," which is the by default port for the PostgreSQL server.

2. What are the tokens in PostgreSQL?

In PostgreSQL, tokens are the building blocks of any source code. Tokens contain several types of special character symbols like constants, quoted identifiers, other identifiers, and keywords. The keywords tokens contain pre-defined SQL commands and meanings. On the other hand, identifiers specify variable names like columns, tables, etc.

3. What are some new characteristics introduced in Postgre 9.1?

The new PostgreSQL 9.1 version is working on important features such as JSON support, synchronous replication, nearest-neighbor geographic searches, SQL/MED external data connections, security labels, and index-only access.

Following is a list of some newly added characteristics in PostgreSQL 9.1:

- Added support for foreign tables.
- Added support for per-column collation.
- Added some extensions to simplify packaging of additions to PostgreSQL.
- Added a true serializable isolation level.
- Added nearest-neighbor (order-by-operator) searching to GiST indexes.
- Added a SECURITY LABEL command and support for SELinux permissions control.

- Along with the above features, PostgreSQL 9.1 has allowed other features such as allowing synchronous replication, allowing data-modification commands (INSERT/UPDATE/DELETE) in WITH clauses, providing support for unlogged tables using the UNLOGGED option in CREATE TABLE, and updating the PL/Python server-side language.

4. What do we call database callback functions? What is its purpose?

The database callback functions are known as PostgreSQL Triggers. The PostgreSQL Triggers are performed or invoked automatically whenever a specified database event occurs.

5. What do you know about the history of PostgreSQL?

The POSTGRES project was started and led by Professor Michael Stonebraker in 1986 and was sponsored by the Defense Advanced Research Projects Agency (DARPA), the Army Research Office (ARO), the National Science Foundation (NSF), and ESL, Inc. It has completed more than 30 years of active development on the core platform. It has been ACID-compliant since 2001 and runs on all the major operating systems. It also has an add-on like PostGIS database extender.

Michel Stonebraker is known as the father of PostgreSQL. He started and led the development of this database in 1986 as a follow-up project to its predecessor, Ingres, now owned by Computer Associates. PostgreSQL was originally called Postgres. It is pronounced PostgreSQL because of its ubiquitous support for the SQL standards among most relational databases. PostgreSQL is used as a by default database in MAC OS.

Postgres was started by Michael Stonebraker in 1986 as a follow-up project to its predecessor, Ingres, which Computer Associates now own. The name Postgres was derived from its predecessor Ingres. Here, Postgres means "after Ingres". The first project of Postgres was developed between 1986 - 1994. It has provided many sophisticated features such as Multi-Version Concurrency Control (MVCC), point in time recovery, tablespaces, asynchronous replication, nested transactions (savepoints), online/hot backups, a sophisticated query planner/optimizer, and write-ahead logging for fault tolerance.

Viva Questions

1. How can you start, stop, and restart the PostgreSQL server on Windows?

To start, stop, and restart the PostgreSQL server on Windows, first, we have to find the PostgreSQL database directory. It would look something like this: C:\Program Files\PostgreSQL\10.4\data. Now, open the Command Prompt and execute the following commands:

To start the PostgreSQL server of Windows:

```
1. pg_ctl -D "C:\Program Files\PostgreSQL\9.6\data" start
```

To stop the PostgreSQL server of Windows:

```
1. pg_ctl -D "C:\Program Files\PostgreSQL\9.6\data" stop
```

To restart the PostgreSQL server of Windows:

```
1. pg_ctl -D "C:\Program Files\PostgreSQL\9.6\data" restart
```

Another way to start, stop and restart the PostgreSQL server on Windows.

There is also another way to start, stop, and restart the PostgreSQL server on Windows. Follow the steps given below:

- First, open the Run Window by pressing the Windows key + R simultaneously.
- Then, type services.msc to find out the PostgreSQL services.
- Search Postgres service based on the version installed.
- Click the stop, start or restart option to do the same.

2. How do you update the stats or statistics in PostgreSQL?

To update stats or statistics in PostgreSQL, we have to call a special function called explicit 'vacuum'. This function creates a Vacuum where the option of Analyze is used to update statistics in PostgreSQL.

Syntax:

```
1. VACUUM ANALYZE;
```

3. What is the difference between clustered index and non clustered index in PostgreSQL?

The main difference between clustered index and non clustered index in PostgreSQL is that the clustered index is an index type used to sort table data rows according to their key values. In RDBMS, a user can create a clustered index based on that column using the primary key. On the other hand, a non-clustered index is an index where the order of the rows does not match the physical order of the actual data. The non-clustered index is ordered by the columns that make up the index.

4. What are the different types of database administration tools used in PostgreSQL?

Following are the different types of database administration tools used in PostgreSQL:

- Phppgadmin
- Psql
- Pgadmin

5. What is the difference between PostgreSQL and MongoDB databases?

PostgreSQL	MongoDB
PostgreSQL is a classical, relational database server that supports most SQL standards. MongoDB is a NoSQL database.	MongoDB belongs to the family of NoSQL databases which is used for storing unstructured documents in JSON format.
PostgreSQL is a traditional relational database management system (RDBMS) or SQL-based databases like Oracle and MySQL. It is open-source and free to use. MongoDB is a no-schema, NoSQL, JSON format database. It also provides a free version, but its enterprise-paid versions are more popular.	MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License which is deemed non-free by several distributions.
PostgreSQL database is written in C language.	MongoDB is written in C++.
PostgreSQL is a Relational Database Management System.	MongoDB is a Non-Relational Database Management System.
PostgreSQL is an Object-Oriented Database.	MongoDB is Document Oriented Database.
PostgreSQL is available in multiple languages.	MongoDB is only available in the English language.
PostgreSQL is 4 to 10 times faster than MongoDB on some parameters.	MongoDB is slower than PostgreSQL. It is best suited for big data.

Viva Questions

1. What is Enterprise Resource Planning (ERP), and what kind of a database is used in an ERP application?

Enterprise Resource Planning (ERP) is an information system used in manufacturing companies and includes sales, inventory, production planning, purchasing and other business functions. An ERP system typically uses a multiuser database.

2. What is SQL, and why is it important?

SQL stands for Structured Query Language, and is the most important data processing language in use today. It is not a complete programming language like Java or C#, but a data sublanguage used for creating and processing database data and metadata. All DBMS products today use SQL.

3. Who is E.F. Codd, and why is he significant in the development of modern database systems?

While working at IBM, E.F. Codd created the relational database model. A paper he published in 1970 presented his ideas to the world at large. His work is the foundation for most of the DBMSs currently in use, and thus forms the basis for database systems as we know and use them today.

4. What is a DBMS?

DBMS stands for Database Management System. A DBMS receives requests from applications and translates those requests into actions on a specific database. A DBMS processes SQL statements or uses other functionality to create, process and administer databases.

5. Why is a database considered to be "self-describing"?

In addition to the users' data, a database contains a description of its own structure. This descriptive data is called "metadata."