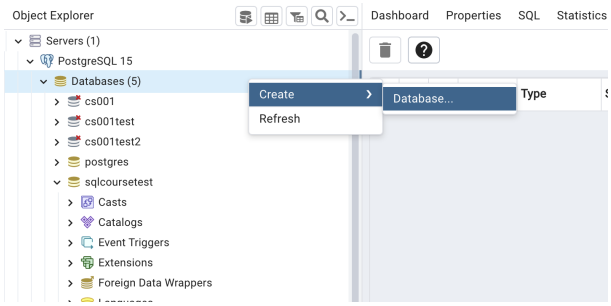
Sample Database: School Management System

We will work on a sample database and understand several basic SQL commands like SELECT, WHERE, ORDER BY and much more.

Let's start by Creating a Database:

You can go ahead and create a database in pgadmin by left clicking on databases and clicking on **CREATE DATABASE**, you can choose to name a database as **SchoolDB** and add some comments over there.



Think of it like building a new school, where you'll keep all the information about students and teachers.

Creating Tables:

We will be creating two tables(Student and Teachers Table) which will be used as a demonstration of commands out there.

```
CREATE TABLE Students (
  StudentID INT PRIMARY KEY,
  Name VARCHAR(50),
  Age INT,
  Class VARCHAR(10)
);
```

This command creates a table called 'Students' with columns for student ID, name, age, and class.

`CREATE TABLE`: This command is used to create a new table in the database.

`Students`: The name of the table being created.

`(...)` contains the columns and data types for the table. Each line in the parentheses defines a column.

Column Definitions in the Students Table:

- `StudentID INT PRIMARY KEY`:
  - `StudentID`: Name of the column.
  - `INT`: Data type integer, used for whole numbers.
  - `PRIMARY KEY`: This signifies that each value in this column is unique and identifies each row in the table.
- `Name VARCHAR(50)`:
  - `Name`: Column name.
  - `VARCHAR(50)`: Data type for variable-length strings (text). `50` specifies the maximum length.
- `Age INT`:
  - `Age`: Column name.
  - `INT`: Data type integer, used for whole numbers.
- `Class VARCHAR(10)`:
  - `Class`: Column name.
  - `VARCHAR(10)`: Data type for variable-length strings, with a maximum length of 10 characters.

Now, we will be creating Teachers Table:

```
CREATE TABLE Teachers (
  TeacherID INT PRIMARY KEY,
  Name VARCHAR(50),
  Subject VARCHAR(30)
);
```

- This creates a 'Teachers' table with teacher ID, name, and the subject they teach.
- Here, `TeacherID` serves as a unique identifier for each teacher.

Now, we have created tables in the databases but we need to insert them right? So for that reason we will insert some sample data into both the tables.

Into students:

```
INSERT INTO Students (StudentID, Name, Age, Class) VALUES

(1, 'Aarav', 14, '8A'),

(2, 'Aditi', 13, '7B'),

(3, 'Rohan', 15, '9A'),

(4, 'Priya', 14, '8B');
```

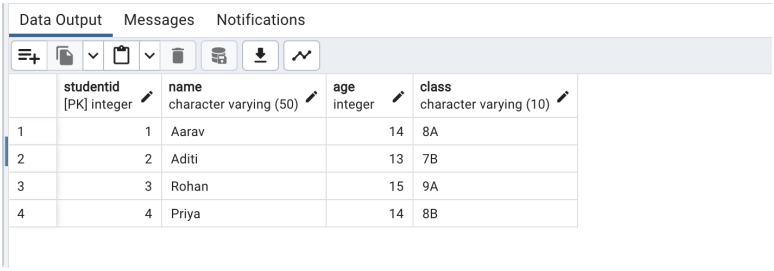
Into Teachers:

```
INSERT INTO Teachers (TeacherID, Name, Subject) VALUES
(1, 'Mr. Sharma', 'Mathematics'),
(2, 'Ms. Gupta', 'Science');
```

Demonstrating Key SQL Commands

Retrieving Data with SELECT:

```
SELECT * FROM Students;
```

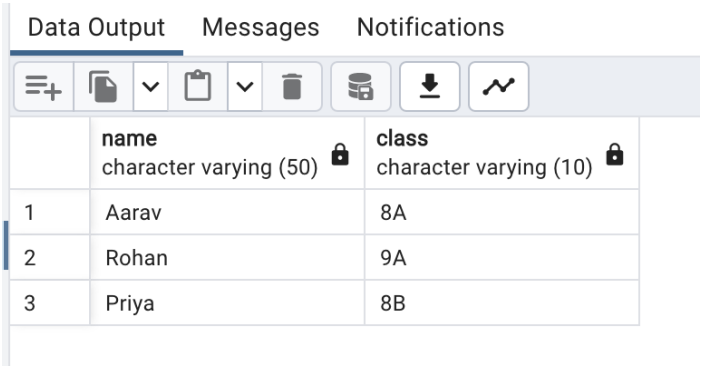


Data Output | Messages | Notifications

	studentid [PK] integer	name character varying (50)	age integer	class character varying (10)
1	1	Aarav	14	8A
2	2	Aditi	13	7B
3	3	Rohan	15	9A
4	4	Priya	14	8B

Filtering Data with WHERE:

```
SELECT Name, Class FROM Students WHERE Age > 13;
```

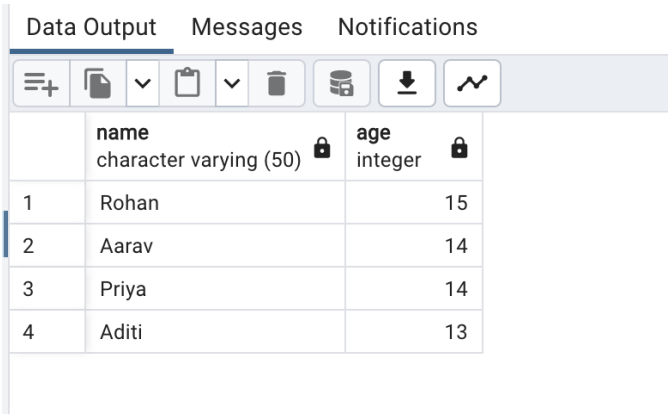


Data Output | Messages | Notifications

	name character varying (50)	class character varying (10)
1	Aarav	8A
2	Rohan	9A
3	Priya	8B

Sorting Data with ORDER BY:

```
SELECT Name, Age FROM Students ORDER BY Age DESC;
```



	name character varying (50)	age integer
1	Rohan	15
2	Aarav	14
3	Priya	14
4	Aditi	13

Grouping Data with GROUP BY:

```
SELECT Class, COUNT(*) as StudentCount FROM Students GROUP BY Class;
```

Explanation of each Part of the query:

SELECT Class:

- `SELECT`: This keyword is used to specify which columns you want to retrieve from the database.
- `Class`: This is the name of the column in the `Students` table. The query will return the value from the `Class` column.

COUNT(*):

- `COUNT`: This is an aggregate function in SQL. Aggregate functions perform a calculation on a set of values and return a single value.
- `(*)`: This symbol represents all rows. When used with `COUNT`, it counts all rows in the specified column or in the table if no column is specified.
- In this context, `COUNT(*)` counts all rows (or students) in each group defined by the `GROUP BY` clause.

as StudentCount:

- `as`: This is used to create an alias in SQL. An alias is a temporary name assigned to a column or a table in your SQL query.
- `StudentCount`: This is the alias name given to the result of the `COUNT(*)` function. Instead of showing a column name like `COUNT(*)` in the results, it will show as `StudentCount`.
- This makes the output more readable and understandable. Instead of seeing `COUNT(*)`, which might be confusing, you see `StudentCount`, clearly indicating the meaning of that number.

FROM Students:

- `FROM`: Specifies from which table the SQL engine should retrieve the data.
- `Students`: The name of the table where the query is being applied.

GROUP BY Class:

- `GROUP BY`: This clause groups rows that have the same value in the specified column(s), allowing aggregate functions like `COUNT`, `SUM`, `AVG`, etc., to be applied to each group.
- `Class`: Indicates that the data should be grouped by the `Class` column. This means the `COUNT(*)` operation will be performed separately for each class.

Using LIMIT:

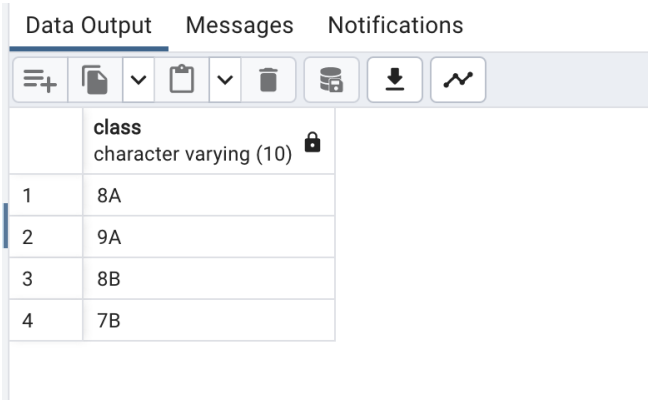
```
SELECT * FROM Students LIMIT 2;
```

Retrieves only the first 2 students from the table.

Eliminating Duplicates with DISTINCT:

```
SELECT DISTINCT Class FROM Students;
```

Lists all unique classes in the Students table.



Using Sub-Queries with WITH:

```
WITH SeniorStudents AS (SELECT * FROM Students WHERE Age > 13)

SELECT Name FROM SeniorStudents;
```

Creates a temporary set of senior students and then selects their names. Here it's Isolating and working with a specific group of students.

- `WITH`: Creates a temporary named result set.
- `SeniorStudents AS`: The name for the temporary result set.
- `(SELECT * FROM Students WHERE Age > 13)`: A subquery that selects all columns from `Students` where `Age` is greater than `13`.

- `SELECT Name FROM SeniorStudents`: Retrieves the `Name` column from the `SeniorStudents` result set.

That’s pretty much it for today!