



Find value in open data and geospatial



✉ hi@bnhr.xyz

🌐 bnhr.xyz

🐙 [@benhur07b](https://github.com/benhur07b)

📘 [@bnhr.xyz](https://www.facebook.com/bnhr.xyz)

📧 fosstodon.org/@bnhrdotxyz

📺 [@bnhrdotxyz](https://www.youtube.com/@bnhrdotxyz)

Introduction to Databases and SQL

Table of Contents

Table of Contents.....	2
Credits and License.....	3
What is a database?.....	4
What is SQL?.....	4
Schemas, Tables, Rows, and Columns.....	5
Roles and permissions.....	5
About data storage.....	7
Relational Database Management Systems.....	7
NoSQL dtabases.....	7
Data warehouses and data lakes.....	7
About Postgres.....	9
Roles.....	9
Privileges.....	9
Host-based access control using pg_hba.conf.....	10
Exercise 01: Basic SQL with Postgres.....	11
View all the data in the table.....	15
Select and sort.....	17
Add and update a field.....	19
Aggregate or compute based on a column.....	20
Wildcards and other select options.....	22



This work and its contents by BNHR is licensed under a
[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

Credits and License

Introduction to Databases and SQL

BNHR

version 2026



This work and its contents by [Ben Hur Pintor](#) is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).

You are free to:

Share — copy and redistribute the material in any medium or format

Adapt — remix, transform, and build upon the material

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial — You may not use the material for commercial purposes.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

Other works (software, source code, etc.) referenced in this work are under their own respective licenses.



This work and its contents by [BNHR](#) is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [yt @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

What is a database?

A database is a structured digital repository that stores, manages, and retrieves various kinds of data. It enables two things that make it invaluable for any data management system: **virtually limitless, scalable storage** and **near-instantaneous data retrieval and processing** through queries.

You can picture a database as being similar to large, well-organized library where a vast collection of books are stored and managed. The library's role is not just to keep these books well-organized but also to provide visitors the book or books they need as quickly as possible.

Think about it this way:

Imagine your database as a library where you store books. Each book you add to this library is equivalent to a piece of data.

To ensure order and accessibility, you categorize your books based on genre and place them in their own sections (schema) of the library.

Each section contains several shelves of books. These shelves are like tables in your database.

Within each shelf, individual books represent rows in your table while the content and information about the book—such as its author, publication date, number of pages—are equivalent to the columns.

What is SQL?

SQL (Structured Query Language)—pronounced either as SEQUEL or S-Q-L depending on your preference—is the language used to interact with a database. It is a domain-specific language used for managing and querying databases providing instructions on data retrieval, manipulation, and management.

With SQL you are able to create, drop, and edit your databases; find, sort, filter, and process your data efficiently; and even assign permissions to different roles.

Think about it this way:

In your library, SQL pertains to the set of instructions that you'd give your library staff to effectively manage your library.

For example, you may give instructions to a librarian in order to find a specific book or set of books. In the same way, SQL lets you tell the database exactly what data you need.

You can also tell library staff to create or destroy sections, shelves, individual books, or even specific information about books. Similarly, SQL allows you to manage the structure of your database.



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
 @bnhr.xyz | @bnhrdotxyz | mastodon.social/@benhur07b

Schemas, Tables, Rows, and Columns

In a database, a schema is a logical container for organizing and structuring data, defining how data is stored and related.

Tables—which is a structured collection of data stored in rows and columns—are housed within schemas.

Lastly, you have rows and columns which are the fundamental components of a database table. Rows represent individual data records (or observations) while columns define the attributes or properties associated with the data (or observation).

Think about it this way:

In your library, the schema is your catalog system that specifies where different types of books can be found. For example, books can be sorted into different sections like fiction, non-fiction, or reference.

The bookshelves in your library correspond to the tables in your database; the books to rows; and the books' content to columns.

Roles and permissions

Roles and permissions in a database determine what kind of interactions that a user or group can have with your database and specify the kind of operations that a user or group is allowed to perform.

Think about it this way:

Roles and permissions can be compared to the different people who can be found in a library such as readers, librarians, and administrators—each of whom have different privileges and access levels.

For example, just as a reader can only read and borrow books but can't add new books to the library; users in a database can have varying roles and permissions—allowing some to read data, some to edit, and others to administer the database.



This work and its contents by BNHR is licensed under a
Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
f @bnhr.xyz | y @bnhrdotxyz | m mastodon.social/@benhur07b



Figure 1: Example database structure showing schemas, tables, etc.



This work and its contents by [BNHR](#) is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).

 [@bnhr.xyz](#) |  [@bnhrdotxyz](#) |  [mastodon.social/@bnhr07b](#)

About data storage

Relational Database Management Systems

What was defined above is the most common type of database known as RDBMS or Relational Database Management Systems that store data in structured tables and where data relationships are established through defined keys and constraints.

[PostgreSQL](#), often referred to as Postgres, is a best-in-class, open-source RDBMS that has gained recognition for its robust features and extensibility. It supports various data types including geospatial data through the PostGIS extension. Other RDBMS include open source MySQL (and the [MariaDB](#) fork) and proprietary options such as Oracle DB and MS SQL Server.

NoSQL dtabases

Aside from RDBMS, NoSQL—which stands for "not only SQL"—databases have gained popularity in recent years, especially in the context of big data and real-time applications. These databases can efficiently handle unstructured and semi-structured data which make them particularly well-suited for use cases that don't fit the traditional tabular structure of RDBMS.

NoSQL databases come in a variety of types based on their data model: document, key-value, wide-column, and graph being the most common. They offer features such as horizontal scalability, excellent handling of semi-structured datasets such as XML and JSON, optimization for real-time processing, and schema-less designs that allow fields to be modified without altering the entire database structure.

Examples of NoSQL databases include: [MongoDB](#) (and its open source alternative [FerretDB](#)), [Apache Cassandra](#) (also open source), and [Redis](#) (also open source).

Data warehouses and data lakes

With the increase in the size and complexity of data, data warehouses and data lakes became more popular. While perfect for structured data, the rigid schema-based approach of relational databases become a limitation when storing unstructured data or large amounts of data from multiple sources.

Data warehouses are optimized for storing large amounts of structured information from various sources. It is essentially a database on steroids with the goal of combining different data sources so that they can easily be used for analysis, processing, and visualization. Examples of technologies that can be classified as data warehouses include: Amazon Redshift, Microsoft Azure Synapse, and Google BigQuery.



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

Data lakes meanwhile can be used for structured, semi-structured, or unstructured data which makes them perfect for storing raw data from multiple sources. They are a cost-effective way of storing large amounts of data especially when there's no immediate need to process or transform the data at the time of storage. It is only when the data needs to be retrieved that some structure is applied. Examples of data lakes include AWS S3, Azure Data Lake, and Google Cloud Storage.

Both data warehouses and data lakes are great for storing historical data.



This work and its contents by BNHR is licensed under a
Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
f @bnhr.xyz | y @bnhrdotxyz | m mastodon.social/@benhur07b

About Postgres

Other resources

- Postgres documentation: <https://www.postgresql.org/docs/>

Roles

PostgreSQL uses the concept of **roles** for managing database permissions. A role can be a database user or a group of database users depending on how the role is set up. Roles can also own database objects (e.g. tables and functions) and are able to assign **privileges** on those objects to other roles in order to control who can access the objects. It is also possible to grant membership in a role to another role which allows the member role to use privileges assigned to another role.

On pgAdmin4, roles can be created by **right-clicking on a server > Create > Login/Group Role**.

Privileges

Privileges in PostgreSQL determine what actions a user or role is allowed to perform on database objects. Key privileges include:

- **SELECT**: Allows reading data from a table.
- **INSERT**: Permits inserting new rows into a table.
- **UPDATE**: Allows modifying existing data in a table.
- **DELETE**: Grants the ability to remove rows from a table.
- **CREATE**: Enables creating new objects, like databases, tables, and views.
- **GRANT**: Allows a user to delegate privileges to others.
- **CONNECT**: Governs the ability to connect to a database.

Privileges can be assigned at various levels such as tables, schemas, and databases.



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
 @bnhr.xyz | @bnhrdotxyz | mastodon.social/@benhur07b

Host-based access control using pg_hba.conf

pg_hba.conf is a key component of controlling access and security to Postgres' network. It determines how Postgres accepts or rejects incoming connections based on client IP addresses, authentication methods, and other criteria.

- **Host-Based Authentication:** pg_hba.conf allows you to define rules for which hosts are allowed to connect to the database and the authentication methods they should use.
- **Authentication Methods:** Postgres supports various authentication methods including password-based, certificate-based, and trust-based. pg_hba.conf is where you specify which method to use for each client and under what circumstances.
- **Network Security:** Properly configuring pg_hba.conf is crucial for protecting your database against unauthorized access and network threats. It serves as the first line of defense against potential attacks.
- **Connection Types:** pg_hba.conf enables the definition of different access rules for local connections, host connections, replication connections, and more, ensuring that the database's access control aligns with its connection requirements.



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
🌐 [@bnhr.xyz](#) | 📺 [@bnhrdotxyz](#) | 🐙 [mastodon.social/@benhur07b](#)

Exercise 01: Basic SQL with Postgres

For this exercise, we restore a database (philippines) and apply some basic SQL commands to select, filter, sort, and compute values.

1. Create a new database in pgAdmin by **right clicking on Databases > Create > Database**.
2. This will open the Create – Database window where you can define the parameters for creating your database.

Figure 2: Create database dialog in pgAdmin4

PRO-TIP:

If you are familiar with the command line or terminal, you can run a **CREATE DATABASE** query or use the [createdb](#) command in [psql](#).

3. Name the database **philippines** and assign a user as the owner. Click Save.

PRO-TIP:

It is generally not considered best practice to use the built-in Postgres superuser (postgres) for your application databases. It is always recommended to create separate and specific roles.



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

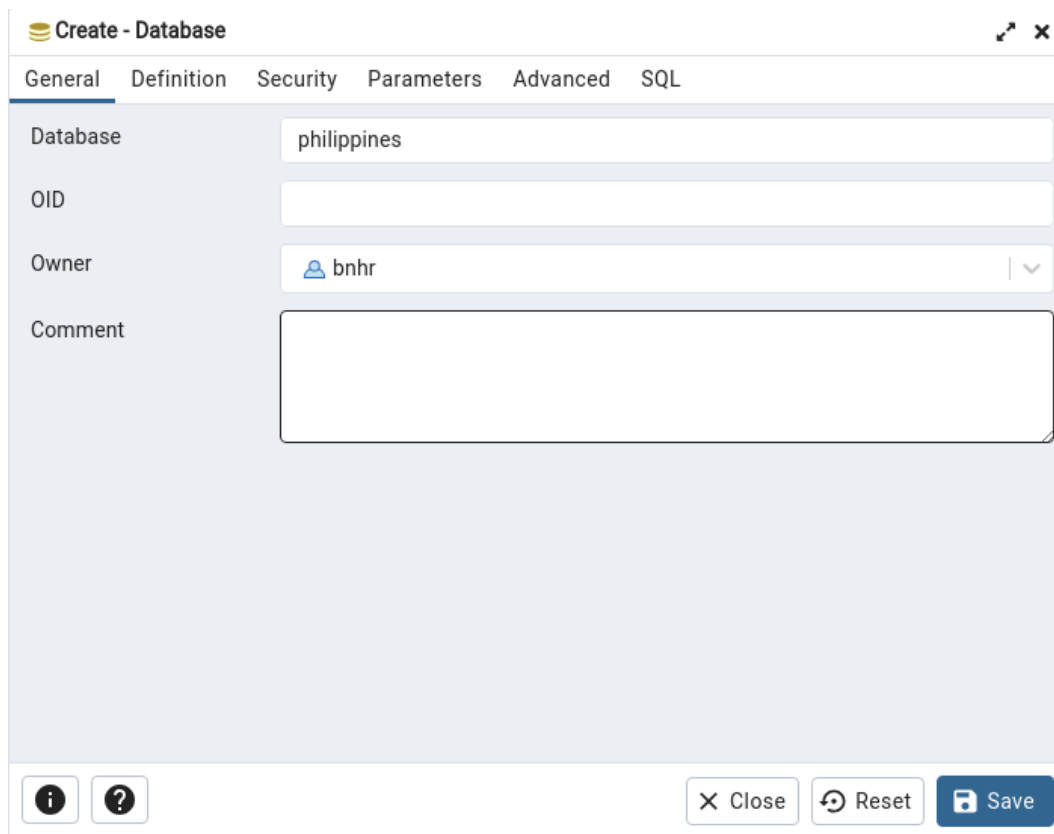


Figure 3: Create database dialog in pgAdmin4 (2)

4. You should now have a **philippines** database available in pgAdmin4.

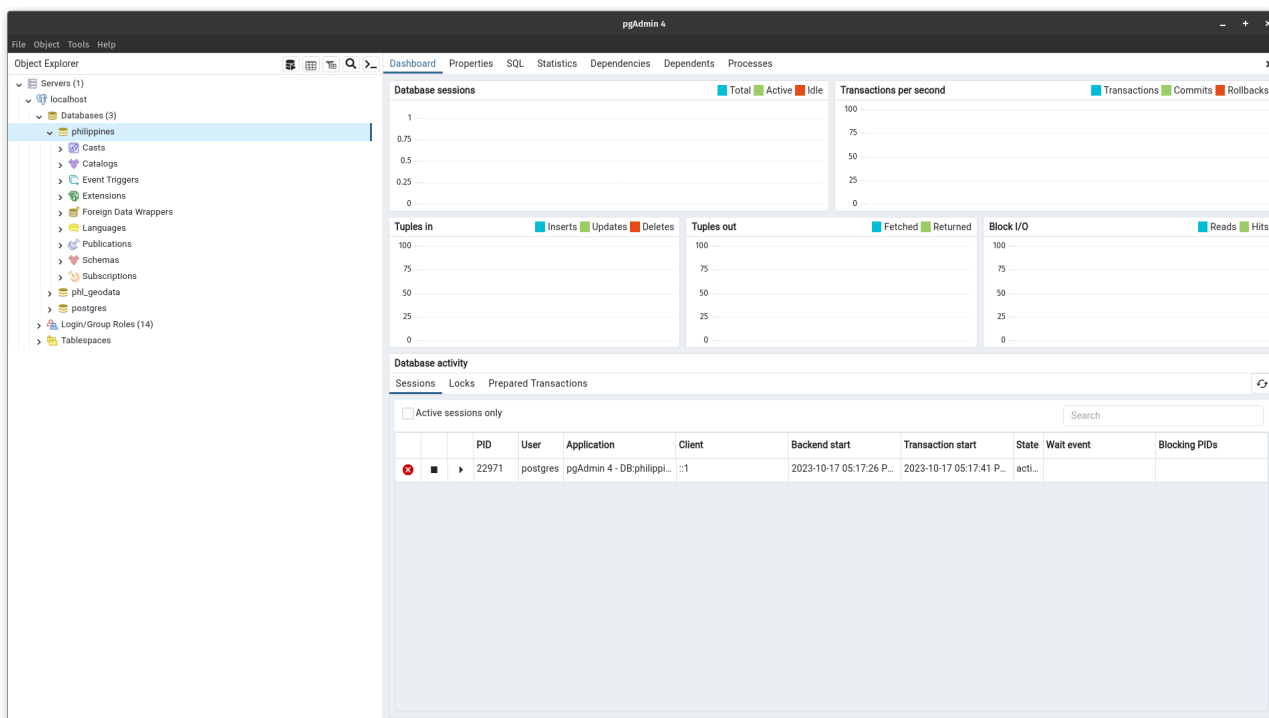


Figure 4: philippines database created



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[@bnhr.xyz](#) | [@bnhrdotxyz](#) | [mastodon.social/@benhur07b](#)

TRY FOR YOURSELF:

Play around and discover the parts of the pgAdmin4 UI and how it relates with your database/s.

5. To **restore** the philippines database, **right-click on the philippines database > Restore...** This will open the Restore database dialog.
6. Add the following parameters and look for the backup file that you want to restore. (**philippines.tar**).

The screenshot shows the 'Restore (Database: philippines)' dialog box in pgAdmin4. The 'General' tab is selected. The 'Format' dropdown is set to 'Custom or tar'. The 'Filename' field contains the path '/media/bnhr/KINGSTON/geospatial/others/sql_queries/philippines.tar'. The 'Number of jobs' field is empty. The 'Role name' dropdown is set to 'bnhr'. At the bottom, there are buttons for 'Close', 'Reset', and 'Restore'.

Figure 5: Restore database dialog in pgAdmin4

Note:

If you can't find the file in the file selection dialog, replace the file type you are searching for in your file search to **All files**.

7. Click Restore.
8. After, pgAdmin should prompt you once the process is completed.



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

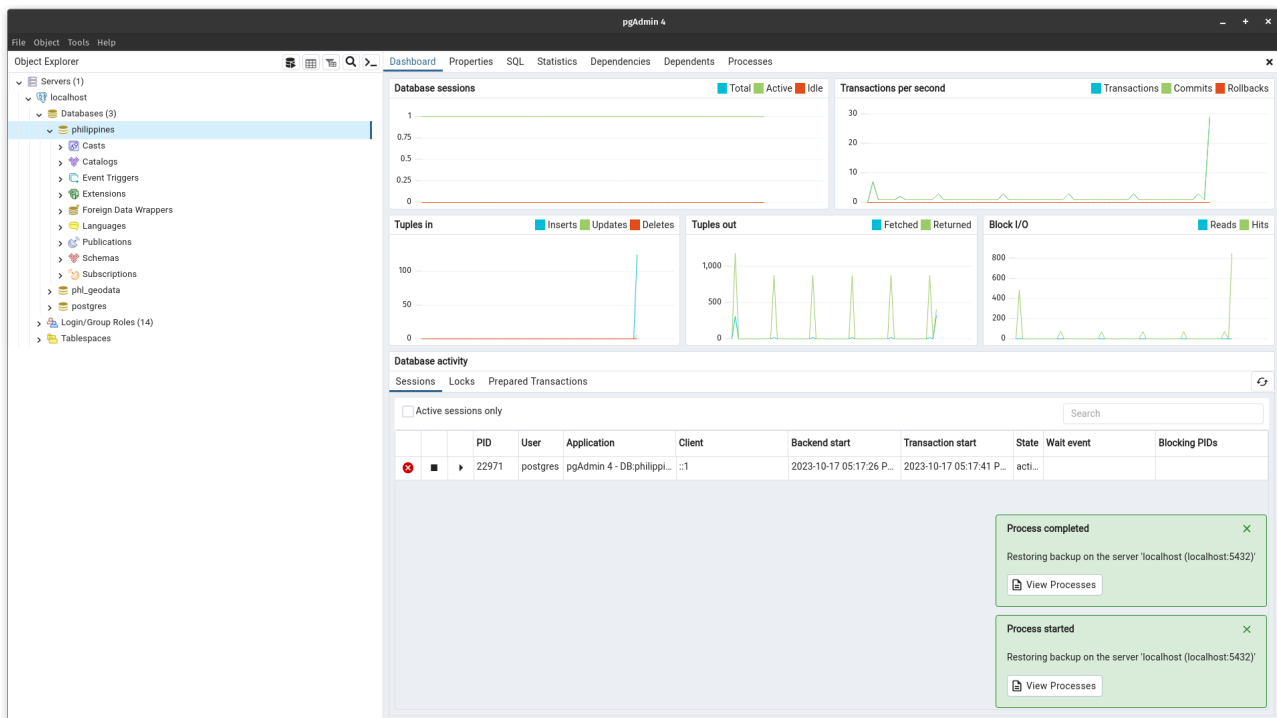
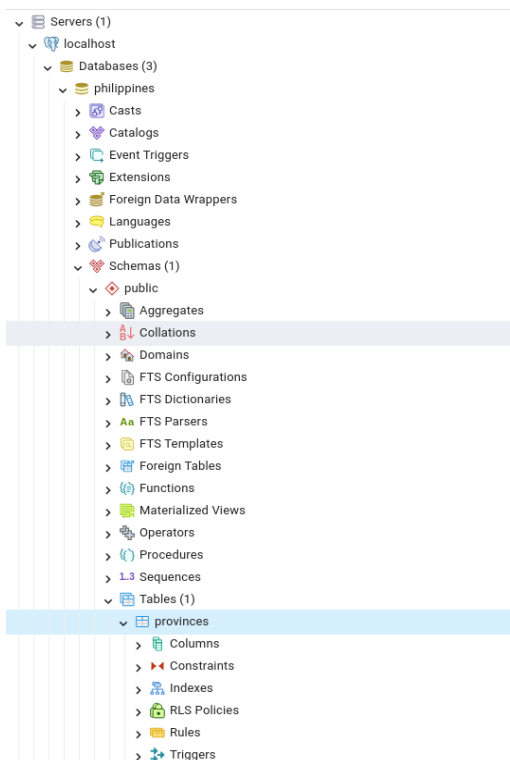


Figure 6: Note the prompt on the bottom right of the window

- Under philippines, you should now see a provinces table under **Schemas > public > Tables > provinces**.



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).
[@bnhr.xyz](https://www.facebook.com/bnhr.xyz) | [@bnhrdotxyz](https://www.youtube.com/channel/UCbnhrdotxyz) | mastodon.social/@benhur07b

View all the data in the table

- Right click on the **provinces** table and click **Query tool**. This will open an interactive window where you can run SQL queries and see their outputs.

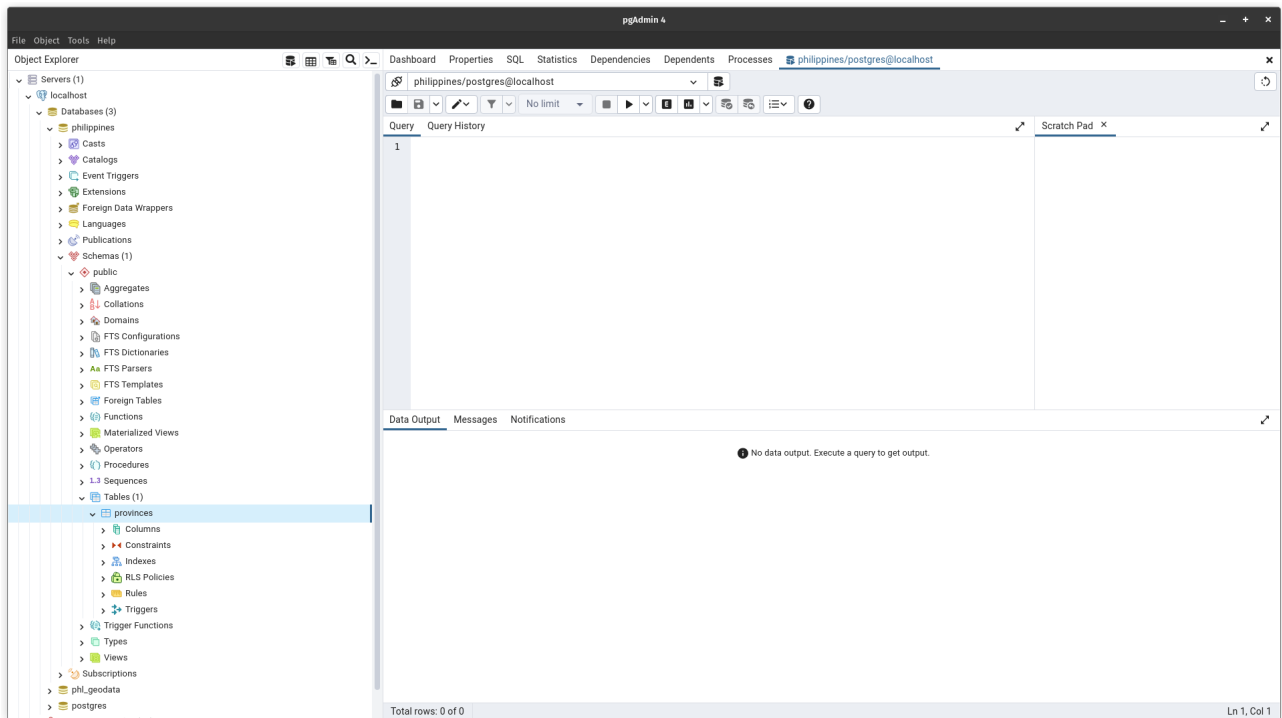


Figure 7: Query tool in pgAdmin4

- Let's look at the entire database by running the following query

```
SELECT * FROM provinces;
```

Note:

The * symbol is a wildcard which usually means ALL when using SELECT queries.

- Click Run/Refresh (F5) or the play button.
- You should see the results below.



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[@bnhr.xyz](#) | [@bnhrdotxyz](#) | [mastodon.social/@benhur07b](#)

Query: `SELECT * FROM provinces;`

	province character varying	psgc_province [PK] character varying	pop2020 integer	pop_perc numeric	notes character varying
1	Mountain Province	PH144400000	158200	0.15	[null]
2	Bohol	PH071200000	1394329	1.28	[null]
3	Maguindanao	PH153800000	1667258	1.53	Includes Cotabato City
4	Zamboanga del Sur	PH097300000	2027902	1.86	Includes Zamboanga City
5	Iloilo	PH063000000	2509525	2.3	Includes Iloilo City
6	Camarines Sur	PH051700000	2068244	1.9	Includes Naga
7	Camarines Norte	PH051600000	629699	0.58	[null]
8	Pampanga	PH035400000	2900637	2.66	Includes Angeles City and San Fernando
9	Zambales	PH037100000	909932	0.83	Includes Olongapo
10	Dinagat Islands	PH168500000	128117	0.12	[null]
11	Nueva Vizcaya	PH025000000	497432	0.46	[null]
12	Pangasinan	PH015500000	3163190	2.9	Includes Dagupan
13	Second District, NCR	PH137400000	4771371	4.38	[null]

Total rows: 85 of 85 Query complete 00:00:00.058 Ln 1. Col 25

Figure 8: Output of the SELECT query showing all 85 rows in the database

TRY FOR YOURSELF:

Examine your database.

TRY FOR YOURSELF:

What if I don't have a backup file but I have a CSV of the data and I want to recreate the philippines database, how should I do it?



This work and its contents by BNHR is licensed under a
[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

Select and sort

PRO-TIP:

You can find the SQL commands used in this exercise under the **sql_queries** folder.

How can we list the 10 most populous provinces in decreasing order?

```
SELECT province, pop2020
FROM provinces
ORDER BY pop2020 DESC
LIMIT 10;
```

- **SELECT province, pop2020:** This line tells the database to retrieve data from two columns, "province" and "pop2020," in the "provinces" table.
- **FROM provinces:** This line specifies that you want to fetch the data from the "provinces" table.
- **ORDER BY pop2020 DESC:** Here, you're asking the database to arrange the data in descending order based on the values in the "pop2020" column. In other words, you want to see the highest population first.
- **LIMIT 10:** This line restricts the results to just the top 10 rows so you'll get data for the 10 provinces with the highest populations.



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
📱 @bnhr.xyz | 📺 @bnhrdotxyz | 🐙 mastodon.social/@benhur07b

Query
Query History

```

1 SELECT province, pop2020
2 FROM provinces
3 ORDER BY pop2020 DESC
4 LIMIT 10;

```

Data Output
Messages
Notifications

	province character varying	pop2020 integer
1	Cebu	5151274
2	Second District, N...	4771371
3	Cavite	4344829
4	Fourth District, NCR	3861951
5	Bulacan	3708890
6	Laguna	3382193
7	Rizal	3330143
8	Negros Occidental	3223955
9	Pangasinan	3163190
10	Third District, NCR	3004627

TRY FOR YOURSELF:

What if you want to show more fields/columns? What if you want to show more rows?



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

Add and update a field

What if you want to add a new field (psgc_region)?

```
ALTER TABLE provinces
ADD psgc_regions character varying;

UPDATE provinces
SET psgc_regions = LEFT(psgc_province, 4) || '0000000';
```

1. **ALTER TABLE provinces ADD psgc_regions character varying;** In this line, you're telling the database to modify the "provinces" table by adding a new column called "psgc_regions." This column will store text (character varying data).
2. **UPDATE provinces SET psgc_regions = LEFT(psgc_province, 4) || '0000000';** In this line, you're updating the data in the "psgc_regions" column for each row in the "provinces" table. You're setting the value of "psgc_regions" to be the first four characters of the "psgc_province" column, followed by seven zeros ('0000000'). This effectively creates values based on the first four characters of the "psgc_province" field, followed by zeroes, and updates the "psgc_regions" column accordingly.

Query
Query History

```

1 ALTER TABLE provinces
2 ADD psgc_regions character varying;
3
4 UPDATE provinces
5 SET psgc_regions = LEFT(psgc_province, 4) || '0000000';

```

Data Output
Messages
Notifications

UPDATE 85

Query returned successfully in 43 msec.

TRY FOR YOURSELF:

How will you check if you actually added the column values?



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

Aggregate or compute based on a column

How do you compute for the total population per region and list the regions with population > 5,000,000 in increasing order?

```
SELECT psgc_regions, SUM(pop2020) AS total_population
FROM provinces
GROUP BY psgc_regions
HAVING SUM(pop2020) > 5000000
ORDER BY total_population ASC;
```

1. **SELECT psgc_regions, SUM(pop2020) AS total_population:** With this line, you're selecting data from two columns, "psgc_regions" and the sum of "pop2020," while giving the sum an alias called "total_population."
2. **FROM provinces:** You're specifying that you want to get this data from the "provinces" table.
3. **GROUP BY psgc_regions:** You're telling the database to group the data based on the values in the "psgc_regions" column.
4. **HAVING SUM(pop2020) > 5000000:** This line filters the grouped data, showing only those groups (regions) where the sum of "pop2020" is greater than 5,000,000
5. **ORDER BY total_population ASC:** Finally, you're instructing the database to arrange the results in ascending order based on the "total_population" column. This means you'll see regions with the lowest total population first.



This work and its contents by BNHR is licensed under a
[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
 @bnhr.xyz | @bnhrdotxyz | mastodon.social/@benhur07b

Query

Query History

1

SELECT psgc_regions, SUM(pop2020) AS total_population

2

FROM provinces

3

GROUP BY psgc_regions

4

HAVING SUM(pop2020) > 5000000

5

ORDER BY total_population ASC;

Data Output

Messages

Notifications

psgc_regions

character varying

total_population

bigint

1

PH100000000

5022768

2

PH110000000

5243536

3

PH010000000

5301139

4

PH050000000

6082165

5

PH060000000

7954723

6

PH070000000

8081988

7

PH030000000

12422172

8

PH130000000

13484462

9

PH040000000

16195042



This work and its contents by BNHR is licensed under a
[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
 @bnhr.xyz | @bnhrdotxyz | mastodon.social/@benhur07b

Wildcards and other select options

How about if you want to list all the provinces whose name starts with B in order of decreasing population?

```
SELECT *
FROM provinces
WHERE province LIKE 'B%'
ORDER BY pop2020 DESC;
```

1. **SELECT *:** With this line, you're instructing the database to select all columns ("*") from the "provinces" table. This means you want to retrieve all available information about the provinces.
2. **FROM provinces:** You're specifying where you want to get this data from.
3. **WHERE province LIKE 'B%':** Here, you're adding a condition. You're telling the database to only include rows where the "province" column starts with the letter 'B'. The '%' symbol acts as a wildcard, matching any characters that follow 'B'.
4. **ORDER BY pop2020 DESC:** This line instructs the database to sort the results in descending order based on the "pop2020" column. In other words, you want to see provinces with the highest population first.

Query

Query History

1

SELECT *

2

FROM provinces

3

WHERE province LIKE 'B%'

4

ORDER BY pop2020 DESC;

Data Output

Messages

Notifications

≡

📄

▼

📋

▼

🗑

🔄

⬇

📈

	province character varying	psgc_province [PK] character varying	pop2020 integer	pop_perc numeric	notes character varying	psgc_regions character varying
1	Bulacan	PH031400000	3708890	3.4	Includes Malolos and San Jose del Monte	PH030000000
2	Batangas	PH041000000	2908494	2.67	Includes Batangas City and Lipa	PH040000000
3	Bukidnon	PH101300000	1541308	1.41	[null]	PH100000000
4	Bohol	PH071200000	1394329	1.28	[null]	PH070000000
5	Bataan	PH030800000	853373	0.78	[null]	PH030000000
6	Benguet	PH141100000	827041	0.76	Includes Baguio	PH140000000
7	Basilan	PH150700000	556586	0.51	Includes Isabela City	PH150000000
8	Biliran	PH087800000	179312	0.16	[null]	PH080000000
9	Batanes	PH020900000	18831	0.02	[null]	PH020000000



This work and its contents by BNHR is licensed under a
 Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
[@bnhr.xyz](https://www.bnhx.org) | [@bnhrdotxyz](https://www.bnhx.org) | [mastodon.social/@benhur07b](https://www.bnhx.org)

Exercise 02: More SQL queries

In this supervised exercise, you will be asked to perform other various SQL Queries to arrive at a desired outcome.

Note:

To perform SQL queries in Postgres, you can either use a DB Manager such as pgAdmin4 (via the Query Tool or PSQL tool) or directly from the command line/terminal using psql.

1. Connect to your server.
2. Create a **database** named **laguna**.

Figure 9: Creating a database named laguna in pgAdmin4

3. Create a table for each CSV file below. Take note to use the appropriate field types and name the tables according to their file name. You can find information about each column inside the **headers** folder.
 1. municity.csv – csv containing information for cities and municipalities in Laguna
 2. alaminos.csv – csv containing information on barangays in Alaminos
 3. liliw.csv – csv containing information on barangays in Liliw
 4. rizal.csv – csv containing information on barangays in Rizal
 5. sanpablo.csv – csv containing information on barangays in San Pablo
 6. victoria.csv – csv containing information on barangays in Victoria



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

TRY FOR YOURSELF:

Open and study the CSVs before creating tables for them.

```

1 CREATE TABLE municipality (
2     name varchar(50),           -- Name of city or municipality
3     psgc varchar(10),          -- PSGC code
4     income_class varchar(3),   -- Municipal or city income class (e.g. 1st to 5th)
5     city_class varchar(20),    -- Component or independent city
6     district varchar(3),       -- Provincial district (e.g. 1st to 4th)
7     voters integer,            -- Number of voters
8     area real                  -- Area in hectares
9 );

```

Figure 10: SQL query for creating the municipality table

```

1 CREATE TABLE alaminos (
2     name varchar(50),           -- Name of city or municipality
3     psgc varchar(10),          -- PSGC code
4     ur varchar(5),             -- Urban or rural
5     pop integer                -- Population
6 );

```

Figure 11: SQL query for creating the alaminos table (barangays)

SOMETHING TO THINK ABOUT:

What could be the difference, pros, and cons of using **varchar(N)** and **character varying** when defining our character fields?

4. Populate the newly created fields by copying the data from the CSVs. You can use the Query Tool or the PSQL tool.

```

laguna=# \COPY municipality FROM '/home/bnhr/municipality.csv' WITH CSV;
COPY 30
laguna=#

```

Figure 12: PSQL query for copying from the municipality.csv to the municipality table



This work and its contents by BNHR is licensed under a
 Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
[f @bnhr.xyz](https://www.facebook.com/bnhr.xyz) | [y @bnhrdotxyz](https://www.youtube.com/channel/UCbnhrdotxyz) | [m mastodon.social/@benhur07b](https://mastodon.social/@benhur07b)

```
laguna=# \COPY alaminos FROM '/home/bnhr/alaminos.csv' WITH CSV;
COPY 15
laguna=#
```

Figure 13: PSQL query for copying from alaminos.csv to the alaminos table

5. Make sure to do the same for all CSVs.

Views

Views in Postgres are **virtual tables** that allow users to query and retrieve data from one or more tables as if they were a single table. They are important for:

1. **Data Abstraction:** Views provide a level of abstraction, allowing users to work with complex data structures without the need to understand the underlying database schema.
2. **Data Security:** Views can restrict access to certain columns or rows, providing an additional layer of data security by controlling what data users can see.
3. **Simplified Queries:** Views can simplify complex queries by encapsulating them into a single, easy-to-understand view. This enhances query performance and readability.
4. **Data Transformation:** Views can be used to transform data before it's presented to users, making them invaluable for data processing and reporting.

To **create a view** in Postgres, you use the **CREATE VIEW** statement followed by a **SELECT** query that defines the view's structure. For example:

```
CREATE VIEW my_view AS
SELECT column1, column2
FROM my_table
WHERE condition;
```

To **use views**, simply treat them as regular tables. For example, to retrieve data from a view, you can use:

```
SELECT * FROM my_view;
```



This work and its contents by BNHR is licensed under a
[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

Create a view of the 2nd district of Laguna and name this view as second district

```
CREATE VIEW "second district" AS
SELECT *
FROM municity
WHERE district = '2nd';
```

Under Views in the public schema, you should now have second district.

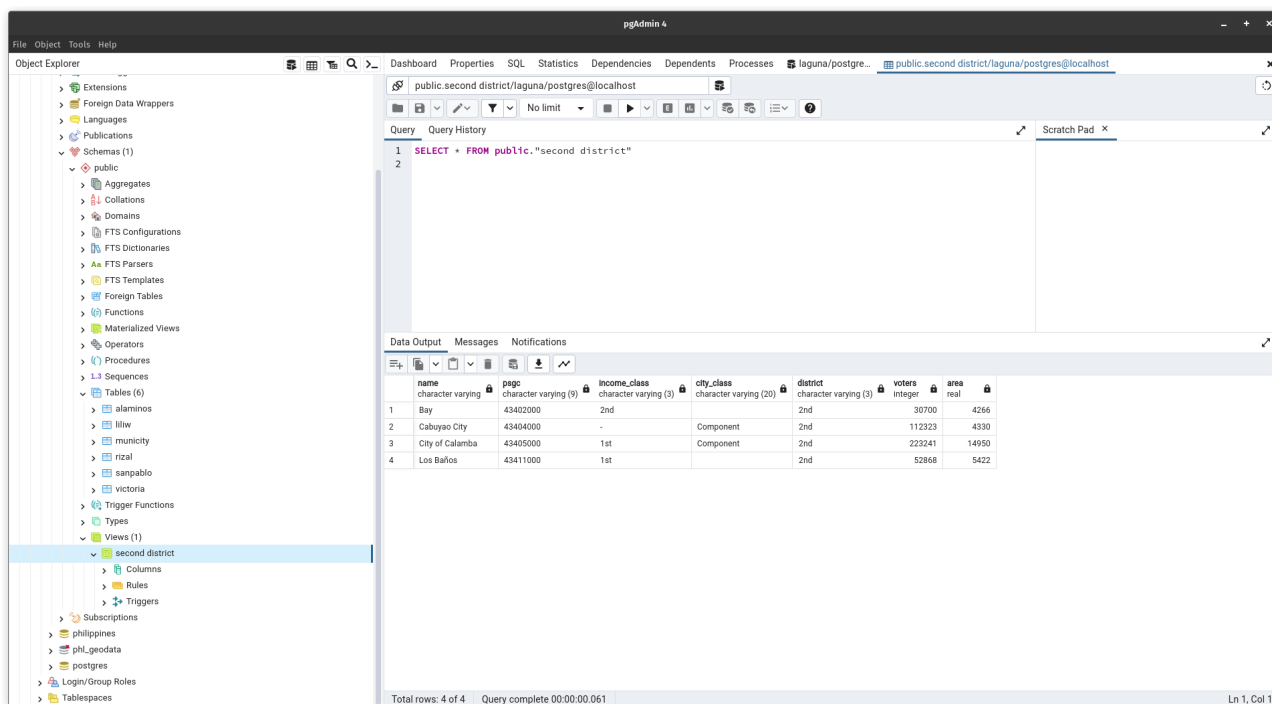


Figure 14: The second district View



This work and its contents by BNHR is licensed under a
[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[@bnhr.xyz](#) | [@bnhrdotxyz](#) | [mastodon.social/@benhur07b](#)

Create a view which unions all municipalities and name it `allbrgys_view`

```
CREATE VIEW allbrgys_view AS
SELECT * FROM alaminos
UNION ALL
SELECT * FROM liliw
UNION ALL
SELECT * FROM rizal
UNION ALL
SELECT * FROM sanpablo
UNION ALL
SELECT * FROM victoria
```

1. **CREATE VIEW allbrgys_view AS:** This line initiates the creation of a new view named "allbrgys_view." A view is a virtual table that doesn't store data itself but provides a convenient way to query data from multiple tables as if it were a single table.
2. **SELECT * FROM alaminos:** This line retrieves all columns (*) from the "alaminos" table. It's the first part of the view's definition, representing the data source from the "alaminos" table.
3. **UNION ALL:** This operator combines the results of the following SELECT statement with the results from the preceding one. In this case, it combines the data from "alaminos" with the data from the next table.
4. **SELECT * FROM liliw:** This line retrieves all columns from the "liliw" table. It's the second part of the view's definition, representing the data source from the "liliw" table.
5. **UNION ALL:** Similar to line 3, this operator continues to combine the results with the data from the "liliw" table with the data from the next table.
6. **SELECT * FROM rizal:** This line retrieves all columns from the "rizal" table, representing the data source from the "rizal" table.
7. **UNION ALL:** This operator continues to combine the results with the data from the "rizal" table with the data from the next table.
8. **SELECT * FROM sanpablo:** This line retrieves all columns from the "sanpablo" table, representing the data source from the "sanpablo" table.
9. **UNION ALL:** Once again, this operator combines the results with the data from the "sanpablo" table with the data from the last table.
10. **SELECT * FROM victoria:** This line retrieves all columns from the "victoria" table, representing the data source from the "victoria" table.



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

The screenshot shows the pgAdmin 4 interface. The left pane displays the database structure, with 'allbrgys_view' selected under 'Views (2)'. The main pane shows the query 'SELECT * FROM public.allbrgys_view' and the resulting data output table.

	name	psgc	ur	pop
	character varying	character varying (9)	character varying (5)	integer
1	Del Carmen	43401001	Rural	1089
2	Palma	43401002	Rural	2251
3	Barangay I (Pop.)	43401003	Urban	2780
4	Barangay II (Pop.)	43401004	Urban	2634
5	Barangay III (Pop.)	43401005	Urban	2629
6	Barangay IV (Pop.)	43401006	Urban	3127
7	San Agustin	43401007	Rural	4650
8	San Andres	43401008	Rural	3180
9	San Benito	43401009	Rural	5720
10	San Gregorio	43401010	Rural	3294
11	San Ildefonso	43401011	Rural	2479
12	San Juan	43401012	Rural	2640
13	San Miguel	43401013	Rural	1938
14	San Roque	43401014	Rural	1536
15	Santa Rosa	43401015	Rural	3579
16	Bagong Anyo (Pop.)	043410002	Urban	1171
17	Bayate	043410003	Rural	210
18	Bubukal	043410004	Rural	228
19	Bongkol	043410005	Rural	605
20	Caluyao	043410007	Rural	90
21	Calumpang	043410008	Rural	3589
22	Culoy	043410009	Rural	64
23	Dagatan	043410010	Rural	472
24	Danin (Daniliw)	043410011	Urban	724
25	Dita	043410012	Rural	353
26	Ibabang Palina	043410013	Rural	118

Total rows: 148 of 148 Query complete 00:00:00.100 Ln 1, Col 1

Figure 15: The allbrgys_view View

SOMETHING TO THINK ABOUT:

Why did we have double quotes for **"second district"** but none in **allbrgys_view**



This work and its contents by BNHR is licensed under a
[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

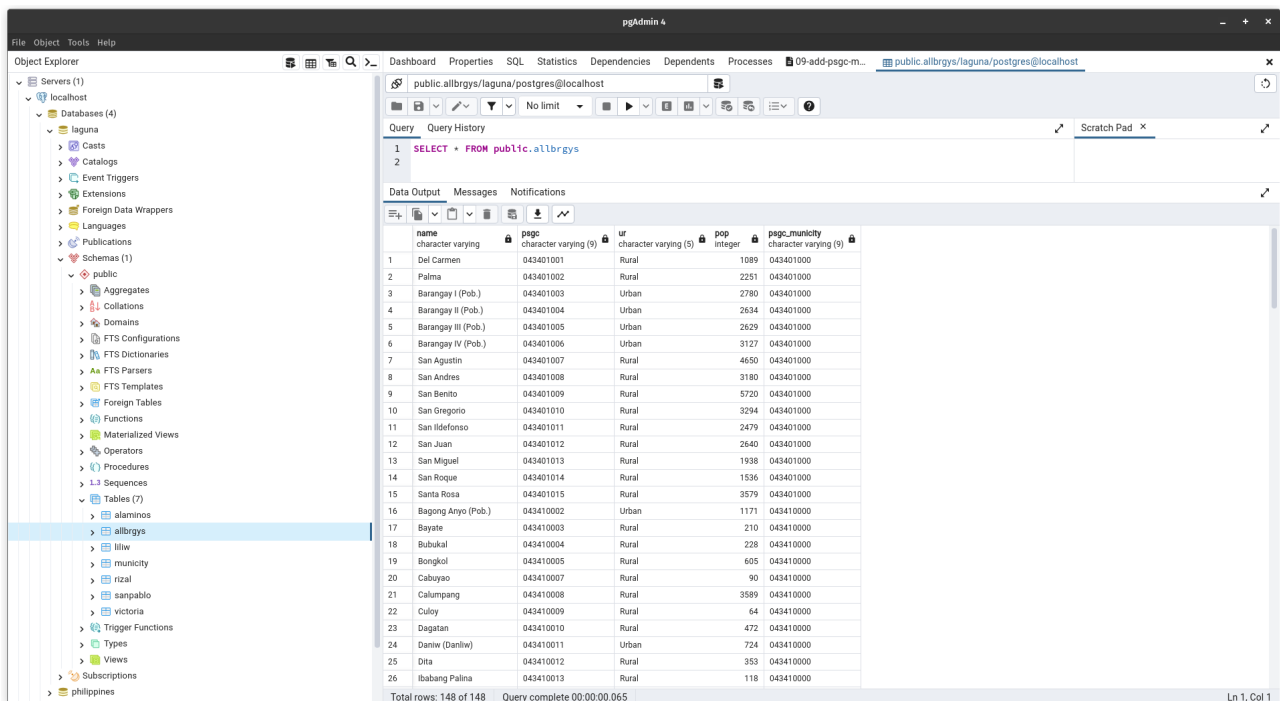
Create a table from the view `allbrgys_view` as `allbrgys`

```
CREATE TABLE allbrgys AS
SELECT * FROM allbrgys_view;
```

Add a column to the `allbrgys` table named `psgc_municipity`. Notice your `psgc_municipity` should turn the last 3 digits of your code to zero

```
ALTER TABLE allbrgys
ADD psgc_municipity character varying(9);

UPDATE allbrgys
SET psgc_municipity = LEFT(psgc, -3) || '000';
```



The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer shows the database structure, with the 'allbrgys' table selected under the 'public' schema. The main pane displays the table's data. The table has columns: name, psgc, ur, pop, and psgc_municipity. The data is as follows:

	name	psgc	ur	pop	psgc_municipity
1	Del Carmen	043401001	Rural	1089	043401000
2	Palma	043401002	Rural	2251	043401000
3	Barangay I (Pop.)	043401003	Urban	2780	043401000
4	Barangay II (Pop.)	043401004	Urban	2634	043401000
5	Barangay III (Pop.)	043401005	Urban	2629	043401000
6	Barangay IV (Pop.)	043401006	Urban	3127	043401000
7	San Agustin	043401007	Rural	4650	043401000
8	San Andres	043401008	Rural	3180	043401000
9	San Benito	043401009	Rural	5720	043401000
10	San Gregorio	043401010	Rural	3294	043401000
11	San Ildefonso	043401011	Rural	2479	043401000
12	San Juan	043401012	Rural	2640	043401000
13	San Miguel	043401013	Rural	1938	043401000
14	San Roque	043401014	Rural	1536	043401000
15	Santa Rosa	043401015	Rural	3579	043401000
16	Bagong Anyo (Pop.)	043410002	Urban	1171	043410000
17	Bayate	043410003	Rural	210	043410000
18	Bubukal	043410004	Rural	228	043410000
19	Bongkol	043410005	Rural	605	043410000
20	Cabuyao	043410007	Rural	90	043410000
21	Calumpang	043410008	Rural	3589	043410000
22	Culoy	043410009	Rural	64	043410000
23	Dagatan	043410010	Rural	472	043410000
24	Danliw (Danliw)	043410011	Urban	724	043410000
25	Dita	043410012	Rural	353	043410000
26	Ibabang Palma	043410013	Rural	118	043410000

Total rows: 148 of 148 Query complete 00:00:00.065 Ln 1, Col 1

Figure 16: `allbrgys` table with `psgc_municipity` field



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

Which municipality has the highest percentage of voters?

```
SELECT psgc_municipity, municipality.name, voters, SUM(pop) AS population,
100.0*voters/SUM(pop) AS "voting percentage"
FROM allbrgys, municipality
WHERE psgc_municipity = municipality.psgc
GROUP BY municipality.name, psgc_municipity, voters
ORDER BY "voting percentage" DESC;
```

1. **SELECT psgc_municipity, municipality.name, voters, SUM(pop) AS population, 100.0 * voters / SUM(pop) AS "voting percentage"**: In this line, you're selecting specific columns and performing calculations:
 - **psgc_municipity**: This is the unique identifier for a municipality.
 - **municipality.name**: This is the name of the municipality.
 - **voters**: The number of registered voters.
 - **SUM(pop) AS population**: It calculates the total population by summing up the "pop" column from the data.
 - **100.0 * voters / SUM(pop) AS "voting percentage"**: This calculates the voting percentage, which is the ratio of registered voters to the total population, expressed as a percentage.
2. **FROM allbrgys, municipality**: This specifies the source tables for your data. You are using two tables, "allbrgys" and "municipity."
3. **WHERE psgc_municipity = municipality.psgc**: This is a condition that filters the data. It restricts the rows selected to those where the "psgc_municipity" in the "allbrgys" table matches the "psgc" in the "municipity" table. This is how you link the data from both tables based on a common identifier.
4. **GROUP BY municipality.name, psgc_municipity, voters**: This line groups the data by specific columns. It groups by the name of the municipality, the "psgc_municipity" identifier, and the number of voters. This allows you to perform aggregate functions, like SUM, on groups of data.
5. **ORDER BY "voting percentage" DESC**: Finally, this line arranges the results in descending order based on the "voting percentage" column. It means that the results with the highest voting percentages will appear at the top.



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
 @bnhr.xyz | @bnhrdotxyz | mastodon.social/@benhur07b

Query
Query History

```

1 SELECT psgc_municipality, municipality.name, voters, SUM(pop) AS population, 100.0*voters/SUM(pop) AS "voting percentage"
2 FROM allbrgys, municipality
3 WHERE psgc_municipality = municipality.psgc
4 GROUP BY municipality.name, psgc_municipality, voters
5 ORDER BY "voting percentage" DESC;

```

Data Output
Messages
Notifications

+

📄

▼

📋

▼

🗑️

🔍

📥

📈

	psgc_municipality character varying (9)	name character varying	voters integer	population bigint	voting percentage numeric
1	043423000	Rizal	10264	15518	66.1425441422863771
2	043430000	Victoria	22443	34604	64.8566639694832967
3	043401000	Alaminos	27235	43526	62.5717961678077471
4	043410000	Liliw	20019	33851	59.1385778854391303
5	043424000	San Pedro	134486	248890	54.0343123468198803



This work and its contents by BNHR is licensed under a
[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
 @bnhr.xyz | @bnhrdotxyz | mastodon.social/@benhur07b

Common Table Expressions (CTE)

A Common Table Expression (CTE) is a temporary result set that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement. CTEs in Postgres allow you to create more readable and manageable SQL queries by breaking them into smaller, named, and self-contained units.

CTEs are important for:

1. **Modularity:** CTEs enhance the modularity of your SQL queries. You can define a CTE that represents a subquery or a specific logical operation, making your main query more concise and easier to understand.
2. **Readability:** Complex SQL queries can become convoluted and challenging to read. CTEs provide a way to give a name to intermediate results, making the main query more readable. They act as building blocks that make the logic of your query explicit.
3. **Reusability:** Once defined, a CTE can be referenced multiple times within the same query. This allows you to reuse a CTE to perform different operations within a single query, reducing redundancy.
4. **Optimization:** PostgreSQL's query planner can often optimize CTEs, improving query performance. In some cases, the planner may choose to materialize the CTE, storing the intermediate result in memory for quicker access.

CTEs can be **created** by using the **WITH** clause followed by the CTE name and its definition. The CTE definition follows the **AS** keyword. The basic structure is:

```
WITH cte_name (column1, column2, ...) AS (
    -- CTE query here
)
```

You can **use** CTEs in various contexts:

- In a SELECT statement to retrieve data.
- In an INSERT statement to add data.
- In an UPDATE statement to modify data.
- In a DELETE statement to remove data.



This work and its contents by BNHR is licensed under a
[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

In the allbrgys table, in terms of population, calculate the percent contribution of each brgy to their city or municipality

```
WITH total_pop AS (
    SELECT psgc_municipity, SUM(pop) AS total_pop
    FROM allbrgys
    GROUP BY psgc_municipity
)
SELECT
    allbrgys.name AS barangay,
    municipality.name AS municipality,
    allbrgys.psgc_municipity,
    allbrgys.pop AS barangay_pop,
    total_pop.total_pop AS municipality_pop,
    100.0 * allbrgys.pop / total_pop.total_pop AS pop_percentage_contribution
FROM
    allbrgys
JOIN total_pop ON allbrgys.psgc_municipity = total_pop.psgc_municipity
JOIN municipality ON allbrgys.psgc_municipity = municipality.psgc;
```

1. **WITH total_pop AS (...):** This is the start of a **Common Table Expression (CTE)** named total_pop. It's a temporary table-like structure that will be used in the main query. Inside the CTE, you're defining a subquery to calculate the total population for each psgc_municipity. The subquery groups data from the "allbrgys" table by psgc_municipity and calculates the sum of the "pop" column for each group.
2. **SELECT allbrgys.name AS barangay, municipality.name AS municipality, allbrgys.psgc_municipity, allbrgys.pop AS barangay_pop, total_pop.total_pop AS municipality_pop, 100.0 * allbrgys.pop / total_pop.total_pop AS pop_percentage_contribution:** In this line, you're selecting specific columns for the main query:

- allbrgys.name AS barangay: The name of the barangay.
- municipality.name AS municipality: The name of the municipality.
- allbrgys.psgc_municipity: The identifier for the municipality.
- allbrgys.pop AS barangay_pop: The population of the barangay.
- total_pop.total_pop AS municipality_pop: The total population of the municipality (retrieved from the CTE).
- 100.0 * allbrgys.pop / total_pop.total_pop AS pop_percentage_contribution: The percentage contribution of the barangay's population to the total population of the municipality.



This work and its contents by BNHR is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

3. **FROM allbrgys, total_pop, municipality:** This line specifies the source tables for your data. You are using three tables: "allbrgys," "total_pop" (the CTE), and "municipality."
4. **WHERE allbrgys.psgc_municipality = total_pop.psgc_municipality:** This is a condition that filters the data. It ensures that data from the "allbrgys" and "total_pop" tables is combined based on matching values in the `psgc_municipality` column.

Query

Query History

```

1 WITH total_pop AS (
2     SELECT psgc_municipality, SUM(pop) AS total_pop
3     FROM allbrgys
4     GROUP BY psgc_municipality
5 )
6 SELECT
7     allbrgys.name AS barangay,
8     municipality.name AS municipality,
9     allbrgys.psgc_municipality,
10    allbrgys.pop AS barangay_pop,
11    total_pop.total_pop AS municipality_pop,
12    100.0 * allbrgys.pop / total_pop.total_pop AS pop_percentage_contribution
13 FROM
14     allbrgys
15 JOIN total_pop ON allbrgys.psgc_municipality = total_pop.psgc_municipality
16 JOIN municipality ON allbrgys.psgc_municipality = municipality.psgc;

```

Data Output

Messages

Notifications

≡

📄

▼

📄

▼

🗑️

📄

📄

📄

📄

	barangay character varying	municipality character varying	psgc_municipality character varying (9)	barangay_pop integer	municipality_pop bigint	pop_percentage_contribution numeric
1	Del Carmen	Alaminos	043401000	1089	43526	2.5019528557643707
2	Palma	Alaminos	043401000	2251	43526	5.1716215595276387
3	Barangay I (Pob.)	Alaminos	043401000	2780	43526	6.3869870881771815
4	Barangay II (Pob.)	Alaminos	043401000	2634	43526	6.0515553921793870
5	Barangay III (Pob.)	Alaminos	043401000	2629	43526	6.0400680053301475
6	Barangay IV (Pob.)	Alaminos	043401000	3127	43526	7.1842117355144052
7	San Agustin	Alaminos	043401000	4650	43526	10.6832697697927675
8	San Andres	Alaminos	043401000	3180	43526	7.3059780361163443
9	San Benito	Alaminos	043401000	5720	43526	13.141570555300280
10	San Gregorio	Alaminos	043401000	3294	43526	7.5678904562790057
11	San Ildefonso	Alaminos	043401000	2479	43526	5.6954463998529614
12	San Juan	Alaminos	043401000	2640	43526	6.0653402563984745
13	San Miguel	Alaminos	043401000	1938	43526	4.4525111427652438
14	San Roque	Alaminos	043401000	1536	43526	3.5289252400863851
15	Santa Rosa	Alaminos	043401000	3579	43526	8.2226715066856591

Total rows: 148 of 148 Query complete 00:00:00.034

Figure 17: Computing the population contribution per barangay in each municipality



This work and its contents by [BNHR](#) is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).

 [@bnhr.xyz](#) |  [@bnhrdotxyz](#) |  [mastodon.social/@bnhr07b](#)