



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)

PostGIS, GeoServer, and QGIS in Multi-user and Enterprise environments

Table of Contents

Table of Contents.....	2
Credits and License.....	3
Building a geospatial database for multi-user or enterprise environments.....	4
Open geospatial tools for multi-user or enterprise geospatial data management systems.....	5
Open geospatial tools.....	6
Using PostGIS, GeoServer, and QGIS in multi-user and enterprise environments.....	7
Creating your database and schema.....	7
Adding Data to Your PostGIS Database.....	9
Connecting your database to QGIS.....	10
Assigning different roles and permissions.....	12
Assigning roles to users.....	12
Assigning database-level permissions.....	16
Assigning schema-level permissions.....	17
Assigning table-level permissions.....	19
Layer access.....	21
Column-level & Row-level permissions.....	22
Column-level permissions.....	22
Row-level permissions.....	23
Default permissions.....	24
Centralized Symbolology and Projects.....	25
Saving default styles for layers in PostGIS.....	25
Saving default styles for layers in GeoServer.....	26
Saving projects in PostGIS.....	26



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

Credits and License

PostGIS, GeoServer, and QGIS in Multi-user and Enterprise environments

BNHR

version 2023



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 [Ben Hur Pintor](#) is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

Building a geospatial database for multi-user or enterprise environments

Depending on what the organization wants to achieve, building an enterprise GIS database is crucial for several reasons:

- **Data organization:** A well-established GIS database allows efficient organization and management of large amounts of spatial data. This can include anything from administrative boundaries to demographic data.
- **Data Integration:** With a proper GIS database, spatial data can be integrated with other data types that your organization collects such as data in the form of spreadsheets.
- **Collaboration:** Enterprise GIS databases are typically designed to allow multiple users across the organization to access or sometimes even manipulate data. This makes it a powerful tool for individuals or teams working together.
- **Scalability:** As your organization grows, the amount of data you collect and manage also grows. An enterprise GIS database can handle this growth allowing you to store and manage increasingly large amounts of spatial data.
- **Data Security:** An enterprise GIS database provides robust security features ensuring that sensitive data is only accessible by authorized users.

Building a geospatial database for multi-user or enterprise environments is a strategic imperative for organizations seeking to harness the full potential of spatial data. The reasons are both compelling and multifaceted.

A well-organized GIS database becomes the central repository for a wealth of spatial information, enabling efficient data management. This extends to seamless data integration, where spatial data harmoniously coexists with diverse data types, from spreadsheets to demographic information.

The collaborative power of an enterprise GIS database empowers teams to work collectively, fostering productivity and data-driven decision-making.

Scalability ensures that as organizations expand, their geospatial data infrastructure grows with them, accommodating the influx of data. Crucially, robust security features safeguard sensitive information, guaranteeing that only authorized users gain access.

In an era of digital transformation and data-driven insights, building an enterprise GIS database isn't merely advantageous—it's an essential tool for achieving organizational goals and ensuring data's integrity, accessibility, and impact.



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

Open geospatial tools for multi-user or enterprise geospatial data management systems

Building an enterprise GIS database using open geospatial technologies offers several additional advantages that align with the principles of openness, collaboration, and cost-effectiveness. Here are some key points highlighting the benefits of using open geospatial tools in the context of an enterprise GIS database:

- **Cost-Effectiveness:** Open source geospatial software and tools are often available for free or at significantly lower costs than proprietary alternatives. This cost-effectiveness allows organizations to allocate resources to other critical areas.
- **Interoperability:** Open geospatial standards ensure that your GIS database can seamlessly integrate with a wide range of data sources and applications. This is crucial for organizations that need to work with diverse datasets and software.
- **Community Support:** Open source geospatial projects benefit from active communities of developers, users, and contributors. These communities provide ongoing support, updates, and enhancements, making open source solutions reliable and sustainable.
- **Flexibility and Customization:** Open geospatial tools provide the flexibility to tailor your GIS database to the specific needs of your organization. Customization options are vast, allowing you to adapt the database to unique requirements.
- **Scalability:** Open source geospatial solutions can scale with your organization's growing data and user needs. You can deploy the software on your infrastructure, whether it's on-premises or in the cloud, providing scalability options.
- **Data Privacy and Security:** Open geospatial tools can be configured to meet stringent data privacy and security requirements. You have full control over the security measures, ensuring that sensitive spatial data remains protected.
- **Open Data Initiatives:** Open geospatial tools align with open data initiatives, allowing organizations to share their spatial data with the public, fostering transparency and collaboration with external stakeholders.
- **Avoiding Vendor Lock-In:** By using open geospatial solutions, organizations can avoid vendor lock-in, reducing dependence on proprietary software vendors and minimizing potential licensing issues in the long term.
- **Collaborative Development:** Open source geospatial projects encourage collaborative development and sharing of best practices. Organizations can leverage the collective knowledge and experience of the open source community.



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

In summary, adopting open geospatial technologies to build an enterprise GIS database not only offers cost-efficiency and flexibility but also ensures compatibility, security, and the ability to collaborate and scale effectively. Open geospatial solutions align with the broader principles of open data and open standards, fostering a culture of transparency and cooperation while meeting an organization's geospatial data management needs.

Open geospatial tools

In a multi-user or enterprise environment, you might use the following open geospatial tools:

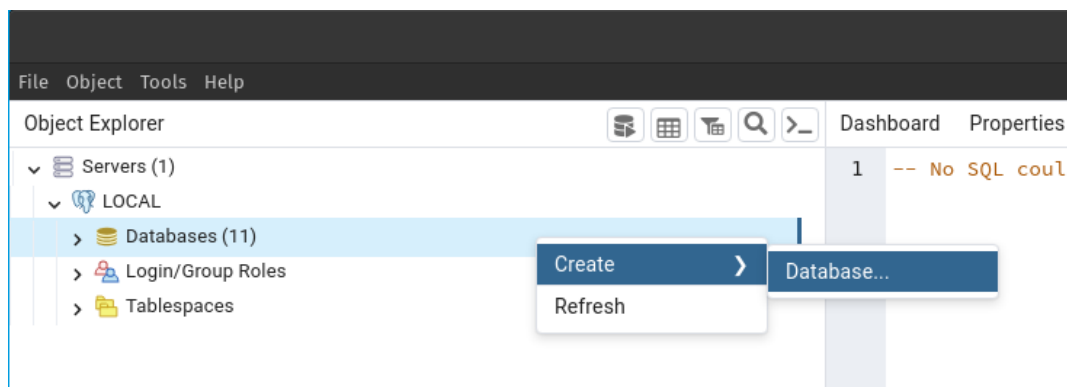


This work and its contents by [Ben Hur Pintor](#) is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
📄 [@bnhr.xyz](#) | 📺 [@bnhrdotxyz](#) | 🐙 [mastodon.social/@benhur07b](#)

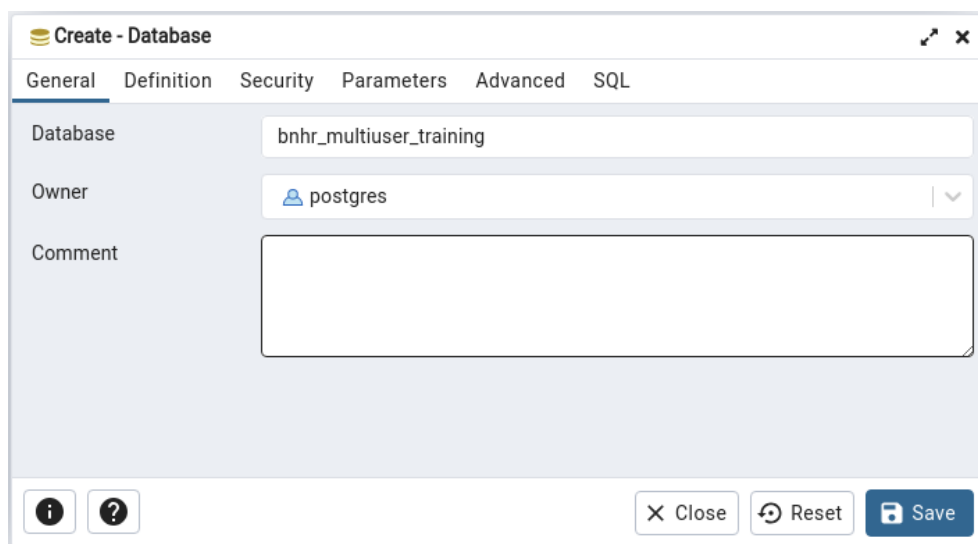
Using PostGIS, GeoServer, and QGIS in multi-user and enterprise environments

Creating your database and schema

1. **Create a new database:** Open pgAdmin4 and connect to a server. Right-click on **"Databases"** then select **"Create > Database"**.



2. In the **"Create - Database"** window, under the **"General"** tab, name your database and add the postgis extension by using your postgis_template. Leave everything as default and click **"Save"**. Under your **"Databases"** in the Object Explorer panel, your new database should now become visible.



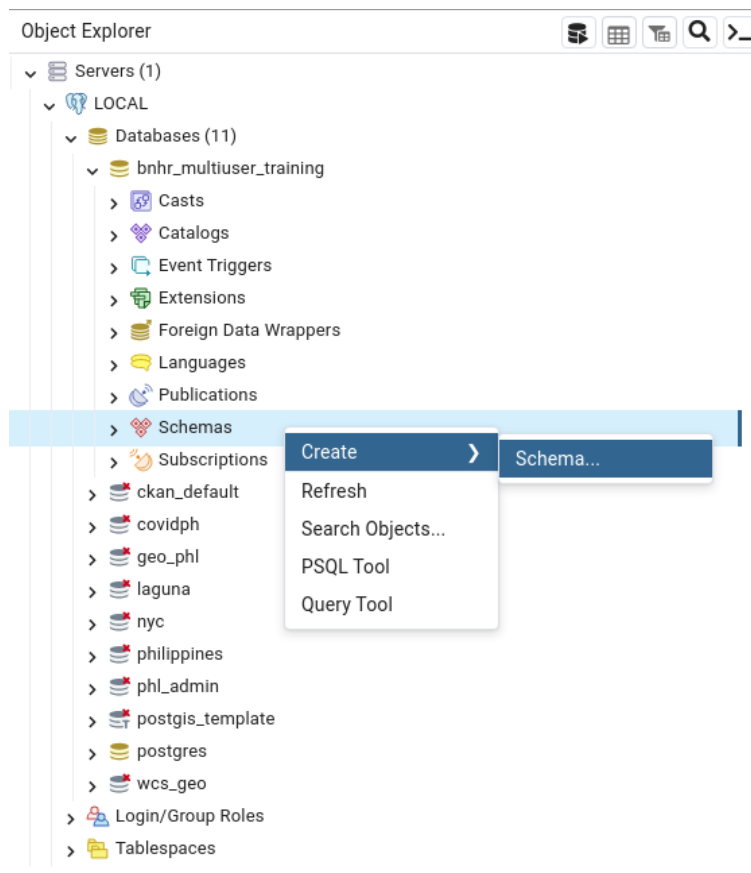
IMPORTANT:

Make sure you have the postgis extension added to your database.

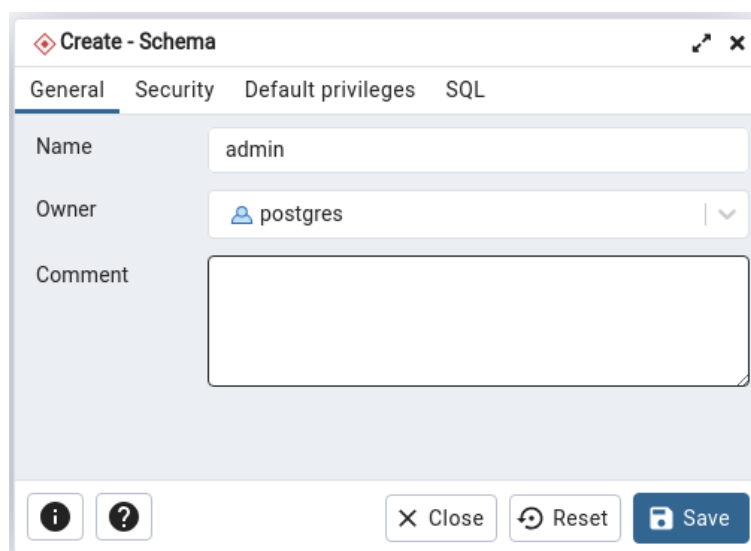


This work and its contents by [Ben Hur Pintor](#) is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

3. **Create a new schema:** Under your new database, you should see “**Schemas**”. Right-click on this and select “**Create > Schema**”.



4. In the “Create - Schema” window, under “**General Tab**”, name the new schema to be created as “**admin**” and leave the rest of the parameters in default. Click “**Save**” and the new schema should now be visible in your Object Explorer panel. You will use this schema later on to store some of your data.



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

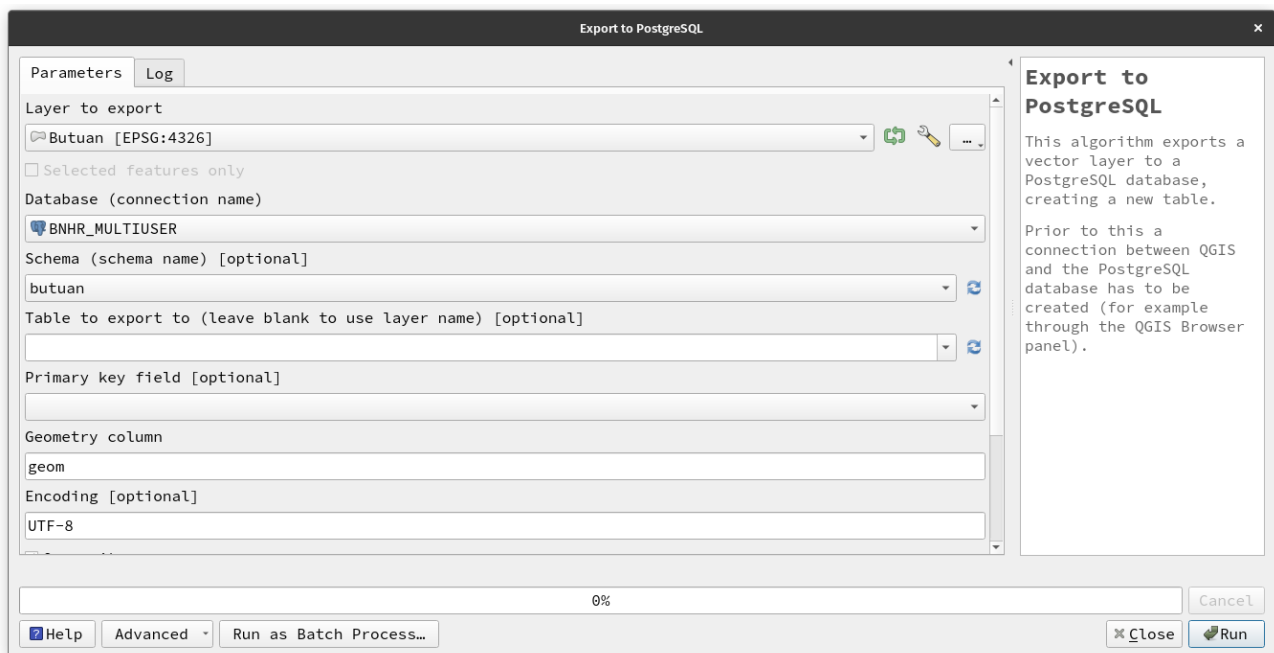
5. Create other schemas (e.g. butuan to hold the butuan data).

IMPORTANT:

It's good practice to use schemas in order to divide/categorize your database tables. In our example, we might use the **public** schema for datasets that we want to expose to the public and the **admin** schema for data that only an admin can access or edit.

Adding Data to Your PostGIS Database

1. **Data preparation:** Download the sample data or use any if you have any data you wish to use. Ensure that your data is in a format that PostGIS can import. Common formats include Shapefile (.shp), GeoJSON (.geojson), and CSV (.csv). After downloading the data, extract the files. For this tutorial, we will use the data named Barangays.shp.
2. **Importing the data:** You can use any of the previously discussed methods to upload your shapefile to your database. For example, you may use the shp2pgsql tool.
3. **Using QGIS to import data:** Another way to upload data to your database is via QGIS functions such as the **Export to PostgreSQL** processing algorithms. For example, you can download building footprints from QuickOSM then upload them to PostGIS.



4. **Add data to different schema:** Try to put data in different schemas in your database. For example: add some layers inside butuan while other layers inside admin.



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

Connecting your database to QGIS

QGIS can interface with Postgres/PostGIS databases directly. To connect your database to QGIS, you should:

1. Open QGIS.
2. Go to **Layer > Add Layer > Add PostGIS Layers** on the Menu bar or **right click on PostgreSQL/PostGIS > New connection** on the Browser. Put a name for your new connection. Enter your database credentials, specifically the host (localhost) and database. When prompted, enter your username (postgres) and password. Click **"Connect"**.

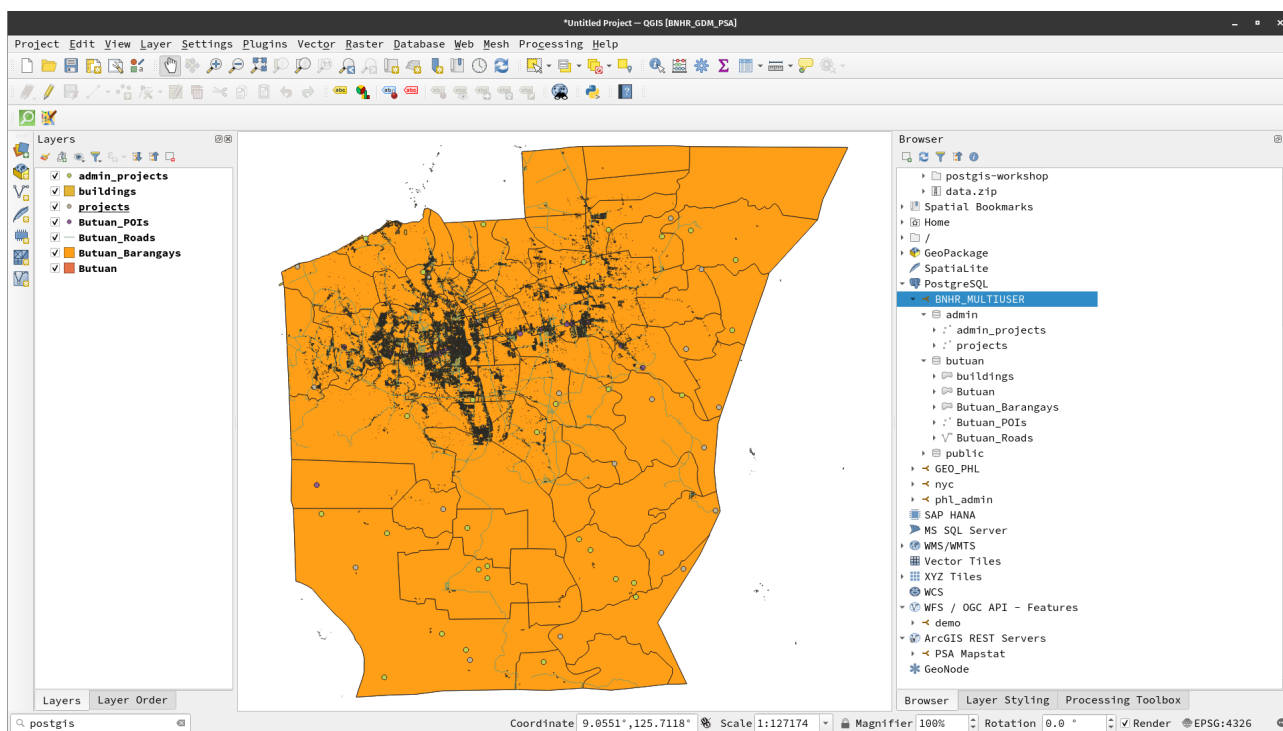
IMPORTANT

As you'll find out later, the user/role that you use to connect QGIS to your database will determine what you can do with the data. For example, if you connect to the database with a role with only read privileges, you will not be able to edit/write new data to the database.



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

3. You should now be able to see the data you have in the database in QGIS.



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

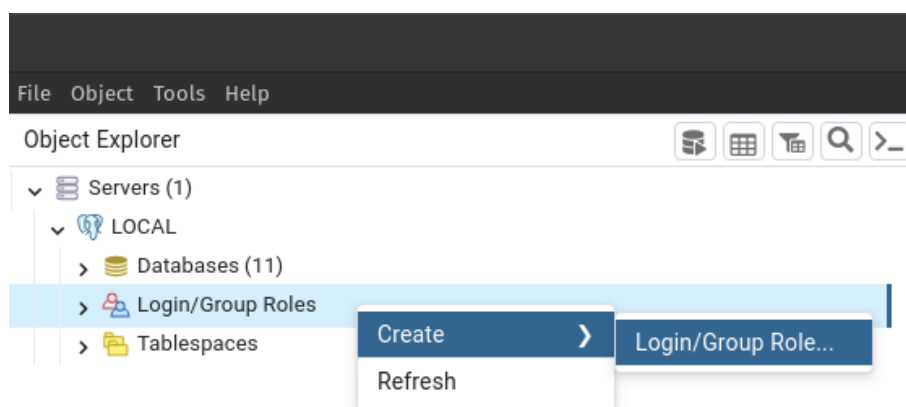
Assigning different roles and permissions

In an enterprise GIS database, it is ideal for different users to have different roles and permissions. These can range from superusers (who have all permissions), to editors (who can edit tables and layers), to readers (who can only view and load data). This system allows for the control of access to sensitive data ensuring that only authorized personnel can make changes to the database. This helps maintain the integrity and security of the data by giving multiple users access to the database with permission only relevant to their role.

This tutorial will guide you through the process of assigning different roles and permissions to users, database-level permissions, and schema and table-level permissions.

Assigning roles to users

1. **Open pgAdmin.**
2. **Connect to your database.**
3. **Create Roles.** In the Object Explorer panel on the left side of the pgAdmin interface, navigate to Login/Group Roles. Right-click on it and select **Create > Login/Group Role....**

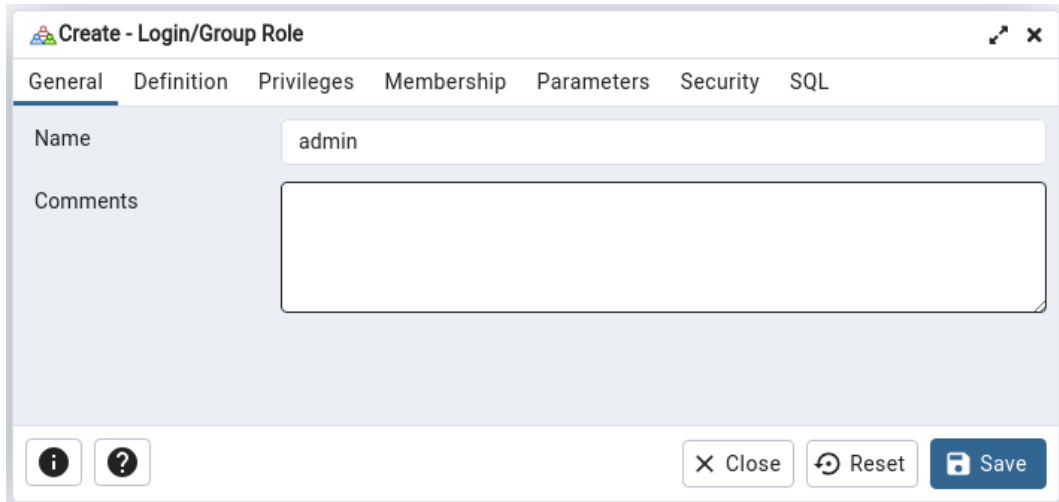


This work and its contents by [Ben Hur Pintor](#) is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
📧 [@bnhr.xyz](#) | 📺 [@bnhrdotxyz](#) | 🐙 [mastodon.social/@benhur07b](#)

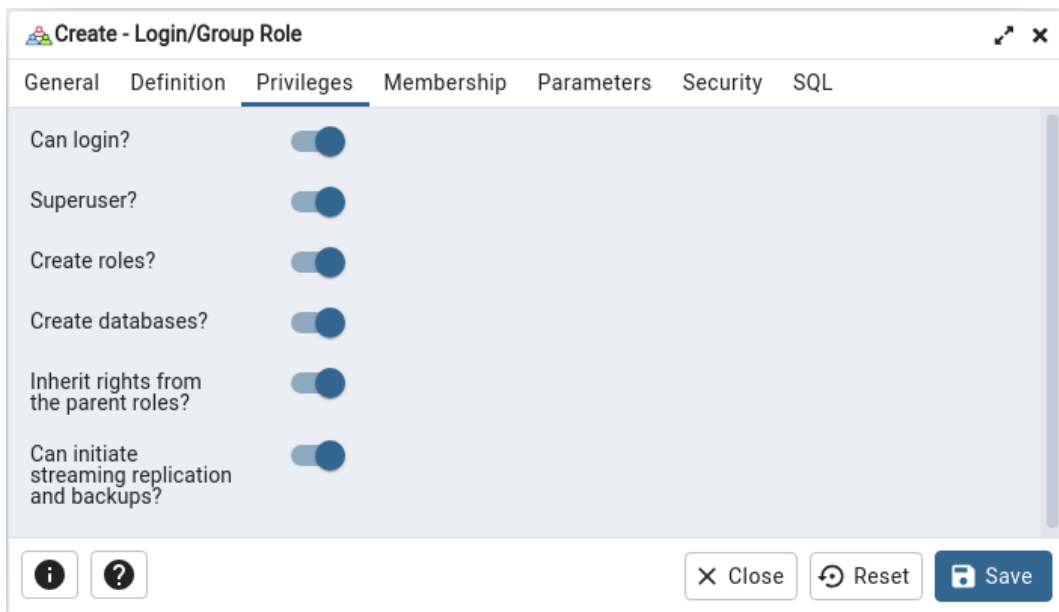
4. **admin Role.** In the “**Create - Login/Group Role**” window, under the General tab, enter a name for the role. In this case, type “**admin**”. This “**admin**” role is a **superuser** and has **complete access and control** to the database.

In the **Privileges** tab, check **all options** to grant this role all privileges. Click Save.

You should also provide a **password** for the role in the **Definition** tab.



The screenshot shows the 'Create - Login/Group Role' window with the 'General' tab selected. The 'Name' field contains the text 'admin'. The 'Comments' field is empty. At the bottom, there are buttons for 'Close', 'Reset', and 'Save'.



The screenshot shows the 'Create - Login/Group Role' window with the 'Privileges' tab selected. The following privileges are listed with toggle switches turned on:

- Can login?
- Superuser?
- Create roles?
- Create databases?
- Inherit rights from the parent roles?
- Can initiate streaming replication and backups?

At the bottom, there are buttons for 'Close', 'Reset', and 'Save'.

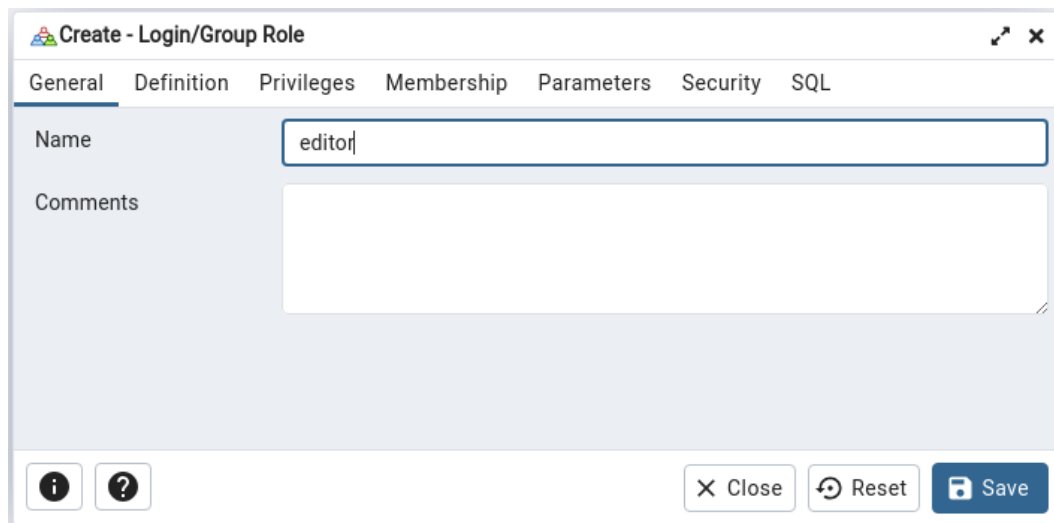


This work and its contents by [Ben Hur Pintor](#) is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

5. **editor Role.** Repeat the steps above but this time create the role of “**editor**”. In the General tab, type a name for the role “**editor**”. An “editor” has only **partial access** to the database but is allowed to **edit tables and layers**.

In the **Privileges** tab, check options for “**Can login**”, “**Can create databases**”, and “**Can create roles**”. Click Save.

You should also provide a **password** for the role in the **Definition** tab.



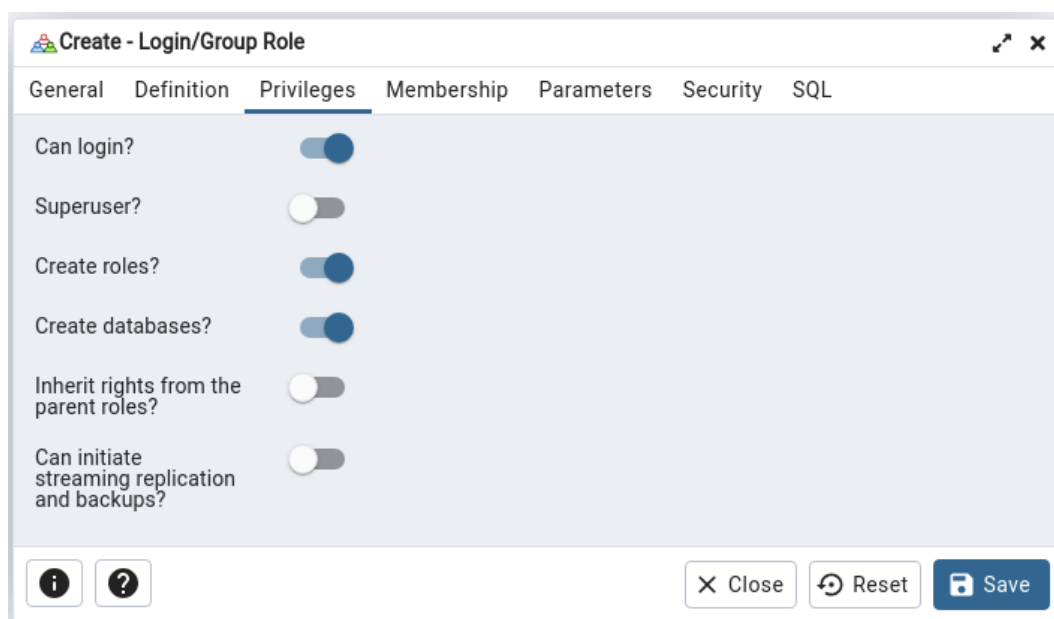
Create - Login/Group Role

General Definition Privileges Membership Parameters Security SQL

Name: editor

Comments:

Close Reset Save



Create - Login/Group Role

General Definition Privileges Membership Parameters Security SQL

Can login? ☒

Superuser? ☐

Create roles? ☒

Create databases? ☒

Inherit rights from the parent roles? ☐

Can initiate streaming replication and backups? ☐

Close Reset Save



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

6. **reader Role.** Do the same steps again this time for the “**reader**” role. In the General tab, enter the “**reader**” name for the role. The “**reader**” role has **very limited access** to the database only **allowed to view and load data but not edit it**.

In the **Privileges** tab, check only the option for “**Can login**” and click Save.

You should also provide a **password** for the role in the **Definition** tab.

The image shows two screenshots of the 'Create - Login/Group Role' dialog box. The top screenshot is the 'General' tab, where the 'Name' field is set to 'reader' and the 'Comments' field is empty. The bottom screenshot is the 'Privileges' tab, where the 'Can login?' toggle is turned on, and all other toggles ('Superuser?', 'Create roles?', 'Create databases?', 'Inherit rights from the parent roles?', 'Can initiate streaming replication and backups?') are turned off. Both screenshots show 'Close', 'Reset', and 'Save' buttons at the bottom right.

IMPORTANT

LOGIN VS NON-LOGIN ROLES. **Login** roles can “log-in” to your database. You may think of them similar to **users**. **Non-login** roles can’t log-in directly but may be used to manage permissions. You may think of them similar to **groups**.



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

Assigning database-level permissions

1. Right-click on your database in the Object Explorer panel and select **“Properties”**.
2. In the **Security** tab, you can assign **permissions to each role**. Under the **“Privileges”** section, **add three rows one row for each role using the + sign**.

In the **“Grantee”** column, click on the drop-down menu and select the appropriate roles.

For the **“Privileges”** column, select **“ALL”** for **“admin”** and the **“editor”** role, and **“CONNECT”** for the **“reader”** role.

3. Click Save.

The screenshot shows the PostgreSQL Security tab for the database 'bnhr_multouser_training'. The 'Privileges' section contains three rows:

Grantee	Privileges	Grantor
admin	CTc	postgres
editor	CTc	postgres
reader	c	postgres

The 'Security labels' section is empty. At the bottom, there are buttons for 'Close', 'Reset', and 'Save'.

TRY FOR YOURSELF:

Open a new QGIS instance (i.e. open/create a new User Profile) and connect to the database using one of the roles above. What do you notice?

IMPORTANT:

You need to assign permissions for the user/role across all database objects—database, schema, table.



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

Assigning schema-level permissions

1. Navigate to your schema in the Object Explorer panel (Databases > [database] > Schemas). Right-click on the **"butuan"** schema and select **"Properties"**.
2. In the **Security** tab, you can assign permissions to each role. Again, under the **"Privileges"** section, **add three rows one row for each role using the + sign**.

In the **"Grantee"** column, click on the drop-down menu and select the appropriate roles.

For the **"Privileges"** column, select **"ALL"** for **"admin"** and **"editor"** role, and **"USAGE"** for the **"reader"** role.

3. Click Save.

The screenshot shows the 'butuan' schema properties dialog box with the 'Security' tab selected. The 'Privileges' section contains the following data:

Grantee	Privileges	Grantor
admin	UC	postgres
editor	UC	postgres
postgres	UC	postgres
reader	U	postgres

The 'Security labels' section is currently empty. At the bottom, there are buttons for 'Close', 'Reset', and 'Save' (which is highlighted).



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

4. Assign the following privileges for the “**admin**” schema.

Grantee	Privileges	Grantor
postgres	UC	postgres
reader	U	postgres

Provider	Security label
----------	----------------

TRY FOR YOURSELF:

Can the reader see the tables inside butuan and/or admin?

IMPORTANT:

You need to assign permissions for the user/role across all database objects—database, schema, table.



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

Assigning table-level permissions

1. Navigate to your table in the Object Explorer panel (Databases > [database] > [schema] > Tables). Right-click on the “**Butuan**” table inside the **butuan schema** and select “**Properties**”.
2. In the **Security** tab, you can assign permissions to each role. Again, under the “**Privileges**” section, add three rows one row for each role using the + sign.

In the “**Grantee**” column, click on the drop-down menu and select the appropriate roles.

For the “**Privileges**” column, select “**ALL**” for “**admin**”; “**SELECT**”, “**INSERT**”, “**UPDATE**”, and “**DELETE**” for the “**editor**” role; and “**SELECT**” for the “**reader**” role.

3. Click Save.

The screenshot shows the 'Butuan' table properties dialog with the 'Security' tab selected. The 'Privileges' section contains the following data:

Grantee	Privileges	Grantor
admin	wdaxrtD	postgres
editor	draw	postgres
postgres	aDdrtwx	postgres
reader	r	postgres

The 'Security labels' section is currently empty. At the bottom, there are buttons for 'Close', 'Reset', and 'Save'.

4. Apply the same privileges for all tables inside the butuan schema.

TRY FOR YOURSELF:

Can the reader see the tables inside butuan and/or admin?

5. For the **projects** table in the **admin** schema, apply the following permissions: “**ALL**” for “**admin**” and “**SELECT**” for “**editor**”.



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

projects

General Columns Advanced Constraints Parameters Security SQL

Privileges +

	Grantee	Privileges	Grantor
	admin v	wdaxrtD	postgres
	editor v	r	postgres
	postgres v	aDdrtwx	postgres

Security labels +

Provider	Security label
----------	----------------

Close Reset Save

6. For the **admin_projects** table in the **admin** schema, apply the following permissions: “**ALL**” for “**admin**”.

admin_projects

General Columns Advanced Constraints Parameters Security SQL

Privileges +

	Grantee	Privileges	Grantor
	admin v	axtdDrw	postgres
	postgres v	raxdtDw	postgres

Security labels +

Provider	Security label
----------	----------------

Close Reset Save

IMPORTANT:

You need to assign permissions for the user/role across all database objects—database, schema, table.



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

Layer access

1. Create 3 QGIS User Profiles—one for admin, editor, and reader.
2. Open 3 QGIS instances—one for each user profile.
3. For each user profile:
 1. Connect to the QGIS database using the appropriate postgis user/role (i.e. connect using admin for the admin user profile, reader for the reader user profile, etc.)
 2. Try to look at what layers are available for each profile.
 3. What can you do or not do for each user profile/postgis role?
 1. Load data?
 2. Open attribute table?
 3. Edit the attribute table?
 4. Add new features?



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

Column-level & Row-level permissions

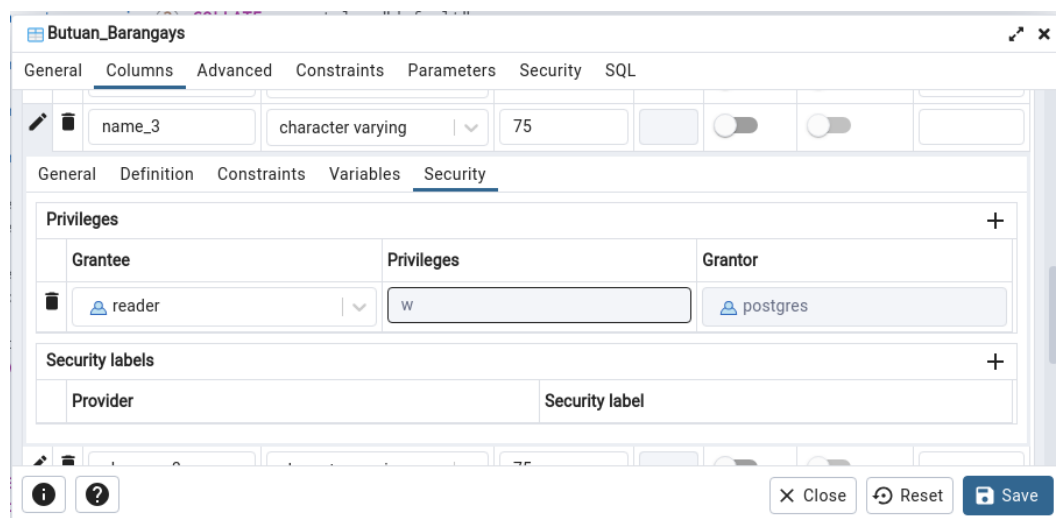
In PostgreSQL, we can set permissions at the column and row levels for more granular control over our data. This is useful when we want to restrict access to certain data within our tables. Additionally, setting default permissions can save time and ensure consistency when creating new tables.

In this tutorial, we will learn how to set:

- **column-level security** (only allow a user to edit a specific column or attribute of a table),
- **row-level security** (only allow users to view specific rows based on rules/permissions), and
- **default permissions** in PostGIS.

Column-level permissions

1. Open pgAdmin.
2. Connect to your database.
3. Navigate to the table you want to modify in the Object Explorer panel (Databases > [database] > [schemas] > butuan > Tables). Right-click on the “**Butuan_Barangays**” table and select “**Properties**”. In the Columns tab, select the column you want to modify. In this case, click the pencil tool at the leftmost side of the ‘name_3’ column.



TRY FOR YOURSELF:

Try to edit the Butuan_Barangays attribute table using the reader user.

4. Repeat the steps above to give the reader UPDATE privileges for other fields.



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

Row-level permissions

1. Navigate to the table you want to modify in the Object Explorer panel (Databases > [database] > [schemas] > butuan > Tables > projects). Right-click on the '**projects**' table and open the '**Query Tool**'. In the 'Query Tool', type this SQL code snippet:

```
ALTER TABLE butuan.projects ENABLE ROW LEVEL SECURITY;

CREATE POLICY reader_select_policy
ON butuan.projects
FOR SELECT
TO reader
USING ("created by" = 'reader');

CREATE POLICY reader_update_policy
ON butuan.projects
FOR UPDATE
TO reader
USING ("created by" = 'reader');
```

This code snippet gives the reader permission to view and update only the rows in the projects layer whose **created by** value is reader.

TRY FOR YOURSELF:

This technique is great for managing projects where you want users to see only select features/data based on their role.

For example, you have a layer with multiple points being managed by 10 users. You can use row level permissions so that each user can only see or update the points that they created.



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

Default permissions

1. Right-click on the database you want to assign default permissions to in the Object Explorer panel and select **"Properties"**.
2. In the **Default privileges** tab, you can set default permissions for new objects created within this database.
3. For the **"Privileges"** column, select **"ALL"** for **"admin"**; **"SELECT"**, **"INSERT"**, **"UPDATE"**, and **"DELETE"** for the **"editor"** role; and **"SELECT"** for the **"reader"** role.
4. Click "Save".
5. This way, whenever a new object is created in the database, these permissions are automatically applied.



Centralized Symbolology and Projects

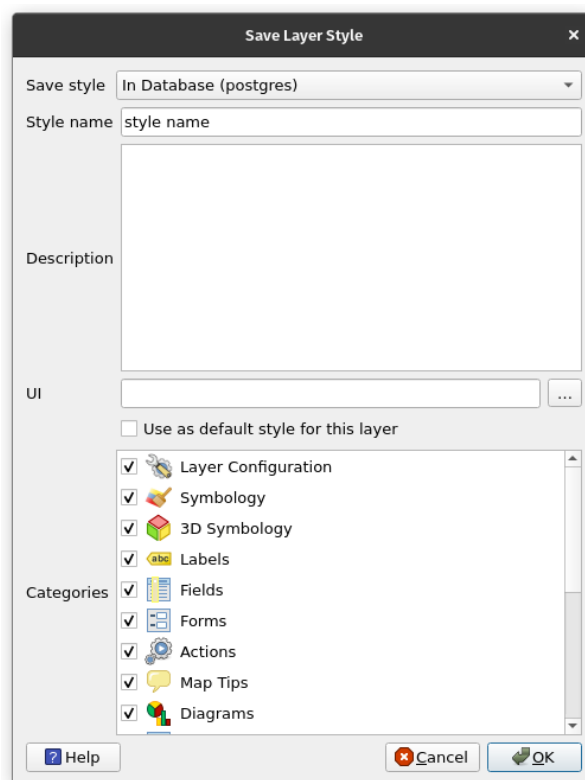
Saving default styles for layers in PostGIS

In order to save default styles for layers inside of PostGIS tables, you can:

1. Load the layer from PostGIS in QGIS.
2. Apply the symbology you want.
3. You can Save Default Style to the Datasource Database.



4. Or you can Save Style to a PostGIS table



IMPORTANT:

Make sure that a user/role has the appropriate privileges for the **layer_styles** table in PostGIS that holds the default and other styles for layers.



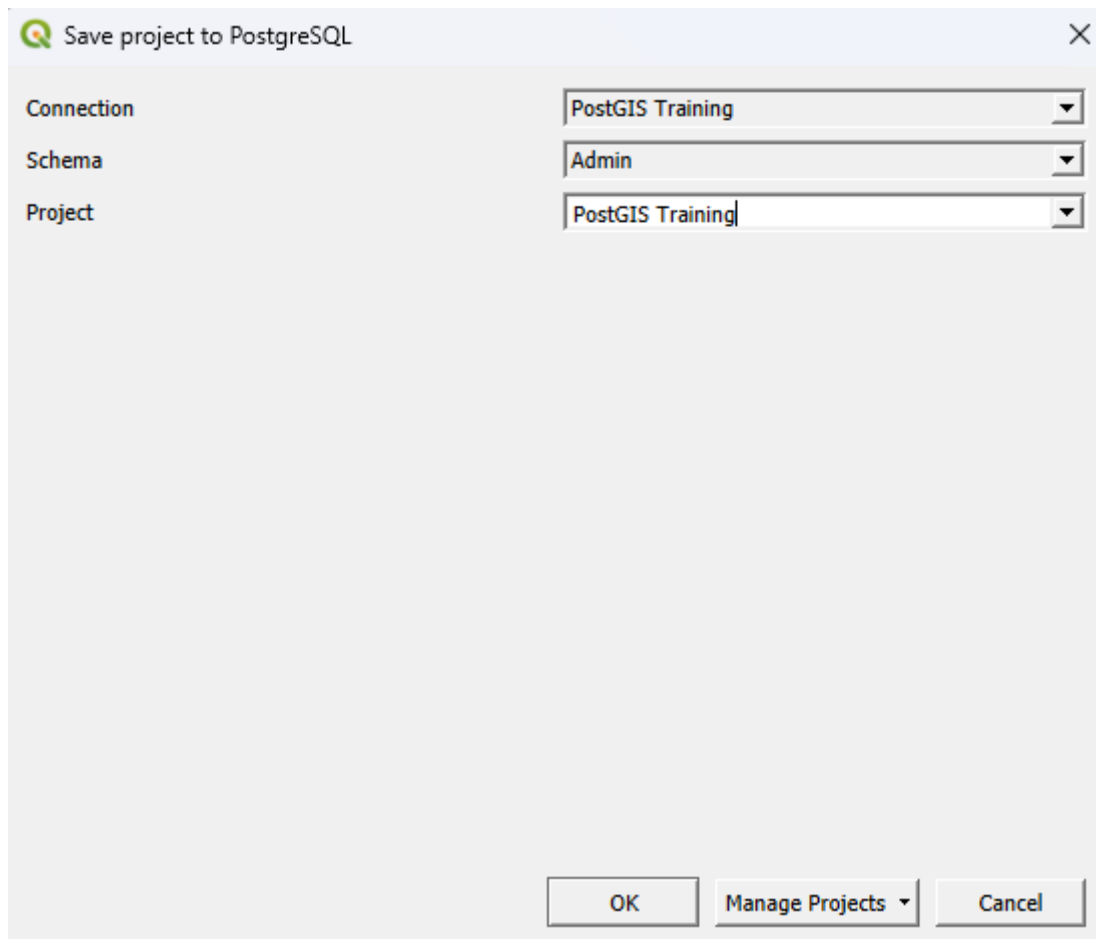
This work and its contents by [Ben Hur Pintor](#) is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).
[f @bnhr.xyz](#) | [y @bnhrdotxyz](#) | [m mastodon.social/@benhur07b](#)

Saving default styles for layers in GeoServer

Refer to the previous topic on styling layers in GeoServer.

Saving projects in PostGIS

You can save projects to a PostGIS database by going to **Project > Save To > PostgreSQL** in the menu bar.



IMPORTANT:

In case you encounter an error saying that your database does not allow QGIS projects to be saved, edit your Connection. Make sure that the option to “Allow saving/loading QGIS Projects in the database” is checked.



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