PostgreSQL

- PostgreSQL Permissions
- PostGreSQL .NET DateTime Usage
- Postgres Setup
- PostgreSQL Bulk Insert
- Postgres Commands
- HowTo: Install PostGreSQL
- Postgres: Troubleshooting Remote Access

PostgreSQL Permissions

Here are three major points you need to understand about object ownership:

1. Only a `superuser` or the `owner` of an object (table, function, procedure, sequence, etc.) can `ALTER`/`DROP` the object.
2. Only a `superuser` or the `owner` of an object can `ALTER` the ownership of that object.
3. Only the `owner` of an object can define default privileges for the objects they create.

PostGreSQL .NET DateTime Usage

.NET EF Core 6 made a breaking change to how DateTime is stored in PostGreSQL.

This article explains some of it and the workaround:

https://github.com/npgsql/efcore.pg/issues/2000

To enable the workaround, add this to the Configure method of your Startup.cs:

```
AppContext.SetSwitch("Npgsql.EnableLegacyTimestampBehavior", true);
```

See this page for what datatypes should be used: https://www.npgsql.org/doc/types/datetime.html

Postgres Setup

Remote Access

For accessing the postgres database from outside the host or from a docker container on the host, you must setup the Postgres instance to listen on a network adapter other than localhost.

To do this, locate the postgresql.conf file in: /etc/postgresql/xx/main.
Locate the line with: “listen_addresses”.
Uncomment it, and set it to listen to a particular network address or “*” for all network addresses.
Like this:

```
# - Connection Settings -

listen_addresses = '10.116.0.2'         # what IP address(es) to listen on;
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for all
                                        # (change requires restart)
port = 5432                             # (change requires restart)
max_connections = 100                   # (change requires restart)
#superuser_reserved_connections = 3     # (change requires restart)
unix_socket_directories = '/var/run/postgresql' # comma-separated list of directories
                                        # (change requires restart)
#unix_socket_group = ''                 # (change requires restart)
```

Next, you will need to add an entry to the pg_hba.conf (same folder as previous), for the subnet or host that will gets access to the postgres database.
Adding entries to this file allows incoming access. For example, the following host entry, allows access from the 172.17.0.0/16 subnet. This subnet happens to be the docker subnet on the example host.

```
# "local" is for Unix domain socket connections only
local   all             all                                     md5
# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256
# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            scram-sha-256
host    replication     all             ::1/128                 scram-sha-256
host    all             all             172.17.0.0/16           md5
host    all             all             10.116.0.2/32           md5
```

> NOTE: The above example has two entries added. This is necessary if your remote host is ina different network, such as a docker network. For example, the above two entries say that 10.116.0.2/32 is the source of connections, that originate from a remote network of 172.17.0.0/16.

Once those two config files are updated, you need to restart the postgres intance for the changes to take effect:

```
sudo service postgresql restart
```

You can now test access to the postgres instance, with this command:

```
psql --username=postgresclient --dbname=dbprojectcontrols --host=10.116.0.2 --port=5432
```

PostgreSQL Bulk Insert

Here’s a means to insert multiple records in a single insert call, which is about 40 times faster than performing an explicit insert for each row.

We will leverage the BeginBinaryImport method on the database connection class of the NPGSQL library.

To make this easy, we’ve created a wrapper call in our Postgres_DAL class of: OGA.Postgres.DAL.

The method to use, is called, PerformBulkBinaryImport.

NOTE: Since this method converts all strings to binary, there is no need to escape or sanitize any string data to prevent SQL injection attacks.
As well, this method also preserves format of any JSON structs, without having to escape quotes and apostrophes.

This method accepts two arguments:

- copyargs
- callback

CopyArgs Parameter

CopyArgs is a string that is formatted as a PostgreSQL COPY statement.
It includes the target table name, a list of columns (in parentheses), and how to receive the data.

For example, here’s the CopyArgs string for bulk inserting log messages into a table called, tbl_LogEntry”

```
string copyargs = "COPY public.\"tbl_LogEntry\" " +
                  "(\"time\", level, host, process, service, runtimeid, callsite, " +
                  "threadname, message, exc_type, exc_message, exc_stacktrace) " +
                  "FROM STDIN (FORMAT BINARY)";
```

The above string includes the target table, ‘public.tbl_LogEntry’.
It includes the ordered list of columns to expect.
And, it includes a FROM clause to say how to receive data (from STDIN and in binary format).

Callback Parameter

The second element is the callback.

This parameter must be set to a method, anonymous function, or lambda, that includes logic write the binary output.

Within that parameterized method, you will use the given ‘writer' object, to create rows, and add columns to each one (following the order in the CopyArgs parameter).

The callback should look like this:

```
var callback = Action<sdfsd> (writer) =>
{
  // Start a new row...
  writer.StartRow();
  writer.Write(etime, NpgsqlDbType.TimestampTz);
  writer.Write(m.host, NpgsqlDbType.Varchar);
  ...
  // Start a new row...
  writer.StartRow();
  writer.Write(etime, NpgsqlDbType.TimestampTz);
  writer.Write(m.host, NpgsqlDbType.Varchar);
}
```

Once you have created these two parameters, pass them to the PerformBulkBinaryImport method, and it will perform the bulk insert of your source data, as defined by your writer callback.

Here’s a working example of how to perform a bulk insert of multiple log entries into a log table, using the above method (taken from the PostGres Log Sink Manager Service in OGA.LogSink.Service):

```
// Declare the setup string for the binary importer...
string import_setup = $"COPY public.\"tbl_LogEntry\" (\"time\", level, host, process, service, runtimeid, callsite, threadname, message, exc_type, exc_message, exc_stacktrace) " +
                      "FROM STDIN (FORMAT BINARY)";

// Declare the row digester callback...
var res = this._dal.PerformBulkBinaryImport(import_setup, (writer) =>
{
    // Iterate entries, and digest each one...
    foreach (var m in msg)
    {
        try
        {
            if (m == null)
                continue;

            writer.StartRow();

            if(!DateTime.TryParse(m.time, out var etime))
            {
                etime = DateTime.Now;
            }

            // Set the UTC flag in the received timestamp...
            etime = DateTime.SpecifyKind(etime, DateTimeKind.Utc);

            writer.Write(etime, NpgsqlDbType.TimestampTz);

            writer.Write(m.level, NpgsqlDbType.Varchar);

            writer.Write(m.host, NpgsqlDbType.Varchar);
            writer.Write(m.process, NpgsqlDbType.Varchar);
            writer.Write(m.service, NpgsqlDbType.Varchar);

            writer.Write(m.RuntimeID, NpgsqlDbType.Varchar);
            writer.Write(m.callsite, NpgsqlDbType.Varchar);
            writer.Write(m.threadname, NpgsqlDbType.Varchar);

            writer.Write(m.message, NpgsqlDbType.Varchar);

            writer.Write(m.exception?.type ?? "", NpgsqlDbType.Varchar);
            writer.Write(m.exception?.message ?? "", NpgsqlDbType.Varchar);
            writer.Write(m.exception?.stacktrace ?? "", NpgsqlDbType.Varchar);
        }
        catch(Exception e)
        {
            int x = 0;
        }
    }
});
if(res != 1)
{
    // Failed to store log messages.
```

Postgres Commands

Access Postgres as SuperUser

Use this terminal command to access the local PostgreSQL instance as the postgres user:

```
sudo -u postgres psql
```

Setting Postgres (Superuser) Password

Normally, the postgres user (superuser) does NOT have a set password.
This is because the postgres user normally, is accessing locally, and through sudo.

But if you are wanting the superuser to have a password, you can set it, with this:

```
sudo -u postgres psql
ALTER ROLE postgres WITH PASSWORD 'Your$trongPassword';
\q
```

Now, the postgres user can log in, remotely, if the pg_hba.conf file allows.

To List Database Privileges

```
SELECT * FROM information_schema. table_privileges
where grantee = 'icore_client';
```

Good reference on how to do admin tasks from the command line: https://medium.com/coding-blocks/creating-user-database-and-adding-access-on-postgresql-8bfcd2f4a91e

Basically, this:

```
// Open a command line session with the postgres engine:
sudo -u postgres psql
// Create a database:
create database mydb;
// Create a user:
create user myuser with encrypted password 'mypass';
// Give the user access to the created database:
grant all privileges on database mydb to myuser;
```

To give a user privilege to create databases, use this:

```
ALTER USER postgresclient CREATEDB;
```

To give a user Super User privileges:

```
ALTER USER librarian WITH SUPERUSER;
```

Restoring Database from Command Line

Taken from here: https://simplebackups.com/blog/postgresql-pgdump-and-pgrestore-guide-examples/

Here’s a quick command to restore a database from a backup file:

```
pg_restore -U postgresclient -Ft -C -d dbProjectControls < /path-to-file/backupfilename.tar
```

Login and connect to a database:

```
// Do this to open a "postgres-#" session...
sudo -u postgres psql

// List databases with...
\l

// Switch databases with...
\c databasename
```

Admin Tasks

Good list of admin tasks: https://chartio.com/resources/tutorials/how-to-list-databases-and-tables-in-postgresql-using-psql/

Another good reference to digest: https://hasura.io/blog/top-psql-commands-and-flags-you-need-to-know-postgresql/

HowTo: Install PostGreSQL

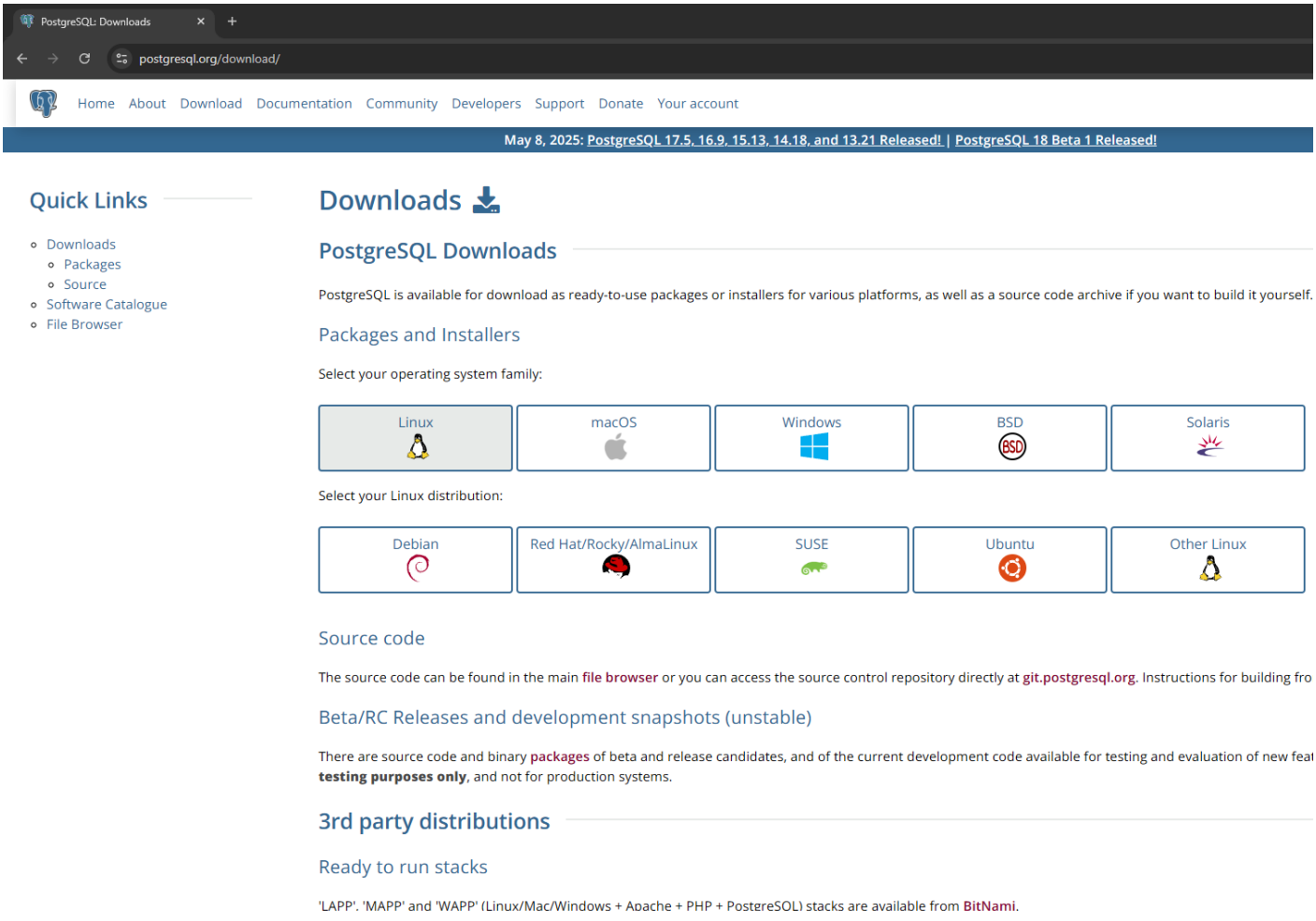
Adapted steps from here:

https://www.c-sharpcorner.com/article/crud-operations-in-postgresql-with-ef-core-and-asp-net-core-web-api/

https://documentation.ubuntu.com/server/how-to/databases/install-postgresql/index.html

Download

PostgreSQL can be downloaded from here: https://www.postgresql.org/download/



> NOTE: We are installing on Ubuntu v24.04 in this tutorial.

Select Linux and Ubuntu from the click boxes.

Install

Run this to install PostgreSQL:

```
sudo apt install postgresql
```

Listener

By default, Postgres only listens on the localhost adapter.
We will change it to listen on all adapters.

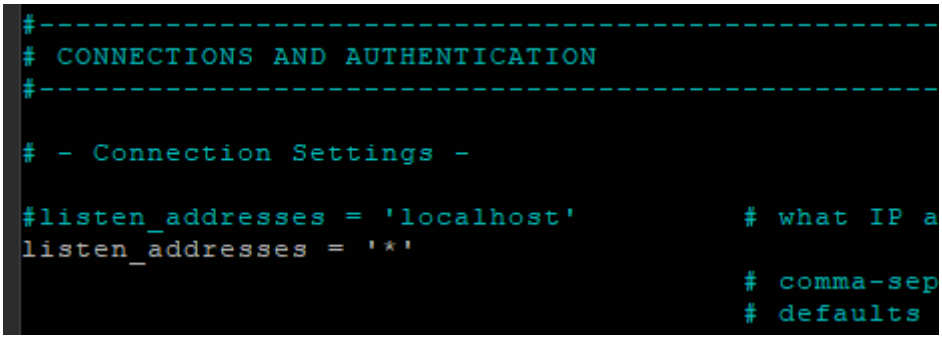
> NOTE: If you only want it to listen on a specific IP, use that.

Navigate to /etc/postgresql/*/main/, and open postgresql.conf.

Change the listen address to '*'. It might not be set, at all.

```
listen_addresses = '*'
```

Like this:



Save the file.

Restart postgres with this:

```
sudo systemctl restart postgresql.service
```

Set Postgres Password

Now, we need to set the password for the postgres user.

Run this in the terminal, to open the default template database:

```
sudo -u postgres psql template1
```

```
glwhite@postgres01:/etc/postgresql/16/main$ sudo -u postgres psql template1
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
Type "help" for help.

template1=#
```

Set the password for the postgres user with this:

```
ALTER ROLE postgres WITH PASSWORD 'Your$trongPassword';
```

Exit psql with 'exit'.

Network Access

With a listener and password set, we need to allow client access from outside the host.

By default, the postgres listener only allows connections from inside the machine.
We need to expand that to the local subnet.

Open the pg_hba.conf file (in /etc/postgresql/*/main/), and allow access from the local subnet.

Find the IPV4 line, that looks like this:

```
host    all             all             127.0.0.1/32            scram-sha-256
```

Change the allowed addresses to your subnet:

```
host    all             all             192.168.60.1/24         scram-sha-256
```

The above allows access to all databases for all users coming from the 60 subnet.

> NOTE: For a production database, your security model may require tighter access.
> If so, you can change the above to specify what users and what databases can access from what addresses.
> Add more than one line if needed, similar to firewall rules.

If you need to provide specific access from multiple subnets or VLANs, you can add entries for each one, like this:

```
host□all□□□□all□□□□192.168.1.0/24□□□md5
host□all□□□□all□□□□192.168.60.0/24□□□md5
host□all□□□□all□□□□192.168.70.0/24□□□md5
```

```
host□all□□□□all□□□□192.168.110.0/24□□md5
host□all□□□□all□□□□192.168.120.0/24□□md5
host□all□□□□all□□□□192.168.150.0/24□□md5
host□all□□□□all□□□□192.168.160.0/24□□md5
```

Once changes have been made, restart the postgres service with:

```
sudo systemctl restart postgresql.service
```

Or, if you are in a query window of pgadmin, you can run this to reload the hba config:

```
SELECT pg_reload_conf()
```

Postgres: Troubleshooting Remote Access

Here's steps to work through, to ensure remote access to a PostgreSQL instance.

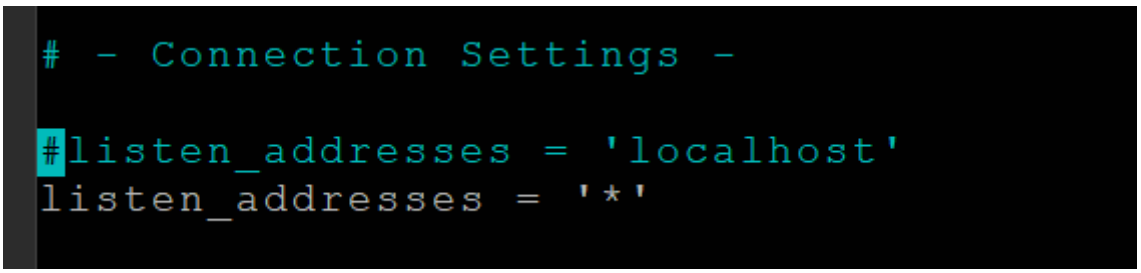
Check PostgreSQL is Listening on All Interfaces

From a terminal, open the postgresql.conf file, with this:

```
sudo nano /etc/postgresql/<version>/main/postgresql.conf
```

> NOTE: Replace <version> with the version you installed. Will be '16' as of this writing.

Locate the line with 'listen_addresses' and ensure it is set to listen on all adapters.
Or, to listen on a specific one for the host.



It is commented out, by default.
So, you will need to open up the listener to at least one adapter, or, all, like above.

Save and close the config file, and restart the postgres instance with this:

```
sudo systemctl restart postgresql
```

Allowed Source Subnets

Open the pg_hba.conf file, to verify allowed source subnets and protocols, with this:

```
sudo nano /etc/postgresql/<version>/main/pg_hba.conf
```

> NOTE: Same as before, replace <version> with the installed version. '16' as of this writing.

Make sure the allowed sources includes the subnet of your source hosts.

For this, you may need to add lines for each allowed subnet, like this:

```
host    all             all             192.168.1.0/24          md5
```

```
# DO NOT DISABLE!
# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local   all             postgres                                peer

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256
# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            scram-sha-256
host    replication     all             ::1/128                 scram-sha-256

host    all             all             192.168.1.0/24          md5
host    all             all             192.168.60.0/24         md5
host    all             all             192.168.70.0/24         md5
host    all             all             192.168.110.0/24        md5
host    all             all             192.168.120.0/24        md5
host    all             all             192.168.150.0/24        md5
host    all             all             192.168.160.0/24        md5
```

The above example allows access from several subnets.

Save and close the config.

And, reload it, with this:

```
sudo systemctl reload postgresql
```

Ensure Correct Listening Port

Run this on the Postgres host, to see what ports the database engine is listening on:

```
sudo ss -tnlp | grep 5432
```

You will see something like this for a default install:

```
glwhite@postgres01:/etc/postgresql/16/main$ sudo ss -tnlp | grep 5432
LISTEN 0      200          0.0.0.0:5432       0.0.0.0:*    users:(("postgres",pid=208491,fd=6))
LISTEN 0      200             [::]:5432          [::]:*    users:(("postgres",pid=208491,fd=7))
glwhite@postgres01:/etc/postgresql/16/main$
```

NOTE: By default, the engine listens on 5432.

Firewall Ingress Rule

Make sure the host firewall allows incoming connections, with this:

```
sudo ufw allow 5432/tcp
```

Test from Remote Client

Now, verify connectivity from your remote client.