

# Advanced Database

COMP412

CHAPTER:02 DATABASE ADMINISTRATION



# Introduction

---

- This chapter covers what we consider basic administration of a PostgreSQL server:
  - Configurations
  - Managing roles and permissions,
  - Creating databases,
  - Backing up and restoring data.

# Configuration Files

---

CONTROL OPERATIONS OF A POSTGRESQL SERVER

# postgresql.conf

---

- Controls general settings, such as:
- memory allocation,
- default storage location for new databases,
- the IP addresses that PostgreSQL listens on,
- location of logs,
- ...

# pg\_hba.conf

---

- Controls access to the server
- dictating which users can log in to which databases
- which IP addresses can connect,
- which authentication scheme to accept
- ...

# pg\_ident.conf

---

- If present, this file maps an authenticated OS login to a PostgreSQL user.
- People sometimes map the OS root account to the PostgreSQL superuser account, postgres.

# Location of configuration files

---

- Query as a superuser while connected to any database.
  - `SELECT name, setting FROM pg_settings WHERE category = 'File Locations';`

name	setting
config_file	/etc/postgresql/9.6/main/postgresql.conf
data_directory	/var/lib/postgresql/9.6/main
external_pid_file	/var/run/postgresql/9.6-main.pid
hba_file	/etc/postgresql/9.6/main/pg_hba.conf
ident_file	/etc/postgresql/9.6/main/pg_ident.conf
(5 rows)	

# Making Configurations Take Effect

---

## ○ Reloading

- `pg_ctl reload -D your_data_directory_here`
- `SELECT pg_reload_conf();`

## ○ Restarting

- `service postgresql-9.6 restart`
- `pg_ctl restart -D your_data_directory_here` (For any PostgreSQL instance not installed as a service)
- Rem: use “`show data_directory`” to get data directory of current instance

# The postgresql.conf File

---

- Changing the postgresql.conf settings
  - using the ALTER SYSTEM SQL command.

```
ALTER SYSTEM SET work_mem = '500MB';
```

“I edited my postgresql.conf and now my server won’t start.”

- The easiest way to figure out what you screwed up is to look at the logfile, open the latest file and read what the last line says. The error raised is usually self-explanatory.

# The postgresql.conf File

---

- Instead of editing **postgresql.conf** directly, you should override settings using an additional file called **post-gresql.auto.conf**.
- Checking postgresql.conf settings without opening the configuration files is to query the view named pg\_settings.

```
SELECT name, context , unit , setting, boot_val, reset_val
FROM pg_settings
WHERE name IN ('listen_addresses','deadlock_timeout','shared_buffers',
'effective_cache_size','work_mem','maintenance_work_mem')
ORDER BY context, name;
```

# The postgresql.conf File

name	context	unit	setting	boot_val	reset_val
-----+-----+-----+-----+-----+-----					
listen_addresses	postmaster		*	localhost	*
deadlock_timeout	superuser	ms	1000	1000	1000
effective_cache_size	user	8kB	16384	16384	16384

- The context is the scope of the setting. Some settings have a wider effect than others, depending on their context.
- Postmaster settings affect the entire server (postmaster represents the PostgreSQL service) and take effect only after a restart.
- If set by the superuser, the setting becomes a default for all users

# The postgresql.conf File

---

- The pg\_file\_settings can be used also to query settings.

```
SELECT name, sourcefile, sourceline, setting, applied
FROM pg_file_settings
WHERE name IN ('listen_addresses','deadlock_timeout','shared_buffers',
'effective_cache_size','work_mem','maintenance_work_mem')
ORDER BY name;
```

name	sourcefile	sourceline	setting	applied
effective_cache_size	E:/data96/postgresql.auto.conf	11	8GB	t
listen_addresses	E:/data96/postgresql.conf	59	*	t

- incorrect changing in postgresql.conf will prevent clients from connecting.

# The postgresql.conf File

---

- Changing values requires a service restart:
  - **listen\_addresses**: Informs PostgreSQL which IP addresses to listen on.
  - **Port**: Defaults to 5432. You may wish to change for security or if you are running multiple PostgreSQL services on the same server.
  - **max\_connections**: The maximum number of concurrent connections allowed.
  - **log\_destination**: specifies the format of the logfiles rather than their physical location.

# The postgresql.conf File

---

- The following settings affect performance.
  - **shared\_buffers**: Allocated amount of memory shared among all connections to store recently accessed pages.
  - **effective\_cache\_size**: An estimate of how much memory PostgreSQL expects the operating system to devote to it. This setting has no effect on actual allocation, but the query planner figures in this setting to guess whether intermediate steps and query output would fit in RAM. If you set this much lower than available RAM, the planner may forgo using indexes.

# The postgresql.conf File

---

- **work\_mem:** Controls the maximum amount of memory allocated for each operation such as sorting, hash join, and table scans. If you have many users running simple queries, you want this setting to be relatively low to be democratic; otherwise, the first user may hog all the memory.
- **maintenance\_work\_mem:** The total memory allocated for housekeeping activities such as vacuuming (pruning records marked for deletion).
- **max\_parallel\_workers\_per\_gather:** This is a new setting introduced in 9.6 for parallelism. The setting determines the maximum parallel worker threads that can be spawned for each gather operation. The default setting is 0, which means parallelism is completely turned off. If you have more than one CPU core, you will want to elevate this.

# The pg\_hba.conf File

---

- The pg\_hba.conf file controls which IP addresses and users can connect to the database.
- Furthermore, it dictates the authentication protocol that the client must follow

#	TYPE	DATABASE	USER	ADDRESS	METHOD
host	all	all	127.0.0.1/32	ident ❶	
host	all	all	:::1/128	trust ❷	
host	all	all	192.168.54.0/24	md5 ❸	
hostssl	all	all	0.0.0.0/0	md5 ❹	

- 1) Authentication method. The usual choices are ident, trust, md5, peer, and password.
- 2) IPv6 syntax for defining network range.
- 3) IPv4 syntax for defining network range.
- 4) SSL connection rule. In our example, we allow anyone to connect to our server outside of the allowed IP range as long as they can connect using SSL.

# The pg\_hba.conf File

---

- This entry allows any IP address to connect to any database with any username using the "md5" authentication method.
- Note that using "0.0.0.0/0" as the address allows connections from any IP address, which can be a potential security risk.
  - # TYPE DATABASE USER ADDRESS METHOD
  - host all all 0.0.0.0/0 md5
- It's generally recommended to restrict access to specific IP addresses or ranges whenever possible.

# Managing Connections

---

# Managing Connections

---

- Retrieve a listing of recent connections and process IDs (PIDs)

```
SELECT * FROM pg_stat_activity;
```

- Cancel active queries on a connection with PID 1234:

```
SELECT pg_cancel_backend(1234);
```

- This does not terminate the connection itself, though.

- Terminate the connection:

```
SELECT pg_terminate_backend(1234);
```

- Kill all connections belonging to a role with a single blow

```
SELECT pg_terminate_backend(pid) FROM pg_stat_activity  
WHERE username = 'some_role'
```

# Managing Connections

---

- You can set certain operational parameters at the server, database, user, session, or function level. Any queries that exceed the parameter will automatically be cancelled by the server. Setting a parameter to 0 disables the parameter:
  - **deadlock\_timeout**: This is the amount of time a deadlocked query should wait before giving up. This defaults to 1000 ms.
  - **statement\_timeout**: This is the amount of time a query can run before it is forced to cancel. This defaults to 0, meaning no time limit.
  - **lock\_timeout**: This is the amount of time a query should wait for a lock before giving up, and is most applicable to update queries. The default is 0, meaning that the query will wait infinitely

# Roles

---

# Roles

---

- PostgreSQL handles credentialing using roles.
- Roles that can log in are called login roles.
- Roles can also be members of other roles; the roles that contain other roles are called group roles.
  - for security, group roles generally cannot log in
- CREATE USER and CREATE GROUP still work in current versions, but shun them and use CREATE ROLE instead.

# Creating Login Roles

---

- pgAdmin has a graphical section for creating user roles, but if you want to create one using SQL, execute an SQL command.
  - `CREATE ROLE leo LOGIN PASSWORD 'king' VALID UNTIL 'infinity' CREATEDB;`
    - Specifying VALID UNTIL is optional. If omitted, the role remains active indefinitely.
    - CREATEDB grants database creation privilege to the new role.
  - `CREATE ROLE regina LOGIN PASSWORD 'queen' VALID UNTIL '2020-1-1 00:00' SUPERUSER;`
    - `CREATE ROLE regina LOGIN PASSWORD 'queen' VALID UNTIL '2020-1-1 00:00' SUPERUSER;`
      - To create a user with superuser privileges.

# Creating Group Roles

---

- Group roles generally cannot log in
  - `CREATE ROLE royalty INHERIT;`
    - INHERIT means that any member of royalty will automatically inherit privileges of the royalty role.
    - except for the superuser privilege
- To add members to a group role, you would do:
  - `GRANT royalty TO leo;`
  - `GRANT royalty TO regina;`
- Let's give the royalty role superuser rights with the command:
  - `ALTER ROLE royalty SUPERUSER;`
  - leo can gain superuser rights by doing:
    - `SET ROLE royalty;`

# Creating Group Roles

---

- Group roles generally cannot log in
  - `CREATE ROLE royalty INHERIT;`
    - INHERIT means that any member of royalty will automatically inherit privileges of the royalty role.
- To add members to a group role, you would do:
  - `GRANT royalty TO leo;`
  - `GRANT royalty TO regina;`
- Let's give the royalty role superuser rights with the command:
  - `ALTER ROLE royalty SUPERUSER;`
- leo is still will not have superuser rights. He can gain superuser rights by doing:
  - `SET ROLE royalty;`

# Creating Group Roles

---

- Example 2-7. SET ROLE and SET AUTHORIZATION

# Database Creation

---

# Database Creation

---

- The minimum SQL command to create a database is:  
`CREATE DATABASE mydb;`
- This creates a copy of the template1 database. Any role with `CREATEDB` privilege can create new databases.
- The basic syntax to create a database modeled after a specific template is:
  - `CREATE DATABASE my_db TEMPLATE my_template_db;`

# Template Databases

---

- A template database serves as a skeleton for new databases.
- PostgreSQL copies all the database settings and data from the template database to the new database.
- The default PostgreSQL installation comes with two template databases: template0 and template1.
- template1 is used as default

# Using Schemas

---

- Schemas organize your database into logical groups.
- `CREATE SCHEMA my_schema;`
- Take advantage of the default search path set in `postgresql.conf` to search in same schema as login roles  
    `set search_path = "$user", public;`

# Privileges

---

# Privileges

---

- A privilege (often called permissions) is **a right to execute a particular type of SQL statement or to access another user's object.**
- Privileges can bore down to the column and row level.
- PostgreSQL has a few dozen privileges
  - The more mundane privileges are SELECT, INSERT, UPDATE, ALTER, EXECUTE, DELETE, and TRUNCATE
- Most privileges must have a context.
  - A role having an ALTER privilege is meaningless unless qualified with a database object such as ALTER privilege on tables1, SELECT privilege on table2, EXECUTE privilege on function1, and so on.
  - Not all privileges apply to all objects: an EXECUTE privilege for a table is nonsense.
- Some privileges make sense without a context.
  - CREATEDB and CREATE ROLE are two privileges where context is irrelevant

# Getting Started

---

1. PostgreSQL creates one superuser and one database for you at installation, both named postgres. Log in to your server as postgres.
2. Before creating your first database, create a role that will own the database and can log in, such as:  

```
CREATE ROLE mydb_admin LOGIN PASSWORD 'something';
```
3. Create the database and set the owner:  

```
CREATE DATABASE mydb WITH owner = mydb_admin;
```
4. Now log in as the mydb\_admin user and start setting up additional schemas and tables.

# GRANT

---

- The GRANT command is the primary means to assign privileges. Basic usage is: *GRANT some\_privilege TO some\_role;*
- A few things to keep in mind when it comes to GRANT:
  - You need to have the privilege you're granting. And, you must have the GRANT privilege yourself.
  - Some privileges always remain with the owner of an object and can never be granted away. These include DROP and ALTER.
  - The owner of an object retains all privileges. That ownership does not drill down to child objects.

# GRANT

---

- A few things to keep in mind when it comes to GRANT:
  - When granting privileges, you can add `WITH GRANT OPTION`. This means that the grantee can grant her own privileges to others, passing them on:
    - `GRANT ALL ON ALL TABLES IN SCHEMA public TO mydb_admin WITH GRANT OPTION;`
    - `ALTER role mydb_admin WITH CREATEROLE`
  - To grant specific privileges on ALL objects of a specific type use `ALL` instead of the specific object name, as in:
    - `GRANT SELECT, REFERENCES, TRIGGER ON ALL TABLES IN SCHEMA my_schema TO PUBLIC;`
  - To grant privileges to all roles, you can use the `PUBLIC` alias, as in:
    - `GRANT USAGE ON SCHEMA my_schema TO PUBLIC;`

# REVOKE

---

- In many cases you might consider revoking some of the defaults with the REVOKE command, as in:

```
REVOKE EXECUTE ON ALL FUNCTIONS IN SCHEMA my_schema FROM PUBLIC;
```

```
REVOKE privilege | ALL  
ON TABLE table_name | ALL TABLES IN SCHEMA schema_name  
FROM role_name;
```

# Backup and Restore

---

RECOVER YOUR DATA FROM MANY FAILURES

# Selective Backup Using pg\_dump

---

- **pg\_dump** can selectively back up tables, schemas, and databases.
- **pg\_dump** can backup to plain SQL, as well as compressed, TAR, and directory formats.
  - **Compressed, TAR, and directory** format backups can take advantage of the parallel restore feature of **pg\_restore**.
  - **Directory** backups allow parallel **pg\_dump** of a large database.
  - <https://www.postgresql.org/docs/9.6/app-pgdump.html>

# Selective Backup Using pg\_dump

---

- To create a compressed, single database backup:
  - `pg_dump` -h localhost -p 5432 -U someuser -F c -v -f mydb.backup mydb
  - -v: This will cause pg\_dump to output detailed object comments
- To create a plain-text single database backup:
  - `pg_dump` -h localhost -p 5432 -U someuser -F p -v -f mydb.backup mydb
- To create a compressed backup of tables whose names start with pay in any schema:
  - `pg_dump` -h localhost -p 5432 -U someuser -F c -v -t \*.pay\* -f pay.backup mydb

# Selective Backup Using pg\_dump

---

- To create a plain-text SQL backup of select tables, useful for porting structure and data to lower versions of PostgreSQL or non-PostgreSQL databases
  - plain text generates an SQL script that you can run on any system that speaks SQL:
  - `pg_dump -h localhost -p 5432 -U someuser -F p --column-inserts -f select_tables.backup mydb`

# Restoring Data

---

- There are two ways to restore data in PostgreSQL from backups created with `pg_dump`
  - Use `psql` to restore plain-text backups generated with `pg_dump`.
  - Use `pg_restore` to restore compressed, TAR, and directory backups created with `pg_dump`.

# Using psql to restore plain-text SQL backups

---

- To restore a backup and ignore errors:
  - `psql -U postgres -f myglobals.sql`
- To restore to a specific database:
  - `psql -U postgres -d mydb -f select_objects.sql`

# Using pg\_restore

---

- To perform a restore using pg\_restore, first create the database using SQL:
  - `CREATE DATABASE mydb;`
- Then restore:
  - `pg_restore --dbname=mydb --jobs=4 --verbose mydb.backup`
- If the name of the database is the same as the one you backed up, you can create and restore the database in one step:
  - `pg_restore --dbname=postgres --create --jobs=4 --verbose mydb.backup`
- You can perform parallel restores using the `-j` (equivalent to `--jobs=`) option to indicate the number of threads to use

# Tablespaces

---

# Tablespaces

---

- PostgreSQL uses tablespaces to ascribe logical names to physical locations on disk.
- To create a new tablespace, specify a logical name and a physical folder
  - `CREATE TABLESPACE secondary LOCATION 'C:/pgdata94_secondary';`
- To move all objects in the database to your secondary tablespace, issue the following SQL command:
  - `ALTER DATABASE mydb SET TABLESPACE secondary;`