PostgreSQL parallel backups with pg_dump

Backing up your PostgreSQL database efficiently is very important, especially for large databases. One way to speed up backups is by using...

Backing up your PostgreSQL database efficiently is very important, especially for large databases. One way to speed up backups is by using pg_dump in directory format with parallel jobs. This method stores each table in a separate file and allows multiple tables to be dumped simultaneously.

1. Checking your data directory

Before starting, let's check where our PostgreSQL data is stored. Typically, the data directory contains multiple subdirectories and configuration files like:

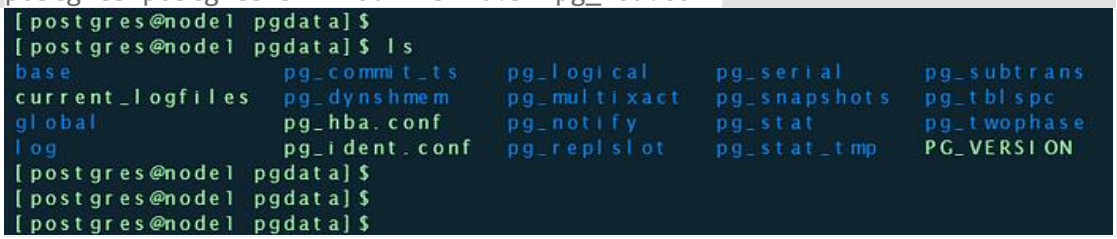
```
[postgres@node1 pgdata]$ ls -rlt /pgdata/

total 72

drwx------ 8 postgres postgres 76 Sep 19 11:28 base
drwx------ 2 postgres postgres 4096 Sep 19 11:30 global
drwx------ 3 postgres postgres 4096 Sep 19 11:31 pg_wal-rw------- 1

postgres postgres 29671 Jul 15 11:01 postgresql.conf-rw------- 1


postgres postgres 5779 Jul 15 10:34 pg_hba.conf
```



- 'base/' — contains the actual database files
- 'pg_wal/' — transaction logs (Write-Ahead Logs)
- 'global/' — cluster-wide metadata

- Configuration files like 'postgresql.conf', 'pg_hba.conf', and 'PG_VERSION'

2. Creating a parallel directory ackup

To create a backup in directory format with parallel jobs, use the -Fd option with the -j option.

```
Copy[postgres@node1 pgdata]$
[postgres@node1 pgdata]$ ls -rlt /pgbackup/
total 0
[postgres@node1 pgdata]$
[postgres@node1 pgdata]$
[postgres@node1 pgdata]$
[postgres@node1 pgdata]$ pg_dump -U postgres -Fd -j 4 -f /pgbackup/
postgres -v

pg_dump: last built-in OID is 16383
pg_dump: reading extensions
pg_dump: identifying extension members
pg_dump: reading schemas
pg_dump: reading user-defined tables
pg_dump: reading user-defined functions
pg_dump: reading user-defined types
pg_dump: reading procedural languages
pg_dump: reading user-defined aggregate functions
pg_dump: reading user-defined operators
pg_dump: reading user-defined access methods
pg_dump: reading user-defined operator classes
pg_dump: reading user-defined operator families
pg_dump: reading user-defined text search parsers
pg_dump: reading user-defined text search templates
pg_dump: reading user-defined text search dictionaries
pg_dump: reading user-defined text search configurations
pg_dump: reading user-defined foreign-data wrappers
pg_dump: reading user-defined foreign servers
pg_dump: reading default privileges
pg_dump: reading user-defined collations
pg_dump: reading user-defined conversions
pg_dump: reading type casts
pg_dump: reading transforms
pg_dump: reading table inheritance information
pg_dump: reading event triggers
pg_dump: finding extension tables
pg_dump: finding inheritance relationships
pg_dump: reading column info for interesting tables
pg_dump: finding table default expressions
pg_dump: flagging inherited columns in subtables
pg_dump: reading partitioning data
pg_dump: reading indexes
pg_dump: flagging indexes in partitioned tables
pg_dump: reading extended statistics
pg_dump: reading constraints
```

```
pg_dump: reading triggers
pg_dump: reading rewrite rules
pg_dump: reading policies
pg_dump: reading row-level security policies
pg_dump: reading publications
pg_dump: reading publication membership of tables
pg_dump: reading publication membership of schemas
pg_dump: reading subscriptions
pg_dump: reading large objects
pg_dump: reading dependency data
pg_dump: saving encoding = UTF8
pg_dump: saving standard_conforming_strings = on
pg_dump: saving search_path =
pg_dump: saving database definition
pg_dump: dumping contents of table "public.all_university"
pg_dump: dumping contents of table "public.olympic_history"
pg_dump: finished item 4345 TABLE DATA all_university
pg_dump: dumping contents of table "public.google_employees"
pg_dump: finished item 4347 TABLE DATA google_employees
pg_dump: dumping contents of table "public.test_unique"
pg_dump: finished item 4350 TABLE DATA test_unique
pg_dump: dumping contents of table "public.products"
pg_dump: finished item 4355 TABLE DATA products
pg_dump: dumping contents of table "public.groups"
pg_dump: finished item 4357 TABLE DATA groups
pg_dump: dumping contents of table "public.students"
pg_dump: finished item 4359 TABLE DATA students
pg_dump: dumping contents of table "public.netflix_shows"
pg_dump: dumping contents of table "public.fifa_players"
pg_dump: finished item 4348 TABLE DATA fifa_players
pg_dump: finished item 4351 TABLE DATA netflix_shows
pg_dump: finished item 4352 TABLE DATA olympic_history
[postgres@node1 pgdata]$
```

pg_dump options:

- -U postgres → the PostgreSQL user - -Fd → directory format (stores each table as a separate file) - -j 4 → use 4 parallel jobs to speed up the backup - -f /pgbackup/ → backup directory - postgres → the database name

This will create /pgbackup/ containing files for all tables and other database objects. Parallel jobs make the backup faster for large databases.

```
[postgres@node1 pgdata]$
[postgres@node1 pgdata]$ ls -rlt /pgbackup/
total 7216
-rw-r--r-- 1 postgres postgres    16265 Sep 26 14:16 toc.dat
-rw-r--r-- 1 postgres postgres   165844 Sep 26 14:16 4345.dat.gz
-rw-r--r-- 1 postgres postgres    29388 Sep 26 14:16 4347.dat.gz
-rw-r--r-- 1 postgres postgres       75 Sep 26 14:16 4359.dat.gz
-rw-r--r-- 1 postgres postgres       63 Sep 26 14:16 4357.dat.gz
-rw-r--r-- 1 postgres postgres       76 Sep 26 14:16 4355.dat.gz
-rw-r--r-- 1 postgres postgres       43 Sep 26 14:16 4350.dat.gz
-rw-r--r-- 1 postgres postgres   375325 Sep 26 14:16 4348.dat.gz
-rw-r--r-- 1 postgres postgres  1369421 Sep 26 14:16 4351.dat.gz
-rw-r--r-- 1 postgres postgres  5403069 Sep 26 14:16 4352.dat.gz
[postgres@node1 pgdata]$
[postgres@node1 pgdata]$
```

2. Restoring from a Parallel Directory Backup

Count postgres DB object list:

```
SELECT
    CASE UPPER(relkind)
        WHEN 'R' THEN 'TABLE'
        WHEN 'P' THEN 'PARTITION TABLE'
        WHEN 'I' THEN 'INDEX'
        WHEN 'S' THEN 'SEQUENCE'
        WHEN 'V' THEN 'VIEW'
        WHEN 'M' THEN 'MATERIALIZED VIEW'
        WHEN 'F' THEN 'FOREIGN TABLE'
        WHEN 'T' THEN 'TOAST TABLE'
        ELSE UPPER(relkind)
    END AS object_type,
    COUNT(*) AS count
FROM pg_class c
JOIN pg_namespace n ON n.oid = c.relnamespace
WHERE n.nspname NOT IN ('pg_catalog', 'information_schema')
GROUP BY UPPER(relkind)
ORDER BY object_type;

object_type      | count
---------------- + ------
INDEX            |    52
SEQUENCE         |     6
TABLE            |     9
VIEW             |     3
T                |    45   <-- this means some relkind not mapped in CASE
 Notice in your output, there is a row T.
That means your query encountered an unexpected relkind = 'T'.
In PostgreSQL, 't' = TOAST table (internal storage for large values).
Since your CASE statement doesn’t map 'T', it just shows T.
```

```
 object_type | count
-------------+-------
 INDEX       |    52
 SEQUENCE    |     6
 T           |    45
 TABLE       |     9
 VIEW        |     3
(5 rows)

postgres=#
```

Dropping Existing Database

If the target database already exists and you want a full restore, first drop it and create a new one:

```
Copypostgres=#
postgres=#
postgres=#
postgres=# \c dvdrental
You are now connected to database "dvdrental" as user "postgres".

dvdrental=#

dvdrental=# DROP DATABASE  postgres ;

DROP DATABASE

dvdrental=#
dvdrental=#
dvdrental=# CREATE DATABASE postgres;
CREATE DATABASE

dvdrental=#

dvdrental=#
```

Restore using pg_restore

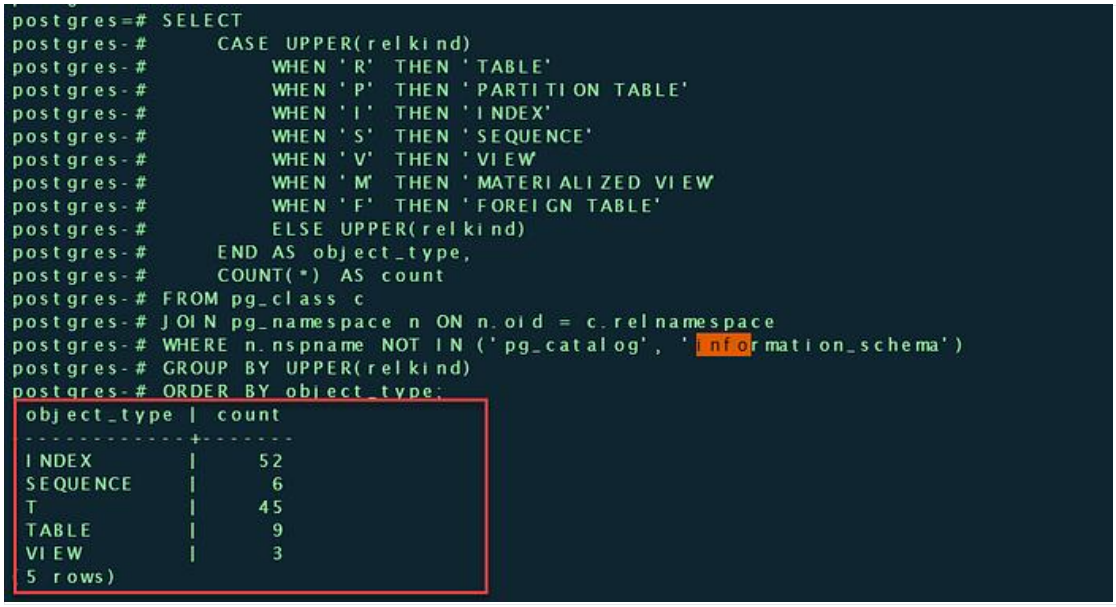
```
Copy[postgres@node1 ~]$
[postgres@node1 ~]$
[postgres@node1 ~]$ pg_restore -U postgres -d postgres -v  -j 4
/pgbackup/

pg_restore: connecting to database for restore
pg_restore: processing item 4362 ENCODING ENCODING
pg_restore: processing item 4363 STDSTRINGS STDSTRINGS
pg_restore: processing item 4364 SEARCHPATH SEARCHPATH
pg_restore: processing item 4365 DATABASE postgres
pg_restore: processing item 2 EXTENSION pg_buffercache
pg_restore: creating EXTENSION "pg_buffercache"
pg_restore: processing item 4366 COMMENT EXTENSION pg_buffercache
pg_restore: creating COMMENT "EXTENSION pg_buffercache"
pg_restore: processing item 3 EXTENSION pg_stat_statements
pg_restore: creating EXTENSION "pg_stat_statements"
```

Count postgres DB object list:

```
SELECT
    CASE UPPER(relkind)
        WHEN 'R' THEN 'TABLE'
        WHEN 'P' THEN 'PARTITION TABLE'
        WHEN 'I' THEN 'INDEX'
        WHEN 'S' THEN 'SEQUENCE'
        WHEN 'V' THEN 'VIEW'
```

```
        WHEN 'M' THEN 'MATERIALIZED VIEW'
        WHEN 'F' THEN 'FOREIGN TABLE'
        WHEN 'T' THEN 'TOAST TABLE'
        ELSE UPPER(relkind)
    END AS object_type,
    COUNT(*) AS count
FROM pg_class c
JOIN pg_namespace n ON n.oid = c.relnamespace
WHERE n.nspname NOT IN ('pg_catalog', 'information_schema')
GROUP BY UPPER(relkind)
ORDER BY object_type;
```



Backup tips

- Directory format with parallel jobs is recommended for large databases. - Always test your backup by restoring it to a separate database. - Keep multiple copies of backups in different locations. -

Use pg_dumpall if you need to backup global objects like roles or tablespaces.

Conclusion

Using pg_dump in directory format with parallel jobs is fast, safe, and flexible. It allows large databases to be backed up and restored efficiently. Always plan your backups, verify them regularly, and take advantage of parallelism to save time.