PostgreSQL Backup & Restore Basics

Fanavaran Anisa
Iran Linux House

Linux & Open Source Training Center

www.anisa.co.ir

Some Questions

```
UPDATE media
SET created_at = crawl_date
WHERE
  crawl_date IS NOT NULL
 AND created_at IS NULL
LIMIT 10000;
```

Why it shows Error? How to correct it ?

Some Questions

```
WITH limited_media AS (
  SELECT id
  FROM media
  WHERE crawl_date IS NOT NULL
    AND created_at IS NULL
  LIMIT 10000
)
UPDATE media
SET created_at = crawl_date
WHERE id IN (SELECT id FROM limited_media);
```

What Does this Code Do?

```
UPDATE media
SET post_date = (
  EXTRACT(EPOCH FROM (
    CURRENT_DATE - (INTERVAL '6 months') +
    (random() * (INTERVAL '2 months'))
  ))::bigint
)
WHERE post_date = 0;
```

Is this Correct?

```
WITH records_to_update AS (
  SELECT m.id
  FROM media m
  JOIN posts p ON m.post_id = p.id
  WHERE m.created_at IS NULL
    AND p.post_date != 0
  LIMIT 10000
)
UPDATE media m
SET created_at = to_timestamp(p.post_date)
FROM posts p
WHERE m.post_id = p.id
  AND m.id IN (SELECT id FROM records_to_update);
```

3 Basic Backup Methods

There are three fundamentally different approaches to backing up PostgreSQL data:

- SQL dump
- File system level backup
- Continuous archiving

https://www.postgresql.org/docs/current/backup.html

pg_dump/pg_restore

Logical Backup

SQL Dump

The idea behind this dump method is **to generate a file with SQL commands** that, when fed back to the server, will recreate the database in the same state as it was at the time of the dump.

pg_dump dbname > dumpfile

pg_dump writes its result to the standard output

Syntax:

pg_dump [connection_option...] [option...] dbname

PG Dump Main Options

Option	Description
-F, --format	Specify the output file format (**c** for custom, **d** for directory, **t** for tar, **p** for plain text). *custom format is a compressed binary format*
-f, --file	Specify the file to write the dump to.
-h, --host	Database server host or socket directory.
-p, --port	Database server port number.
-U, --username	User name to connect as.
--no-password	Never prompt for a password.
--schema	Dump only the named schema, or none to exclude schemas.
--table	Dump only the named table, or none to exclude tables.

Some Practical Samples

```
pg_dump -U [your_username] -h [your_host] -d [your_database]

pg_dump -U [your_username] -h [your_host] -d [your_database]
--table orders -F c -f northwind_orders_table.custom

pg_dump -U [your_username] -h [your_host] -d [your_database]
--format p --file northwind.txt  --exclude-table=employees

pg_dump -U [your_username] -h [your_host] -d [your_database]
--schema-only -F c -f northwind_schema_only.custom
```

Pg_Dump notes

- Like any other PostgreSQL client application, pg_dump will by default **connect with the database user name that is equal to the current operating system user name**. To override this, either specify the -U option or set the environment variable PGUSER.
- Ensure that the **user** running pg_dump **has the necessary permissions** to access the database and perform the dump operation.
- Depending on your needs, you might want to use **--schema-only** or **--data-only** options to dump only the database schema or data, respectively.
- Use **--exclude-table** or **--exclude-table-data** options to exclude specific tables or table data from the dump.

Pg_Dump notes

- Dumps will usually not block other operations, but they can be **long-running**
- Because of MVCC, long running backups may cause Postgres to experience **performance degradation until the dump completes** (all records-dead or alive- have to be exist until dump is completes).
- pg_dump as a **corruption check** : pg_dump sequentially scans through the entire data set as it creates the file. Reading the entire database is a rudimentary corruption check for all the table data, but **not for indexes**. If your data is corrupted, pg_dump will throw an exception
- **Indexes are not stored in the SQL dump**. Only the CREATE INDEX command is stored and indexes must be rebuilt when restoring from a logical backup.

Pg_Dump notes

- For large databases, you can use the --jobs (-j) option to enable **parallel dump operations**, which can speed up the process. This can significantly speed up the backup process for large databases by **leveraging multiple CPU cores.**
  - Parallel dumps are only supported for the "directory" archive format.
- The **--no-sync** option can be used to skip the synchronization of data files before the actual dump. While synchronization ensures a more consistent state, **it can be time-consuming, especially for large databases**. Skipping synchronization can speed up the dump process, but it might result in a less consistent backup.

When to use pg_dump?

- For version upgrades and migrations
- Ensure that the **user** running pg_dump **has the necessary permissions** to access the database and perform the dump operation.
- **pg_dump** can be configured to back up specific database objects and ignore others.
- Postgres dumps are also internally consistent, which means the **dump represents a snapshot of the database at the time the process started**. Dumps will usually not block other operations, but they can be long-running

pg_dumpall

- pg_dump dumps only a single database at a time, and it does **not dump information about roles or tablespaces** (because those are cluster-wide rather than per-database).
- **pg_dumpall backs up each database in a given cluster**, and also **preserves cluster-wide data such as role and tablespace definitions**. The basic usage of this command is:

```
pg_dumpall > dumpfile
```

- The resulting dump can be restored with psql:

```
psql -f dumpfile postgres
```

- The **postgres** specifies the database to connect to. In this case, it's the default PostgreSQL database.

pg_dumpall

- pg_dumpall works by **emitting commands to re-create roles, tablespaces**, and empty databases, **then invoking pg_dump for each database**. This means that while each database will be internally consistent, the snapshots of different databases are not synchronized.

- Cluster-wide data can be dumped alone using the **pg_dumpall --globals-only** option(roles and tablespaces). This is necessary to fully backup the cluster if running the pg_dump command on individual databases..

pg_dumpall

Option	Description
-h, --host	Database server host or socket directory.
-p, --port	Database server port number.
-U, --username	User name to connect as.
--no-password	Never prompt for a password.
--file	Output file for the dump.
--clean	Clean (drop) databases before recreating them.
--globals-only	dump only global objects, no databases
--encoding	Dump encoding for the database.
--exclude-database	Exclude the specified database from the dump.
--roles-only	Dump only roles, no databases.
--schema-only	Dump only the schema, no data.
--data-only	Dump only data, no schema.
--no-sync	Do not synchronize system catalogs with data.
--verbose	Produce more detailed output.
--inserts	Use INSERT statements instead of COPY during the dump generation . pg_dump will generally default to using the COPY command for efficiency reasons, especially when dealing with large datasets.
--if-exists	Use conditional commands (IF EXISTS) in the dump.

No Format Options! No -j

Restoring the Dump

- The general command form to restore a dump is

  psql dbname [--set ON_ERROR_STOP=on] < dumpfile

  or

  psql dbname [--set ON_ERROR_STOP=on] –f dumpfile

- Use the **-1 or --single-transaction** command-line options to specify that the whole dump should be restored as a single transaction

- Use Pipes as an transfer medium

  pg_dump -h host1 dbname | **psql** -h host2 dbname

pg_restore

pg_restore is a PostgreSQL utility used to restore a PostgreSQL database from an archive created by **pg_dump**

```
pg_restore [connection_option...] [option...] [filename]
```

Option	Description
--list	Display the contents of the archive file.
-v, --verbose	Produce more detailed output.
--no-privileges	Do not restore access privileges.

pg_restore

Option	Description
-c, --clean	Clean (drop) database objects before recreating them.
-C, --create	Create the database before restoring into it.
--if-exists	Use conditional commands (IF EXISTS) in the restore.
--jobs	Number of parallel jobs to run (for parallel restore).
-e, --exit-on-error	Exit on error, do not proceed with the restore.
--no-security-labels	Do not restore security labels.
--data-only	Restore only the data, no schema.
--schema-only	Restore only the schema, no data.
--table	Restore only the specified table.
--tablespace	Specify the tablespace for the table.
--exclude-table	Do not restore the specified table.
--disable-triggers	Disable triggers during data-only restore.
--single-transaction	Execute the restore as a single transaction.
--use-set-session-authorization	Use SET SESSION AUTHORIZATION commands during restore.

Handling Large Databases

Use compressed dumps.
You can use your favorite compression program, for example gzip:

```
pg_dump dbname | gzip > filename.gz
```

Reload with:

```
gunzip -c filename.gz | psql dbname
```

or:

```
cat filename.gz | gunzip | psql dbname
```

Use split.
The `split` command allows you to split the output into smaller files that are acceptable in size to the underlying file system.

```
pg_dump dbname | split -b 2G - filename
```

Reload with:

```
cat filename* | psql dbname
```

If using GNU split, it is possible to use it and gzip together:

```
pg_dump dbname | split -b 2G --filter='gzip > $FILE.gz'
```

It can be restored using zcat.

Handling Large Databases

Use pg_dump's custom dump format. If PostgreSQL was built on a system with the zlib compression library installed, the custom dump format will compress data as it writes it to the output file. This will produce dump file sizes similar to using `gzip`, but it has the added advantage that tables can be restored selectively. The following command dumps a database using the custom dump format:

```
pg_dump -Fc dbname > filename
```

A custom-format dump is not a script for psql, but instead must be restored with pg_restore, for example:

```
pg_restore -d dbname filename
```

For very large databases, you might need to combine `split` with one of the other two approaches.

Handling Large Databases

Use pg_dump's parallel dump feature.
To speed up the dump of a large database, you can use pg_dump's parallel mode. This will dump multiple tables at the same time. You can control the degree of parallelism with the `-j` parameter.
Parallel dumps are only supported for the "directory" archive format.

```
pg_dump -j num -F d -f out.dir dbname
```

You can use `pg_restore -j` to restore a dump in parallel. This will work for any archive of either the "custom" or the "directory" archive mode, whether or not it has been created with `pg_dump -j`.

Physical Backup

File System Level Backup

```
tar -cf backup.tar /usr/local/pgsql/data
```

There are two restrictions:
1.The database server *must* be shut down in order to get a usable backup. Half-way measures such as disallowing all connections will *not* work (in part because `tar` and similar tools do not take an atomic snapshot of the state of the file system, but also because of internal buffering within the server).

2.If you have dug into the details of the file system layout of the database, you might be tempted to try to back up or restore only certain individual tables or databases from their respective files or directories. This will *not* work because the information contained in these files is not usable without the commit log files, `pg_xact/*`, which contain the commit status of all transactions. A table file is only usable with this information. So **file system backups only work for complete backup** and restoration of an entire database cluster.

File System Level Backup - Rsync

Another option is to use rsync to perform a file system backup.

- This is done by first running rsync while the database server is running
- then shutting down the database server long enough to do an `rsync --checksum`. (`--checksum` is necessary because `rsync` only has file modification-time granularity of one second.)
- The second rsync will be quicker than the first, because it has relatively little data to transfer, and the end result will be consistent because the server was down. This method allows a file system backup to be performed with minimal downtime.

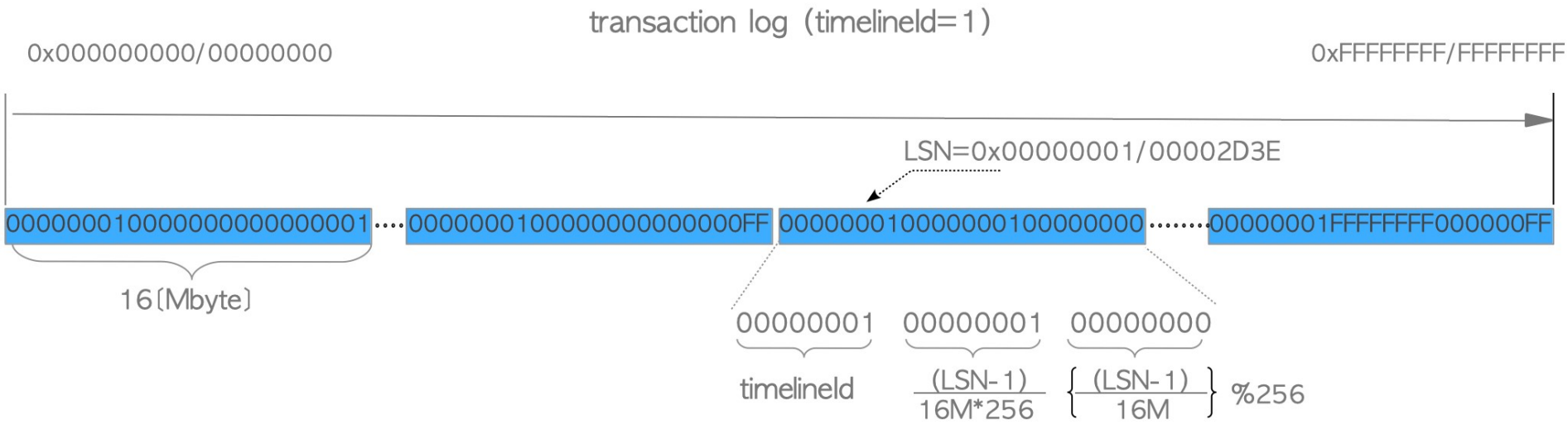
Note that a file system backup will typically be larger than an SQL dump. (pg_dump does not need to dump the contents of indexes for example, just the commands to recreate them.) However, taking a file system backup might be faster.

Continuous Archiving

WAL files

PostgreSQL maintains a write ahead log (**WAL**) in the pg_wal/ subdirectory of the cluster's data directory. The log records every change made to the database's data files. This log exists primarily for crash-safety purposes:

if the system crashes, the database can be restored to consistency by “replaying” the log entries made since the last checkpoint



WAL Segment Files

The first WAL segment file is 000000010000000000000001. If the first one has been filled up with the writing of XLOG records, the second one 000000010000000000000002 would be provided. Files are used in ascending order in succession. After 0000000100000000000000FF has been filled up, the next one 000000010000000100000000 will be provided. In this way, whenever the last 2-digit carries over, the middle 8-digit number increases one.

Similarly, after 0000000100000001000000FF has been filled up, 000000010000000200000000 will be provided, and so on.

```
select pg_current_wal_lsn(), pg_current_wal_insert_lsn();
```

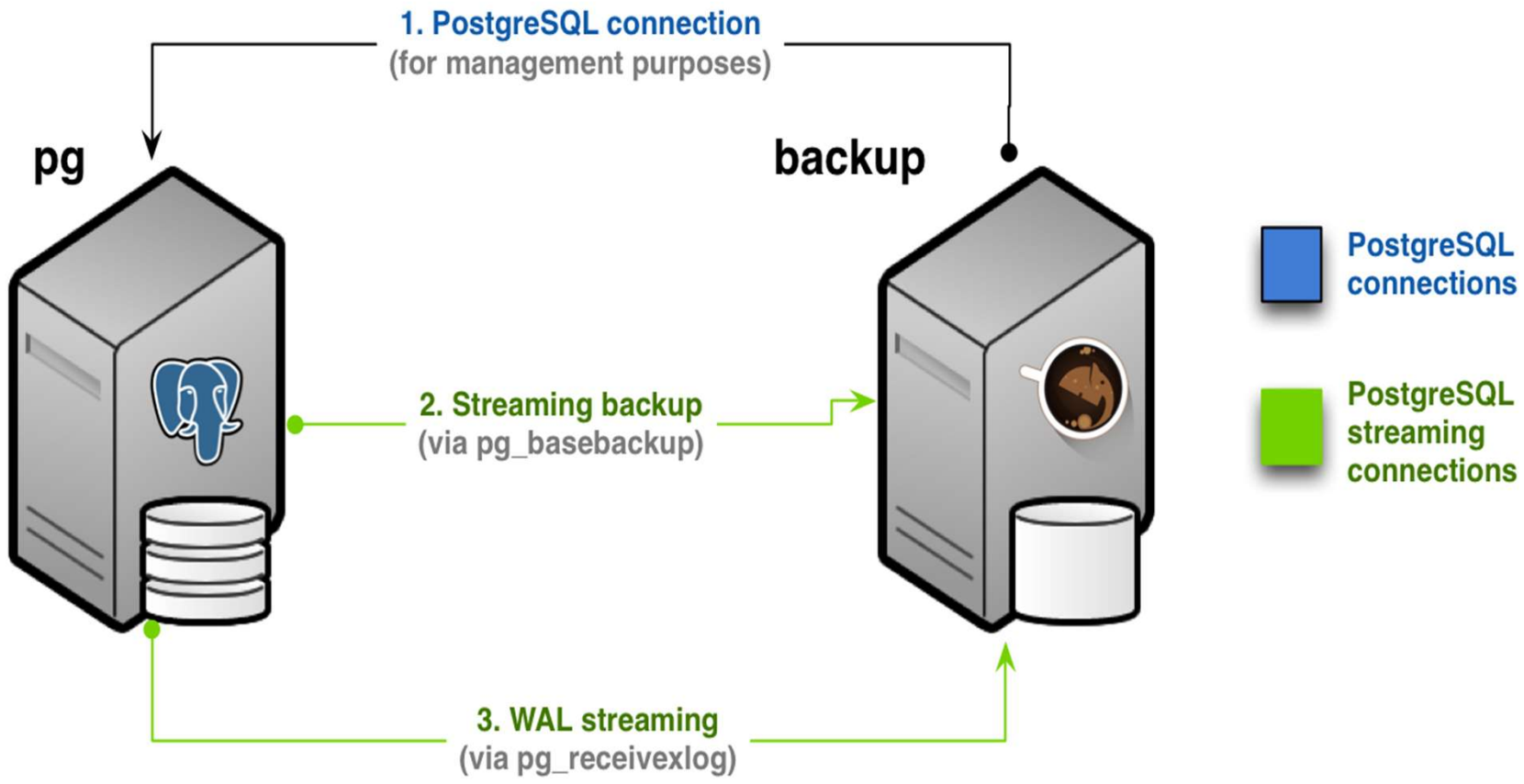
```
select pg_walfile_name('76/7D000028');
pg_walfile_name
--------------------------
000000010000000760000007D
```

https://www.crunchydata.com/blog/postgres-wal-files-and-sequuence-numbers

WAL WRITER PROCESS

The WAL writer is a background process that periodically checks the WAL buffer and writes all unwritten XLOG records to the WAL segments. This process helps to avoid bursts of XLOG record writing. If the WAL writer is not enabled, writing XLOG records could be bottlenecked when a large amount of data is committed at once

Basebackup + Wal Archiving : the main approach



Wrap Up

In general, it is best to think of pg_dump is as a utility for doing specific database tasks. pg_basebackup can be an option if you’re ok with single physical backups on a specific time basis. If you have a production system of size and need to create a disaster recovery scenario, it's best to implement pgBackRest or a more sophisticated tool using WAL segments on top of a base backup

Any Question?