Learning Objectives

Learners will be able to...

- Explain what backups are and why they are needed
- Explore a database backup
- Restore database from backup

info

Make Sure You Know

Knowing how to manage permissions within a DBMS and understanding security best practices is crucial, especially when dealing with backups.

You should know the importance of data integrity, including concepts like primary keys, foreign keys, and constraints.

What is a backup?

Backup is necessary to form a data archive that is protected from changes and damage, as well as to restore in case of damage or failures in the original source.

> info
>
> ## Backup
>
> Backup - creating copies of files, folders or systems on additional (local or cloud) storage media.

There are different types of backups:
- **Full** — creates a copy of all data on the device or in a separate section of memory. Subsequent copies are also created in full. For example, a source data size of 50 GB and 10 full copies would add up to 500 GB in the archive.
- **Incremental** - Creates a backup that only contains changes to files and automatically adds to or modifies them in the original full backup. With an incremental backup, the files are not replaced, but supplemented. For example, the source data size is 50 GB. Changes in one day take up 5 GB, and 10 incremental copies are also made, which gives a total of 100 GB in the archive (one full copy and changed data).
- **Differential** - Creates a copy that includes only the files and data that have changed since the previous copy. For example, the size of the original data is 50 GB, the changes in one day are 5 GB. Relative to the previous day's copy, unique data is 3 GB. As a result, 10 differential copies will total 80 GB in the archive.

The importance of a backup is due to the fact that there are many situations where data loss is possible.

Backup allows you not to worry about the safety of critical files. Even if they are lost or deleted, you can quickly restore all the necessary data.

SQL Backup Types and SQL dump

PostgreSQL Backup types

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

- SQL dump
- File system level backup
- Continuous archiving

Each approach has its advantages and disadvantages. The idea behind this method is to generate a text file with SQL commands that, when executed on the server, will recreate the database in the same state it was in when the command was run.

Dump

To create a backup in this way, there are two commands: `pg_dump` and `pg_dumpall`.

The second command will make a complete dump of all data, with the help of the first one you can selectively create copies of databases or even individual tables.

General view of the command to create a backup.

```
pg_dump dbname > dumpfile
```

So we can create a bump of the current database.

```
# create a dump
pg_dump codio > codio.dump
# show last 20 lines of created file
tail -n 20 codio.dump
```

Restore

You can restore the database back using the PostgreSQL client - `psql`. The general command form to restore a dump is

```
psql dbname < dumpfile
```

where dumpfile is the file output by the pg_dump command.

In relation to our situation, the command will look like this.

```
psql codio < codio.dump
```

Large Databases

You can compress the resulting database bump to reduce the file size, with gzip for example:

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

Compare file size:

```
pg_dump codio | gzip > codio.dump.gz
```

Benefit is more then 4 times:

```
ls -alsh codio.*

2.6M -rw-rw-r-- 1 codio codio 2.6M Aug 23 09:52 codio.dump
616K -rw-rw-r-- 1 codio codio 613K Aug 23 10:10 codio.dump.gz
```

Reload with:

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

```
gunzip -c  codio.dump.gz | psql codio
```

PostgreSQL Backup and Continuous Archiving

An alternative backup strategy is to directly copy the files that PostgreSQL uses to store the data in the database.

You can see the directory where the data is located in the configuration, but sometimes it can be set additionally when starting the service.

```
cat /etc/postgresql/14/main/postgresql.conf | grep
        data_directory
```

You can use whatever method you prefer for doing file system backups; for example:

```
tar -cf backup.tar /var/lib/postgresql/14/main
```

The following restrictions apply to this method:

1. The database must be completely stopped
2. The structure must be copied completely, there is no way to make a table-by-table backup
3. When switching to a new version, the structures may be incompatible with each other

Note that a file-level copy is typically larger than an SQL dump. However, file-level copying can be faster.

Continuous Archiving

Continuous archiving is a combination of the above methods into a single process. The backup and WAL files are used to restore the database. Because playback of WAL files can be stopped at any timestamp, one of the benefits of this process is that databases can be easily restored to any available point in time. Absolutely any number of WAL files can be played, assuming they go back to before the last backup started.

The key to using WAL files for point-in-time recovery is to stream them to the database server. PostgreSQL moves through files, overwriting them as space usage is adjusted. Streaming allows you to avoid overwriting a copy of files. It is included in the PostgreSQL configuration to run WAL archiving and define the command used to copy files to another location. Note that

PostgreSQL keeps WAL files until they are successfully copied. Therefore, make sure that monitoring is enabled to avoid stopping work due to insufficient disk space, network failures, etc.

This process uses PostgreSQL's own pg_basebackup tool to get the original backup file. Along with this command, a maintenance data file is used to record backup start/end time points and write-back logs (WAL).

A complete overview of this functionality can be found in the documentation.