7. Backup and Recovery

Introduction to PostgreSQL

AGENDA

- Logical vs physical backup
- Tools: pg_dump, pg_dumpall
- Recovering from a backup
- pg_restore, pg_basebackup, pg_backrest

BACKUPS

- Three basic types of backups
- SQL dump
  - Create a set of SQL files that, when executed, recreate the server data
- File system level backup
  - Makes a physical copy of the file system
- Continuous archiving
  - Combination of a file system back up and WALL file backup
- There various tools that can perform each of these

PHYSICAL BACKUP

- Copying the physical files that PostgreSQL uses to store data
  - Includes the data files, write-ahead logs (WAL), configuration files, and other system files.
- File System Backup:
  - A physical copy of the data directory using tools like rsync, tar, or cp.
  - The database must be in a consistent state
  - Can be done by stopping the server
  - Or by taking the backup while the database is in a backup mode using pg_start_backup() and pg_stop_backup()
- pg_basebackup:
  - A built-in tool that creates a physical backup of the entire database cluster. It can be run while the server is running and supports streaming WAL files for point-in-time recovery.

PHYSICAL BACKUP

- pg_start_backup()
  - Used to prepare the database for a file system-level backup by putting it into backup mode.
  - Creates a backup label file and triggers a checkpoint, ensuring that all data files are flushed to disk and the database's state is consistent at the start of the backup.
  - “SELECT pg_start_backup('backup_label');”
    - *backup_label: A text label used to identify the backup session*
  - A checkpoint is created to ensure all data files are in a consistent state.
  - A backup label file is created in the data directory, marking the start of the backup.
  - PostgreSQL begins to track which WAL files are necessary to recover the database from the backup point.

PHYSICAL BACKUP

- pg_stop_backup()
  - Used to end the backup mode that was started with pg_start_backup().
  - Finalizes the backup and removes the backup label file
  - Records the end of the backup in the WAL stream, ensuring that all changes since the backup began are properly accounted for
  - A backup history file is created in the WAL archive, which logs the backup start and end times and required WAL files.
  - PostgreSQL is returned to normal operation, and the system continues processing WAL as usual.

PHYSICAL BACKUP

- pg_basebackup
  - Performs a physical backup by copying the entire data directory, including all database files, configuration files, and the Write-Ahead Log (WAL) files necessary to ensure the consistency of the backup
  - Automatically manages the WAL files needed to restore the backup to a consistent state.
  - If used with WAL archiving, it supports point-in-time recovery (PITR).
  - Can be run while the server is online, allowing backups to be taken without stopping the database.
  - Supports parallel mode, allowing multiple data streams to be used during the backup process to speed up the process for large databases.
  - Supports compression of the backup files to reduce disk space usage and improve transfer speed.

PHYSICAL BACKUP

- “pg_basebackup -D /path/to/backup_directory -F format -h host -p port -U user -W”
  - -D /path/to/backup_directory: directory where the backup will be stored.
  - -F format: p for plain directory, t for tar.
  - -h host: The hostname of the PostgreSQL server.
  - -p port: The port number where PostgreSQL is running.
  - -U user: The username to connect with, typically a replication user.
  - -W: Prompts for a password.
    - *pg_basebackup -D /var/lib/postgresql/backups/ -F t -h localhost -p 5432 -U pgadmin -W*

PHYSICAL BACKUP

- Consistency:
  - Physical backups ensure a consistent state of the database at the file level.
  - When used with WAL archiving, they support point-in-time recovery.
- Performance:
  - Generally faster and more efficient for large databases because they operate at the file level.
- Restoration:
  - Restoration involves copying the backed-up files back into the PostgreSQL data directory and starting the server.
  - Point-in-time recovery can be achieved using archived WAL segments.
- Ideal for disaster recovery, full cluster backup, and high-availability setups (e.g., setting up replicas).

PHYSICAL BACKUP

- Advantages
  - Efficient for large datasets.
  - Supports point-in-time recovery when combined with WAL archiving.
  - Suitable for creating replicas in high-availability configurations.
- Disadvantages
  - Less flexible for selective restoration of individual tables or databases.
  - Larger backup size compared to logical backups.

RESTORING PHYSICAL BACKUP

- Stop the PostgreSQL server
  - Restoring involves replacing the data directory, and running processes can interfere with this.
- Prepare the data directory
  - Ensure that the directory where the backup will be restored is empty.
- Extract or copy the backup
  - If the backup is archived, extract to the data directory

    “tar -xzvf /path/to/backup/base.tar.gz -C /var/lib/postgresql/16/main/”
  - If the backup is not archived, copy it to the data directory

    “sudo cp -R /path/to/backup/ /var/lib/postgresql/16/main/”*
- Set correct permissions
  - Data directory and all its contents should owned by the PostgreSQL user, usually postgres.

RESTORING PHYSICAL BACKUP

- Configure PostgreSQL for recovery
  - Create a *recovery.signal* file in the data directory.
  - This tells PostgreSQL that it should start in recovery mode.
- Start the PostgreSQL server
  - Monitor the logs to ensure that the recovery process completes successfully.

PHYSICAL BACKUP

- pgBackRest
  - Open-source backup and restore tool for PostgreSQL
  - Designed for full and incremental physical backups, point-in-time recovery, and replication.
  - Allows for efficient backups by only saving changes since the last full backup (incremental) or since the last backup of any type (differential).
  - Supports PITR using archived WAL files, which allows restoring the database to a specific point in time.
  - Supports compressing and encrypting backups
  - Uses multiple processes to speed up backup and restore operations, making it efficient for large databases.

POINT IN TIME RECOVERY

- Point-in-Time Recovery (PITR) allows you to restore a database to a specific moment in time.
  - commonly used for disaster recovery, undoing unintended changes, or recovering from corruption.
  - PITR combines the use of base backups and Write-Ahead Log (WAL) files to perform the recovery
  - The steps are outlined in the notes

LOGICAL BACKUP

- Involve exporting the database's schema and data in a human-readable format (e.g., SQL scripts).
  - Captures the logical content of the database, such as tables, indexes, sequences, and their data, without concern for the underlying file structure.
- pg_dump:
  - Utility that exports data and schema definitions of individual tables, databases, or clusters into SQL scripts or other formats like plain text, custom, or directory formats.
- pg_dumpall:
  - Utility that exports all databases in a cluster, including global objects like roles and tablespaces.

LOGICAL BACKUP

- pg_dump:
  - Performs logical backups by generating SQL scripts or other formats that describe the database objects and their data, which can be executed to recreate the database.
- Multiple output Formats:
  - *Plain Text Format (-Fp):* Generates a plain SQL script with CREATE, INSERT, and COPY commands.
    - *Easy to read and edit but requires manual parsing for selective restores.*
  - *Custom Format (-Fc):* A compressed, non-text format that allows selective restoration of database objects using pg_restore.
    - *This format is often preferred for flexibility.*
  - *Directory Format (-Fd):* Dumps the database into a directory with separate files for each table and its data, enabling parallel processing and easy inspection of individual components.
  - *Tar Format (-Ft):* Outputs the dump in a tar archive. Suitable for easy transport and storage, similar to the directory format but within a single file.

LOGICAL BACKUP

- Selectively back up specific tables, schemas, or data without dumping the entire database
  - Backing up a table

    pg_dump -t my_table -Fc -f my_table.dump mydatabase
  - Backing up a schema

    pg_dump -n my_schema -Fc -f my_schema.dump mydatabase
  - Can back up tables, views, indexes, sequences, functions, triggers, and other database objects
  - It preserves object ownership, permissions, and other metadata essential for a complete database restoration.

LOGICAL RESTORE

- Restoring with pg_restore
  - When using custom or directory formats, the backups can be restored using pg_restore
  - Provides options for selective restoration of objects and data
    - *-d: Specifies the target database.*
    - *-1: Runs the restore in a single transaction, which is useful for rollback on failure.*
    - *pg_restore -U postgres -d mydatabase -1 mydatabase.dump*
- Selective Restore
  - Restore only specific tables or schemas from a backup:
    - *pg_restore -U postgres -d mydatabase -t my_table mydatabase.dump*

LOGICAL BACKUP

- Database Migrations
  - Export databases for migration between PostgreSQL versions
- Selective Restores
  - Restore individual tables, schema, or specific parts of a database without affecting other objects.
- Development and Testing
  - Create logical backups for setting up development and testing environments quickly.
- Regular Backup Strategy
  - Regular, scheduled logical backups protect against data loss.

LOGICAL BACKUP

- Disadvantages
  - For very large databases, pg_dump can be slower than physical backups, especially if using plain text format.
  - While pg_dump provides consistent snapshots, it does not support point-in-time recovery directly.
  - Running pg_dump can consume significant CPU and I/O resources

LOGICAL BACKUP

- pg_dumpall
  - Used to back up an entire database cluster, including all databases, global objects, roles, tablespaces, and configuration settings.
  - Performs a logical backup of everything within a cluster to support full cluster backups and migrations.
- Cluster-Wide Backups
  - pg_dumpall creates a comprehensive backup of all databases in a PostgreSQL cluster, including global objects that are not tied to any specific database.
  - Includes global objects such as roles (users), group roles, tablespaces, and database configurations, which are not included when using pg_dump alone.
- Output Format:
  - Outputs a plain text SQL script. Unlike pg_dump, it does not support custom, directory, or tar formats.
  - Useful reading, editing, and applying during restore.

LOGICAL BACKUP

- Uses pg_dump Internally:
  - pg_dumpall uses pg_dump to back up each database within the cluster.
  - It sequentially dumps each database and combines these dumps into a single output file.
- Basic Usage
  - “pg_dumpall [OPTIONS]”
  - Options
    - *-U: Specifies the user to connect as.*
    - *-h: Specifies the host of the PostgreSQL server.*
    - *-p: Specifies the port number.*
    - *-f: Specifies the output file for the dump.*
    - *-l: Specifies a file containing the list of databases to be dumped.*
  - “pg_dumpall -U postgres -h localhost -p 5432 -f backup.sql”

LOGICAL BACKUP

- Options for pg_dumpall
  - -g or –globals-only:
    - *Dumps only global objects (roles, tablespaces) without the individual databases.*
    - *“ pg_dumpall -U postgres --globals-only -f globals_backup.sql”*
  - -r or --roles-only:
    - *Dumps only the roles (users) defined in the cluster.*
    - *“pg_dumpall -U postgres --roles-only -f roles_backup.sql”*
  - --clean:
    - *Includes commands to drop the objects before recreating them, useful for restoring into a cluster where objects might already exist.*
    - *“pg_dumpall -U postgres --clean -f clean_full_cluster_backup.sql”*

LOGICAL BACKUP

- Full Cluster Backups:
  - Use pg_dumpall for full backups of the whole cluster, including all databases and global configurations
  - For disaster recovery scenarios where the entire cluster needs to be restored.
- Cluster Migration:
  - Migrate the entire, including roles and tablespaces, from one server to another
  - Or migrate the cluster to different versions of PostgreSQL.
- Consistency Across Databases:
  - Ensure consistency when backing up multiple related databases within the same cluster, since pg_dumpall captures a snapshot of the entire cluster.
- Backing Up Global Objects:
  - Use to back up global objects like roles and tablespaces, which are not included in individual pg_dump backups.

LOGICAL BACKUP

- Output Size:
  - The plain text output of pg_dumpall can be very large – mitigated with compression tools
- Performance:
  - Each database processed sequentially
  - The backup process can take a long time, especially for large clusters.
  - Schedule backups during off-peak hours to minimize impact.
- Restore Process:
  - Restoring executes a large SQL script, which can be slow for large datasets.
- No Custom Formats:
  - Does not support custom or parallelizable formats which may impact performance

LOGICAL RESTORE

- Restoring from a pg_dumpall backup
  - “psql -U postgres -f backup.sql”
  - This executes the SQL in the backup file
  - Recreates the entire cluster, including roles, configurations, and all databases.

LAB 7-1

- The lab description and documentation is in the Lab directory in the class repository

End Module