Reading: Performing MySQL Physical Backup and Restoration

Estimated Time: 10 minutes

Objectives:

After completing this reading, you will be able to:

- Understand the Significance of Physical Backups in Database Management.
- An Overview of Key Linux Commands used for Conducting Physical Backup and Restoration of MySQL Database.

Understand the Significance of Physical Backups in Database Management.

A physical backup involves duplicating all the physical storage components, such as files and directories, of a specific table, database, or other objects within a database system. These components include essential elements like data files, configuration files, and log files, capturing the entire storage structure as it exists on the disk.

Compared to logical backups, which involve exporting data in a structured format, physical backups are generally more compact and quicker to execute. This makes them particularly well-suited for scenarios where large or critical databases must be backed up swiftly and efficiently.

Common Techniques for Physical Backups:

This section explores various methods and techniques used for performing physical backups in database systems.

File-based Backups: This technique involves copying database files directly from disk to backup storage. It is a straightforward approach but may require downtime or database locking during backup operations.
This typically includes data files, control files, and archived redo logs.

Example: In an Oracle database environment, a file-based backup might involve copying the data files **(.dbf files), control files (controlfile.ctl), and archived redo logs (*.arc)** to a backup destination using operating system utilities like **cp (copy) or rsync**.

Block-level Backups: Block-level backups operate at a lower level, capturing data at the disk block level rather than file level. This technique allows for incremental backups, where only changed blocks are backed up, reducing backup time and storage requirements.

Example: In a PostgreSQL database, block-level backups can be achieved using tools like pg_basebackup or third-party solutions that utilize PostgreSQL's write-ahead logging (WAL) mechanism to capture changed blocks since the last backup.

Online Backups: Online backups enable database backups to be performed while the database remains operational, minimizing downtime and disruptions to business operations.

Example: In a MySQL database, online backups can be achieved using tools like **mysqldump** for logical backups or **Percona XtraBackup** for physical backups. These tools allow backups to be taken without locking tables, ensuring that the database remains accessible to users during the backup process.

Snapshot Backups: Snapshot backups use storage snapshots to create point-in-time copies of the database. This technique provides a consistent view of the database without interrupting ongoing transactions.

This approach is sometimes used in the cloud by taking snapshots of Cloud based storage volumes, and restarting the database on a different VM.

Example: In a Microsoft SQL Server environment, snapshot backups can be created using storage technologies such as Storage Spaces Direct (S2D) or SAN/NAS-based snapshot solutions. These technologies leverage the storage infrastructure to create instant, space-efficient snapshots of database volumes.

Backup Compression: Compressing backup files reduces storage space requirements and improves backup efficiency without sacrificing data integrity.

Example: In a MongoDB database, backup compression can be enabled using built-in compression options in backup utilities like mongodump or by utilizing third-party backup solutions that offer compression features.

Best Practices for Physical Backups:

This section offers recommendations and guidelines for implementing and managing physical backup strategies effectively.

Regular Backup Schedule: Here a regular backup schedule is established based on business requirements and data sensitivity. Consider factors such as data volatility, recovery time objectives (RTO), and recovery point objectives (RPO).

Offsite Storage: Store backup copies in secure, offsite locations to protect against site-wide disasters such as fires, floods, or theft.

Backup Verification: Regularly test backup files so that we can ensure that they are valid files,thus enabling successful restoration. Verification helps identify any issues with backup processes or storage media before they become critical.

Retention Policies: Define retention policies for backup files based on compliance requirements, regulatory standards, and business needs. Retain backups for an appropriate duration to support data recovery and legal obligations.

Encryption: Implement encryption for backup data in transit as well as at rest to safeguard sensitive information from unauthorized access or data breaches.

Monitoring and Alerting: Here, monitoring and alerting mechanisms are set up to track backup performance, storage usage, and potential issues. Proactive monitoring helps identify and address backup failures or anomalies promptly.

An Overview of Key Linux Commands used for Conducting Physical Backup and Restoration of MySQL Database

Let us explore a scenario where you are tasked with taking a complete physical backup of the MySQL server that houses a database named **world**. Within this database, you will encounter tables like **cities**, **countries** and **languages**, each holding data related to **cities**, **countries** and **languages** spoken worldwide.
The necessary SQL statements related to this data is present in a sql script file **world_mysql_script.sql**.

You can first download this script in the linux terminal by executing the command below:

```
wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0231EN-SkillsNetwork/datasets/World/world_mysql_script
```

Once executed, this command will fetch the SQL script file from the internet and save it to the current directory on your local machine.

Pre-requisite: To interact with MySQL databases on your local machine, you need to install and start the MySQL server. To install the MySQL server run the following command:

```
sudo apt-get install mysql-server
```

This command will prompt you to enter your password, and once you confirm, it will download and install the MySQL server package along with any necessary dependencies.

> Note: In case you encounter errors regarding unreachable packages, execute the command `sudo apt update` to fetch the most recent information concerning available packages and their versions from the repositories set up on your system. Once done you can then execute the command to install the mysql-server.

After the installation start the MySQL service using the command:

```
sudo systemctl start mysql
```

This command is used to start the MySQL service on a system running systemd, which is a system and service manager for Linux.

Here's what each part of the command does:

- **sudo**: This prefix grants temporary administrative privileges to the command that follows it. It allows the user to execute the subsequent command with superuser (root) privileges.
- **systemctl**: This is a command-line utility used to manage **systemd**, which is a system and service manager for Linux operating systems. It allows users to start, stop, enable, disable, and manage services and other units.
- **start**: This subcommand of **systemctl** instructs systemd to start the specified service. In this case, **mysql** refers to the MySQL service.

Next, to access the MySQL CLI(Command Line Interface), you can use the command below in the terminal:

```
sudo mysql –u root –p
```

Here's what each part of the command does:

- **mysql**: This is the command used to launch the MySQL command-line interface.
- **-u root**: This option specifies the username to use when connecting to the MySQL server. In this case, it specifies that the root user should be used.
- **-p**: This option prompts the user to enter the password for the specified MySQL user (in this case, the root user). After entering the password, if it's correct, the user gains access to the MySQL CLI with administrative privileges.

Create the database and navigate to the database using the commands below in the terminal:

```
create database world;
use world;
```

Use **world_mysql_script.sql** script to complete the world database creation process. This script holds the data of the tables **cities** ,**countries** and **languages**.

```
source world_mysql_script.sql;
```

This command will execute all the SQL commands contained in the script file within the MySQL command-line interface.

Once the operation is completed, we can quit from the MySQL command line interface using the command `\q` in the terminal.

Before taking the backup we need to stop the MySQL service using the command below:

```
sudo systemctl stop mysql
```

Here's what each part of the command does:

- **systemctl**: This is a command-line utility used to manage **systemd**, which is a system and service manager for Linux operating systems. It allows users to start, stop, enable, disable, and manage services and other units.
- **stop**: This subcommand of **systemctl** instructs systemd to stop the specified service. In this case, **mysql** refers to the MySQL service.

You can then perform a backup of the complete database using the command below to recursively copy all the directories and subdirectories of the MySQL data directory to the specified backup directory:

```
sudo cp –R /var/lib/mysql /path/to/backup_directory
```

Here's what each part of the command does:

- **cp**: This is the command used to copy files and directories.
- **-R**: This option is used to specify a recursive copy, meaning it will copy the specified directory and all of its contents, including subdirectories and files.
- **/var/lib/mysql**: MySQL data files, including databases, tables, and other related files are present in this directory.
- **/path/to/backup_directory**: This is the destination directory where the MySQL data directory will be copied. You should replace “/path/to/backup_directory” with the directory which you want the backup to be stored.

To verify the successful creation of the backup in the designated backup directory, execute the following command in the terminal:

```
sudo ls –l /path/to/backup_directory
```

This will list detailed information about the contents of the specified backup directory and you can view MySQL database related files here.

You can restore the MySQL database from the backup by following the commands below:

```
sudo mv /var/lib/mysql /var/lib/mysql_old
```

This command will rename the current MySQL data directory `/var/lib/mysql` to `/var/lib/mysql_old`

Next, we can recursively copy the contents of the backup directory which we created earlier into the MySQL data directory by using the command below:

```
sudo cp -R /path/to/backup_directory /var/lib/mysql
```

> Note: Please be aware that during the restoration of the backup, you might encounter insufficient space on the device. This can happen if the backup file or the data being restored exceeds the available disk space on the device where MySQL is installed or where the backup is being restored. It's essential to ensure that there is enough disk space available to accommodate the restoration process to avoid encountering this error.

We can change the ownership of the MySQL data directory and all its contents recursively to the user **mysql** and the group **mysql** as below:

```
sudo chown -R mysql:mysql /var/lib/mysql
```

Here's what each part of the command does:

- **chown**: Stands for **change ownership,** a command used to change the ownership of files or directories in Linux.
- **-R**: Specifies a recursive operation,including subdirectories and files.
- **mysql:mysql**: Specifies the new owner and group for the directory and its contents. In this case, it sets the user to "mysql" and the group to "mysql".
- **/var/lib/mysql**: Specifies the directory whose ownership will be changed. This is typically the MySQL data directory.

Once completed you can restart the MySQL service using the command: `sudo systemctl start mysql`

Conclusion:

In conclusion, this reading stressed the vital role of physical backups in database management. Understanding their importance helps maintain smooth operations and prevents data loss. Exploring MySQL Physical Backup and Restoration on Linux provides practical skills to implement reliable backup strategies, enhancing the resilience of MySQL databases in Linux environments.

Author(s)

Lakshmi Holla

Other Contributors

Malika Singla