How to perform Point-In-Time-Recovery in MySQL|PostgreSQL databases

Understanding Point-In-Time Recovery

PITR involves restoring the database to a state it was in at a specific point in time. This process typically involves restoring a base backup (either logical or physical) and then applying incremental changes from transaction logs (binlogs in MySQL or WAL logs in PostgreSQL) up to the desired recovery point.

Point-in-time recovery (PITR) capability is invaluable in data corruption, accidental deletions, or other data integrity issues.

What is Binary log (MySQL) | WAL log (PostgreSQL)

Binary Logs | WAL logs are a set of log files that record all changes made to the database. They include data modification statements like INSERT, UPDATE, DELETE, and CREATE, which allows for data recovery and replication

In this blog, we'll cover both logical and physical backup-based PITR for MySQL and PostgreSQL databases, **with examples of restoring from a full db backup taken at 12 AM and logs after a crash at 4 AM.**

MySQL Point-In-Time Recovery Example

Scenario

- **Backup time:** 12 AM
- **Crash time:** 4 AM
- **Binlog files:** mysql-bin.000001, mysql-bin.000002, mysql-bin.000003
- **Binlog position after backup:** File: mysql-bin.000001**, Position:** 120 [from backup file with --master-data=2]
- **Desired recovery time:** Just before 4 AM

Logical Backup-Based PITR

1. **Identify Binlog Files and Positions:** get the binlog file and position from the backup file when the backup was taken . Let's assume the output was **mysql-bin.000001 | 120**
2. **Restore the Full Backup:** Restore the full backup taken at 12 AM. This step assumes we already have the backup file full_backup_12am.sql.

```
mysql -u root -p < /backups/full_backup_12am.sql
```

3. Apply Binary Logs: Use the mysqlbinlog utility to apply the changes from the binary logs up to the desired point in time (4 AM)

```
mysqlbinlog --start-position=120 --stop-datetime="2024-07-11 04:00:00"  mysql-bin.000001 mysql-bin.000002 mysql-bin.000003 | mysql -u root -p
```

This command will replay the transactions from the binary logs starting from position 120 in mysql-bin.000001 up until just before 4 AM.

4. Verify Recovery: Check the database to ensure it is in the expected state. Verify MySQL -error log too

```
SHOW TABLES ;
```

Physical Backup-Based PITR

1. **Identify Binlog Files and Positions:** same as above, ensure we have the binlog file and position recorded.

2. **Restore the Full Physical Backup:** Restore the physical backup taken at 12 AM using Percona XtraBackup. let's assume we have the backup directory /backups/mysql_xtrabkp

```
xtrabackup --copy-back --target-dir=/backups/mysql_xtrabkp
chown -R mysql:mysql /var/lib/mysql
```

3. Apply Binary Logs: Use the mysqlbinlog utility as described in the logical backup example to apply the binary logs.

```
mysqlbinlog --start-position=120 --stop-datetime="2024-07-11 04:00:00"  mysql-bin.000001 mysql-bin.000002 mysql-bin.000003 | mysql -u root -p
```

4. Verify Recovery: similar to logical restore, check the database to ensure it is in the expected state and check error log.

==
==

PostgreSQL Point-In-Time Recovery Example

Scenario

- **Backup time:** 12 AM
- **Crash time:** 4 AM
- **WAL files:** 00000001000000000000000A, 00000001000000000000000B, 00000001000000000000000C
- **WAL position after backup:** 0/2000000
- **Desired recovery time:** Just before 4 AM

Logical Backup-Based PITR

1. **Identify WAL Files and Positions:** After taking the backup at 12 AM, we should have recorded the WAL file and position. We can add in our backup script to record LSN at the end

```
SELECT pg_current_wal_lsn();
```

lets assume we have 0/2000000

2. **Restore the Full Backup:**

   Restore the full backup taken at 12 AM. This step assumes we already have the backup file full_backup_12am.dump

```
pg_restore -U postgres -d postgres /backups/full_backup_12am.dump
```

3. **Configure** recovery.conf **for Point-In-Time Recovery**:

   Create or edit the recovery.conf file in our PostgreSQL data directory (/var/lib/postgresql/data). Configure it to restore up to the point just before the crash (4 AM).

   vi postgresql.conf or postgersql.auto.conf

```
restore_command = 'cp /var/lib/postgresql/wal_archive/%f %p'
recovery_target_time = '2024-07-11 04:00:00'
```

4. Restart PostgreSQL to Start the Recovery Process: Restart PostgreSQL to begin the recovery process. PostgreSQL will automatically apply the WAL files up to the specified recovery target time.

```
pg_ctl -D /var/lib/postgresql/data restart
```

5. Verify Recovery: Check the PostgreSQL logs to ensure recovery is completed successfully. Verify the database state to ensure it matches the expected state as of just before 4 AM.

```
SELECT * FROM pg_stat_activity;
```

Conclusion

Point-in-time recovery is an essential feature for maintaining data integrity and ensuring business continuity. Whether using logical or physical backups, both MySQL and PostgreSQL provide robust tools for performing PITR. By following the steps outlined above, DBA's can effectively recover databases to a specific point in time, minimizing data loss and maintaining operational stability.