**BACKUP & RECOVERY REQUIREMENTS**

**Group 11**

**Blossom Anukposi**

**Cristian Trifan**

**Mihail Josan**

**Stefan Cazacu**

Table of Contents

[1. Recommended Backup Strategy 2](#_Toc188037010)

[2. Sample SQL Backup Commands 3](#_Toc188037011)

[3. Backup Protocol Description 4](#_Toc188037012)

[4. Backup Strategy Advice 5](#_Toc188037013)

[5. Preventing Database Downtime 7](#_Toc188037014)

[6. Backup Strategy Implementation Explanation 8](#_Toc188037015)

## 1. Recommended Backup Strategy

**1.1 Daily Full Backups**

* **Reason: The database size is manageable (in the GB range), making daily full backups feasible. This ensures that we have a complete snapshot of the database each day.**

**1.2 Point-in-Time Recovery (PITR)**

* **Method: Utilize PostgreSQL's Write-Ahead Logs (WAL) for PITR.**
* **Reason: PITR allows recovery to a specific point in time, minimizing data loss in case of accidental deletion or corruption.**

**1.3 Retention Period**

* **Policy: Keep 7 days of backups.**
* **Reason: This balances storage requirements with the ability to recover from recent issues. A week-long history is typically sufficient to address most common data incidents.**

## 2. Sample SQL Backup Commands

**Full Backup**

**pg\_dump -U username -d database\_name -F custom -f "/path/to/backup/netflix\_backup\_%date%.dump"**

* **Custom Format (-F custom):**
  + **Offers compression to save space.**
  + **Supports parallel restores for faster recovery.**
  + **More flexible than plain text format.**

**Backup Specific Tables**

**pg\_dump -U username -d database\_name -t users -t movies -F custom -f "/path/to/backup/partial\_backup\_%date%.dump"**

* **Reason: Backing up critical tables separately ensures faster recovery of essential data without requiring full database restoration.**

**Restore Command**

**pg\_restore -U username -d database\_name "/path/to/backup/netflix\_backup.dump"**

* **Custom Format Advantage: Allows selective restoration of specific database components.**

## 

## 3. Backup Protocol Description

**3.1 Daily Backup Process**

* **Schedule: Perform a full database backup at midnight.**
* **WAL Archiving: Enabled to facilitate continuous PITR.**
* **Cleanup: Automate the removal of backups older than 7 days to manage storage.**

**3.2 Backup Verification**

* **Weekly Tests: Restore backups in a separate environment to verify their integrity.**
* **Automated Checks: Ensure backup completion and verify file sizes.**

**3.3 Storage Location**

* **Primary Location: Local storage for quick access and immediate recovery.**
* **Secondary Location: Cloud storage (e.g., GitHub or Azure Blob Storage).**
  + **Reason for Two Locations:**
    - **Protects against local hardware failures or accidental deletions.**
    - **Ensures backups are accessible even in case of primary storage issues.**

## 4. Backup Strategy Advice

**4.1 PostgreSQL Configuration (postgresql.conf)**

**wal\_level = replica**

**archive\_mode = on**

**archive\_command = 'cp %p /path/to/archive/%f'**

* **Explanation:**
  + **wal\_level = replica: Enables detailed logging required for PITR.**
  + **archive\_mode = on: Activates WAL archiving.**
  + **archive\_command: Specifies how WAL files are copied to the archive directory.**

**4.2 Automation Script Example**

**#!/bin/bash**

**# Filename: backup\_netflix.sh**

**DATE=$(date +%Y%m%d)**

**BACKUP\_DIR="/path/to/backup"**

**LOG\_FILE="/path/to/logs/backup\_$DATE.log"**

**# Full backup command**

**pg\_dump -U username -d database\_name -F custom -f "$BACKUP\_DIR/netflix\_backup\_$DATE.dump" > "$LOG\_FILE" 2>&1**

**# Verify backup success**

**if [ $? -eq 0 ]; then**

**echo "Backup completed successfully on $DATE" >> "$LOG\_FILE"**

**else**

**echo "Backup failed on $DATE" >> "$LOG\_FILE"**

**fi**

* **Usage:**
  1. **Save the script as backup\_netflix.sh.**
  2. **Run the following commands in Git Bash:**
  3. **chmod +x backup\_netflix.sh**

**./backup\_netflix.sh**

## 5. Preventing Database Downtime

**5.1 Best Practices**

* **Low-Usage Periods: Schedule backups at midnight to minimize user impact.**
* **Custom Format: Use pg\_dump -F custom for faster and more reliable backups.**
* **WAL Archiving: Ensure continuous backup through WAL files.**
* **Regular Testing: Restore backups in a separate environment weekly.**

**5.2 Performance Optimization**

* **Compression: Reduces storage requirements for backups.**
* **Parallel Operations: Speeds up large backups and restores.**
* **Monitoring: Track backup duration and adjust schedules if necessary.**

**5.3 Additional Recommendations**

* **Version Control: Store backup scripts in GitHub or similar platforms.**
* **Restore Documentation: Maintain clear, step-by-step restore procedures.**
* **Monitoring & Notifications: Set up alerts for backup completion and errors.**
* **Backup Logs: Keep logs for audit and troubleshooting purposes.**

## 6. Backup Strategy Implementation Explanation

**6.1 Why This Approach?**

* **Full Database Backups:**
  + **Simple and reliable.**
  + **Easy to restore, especially for a small database like the Netflix clone.**
* **Custom Format (-F custom):**
  + **Better compression.**
  + **Parallel restore support.**
  + **Flexible restoration options.**

**6.2 Why Include Testing?**

* **Verifies database connectivity and backup integrity.**
* **Provides immediate feedback for troubleshooting.**

**6.3 Why Include Error Handling?**

* **Prevents silent failures.**
* **Logs issues for faster resolution.**
* **Ensures reliability of the backup process.**

**6.4 Benefits of This Implementation**

**1. Simplicity**

* **Straightforward execution and maintenance.**
* **Easy to adapt to the project’s scope and resources.**

**2. Reliability**

* **Connection checks and error handling ensure consistency.**
* **Backups are verified through weekly restoration tests.**

**3. Flexibility**

* **Adaptable to different environments or database configurations.**
* **Supports both automated and manual execution.**

## 7. Advice for Preventing Database Downtime

**7.1 Proactive Monitoring**

* **Set Alerts**: Monitor database performance metrics and set up alerts for unusual activity or high resource usage.
* **Health Checks**: Implement periodic health checks to detect issues early.

**7.2 Redundancy**

* **Replicas**: Use read replicas to distribute load and provide a fallback in case the primary database fails.
* **Failover Mechanisms**: Configure automatic failover to a standby database.

**7.3 Maintenance Practices**

* **Index Optimization**: Regularly rebuild and analyze indexes to maintain query performance.
* **Vacuuming**: Run VACUUM and ANALYZE commands to prevent bloat and ensure accurate query planning.

**7.4 Backup and Recovery**

* **Frequent Backups**: Ensure regular backups to minimize data loss during downtime.
* **Restore Testing**: Regularly test restores to validate the reliability of backups.

**7.5 Scaling**

* **Horizontal Scaling**: Distribute load across multiple nodes.
* **Vertical Scaling**: Upgrade hardware or allocate more resources during peak usage periods.

**7.6 Scheduling**

* **Off-Peak Hours**: Schedule backups, maintenance, and updates during periods of low activity.

**7.7 Documentation and Training**

* **Restore Procedures**: Maintain detailed documentation for recovery processes.
* **Team Preparedness**: Train team members on handling database issues and performing restores.