PITR with PostgreSQL 8

Crash recovery.
It's not a matter of if, but when.

PITR in a Nutshell

If pg_dump is to DBAs what level zero backups are to System Administrators, then PITR is to DBAs what incremental backups are to System Administrators.

Is PITR right for you?

- PITR is not for everyone: it adds administrative complexity and increased disk use and IO.
- Small databases may find performing frequent full dumps via pg_dump to be more acceptable than implementing and deploying PITR.
- For non-readonly databases with 24/7 operation and DBAs who want a high degree of recoverability, PITR is a must!

Is PITR appropriate?

Use PITR if:

- Obtaining a frequent dump of the database is performance or space prohibitive.
- It is not acceptable to loose data between the last dump and the time of a crash.
- File system snapshots aren't appealing.
- There is sufficient man power available to monitor the health of a PITR setup!

How PITR works...

- PITR works by invoking an external command that archives WAL files when they become eligible to be recycling.
- A WAL file will be recycled upon successful completion of the external command.
- The postmaster will execute the WAL file archival command as many times as is necessary until it returns non-zero.

postgresql.conf

```
archive_command = '/any/command'
```

- `%p` = absolute path
- `%f` = filename only
- `%%` = The % character
- Example:

```
archive_command = 'cp -n %p /nfs1/pitr/%f'
```

PITR Full Backup Procedure

Backups of $PGDATA can now be done with the database online without the use or need for filesystem snapshots.

```
1. SELECT pg_start_backup('my_backup');

2. tar --exclude $PGDATA/pg_xlog \
   cvjpf pgbackup.tar.bz2 $PGDATA/

3. SELECT pg_stop_backup();
```

PITR Backup Notes

- Be mindful of tablespaces and data directories outside $PGDATA

- Time interval between issuing pg_start_backup() and pg_end_backup() statements is not time sensitive

- Make sure WAL archiving is working before starting this procedure

PITR Recovery

- Start fresh (ie: move $PGDATA and its data subdirectories - including table spaces - and install copies of its configs
- Restore database from last full dump/tar
- Clean out pg_xlog
- Copy unarchived WAL logs into $PGDATA/pg_xlog/, if available
- Create **`recovery.conf`** in $PGDATA/
- Start postmaster

recovery.conf

- restore_command = '/any/sh/command'
- `%p` = absolute path
- `%f` = filename only
- `%%` = The % character
- Example:

```
restore_command = 'cp -n /nfs1/pitr/%f %p'
```

Recovery Notes

- restore_command will be asked for files that don't exist
- Command must return zero on success
- Command must return non-zero on non-existent files
- Can restore to any time after a completed full dump to a given time or transaction ID

PITA Food for PITR Thought

- Test, practice, and verify that your site's PITR procedure works before depending on its recoverability!
- WAL logs aren't small. Choose data expiration policies for archived WAL data wisely (ie: only after last full backup completes)
- WAL files must archived faster than WAL files are generated!!!
- PITR files should be kept with with full backups
- Why not backup unarchived WAL files once a minute via cron(8)? Just remember to discard unofficially archived WAL files when the official archival of a WAL file happens.

Final Thoughts

- Read the docs, this was a lightning talk to familiarize and introduce the concept and use of PITR. This is not meant to be used for training or replace a tutorial.
- Having DBA depts contracting PITRitis is not fatal, but it can be a PITA to setup, but pays huge dividends come failure time.
- Database administration 101: It's not a matter of if the database hardware/disks crash, it's a matter of when. PITR is the extra safety belt.

Thank you!

Questions? Let me know!

Sean Chittenden
sean@gigave.com