psql : l'essentiel

```
\l                    liste les bases       \c db     se connecter
\dt  \di  \dv  \df    tables / index / vues / fonctions
\d+ name              décrit, avec index et stockage
\du  rôles            \dn  schémas          \dx  extensions
\dp name              privilèges d'une table
\x                    affichage étendu (bascule)
\timing               affiche le temps de chaque requête
\e                    édite la requête dans $EDITOR
\i file.sql           exécute un fichier SQL
\copy t TO 'out.csv' CSV HEADER    export côté client
\set VERBOSITY verbose affiche le SQLSTATE avec chaque erreur
\watch 2              relance la dernière requête toutes les 2s
```

Ce qui tourne

```
SELECT pid, state, now() - query_start AS runtime,
       wait_event_type, left(query, 80) AS query
FROM pg_stat_activity
WHERE state <> 'idle'
ORDER BY query_start;
```

```
-- Qui bloque qui :
SELECT w.pid AS waiting, b.pid AS blocking,
       left(b.query, 60) AS blocking_query
FROM pg_stat_activity w
JOIN pg_stat_activity b
  ON b.pid = ANY (pg_blocking_pids(w.pid));
```

`state = 'idle in transaction'` + `xact_start` ancien = ennuis. Plus : voir les requêtes en cours.

Annuler et tuer

```
SELECT pg_cancel_backend(pid);     -- annule la requête, garde la session
SELECT pg_terminate_backend(pid);  -- ferme toute la connexion
```

Escaladez dans cet ordre. Jamais de `kill -9` sur un backend depuis le shell : tout le cluster redémarre. Détails : tuer une requête en sécurité.

Timeouts

```
SET statement_timeout = '30s';   -- plafond de durée totale (attentes de verrous incl.)
SET lock_timeout = '3s';         -- attente max pour un verrou (idéal pour le DDL)
SET idle_in_transaction_session_timeout = '5min';

BEGIN;
SET LOCAL statement_timeout = '30min';  -- cette transaction seulement
-- ... longue migration ou rapport ...
COMMIT;
```

Quand ils se déclenchent : statement timeout · lock timeout.

Tailles

```
SELECT pg_size_pretty(pg_total_relation_size('t')); -- heap+TOAST+index
SELECT pg_size_pretty(pg_table_size('t'));          -- sans les index
SELECT pg_size_pretty(pg_indexes_size('t'));        -- index seulement
SELECT pg_size_pretty(pg_database_size(current_database()));
```

La requête top-20 et les détails : tailles des tables et index. Taille ≠ données vivantes — voir bloat.

Maintenance

```
ANALYZE t;                        -- rafraîchit les statistiques du planificateur
VACUUM (VERBOSE, ANALYZE) t;      -- récupère les lignes mortes + stats
CREATE INDEX CONCURRENTLY idx ON t (col);  -- sans verrou d'écriture
REINDEX INDEX CONCURRENTLY idx;

-- surveillance du wraparound (alerte bien avant ~2 milliards) :
SELECT datname, age(datfrozenxid)
FROM pg_database ORDER BY 2 DESC;
```

Le contexte : VACUUM, autovacuum et bloat · urgence : wraparound.

EXPLAIN

```
EXPLAIN SELECT ...;                    -- plan seul, n'exécute pas
EXPLAIN (ANALYZE, BUFFERS) SELECT ...; -- EXÉCUTE la requête, temps réels

-- écritures, en sécurité :
BEGIN; EXPLAIN (ANALYZE) UPDATE ...; ROLLBACK;
```

Collez la sortie (texte ou JSON) dans le visualiseur EXPLAIN : flame graph et avertissements automatiques.

Sauvegarde et restauration

```
pg_dump -Fc dbname > db.dump        # format custom : compressé,
pg_restore -j4 -d dbname db.dump    #   restauration parallèle avec -j
pg_dump -Fc -t mytable dbname > table.dump
pg_dumpall --globals-only > roles.sql   # les rôles ne sont PAS dans pg_dump
psql dbname < plain.sql             # restaure les dumps SQL simples
```

`pg_dump` est cohérent sans bloquer les écritures (il s'exécute dans une transaction Repeatable Read).

Privilèges (rôle applicatif type)

```
GRANT USAGE ON SCHEMA app TO app_user;
GRANT SELECT, INSERT, UPDATE, DELETE
  ON ALL TABLES IN SCHEMA app TO app_user;
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA app TO app_user;

-- aussi les tables FUTURES (créées par le rôle de migration) :
ALTER DEFAULT PRIVILEGES FOR ROLE migrator IN SCHEMA app
  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO app_user;
```

Les GRANT ne couvrent pas les tables futures — le piège classique. Plus : permission denied.

Configuration

```
SHOW work_mem;
SELECT name, setting, source FROM pg_settings
WHERE source <> 'default';        -- ce qui a été personnalisé

ALTER SYSTEM SET work_mem = '64MB';
SELECT pg_reload_conf();          -- certains paramètres exigent un redémarrage
```

`work_mem` vaut par nœud sort/hash, par requête — ça se multiplie. `max_connections` et `shared_buffers` exigent un redémarrage.

Réplication : vérifications rapides

```
SELECT pg_is_in_recovery();            -- suis-je un réplica ?
SELECT client_addr, state, replay_lag
FROM pg_stat_replication;              -- sur le primaire

-- slots retenant du WAL (active = f est un signal d'alerte) :
SELECT slot_name, active, pg_size_pretty(
  pg_wal_lsn_diff(pg_current_wal_lsn(), restart_lsn)) AS retained
FROM pg_replication_slots;
```

Les slots oubliés remplissent les disques : No space left on device · requêtes annulées sur le réplica : conflict with recovery.

Verrous et transactions

```
BEGIN ISOLATION LEVEL SERIALIZABLE;  -- réessayez sur SQLSTATE 40001
SELECT ... FOR UPDATE;               -- verrouille les lignes à mettre à jour
SELECT ... FOR UPDATE SKIP LOCKED;   -- files de jobs sans contention
SELECT ... FOR UPDATE NOWAIT;        -- échoue vite au lieu d'attendre
SHOW default_transaction_isolation;  -- le niveau réellement utilisé
```

Ce que garantit chaque niveau et comment réessayer correctement : niveaux d'isolation en pratique · deadlock detected.

Connexions

```
SHOW max_connections;
SELECT count(*) FROM pg_stat_activity;
SELECT usename, application_name, count(*)
FROM pg_stat_activity GROUP BY 1, 2 ORDER BY 3 DESC;

ALTER ROLE app_user CONNECTION LIMIT 50;   -- plafond par rôle
```

Slots épuisés ? La vraie solution est un pooler, pas une limite plus haute : too many clients.

SQLSTATE courants

`40P01` deadlock detected · `40001` serialization failure · `23505` duplicate key · `53300` too many clients · `57014` query canceled · `55P03` lock timeout · `42P01` relation does not exist · `42501` permission denied · `25P02` transaction aborted · `22P02` invalid input syntax

Les 30, avec causes et solutions : référence des erreurs. Dans psql, `\set VERBOSITY verbose` affiche le code avec chaque erreur.