psql: l'essenziale

```
\l                    elenca i database     \c db     connettiti
\dt  \di  \dv  \df    tabelle / indici / viste / funzioni
\d+ name              descrivi, con indici e storage
\du  ruoli            \dn  schemi            \dx  estensioni
\dp name              privilegi di una tabella
\x                    output espanso (toggle)
\timing               mostra il tempo di ogni query
\e                    modifica la query in $EDITOR
\i file.sql           esegui un file SQL
\copy t TO 'out.csv' CSV HEADER    export lato client
\set VERBOSITY verbose mostra lo SQLSTATE con ogni errore
\watch 2              riesegui l'ultima query ogni 2s
```

Cosa sta girando

```
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;
```

```
-- Chi blocca chi:
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` vecchio = guai.
Approfondimento: vedere le query in esecuzione.

Annullare e terminare

```
SELECT pg_cancel_backend(pid);     -- annulla la query, la sessione resta
SELECT pg_terminate_backend(pid);  -- chiude l'intera connessione
```

Scala in quest'ordine. Mai `kill -9` su un backend dalla shell: riparte l'intero cluster. Dettagli: terminare una query in sicurezza.

Timeout

```
SET statement_timeout = '30s';   -- tetto al tempo totale (incl. attese di lock)
SET lock_timeout = '3s';         -- attesa massima per un lock (ottimo per i DDL)
SET idle_in_transaction_session_timeout = '5min';

BEGIN;
SET LOCAL statement_timeout = '30min';  -- vale solo per questa transazione
-- ... migrazione o report lunghi ...
COMMIT;
```

Quando scattano: statement timeout · lock timeout.

Dimensioni

```
SELECT pg_size_pretty(pg_total_relation_size('t')); -- heap+TOAST+indici
SELECT pg_size_pretty(pg_table_size('t'));          -- senza indici
SELECT pg_size_pretty(pg_indexes_size('t'));        -- solo indici
SELECT pg_size_pretty(pg_database_size(current_database()));
```

La query delle top-20 e i dettagli: dimensioni di tabelle e indici.
Dimensione ≠ dati vivi — vedi bloat.

Manutenzione

```
ANALYZE t;                        -- aggiorna le statistiche del planner
VACUUM (VERBOSE, ANALYZE) t;      -- recupera le righe morte + statistiche
CREATE INDEX CONCURRENTLY idx ON t (col);  -- senza lock in scrittura
REINDEX INDEX CONCURRENTLY idx;

-- sentinella wraparound (allarme ben prima di ~2 miliardi):
SELECT datname, age(datfrozenxid)
FROM pg_database ORDER BY 2 DESC;
```

Il contesto: VACUUM, autovacuum e bloat · emergenza: wraparound.

EXPLAIN

```
EXPLAIN SELECT ...;                    -- solo il piano, non esegue
EXPLAIN (ANALYZE, BUFFERS) SELECT ...; -- ESEGUE la query, tempi reali

-- scritture, in sicurezza:
BEGIN; EXPLAIN (ANALYZE) UPDATE ...; ROLLBACK;
```

Incolla l'output (testo o JSON) nel visualizzatore EXPLAIN: flame graph e warning automatici.

Backup e restore

```
pg_dump -Fc dbname > db.dump        # formato custom: compresso,
pg_restore -j4 -d dbname db.dump    #   restore parallelo con -j
pg_dump -Fc -t mytable dbname > table.dump
pg_dumpall --globals-only > roles.sql   # i ruoli NON stanno in pg_dump
psql dbname < plain.sql             # restore dei dump SQL semplici
```

`pg_dump` è consistente senza bloccare le scritture (gira in una transazione Repeatable Read).

Privilegi (ruolo applicativo tipico)

```
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;

-- anche le tabelle FUTURE (create dal ruolo delle migrazioni):
ALTER DEFAULT PRIVILEGES FOR ROLE migrator IN SCHEMA app
  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO app_user;
```

I GRANT non valgono per le tabelle future — la trappola classica. Approfondimento: permission denied.

Lock e transazioni

```
BEGIN ISOLATION LEVEL SERIALIZABLE;  -- riprova su SQLSTATE 40001
SELECT ... FOR UPDATE;               -- blocca le righe che
aggiornerai
SELECT ... FOR UPDATE SKIP LOCKED;   -- code di job senza contesa
SELECT ... FOR UPDATE NOWAIT;        -- fallisci subito invece di
aspettare
SHOW default_transaction_isolation;  -- il livello a cui giri
davvero
```

Cosa garantisce ogni livello e come riprovare correttamente: livelli di isolamento in pratica · deadlock detected.

Configurazione

```
SHOW work_mem;
SELECT name, setting, source FROM pg_settings
WHERE source <> 'default';        -- cosa è stato personalizzato

ALTER SYSTEM SET work_mem = '64MB';
SELECT pg_reload_conf();          -- alcuni parametri richiedono
invece un riavvio
```

`work_mem` vale per ogni nodo sort/hash, per query — si moltiplica. `max_connections` e `shared_buffers` richiedono un riavvio.

Connessioni

```
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;   -- tetto per ruolo
```

Slot esauriti? La soluzione che dura è un pooler, non un limite più alto: too many clients.

Replica: controlli rapidi

```
SELECT pg_is_in_recovery();            -- sono una replica?
SELECT client_addr, state, replay_lag
FROM pg_stat_replication;              -- sul primario

-- slot che trattengono WAL (active = f è un campanello d'allarme):
SELECT slot_name, active, pg_size_pretty(
  pg_wal_lsn_diff(pg_current_wal_lsn(), restart_lsn)) AS retained
FROM pg_replication_slots;
```

Gli slot dimenticati riempiono i dischi: No space left on device · query annullate sulla replica: conflict with recovery.

SQLSTATE comuni

`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

Tutti e 30, con cause e soluzioni: riferimento errori. In psql, `\set VERBOSITY verbose` mostra il codice con ogni errore.