psql-Grundlagen

```
\l                    Datenbanken auflisten \c db     verbinden
\dt  \di  \dv  \df    Tabellen / Indexe / Views / Funktionen
\d+ name              beschreiben, inkl. Indexe & Storage
\du  Rollen           \dn  Schemata          \dx  Erweiterungen
\dp name              Berechtigungen einer Tabelle
\x                    erweiterte Ausgabe (Umschalter)
\timing               Zeit pro Abfrage anzeigen
\e                    Abfrage im $EDITOR bearbeiten
\i file.sql           SQL-Datei ausführen
\copy t TO 'out.csv' CSV HEADER    Export auf Client-Seite
\set VERBOSITY verbose SQLSTATE bei jedem Fehler anzeigen
\watch 2              letzte Abfrage alle 2s wiederholen
```

Was gerade läuft

```
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;
```

```
-- Wer blockiert wen:
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'` + altes `xact_start` = Ärger. Mehr: laufende Abfragen anzeigen.

Abbrechen & beenden

```
SELECT pg_cancel_backend(pid);     -- Abfrage abbrechen, Sitzung
behalten
SELECT pg_terminate_backend(pid);  -- die ganze Verbindung
schließen
```

In dieser Reihenfolge eskalieren. Niemals `kill -9` auf einen Backend-Prozess: der ganze Cluster startet neu. Details: eine Abfrage sicher beenden.

Timeouts

```
SET statement_timeout = '30s';   -- Obergrenze der Gesamtlaufzeit
(inkl. Lock-Wartezeiten)
SET lock_timeout = '3s';         -- max. Wartezeit auf einen Lock
(ideal für DDL)
SET idle_in_transaction_session_timeout = '5min';

BEGIN;
SET LOCAL statement_timeout = '30min';  -- nur für diese
Transaktion
-- ... lange Migration oder Report ...
COMMIT;
```

Wenn sie zuschlagen: statement timeout · lock timeout.

Größen

```
SELECT pg_size_pretty(pg_total_relation_size('t')); --
Heap+TOAST+Indexe
SELECT pg_size_pretty(pg_table_size('t'));          -- ohne Indexe
SELECT pg_size_pretty(pg_indexes_size('t'));        -- nur Indexe
SELECT pg_size_pretty(pg_database_size(current_database()));
```

Die Top-20-Abfrage und das Kleingedruckte: Tabellen- und Indexgrößen. Größe ≠ lebende Daten — siehe Bloat.

Wartung

```
ANALYZE t;                          -- Planer-Statistiken
aktualisieren
VACUUM (VERBOSE, ANALYZE) t;        -- tote Zeilen freigeben +
Statistiken
CREATE INDEX CONCURRENTLY idx ON t (col);  -- ohne Schreib-Lock
REINDEX INDEX CONCURRENTLY idx;

-- Wraparound-Wache (Alarm lange vor ~2 Milliarden):
SELECT datname, age(datfrozenxid)
FROM pg_database ORDER BY 2 DESC;
```

Hintergrund: VACUUM, Autovacuum und Bloat · Notfall: Wraparound.

EXPLAIN

```
EXPLAIN SELECT ...;                    -- nur der Plan, führt nicht
aus
EXPLAIN (ANALYZE, BUFFERS) SELECT ...; -- FÜHRT die Abfrage AUS,
echte Zeiten

-- Schreibzugriffe, sicher:
BEGIN; EXPLAIN (ANALYZE) UPDATE ...; ROLLBACK;
```

Fügen Sie die Ausgabe (Text oder JSON) in den EXPLAIN-Visualizer ein: Flame Graph und automatische Warnungen.

Backup & Wiederherstellung

```
pg_dump -Fc dbname > db.dump        # Custom-Format: komprimiert,
pg_restore -j4 -d dbname db.dump    #   parallele Wiederherstellung
mit -j
pg_dump -Fc -t mytable dbname > table.dump
pg_dumpall --globals-only > roles.sql   # Rollen sind NICHT in
pg_dump
psql dbname < plain.sql             # Plain-SQL-Dumps
wiederherstellen
```

`pg_dump` ist konsistent, ohne Schreibzugriffe zu blockieren (läuft in einer Repeatable-Read-Transaktion).

Berechtigungen (typische Anwendungsrolle)

```
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;

-- auch KÜNFTIGE Tabellen (von der Migrationsrolle erstellt):
ALTER DEFAULT PRIVILEGES FOR ROLE migrator IN SCHEMA app
  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO app_user;
```

GRANTs gelten nicht für künftige Tabellen — die klassische Falle. Mehr: permission denied.

Konfiguration

```
SHOW work_mem;
SELECT name, setting, source FROM pg_settings
WHERE source <> 'default';        -- was angepasst wurde

ALTER SYSTEM SET work_mem = '64MB';
SELECT pg_reload_conf();          -- manche Parameter erfordern
einen Neustart
```

`work_mem` gilt pro Sort/Hash-Knoten, pro Abfrage — es multipliziert sich. `max_connections` und `shared_buffers` erfordern einen Neustart.

Replikation: Schnellchecks

```
SELECT pg_is_in_recovery();            -- bin ich ein Replikat?
SELECT client_addr, state, replay_lag
FROM pg_stat_replication;              -- auf dem Primärserver

-- Slots, die WAL festhalten (active = f ist ein Warnsignal):
SELECT slot_name, active, pg_size_pretty(
  pg_wal_lsn_diff(pg_current_wal_lsn(), restart_lsn)) AS retained
FROM pg_replication_slots;
```

Vergessene Slots füllen Platten: No space left on device · abgebrochene Abfragen auf dem Replikat: conflict with recovery.

Sperren & Transaktionen

```
BEGIN ISOLATION LEVEL SERIALIZABLE;  -- bei SQLSTATE 40001
wiederholen
SELECT ... FOR UPDATE;               -- die zu ändernden Zeilen
sperren
SELECT ... FOR UPDATE SKIP LOCKED;   -- Job-Queues ohne Konkurrenz
SELECT ... FOR UPDATE NOWAIT;        -- sofort fehlschlagen statt
warten
SHOW default_transaction_isolation;  -- die tatsächlich genutzte
Stufe
```

Was jede Stufe garantiert und wie man korrekt wiederholt: Isolationsstufen in der Praxis · deadlock detected.

Verbindungen

```
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;   -- Limit pro Rolle
```

Slots erschöpft? Die dauerhafte Lösung ist ein Pooler, kein höheres Limit: too many clients.

Häufige SQLSTATEs

`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

Alle 30, mit Ursachen und Lösungen: Fehlerreferenz. In psql zeigt `\set VERBOSITY verbose` den Code bei jedem Fehler.