

PostgreSQL Indexing & Query Optimization

Production Guide to Index Types, EXPLAIN Analysis and Planner Settings

⚡ PostgreSQL 15 / 16 / 17 compatible

Updated: October 2026 · Production-grade



PG

01 Index Types FUNDAMENTALS



TYPE	OPERATORS	WHEN?
B-tree	= < > BETWEEN IN, LIKE 'abc%'	Default. Equality, range, sorting (ORDER BY).
Hash	=	Equality only. WAL-logged since PG10; rarely beats B-tree.
GIN	@> ? && @@	JSONB, array, full-text, pg_trgm (LIKE '%x%').
GiST	&& @> <->	Geometry (PostGIS), ranges, KNN, exclusion constraints.
SP-GiST	<< >> ~=	Unbalanced data: IP (inet), quad-tree, prefix search.
BRIN	= < >	Huge physically ordered tables (time series). Tiny size.
Bloom	=	Extension. Many columns, arbitrary equality combinations.

CHOOSE THE TYPE — EXAMPLES

```
-- B-tree (default)
CREATE INDEX idx_orders_created ON orders (created_at);

-- JSONB containment search
CREATE INDEX idx_evt_payload ON events
  USING gin (payload jsonb_path_ops);

-- Trigram for LIKE '%word%'
CREATE EXTENSION IF NOT EXISTS pg_trgm;
CREATE INDEX idx_cust_name_trgm ON customers
  USING gin (name gin_trgm_ops);

-- 1-billion-row log table: index of a few MB
CREATE INDEX idx_logs_ts_brin ON logs
  USING brin (ts) WITH (pages_per_range = 64);

-- Range overlap (reservations)
CREATE INDEX idx_booking_period ON bookings
  USING gist (period);
```

INDEX SIZES

```
SELECT indexrelid::regclass AS idx,
       pg_size_pretty(pg_relation_size(indexrelid)) AS size
FROM   pg_index WHERE indrelid = 'orders'::regclass;
```

BRIN rule: Only effective when column values strongly correlate with physical row order. Check: `pg_stats.correlation ≈ ±1`.

02 Reading EXPLAIN CRITICAL



PROPER USAGE

```
EXPLAIN (ANALYZE, BUFFERS, SETTINGS, VERBOSE)
SELECT * FROM orders WHERE customer_id = 42;

-- Analyze DML without keeping its effects
BEGIN; EXPLAIN ANALYZE DELETE FROM t WHERE ...; ROLLBACK;
```

PLAN NODE	MEANING / WHEN IS IT A PROBLEM?
Seq Scan	Reads the whole table. Normal on small tables; on big tables with a selective filter, an index is missing.
Index Scan	Index + heap access. Ideal for queries returning few rows.
Index Only Scan	Skips the heap. High Heap Fetches → VACUUM.
Bitmap Heap Scan	Medium selectivity; multiple indexes combined via BitmapAnd/Or.
Nested Loop	Small outer set + index on inner side. Disaster if outer set is large.
Hash Join	Large equality joins. Batches > 1 → work_mem too low.
Merge Join	Efficient when both inputs are already sorted (index).
Sort / HashAgg	external merge Disk: → spilling to disk, raise work_mem.

WHAT TO LOOK FOR IN THE OUTPUT

```
Index Scan using idx_orders_cust on orders
(cost=0.43..8.45 rows=5 width=64)
(actual time=0.021..0.030 rows=48210 loops=1)
Index Cond: (customer_id = 42)
Rows Removed by Filter: 0
Buffers: shared hit=1204 read=3310
Planning Time: 0.112 ms
Execution Time: 41.733 ms
```

SIGNAL	DIAGNOSIS
estimated rows ≠ actual	10× gap → stale statistics: ANALYZE
Rows Removed by Filter	If high, the filter column is not indexed
shared read >> hit	Data not cached, disk I/O dominates
loops=N	Time × loops = real total cost

✓ **Best Practice:** Capture the plan with `FORMAT JSON` and visualize it on [explain.dalibo.com](#) or [explain.depesz.com](#). PG17 added the `SERIALIZE` and `MEMORY` options.

03 Index Creation Strategies



TECHNIQUE	USAGE
Composite	Equality columns first, range column last.
Partial	Subset via WHERE — small, fast index.
Expression	Filters using functions like <code>lower(email)</code> .
Covering	<code>INCLUDE</code> (PG11+) → Index Only Scan.
CONCURRENTLY	Build without blocking writes (production).

COMPOSITE — COLUMN ORDER

```
-- WHERE status = 'paid' AND created_at > now()-'7d'
CREATE INDEX idx_o_status_created
ON orders (status, created_at); -- ✓ equality → range
```

PARTIAL & EXPRESSION

```
-- Active rows only (3% of rows)
CREATE INDEX idx_o_pending ON orders (created_at)
WHERE status = 'pending';

-- Unique for soft-delete tables
CREATE UNIQUE INDEX uq_users_email ON users (lower(email))
WHERE deleted_at IS NULL;
-- Query must use the EXACT expression:
SELECT * FROM users WHERE lower(email) = 'a@b.com'
AND deleted_at IS NULL;
```

COVERING INDEX (INCLUDE)

```
CREATE INDEX idx_o_cust_cov ON orders (customer_id)
INCLUDE (total, status);
-- SELECT total, status ... WHERE customer_id = $1
-- → Index Only Scan (heap skipped)
```

SAFE CREATION IN PRODUCTION

```
SET maintenance_work_mem = '2GB';
CREATE INDEX CONCURRENTLY idx_o_cust ON orders (customer_id);
-- Progress tracking (PG12+)
SELECT phase, blocks_done, blocks_total
FROM   pg_stat_progress_create_index;
```

⚠ **Warning:** `CONCURRENTLY` cannot run inside a transaction block. If it fails, an `INVALID` index is left behind and still adds write overhead — clean up with `DROP INDEX CONCURRENTLY`.

04
 Planner & Cost Parameters

Parameter	Default	Recommended
random_page_cost Random page read cost. Lower it on SSD → indexes get preferred.	4.0	1.1 (SSD)
effective_cache_size Estimate of OS + shared_buffers cache. Allocates no memory.	4GB	RAM × 0.75
work_mem Memory per Sort/Hash operation. Multiplied by connections × nodes!	4MB	16–64MB
hash_mem_multiplier work_mem multiplier for hash operations.	2.0	2.0–4.0
effective_io_concurrency Bitmap Heap Scan prefetch depth.	1	200 (SSD)
default_statistics_target Histogram / MCV sample size.	100	100–500
jit Adds compile latency to short OLTP queries.	on	off (OLTP)
max_parallel_workers_per_gather Parallel workers per query.	2	2–4

plan_cache_mode	auto	auto
-----------------	------	------

Prepared statements: generic / custom plan.

PostgreSQL.conf (64 GB RAM · SSD · OLTP)

shared_buffers	= 16GB
effective_cache_size	= 48GB
work_mem	= 32MB
maintenance_work_mem	= 2GB
random_page_cost	= 1.1
effective_io_concurrency	= 200
default_statistics_target	= 200
jit	= off

Session-Level Testing

```
-- Why isn't the index chosen? Force & compare
SET enable_seqscan = off;
EXPLAIN ANALYZE SELECT ...;
RESET enable_seqscan;
```

```
-- For a single heavy report
SET LOCAL work_mem = '512MB'; -- this TX only
```

⚠️ ⚠️ **Warning:** `enable_*` parameters are for diagnosis only — never disable them globally. Raising `work_mem` globally can cause OOM; set it per role: `ALTER ROLE report SET work_mem='256MB'.`

05
 Statistics & ANALYZE

pg_stats column	Meaning
null_frac	Fraction of NULLs
n_distinct	Distinct values (negative = fraction of rows)
most_common_vals	Most common values (MCV) and frequencies
histogram_bounds	Distribution bounds for range estimates
correlation	Physical order correlation (±1 → BRIN / cheap Index Scan)

🔍 Inspect Column Statistics

```
SELECT attname, null_frac, n_distinct, correlation,
       most_common_vals, most_common_freqs
FROM   pg_stats
WHERE  tablename = 'orders' AND attname = 'status';
```

⚙️ Per-Column Precision

```
-- More samples for a skewed column
ALTER TABLE orders ALTER COLUMN customer_id
  SET STATISTICS 1000;
ANALYZE orders;
```

🔗 Extended Statistics — Correlated Columns

```
-- city and zip are dependent → planner multiplies
-- selectivities and underestimates rows
CREATE STATISTICS st_addr (dependencies, ndistinct, mcv)
  ON city, zip_code FROM addresses;
ANALYZE addresses;
```

```
SELECT statistics_name, dependencies
FROM   pg_stats_ext WHERE tablename = 'addresses';
```

🔄 Last Analyze & Modified Rows

```
SELECT relname, n_mod_since_analyze,
       last_analyze, last_autoanalyze
FROM   pg_stat_user_tables
ORDER BY n_mod_since_analyze DESC LIMIT 10;

-- Large table: more frequent autoanalyze
ALTER TABLE orders SET (
  autovacuum_analyze_scale_factor = 0.02,
  autovacuum_analyze_threshold   = 1000);
```

⚠️ ⚠️ **Warning:** After `pg_upgrade` or large bulk loads statistics are missing/stale. Run `vacuumdb --all --analyze-in-stages` immediately, or plans will collapse.

💡 **Temp tables** are never analyzed by autovacuum — run `ANALYZE` manually after loading them.

06
 Query Anti-Patterns

⚡ Common Mistake

✗ BAD	✓ GOOD
WHERE date(created_at) = '2026-10-13'	WHERE created_at >= '2026-10-13' AND created_at < '2026-10-14'
WHERE lower(email) = \$1 (index is on email)	Expression index: (lower(email))
WHERE phone = 5551234 (phone is text → implicit cast)	WHERE phone = '5551234'
WHERE name LIKE '%ali%'	pg_trgm GIN index
WHERE a = 1 OR b = 2	Separate indexes (BitmapOr) or UNION ALL
WHERE id NOT IN (SELECT ...)	WHERE NOT EXISTS (...)
SELECT * ...	Only the columns you need
COUNT(*) > 0 check	EXISTS (SELECT 1 ...)

▶▶ Keyset Pagination Instead of Offset

```
-- ✗ Page 5000: 100,000 rows read and discarded
SELECT * FROM orders ORDER BY id LIMIT 20 OFFSET 100000;

-- ✓ Constant cost, uses the index
SELECT id, total FROM orders
WHERE (created_at, id) < (:last_ts, :last_id)
ORDER BY created_at DESC, id DESC
LIMIT 20;
```

✂️ Batch Query Instead of N+1

```
-- ✗ 1000 times in a loop: WHERE id = $1
-- ✓ In a single round trip:
SELECT * FROM products WHERE id = ANY($1::bigint[]);
```

🏗️ CTE & Materialization (PG12+)

```
-- PG12+: CTEs are inlined by default
WITH recent AS NOT MATERIALIZED (
  SELECT * FROM orders WHERE created_at > now() - '1 day'::interval
)
SELECT * FROM recent WHERE customer_id = 42;
```

⚠️ ⚠️ **Warning:** If a `NOT IN` subquery returns a single NULL, the result is **empty** — both wrong and slow. Always prefer `NOT EXISTS`.

87 Index Maintenance & Diagnostics

UNUSED INDEXES

```
SELECT s.schemaname, s.relname AS table, s.indexrelname AS idx,
       pg_size_pretty(pg_relation_size(s.indexrelid)) AS size
FROM   pg_stat_user_indexes s
JOIN   pg_index i USING (indexrelid)
WHERE  s.idx_scan = 0
       AND NOT i.indisunique AND NOT i.indisprimary
ORDER BY pg_relation_size(s.indexrelid) DESC;
```

DUPLICATE INDEXES

```
SELECT indrelid::regclass AS table,
       array_agg(indexrelid::regclass) AS duplicates
FROM   pg_index
GROUP BY indrelid, indkey::text, indclass::text,
         coalesce(indexprs::text, ''), coalesce(indpred::text, '')
HAVING count(*) > 1;
```

INVALID INDEXES

```
SELECT indexrelid::regclass FROM pg_index WHERE NOT indisvalid;
```

REMOVING BLOAT — REINDEX

```
-- PG12+: rebuild without blocking
REINDEX INDEX CONCURRENTLY idx_orders_created;
REINDEX TABLE CONCURRENTLY orders;

-- Bloat estimate (pgstattuple extension)
CREATE EXTENSION IF NOT EXISTS pgstattuple;
SELECT avg_leaf_density, leaf_fragmentation
FROM   pgstattuple('idx_orders_created');
-- avg_leaf_density < 50 → REINDEX candidate
```

MOST EXPENSIVE QUERIES (PG_STAT_STATEMENTS)

```
SELECT round(total_exec_time::numeric, 0) AS total_ms,
       calls, round(mean_exec_time::numeric, 2) AS mean_ms,
       rows, shared_blks_read, left(query, 80) AS query
FROM   pg_stat_statements
ORDER BY total_exec_time DESC LIMIT 10;
```

⚠️ ⚠️ **Warning:** `idx_scan = 0` only counts since the last stats reset. Before dropping, **check replicas too** and account for month-end reports.

88 Optimization Checklist

SLOW QUERY WORKFLOW

CHECK ITEM	PRIORITY
✓ pg_stat_statements installed, top 10 queries known ORDER BY total_exec_time DESC	CRITICAL
✓ Plan reviewed with EXPLAIN (ANALYZE, BUFFERS) estimated rows ≈ actual rows ?	CRITICAL
✓ Foreign key columns are indexed JOIN and ON DELETE CASCADE performance	CRITICAL
✓ Statistics up to date pg_stat_user_tables.last_autoanalyze	CRITICAL
✓ random_page_cost tuned for SSD SHOW random_page_cost; -- 1.1	IMPORTANT
✓ Unused / duplicate indexes removed pg_stat_user_indexes.idx_scan = 0	IMPORTANT
✓ No INVALID indexes pg_index WHERE NOT indisvalid	IMPORTANT
✓ No sort/hash spilling to disk log_temp_files = 0 · temp_blks_written	IMPORTANT
✓ Slow query log and auto_explain active log_min_duration_statement = 500	RECOMMENDED
✓ Index bloat monitored regularly pgstattuple() · REINDEX CONCURRENTLY	RECOMMENDED
CRITICAL fix now IMPORTANT this sprint RECOMMENDED plan	

FIND UNINDEXED FOREIGN KEYS

```
-- dbaguide.com · missing FK index finder
SELECT c.conrelid::regclass AS table,
       c.conname AS fk,
       pg_get_constraintdef(c.oid) AS definition
FROM   pg_constraint c
WHERE  c.contype = 'f'
       AND NOT EXISTS (
         SELECT 1 FROM pg_index i
         WHERE  i.indrelid = c.conrelid
               AND (i.indkey::int2[])[0:cardinality(c.conkey)-1]
                   OPERATOR(pg_catalog.@>) c.conkey);
```

AUTO_EXPLAIN — AUTO-LOG SLOW PLANS

```
shared_preload_libraries = 'pg_stat_statements,auto_explain'
auto_explain.log_min_duration = '1s'
auto_explain.log_analyze      = on
auto_explain.log_buffers      = on
auto_explain.log_timing       = off # overhead ↓
auto_explain.sample_rate      = 0.1
log_temp_files               = 0
```

✓ **Workflow:** `pg_stat_statements` → most expensive query → `EXPLAIN (ANALYZE, BUFFERS)` → estimate error? (`ANALYZE`) · Seq Scan? (index) · Disk sort? (`work_mem`) → measure again.