Finding & Fixing Missing Indexes in Under 10 Minutes

> "Most slow queries are just an index away from being fast." — every seasoned DBA

Unindexed columns turn your sleek Postgres into a table-scanning slug. This crash course shows how to detect, prioritize, and create the right indexes before your next coffee refill. No extensions needed — just core catalog views (pg_stat_*, pg_index, pg_constraint) and a dash of SQL.

1 · Spot Sequential Scans at Scale (1 min)

```
SELECT relname            AS table,
    seq_scan          AS seq_scans,
    idx_scan          AS idx_scans,
    round(100*seq_scan/NULLIF(seq_scan+idx_scan,0),2) AS seq_pct,
    n_live_tup        AS rowsFROM   pg_stat_user_tablesWHERE  n_live_tup > 10000        -- big
enough to matterORDER  BY seq_scan DESC
LIMIT 15;
```

- **seq_pct > 10 %** on tables > 10 k rows → likely missing or unused indexes.
- Small tables can live with seq scans; indexes add overhead.

2 · Pinpoint Offending Columns with EXPLAIN (2 min)

1. Get top queries for the table:

```
SELECT queryFROM   pg_stat_statementsWHERE  query LIKE '%big_table%'ORDER  BY
total_exec_time DESC
LIMIT 5;
```

1. Run EXPLAIN (ANALYZE, BUFFERS) on the slowest query.

Look for:

- Seq Scan with a filter → **add normal / partial index**
- Bitmap Heap Scan removing many rows → **covering index**
- Joins lacking Index Scan on FK side → **index the FK column**

Need help reading plans? See **EXPLAIN ANALYZE Demystified**.

3 · Automatic Foreign-Key Audit (3 min)

Find FK columns missing indexes:

```
Copy

WITH fks AS (
  SELECT conrelid, conname, conkey
  FROM   pg_constraint
  WHERE  contype = 'f'
), ix AS (
  SELECT indrelid, indkey
  FROM   pg_index
  WHERE  indisvalid AND indpred IS NULL
)SELECT  n.nspname || '.' || c.relname AS table,
       f.conname                  AS fk_name,
       array_to_string(ARRAY(
         SELECT a.attname
         FROM   pg_attribute a
         WHERE  a.attrelid = f.conrelid
           AND  a.attnum   = ANY(f.conkey)
         ORDER  BY array_position(f.conkey, a.attnum)
       ), ',') AS fk_colsFROM    fks fJOIN    pg_class c ON c.oid = f.conrelidJOIN    pg_namespace n ON
n.oid = c.relnamespace
LEFT JOIN ix ON ix.indrelid = f.conrelid
         AND f.conkey   = ix.indkey[0:array_length(f.conkey,1)-1]WHERE   ix.indrelid IS NULL   -- no
supporting indexORDER BY table;
```

Result lists FK constraints that need indexes.

4 · Generate Index DDL (1 min)

Auto-craft index statements:

```
Copy

SELECT format(
  'CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_%I_%s ON %s USING btree (%s);',
  relname,              -- table name
  string_agg(col, '_'),  -- suffix
  relid::regclass,      -- schema.table
  string_agg(col, ', ')  -- column list
) AS ddlFROM   your_fk_missing_index_queryGROUP  BY relid, relname;
```

Run the generated `CREATE INDEX CONCURRENTLY` outside a transaction block.

5 · Partial & Covering Index Patterns (2 min)

Scenario	Index Recipe	Why
Soft deletes (WHERE deleted_at IS NULL)	CREATE INDEX … WHERE deleted_at IS NULL	Smaller, faster scans

Recent rows (created_at > NOW()-30d) | CREATE INDEX … ON … (created_at) WHERE created_at > … | Keeps old data out of index

Filter + sort (status='paid' ORDER BY date) | CREATE INDEX … (status, created_at DESC) | Supports filter *and* order by

Covering lookup (select few cols) | CREATE INDEX … (id) INCLUDE (col1, col2) | Enables index- only scan

Use `pg_size_pretty(pg_relation_size('index_name'))` to verify size savings.

6 · Validate Impact (1 min)

```
Copy

EXPLAIN (ANALYZE, BUFFERS)<your query again>;
```

`Seq Scan` should disappear and time should drop. Monitor:

```
Copy

SELECT idx_scan, seq_scanFROM   pg_stat_user_tablesWHERE  relname = 'big_table';
```

Expect `idx_scan` to climb after deployment.

7 · Gotchas & Best Practices

1. **Don't over-index** — writes pay the price; review `idx_scan = 0` quarterly.
2. **CREATE INDEX CONCURRENTLY** in prod to avoid write locks.
3. Drop unused indexes:

```
Copy

SELECT relnameFROM   pg_stat_user_indexesWHERE  idx_scan = 0
  AND  pg_relation_size(indexrelid) > 10*8192;
```

1. Avoid duplicates (compare `pg_index.indkey`).
2. Use `fillfactor` for hot rows (`ALTER INDEX … SET (fillfactor=90)`).

TL;DR

1. Use `pg_stat_user_tables` to locate heavy seq scans.
2. `EXPLAIN ANALYZE` reveals columns to index.
3. Audit FKs for missing indexes.
4. Create indexes **concurrently**; favour partial and covering patterns.
5. Re-run stats and queries to confirm wins.