EcomLabs

STEP-BY-STEP GUIDE

EXPLAIN ANALYZE and indexes in Postgres

How to read an EXPLAIN ANALYZE plan, create the right index for WHERE and ORDER BY, measure before and after and find the indexes nobody uses. With the test table and the measured results.

FOR	Apps on PostgreSQL with a slow query
TESTED WITH	Node 22.22.3 · PGlite 0.5.10 (PostgreSQL 18.6) · Windows 11
UPDATED	November 12, 2026
DOCS	postgresql.org/docs/current/using-explain.html
ARTICLE	ecomlabs.dev/g/indices-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What you will do
Step 01	The test table
Step 02	Reading the plan
Step 03	Create the index and measure
Step 04	Review the indexes you already have
When you finish	Checklist

BEFORE YOU START

What you will do

A four-step loop: measure the query with `EXPLAIN ANALYZE`, read the plan, create the index it asks for and measure again. We tested it on November 12, 2026 on PGlite 0.5.10 (PostgreSQL 18.6). `EXPLAIN ANALYZE` runs the query: use it on a copy or a test database, and wrap any `INSERT`, `UPDATE` or `DELETE` in `BEGIN` and `ROLLBACK`.

STEP 01

The test table

200,000 invented orders, 5,000 customers, no index besides the primary key. Swap in your own table; what matters is a volume close to the real one:

```
CREATE TABLE orders (
  id          bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  customer_id int           NOT NULL,
  status      text          NOT NULL,
  created_at  timestamptz   NOT NULL,
  total       numeric(10,2) NOT NULL
);

INSERT INTO orders (customer_id, status, created_at, total)
SELECT (i * 7919) % 5000 + 1,
       (ARRAY['paid','paid','paid','fulfilled','refunded'])[i % 5 + 1],
       timestamptz '2026-01-01' + i * interval '2 minutes',
       round(((i * 37) % 20000) / 100.0, 2)
FROM generate_series(1, 200000) AS i;

ANALYZE orders;
```

STEP 02

Reading the plan

A customer's last 20 orders:

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT id, created_at, total
FROM orders
WHERE customer_id = 4242
ORDER BY created_at DESC
LIMIT 20;
```

PLAN LINE	WHAT IT SAYS	BAD SIGN
`Seq Scan`	It read the whole table	To return a few rows
`Rows Removed by Filter`	Rows read and discarded	Much larger than `rows`
`Sort`	It sorted the rows after reading them	Over many rows, with a small `LIMIT`
`actual time` and `loops`	Milliseconds per run and how many times it ran	Large time times `loops`
`Buffers: shared hit/read`	Pages read from cache and from disk	Thousands of pages for a few rows

Our result with no index: `Seq Scan` with `Rows Removed by Filter: 199960`, 1667 buffers and 15.432 ms.

STEP 03

Create the index and measure

The `=` columns of the `WHERE` first, then the `ORDER BY` ones:

```
CREATE INDEX orders_customer_created_idx ON orders (customer_id, created_at DESC);
ANALYZE orders;
```

Run the same query again, three times, and keep the third. Compare `Buffers` and milliseconds. In production use `CREATE INDEX CONCURRENTLY`: it does not block writes, but it takes longer, cannot run inside a transaction and, if it fails, leaves an `INVALID` index you must drop and recreate.

VARIANT	READ NODE	BUFFERS	EXECUTION TIME
A · no index	Seq Scan + Sort	1667	15.432 ms
B · `(customer_id)`	Bitmap Heap Scan + Sort	42	0.468 ms
D · `(created_at, customer_id)`	Index Scan Backward	400	3.596 ms
C · `(customer_id, created_at DESC)`	Index Scan, no Sort	24	0.123 ms

Inserting 100,000 rows took 703 ms (median of three) with only the primary key and 1257 ms with three more indexes: every index charges on writes.

STEP 04

Review the indexes you already have

The first query lists indexes with no recorded scan (unique ones excluded); the second, the ones left `INVALID` by a failed `CONCURRENTLY` build:

```
SELECT s.relname AS table_name, s.indexrelname AS index_name, s.idx_scan,
       pg_size_pretty(pg_relation_size(s.indexrelid)) AS size
FROM pg_stat_user_indexes s
JOIN pg_index i ON i.indexrelid = s.indexrelid
WHERE s.idx_scan = 0 AND NOT i.indisunique
ORDER BY pg_relation_size(s.indexrelid) DESC;

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

> **CAUTION**
>
> Statistics are reset after a crash, an immediate shutdown, a base backup or point-in-time recovery. Look at a representative period, month-end reports included, before dropping, and keep the `CREATE INDEX` statement at hand to recreate it.

WHEN YOU FINISH

Checklist

- [ ] You read `actual time`, `Rows Removed by Filter` and `Buffers`, not just `cost`.
- [ ] The index has the `=` columns first and the `ORDER BY` columns after.
- [ ] You run `ANALYZE` and compare before and after with the same data volume.
- [ ] In production, `CONCURRENTLY` and a check for `INVALID` indexes.
- [ ] `pg_stat_user_indexes` reviewed over a representative period before dropping.