EcomLabs

STEP-BY-STEP GUIDE

The N+1 query problem: before and after

A route that runs 101 queries, the same route with 1 or 2, and how to find the pattern in your own project: a per-request query counter, the PostgreSQL log and pg_stat_statements. With a checklist.

FOR	Node.js APIs on PostgreSQL, with hand-written SQL (postgres.js) or an ORM
TESTED WITH	Node 22.22.3 · PGlite 0.5.10 (PostgreSQL 18.6) · postgres.js 3.4.9 · Drizzle ORM 0.45.4 and 1.0.0-rc.5
UPDATED	November 19, 2026
DOCS	postgresql.org/docs/current/pgstatstatements.html
ARTICLE	ecomlabs.dev/g/consultas-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	How to recognize an N+1
Step 01	Sample data
Step 02	Before and after
Step 03	Find it in your project: queries per request
Step 04	Find it in the database: the log and pg_stat_statements
Step 05	Measure before and after
When you finish	Checklist

BEFORE YOU START

How to recognize an N+1

An N+1 is one query to fetch a list plus one more query for every item in that list. Three signs give it away:

- The number of queries a route runs grows with the size of the list: 10 orders give 11 queries, 100 give 101.
- In the log, the same statement repeats with a different value in `$1`.
- In `pg_stat_statements`, one statement has far more `calls` than the list query, with few rows per call.

STEP 01

Sample data

Create two tables in a test database: 1,000 orders with 1 to 5 items each. You need Node.js 21 or later and postgres.js. Everything ran on November 19, 2026 against PGlite 0.5.10 (PostgreSQL 18.6) over TCP.

```
CREATE TABLE orders (id bigint PRIMARY KEY, created_at timestamptz NOT NULL, total numeric(10,2) NOT NULL);
CREATE TABLE order_items (id bigint PRIMARY KEY, order_id bigint NOT NULL REFERENCES orders(id),
  sku text NOT NULL, qty int NOT NULL, price numeric(10,2) NOT NULL);
CREATE INDEX order_items_order_id_idx ON order_items (order_id); -- the foreign key does not create it

INSERT INTO orders SELECT g, timestamptz '2026-11-01' + g * interval '1 minute', 0 FROM generate_series(1,
1000) g;
INSERT INTO order_items
  SELECT row_number() OVER (), o.id, 'SKU-' || (o.id * 7 + n) % 300, 1 + n % 3, 100 + (o.id + n) % 50
  FROM orders o, generate_series(1, 1 + (o.id % 5)::int) n;
```

The 100 most recent orders (901 to 1000) have 300 items: that is what every version must return.

STEP 02

Before and after

```
// GET /api/orders, N+1 version: 101 queries
const orders = await sql`SELECT id, created_at, total FROM orders ORDER BY id DESC LIMIT 100`;
for (const o of orders) {
  o.items = await sql`SELECT sku, qty, price FROM order_items WHERE order_id = ${o.id} ORDER BY id`;
}
return orders;
```

One query for the list and 100 inside the `for`, in a row. Look for the same shape in your code: an `await` on the database inside a loop, a `.map` that runs queries, or an ORM relation read inside a loop.

After, with `ANY`: every id travels in a single array parameter.

```
// GET /api/orders, ANY version: 2 queries
const orders = await sql`SELECT id, created_at, total FROM orders ORDER BY id DESC LIMIT 100`;
const ids = orders.map((o) => o.id);
const items = await sql`
  SELECT order_id, sku, qty, price FROM order_items
  WHERE order_id = ANY(${ids}::bigint[]) ORDER BY id`;

const byOrder = Map.groupBy(items, (i) => i.order_id);
for (const o of orders) o.items = (byOrder.get(o.id) ?? []).map(({ order_id, ...i }) => i);
return orders;
```

With a JOIN (1 query), limit the orders in a subquery, `FROM (SELECT * FROM orders ORDER BY id DESC LIMIT 100) o LEFT JOIN order_items i ON i.order_id = o.id`: a `LIMIT 100` on the JOIN counts rows, and in the test it returned only 33 orders.

With Drizzle ORM 0.45, declare the relations with `relations()`, pass them in `schema` and call `db.query.orders.findMany({ limit: 100, with: { items: true } })`: it sent a single query. In Drizzle 1.0 (currently 1.0.0-rc.5) they are declared with `defineRelations`; in Prisma, the equivalent is `include` (we did not run it).

STEP 03

Find it in your project: queries per request

This file counts each request's queries with `AsyncLocalStorage` and postgres.js's `debug` function, which is called for every query. When a request goes over 10, it logs a warning with the statement that repeated the most:

```
// db.js: counts the queries of each request (postgres.js + AsyncLocalStorage)
import { AsyncLocalStorage } from "node:async_hooks";
import postgres from "postgres";

const requestStore = new AsyncLocalStorage();
const LIMIT = 10; // warn above this many queries in one request

export const sql = postgres(process.env.DATABASE_URL, {
  debug: (_connection, query) => {
    const r = requestStore.getStore();
    if (r) r.queries.push(query);
  },
});

// Wraps a node:http handler and warns when one request runs more than LIMIT queries.
export function countQueries(label, handler) {
  return (req, res) => {
    const r = { queries: [] };
    res.on("finish", () => {
      if (r.queries.length <= LIMIT) return;
      const [query, runs] = Object.entries(Object.groupBy(r.queries, (q) => q))
        .sort((a, b) => b[1].length - a[1].length)[0];
      console.warn(`[n+1?] ${label}: ${r.queries.length} queries, ${runs.length}x ${query.replace(/\s+/g, "
```

```
")}`);
    });
    requestStore.run(r, () => handler(req, res));
  };
}
```

Wrap each route: `http.createServer(countQueries("GET /orders", handler))`. With the before version it printed this; with the after version, nothing, because it ran 2 queries:

```
[n+1?] GET /orders (before): 101 queries, 100x SELECT sku, qty, price FROM order_items WHERE order_id = $1
ORDER BY id
```

postgres.js's first connection also queries the array types: if your first request shows one extra query, that is it. In Express, the same body goes in a middleware that calls `next()` inside `requestStore.run`; with Drizzle you can count in `logger.logQuery` instead of `debug`, but not in both. Use it in development and tests.

STEP 04

Find it in the database: the log and pg_stat_statements

To see one request in the PostgreSQL log, lower the threshold for your test session only (it takes a superuser or the `SET` privilege on the parameter):

```
SET log_min_duration_statement = 0;  -- 0 logs every statement; -1 (default) none
```

Each N+1 query takes under a millisecond: a 250 ms threshold never logs it. Look for the same statement with a different `Parameters: $1` over and over. For all traffic, enable `pg_stat_statements`: add `shared_preload_libraries = 'pg_stat_statements'` to `postgresql.conf`, restart, and run `CREATE EXTENSION pg_stat_statements;` in the database. On a managed database, follow your provider's documentation. Then:

```
SELECT pg_stat_statements_reset();   -- zero the counters
-- call the route once
SELECT calls, rows, round(mean_exec_time::numeric, 3) AS ms, query
FROM pg_stat_statements
ORDER BY calls DESC
LIMIT 10;
```

With the before route, the top row was the `order_items` query with `calls = 100` and `rows = 300`. After the change, that row is gone and there is one query with `ANY($1::bigint[])` and `calls = 1`.

> **CAUTION**
>
> `pg_stat_statements_reset()` wipes the statistics for the whole server and, by default, only a superuser can run it. Do it on a test database, not in production, where you would lose the history your team relies on.

STEP 05

Measure before and after

Write down two numbers per version: queries per request and route time, with the database on another machine as in production. To estimate your case before measuring: queries × round-trip time to your database, which you measure with a `SELECT 1` from your application server. In our test (illustrative: median of 30 runs at 1.5 ms per trip), the N+1 version took 161.5 ms and the `ANY` one 4.7 ms: 101 × 1.5 ms is about 150 ms of waiting alone.

WHEN YOU FINISH

Checklist

- [ ] No route runs a query inside a loop over a list.
- [ ] The per-request query counter is on in development and warns above 10.
- [ ] Relations load with a JOIN, with `= ANY($1)` or with `with` / `include`.
- [ ] Lists of ids travel as an array, not as one parameter per id.
- [ ] The foreign key column has an index.
- [ ] You wrote down queries and time before and after the change.