GUÍA PASO A PASO

El problema N+1: antes y después

Una ruta que hace 101 consultas, la misma ruta con 1 o 2, y cómo encontrar el patrón en tu propio proyecto: un contador de consultas por petición, el log de PostgreSQL y pg_stat_statements. Con la checklist.

PARA	APIs en Node.js sobre PostgreSQL, con SQL a mano (postgres.js) o con un ORM
PROBADO CON	Node 22.22.3 · PGlite 0.5.10 (PostgreSQL 18.6) · postgres.js 3.4.9 · Drizzle ORM 0.45.4 y 1.0.0-rc.5
ACTUALIZADA	19 de noviembre de 2026
DOCUMENTACIÓN	postgresql.org/docs/current/pgstatstatements.html
ARTÍCULO	ecomlabs.dev/g/consultas
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Cómo se reconoce un N+1
Paso 01	Datos de ejemplo
Paso 02	Antes y después
Paso 03	Detéctalo en tu proyecto: consultas por petición
Paso 04	Detéctalo en la base: log y pg_stat_statements
Paso 05	Mide antes y después
Al terminar	Checklist

ANTES DE EMPEZAR

Cómo se reconoce un N+1

Un N+1 es una consulta para traer una lista y una consulta más por cada elemento de esa lista. Tres señales te lo delatan:

- El número de consultas de una ruta crece con el tamaño de la lista: 10 pedidos dan 11 consultas, 100 dan 101.
- En el log, la misma sentencia se repite con otro valor en `$1`.
- En `pg_stat_statements`, una sentencia tiene muchas más `calls` que la consulta de la lista, con pocas filas por llamada.

PASO 01

Datos de ejemplo

Crea dos tablas en una base de pruebas: 1,000 pedidos con 1 a 5 productos cada uno. Necesitas Node.js 21 o superior y postgres.js. Todo corrió el 19 de noviembre de 2026 contra PGlite 0.5.10 (PostgreSQL 18.6) por 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); -- la llave foránea no lo crea

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;
```

Los 100 pedidos más recientes (901 a 1000) tienen 300 productos: es lo que deben devolver todas las versiones.

PASO 02

Antes y después

```
// GET /api/orders, versión con N+1: 101 consultas
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;
```

Una consulta para la lista y 100 dentro del `for`, en fila. Busca este mismo patrón en tu código: un `await` a la base dentro de un ciclo, un `.map` con consultas, o una relación del ORM leída dentro de un ciclo.

Después, con `ANY`: todos los ids viajan en un solo parámetro de tipo arreglo.

```
// GET /api/orders, versión con ANY: 2 consultas
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;
```

Con un JOIN (1 consulta), limita los pedidos en una subconsulta, `FROM (SELECT * FROM orders ORDER BY id DESC LIMIT 100) o LEFT JOIN order_items i ON i.order_id = o.id`: un `LIMIT 100` sobre el JOIN cuenta filas y en la prueba devolvió solo 33 pedidos.

Con Drizzle ORM 0.45, declara las relaciones con `relations()`, pásalas en `schema` y pide `db.query.orders.findMany({ limit: 100, with: { items: true } })`: mandó una sola consulta. En Drizzle 1.0 (hoy 1.0.0-rc.5) se declaran con `defineRelations`; en Prisma, lo equivalente es `include` (no lo ejecutamos).

PASO 03

Detéctalo en tu proyecto: consultas por petición

Este archivo cuenta las consultas de cada petición con `AsyncLocalStorage` y la función `debug` de postgres.js, que se llama con cada consulta. Cuando una petición pasa de 10, escribe un aviso con la sentencia que más se repitió:

```
// 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));
  };
}
```

Envuelve cada ruta: `http.createServer(countQueries("GET /orders", handler))`. Con la versión de antes escribió esto; con la de después, nada, porque hizo 2 consultas:

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

La primera conexión de postgres.js también consulta los tipos de arreglo: si tu primera petición marca una consulta de más, es esa. En Express, el mismo cuerpo va en un middleware que llama a `next()` dentro de `requestStore.run`; con Drizzle puedes contar en `logger.logQuery` en lugar de `debug`, pero no en los dos. Úsalo en desarrollo y pruebas.

PASO 04

Detéctalo en la base: log y pg_stat_statements

Para ver una petición en el log de PostgreSQL, baja el umbral solo en tu sesión de prueba (requiere superusuario o el privilegio `SET` sobre el parámetro):

```
SET log_min_duration_statement = 0;  -- 0 registra todas; -1 (por defecto) ninguna
```

Cada consulta del N+1 tarda menos de un milisegundo: un umbral de 250 ms no la registra. Busca la misma sentencia con otro `Parameters: $1` una y otra vez. Para todo el tráfico, activa `pg_stat_statements`: agrega `shared_preload_libraries = 'pg_stat_statements'` en `postgresql.conf`, reinicia y corre `CREATE EXTENSION pg_stat_statements;` en la base. En una base administrada, sigue la documentación de tu proveedor. Luego:

```
SELECT pg_stat_statements_reset();   -- contadores en cero
-- llama la ruta una vez
SELECT calls, rows, round(mean_exec_time::numeric, 3) AS ms, query
FROM pg_stat_statements
ORDER BY calls DESC
LIMIT 10;
```

Con la ruta de antes, la primera fila fue la consulta de `order_items` con `calls = 100` y `rows = 300`. Después del cambio, esa fila desaparece y queda una consulta con `ANY($1::bigint[])` y `calls = 1`.

> **CUIDADO**
>
> `pg_stat_statements_reset()` borra las estadísticas de todo el servidor y por defecto solo lo puede correr un superusuario. Hazlo en una base de pruebas, no en producción, donde perderías el historial que usa tu equipo.

PASO 05

Mide antes y después

Anota dos números por versión: consultas por petición y tiempo de la ruta, con la base en otra máquina como en producción. Para estimar tu caso antes de medir: consultas × tiempo de ida y vuelta a tu base, que mides con un `SELECT 1` desde tu servidor de aplicación. En nuestra prueba (ilustrativa: mediana de 30 corridas con 1.5 ms por viaje), la versión N+1 tardó 161.5 ms y la de `ANY`, 4.7 ms: 101 × 1.5 ms son unos 150 ms solo de espera.

AL TERMINAR

Checklist

- [ ] Ninguna ruta hace una consulta dentro de un ciclo sobre una lista.
- [ ] El contador de consultas por petición está activo en desarrollo y avisa sobre 10.
- [ ] Las relaciones se cargan con JOIN, con `= ANY($1)` o con `with` / `include`.
- [ ] Las listas de ids viajan como arreglo, no como un parámetro por id.
- [ ] La columna de la llave foránea tiene índice.
- [ ] Anotaste consultas y tiempo antes y después del cambio.