EcomLabs

GUÍA PASO A PASO

Claude Code y Postgres por MCP, sin permiso de borrar

El SQL del rol de solo lectura, la configuración del servidor MCP para Claude Code sin contraseñas en el repo y la prueba de seguridad que confirma que DELETE, UPDATE e INSERT se rechazan. Probado en PGlite y con DBHub 1.6.1.

PARA	Equipos que conectan Claude Code a una base PostgreSQL
PROBADO CON	Node 22.22.3 · PGlite 0.5.10 (PostgreSQL 18.6) · DBHub 1.6.1 · Windows 11
ACTUALIZADA	11 de noviembre de 2026
DOCUMENTACIÓN	code.claude.com/docs/en/mcp
ARTÍCULO	ecomlabs.dev/g/mcp
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a montar
Paso 01	El rol de solo lectura
Paso 02	La configuración de DBHub
Paso 03	Registrar el servidor en Claude Code
Paso 04	Apúntalo a una réplica
Paso 05	La prueba de seguridad
Al terminar	Checklist

ANTES DE EMPEZAR

Qué vas a montar

Un rol de Postgres que solo puede leer, el servidor MCP DBHub configurado con ese rol en Claude Code y una prueba que confirma que las escrituras se rechazan. El SQL lo probamos el 11 de noviembre de 2026 en PGlite 0.5.10 (PostgreSQL 18.6) con `node --test`: 18 de 18 pruebas pasaron. El servidor lo probamos con DBHub 1.6.1 y un cliente MCP simulado, sin Claude Code ni ningún modelo.

La idea central: el modo de solo lectura del servidor ayuda, pero quien rechaza un `DELETE` es Postgres, por los privilegios del rol.

PASO 01

El rol de solo lectura

Guárdalo como `role.sql` y ejecútalo con un usuario que pueda crear roles. Cambia `app_owner` por el rol dueño de tus tablas (el que corre tus migraciones):

```
CREATE ROLE mcp_readonly WITH LOGIN CONNECTION LIMIT 3;

GRANT USAGE ON SCHEMA public TO mcp_readonly;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO mcp_readonly;

ALTER DEFAULT PRIVILEGES FOR ROLE app_owner IN SCHEMA public
  GRANT SELECT ON TABLES TO mcp_readonly;

ALTER ROLE mcp_readonly SET default_transaction_read_only = on;
ALTER ROLE mcp_readonly SET statement_timeout = '10s';
```

- **Contraseña:** no la escribas en el SQL. En psql, como administrador: `\password mcp_readonly`.
- `FOR ROLE`**:** solo cuentan los privilegios por defecto del rol que crea la tabla. Si otro rol crea tablas, repite la línea para él.
- **Base anterior a PostgreSQL 15:** ejecuta también `REVOKE CREATE ON SCHEMA public FROM PUBLIC;`
- **Datos personales:** si Claude no los necesita, da `SELECT` por tabla o por columna en vez de `ALL TABLES`.

PASO 02

La configuración de DBHub

DBHub solo acepta `readonly` en su archivo TOML (el flag `--readonly` ya no existe). Guarda `dbhub.toml`:

```
[[sources]]
id = "replica"
dsn = "${MCP_PG_DSN}"

[[tools]]
name = "execute_sql"
source = "replica"
readonly = true
max_rows = 200
```

El TOML no lleva la contraseña: la lee de la variable `MCP_PG_DSN`. Defínela en tu shell o gestor de secretos con esta forma:

```
postgres://mcp_readonly:<contraseña>@<host-de-la-réplica>:5432/<base>?sslmode=require
```

`sslmode=require` cifra sin verificar el certificado; `verify-full` también verifica certificado y host.

PASO 03

Registrar el servidor en Claude Code

Solo para ti (alcance local, se guarda en `~/.claude.json`):

```
claude mcp add --env MCP_PG_DSN="$MCP_PG_DSN" --transport stdio postgres-ro \
  -- npx -y @bytebase/dbhub@1.6.1 --config /ruta/absoluta/dbhub.toml
```

- `--transport stdio` va entre `--env` y el nombre: si el nombre va justo después de `--env`, la CLI lo rechaza.
- Usa ruta absoluta en `--config`: la documentación no dice en qué carpeta arranca un servidor stdio.
- Comprueba con `/mcp` en Claude Code que el servidor tenga la marca de verificación.

Para el equipo (alcance `project`, archivo `.mcp.json` en la raíz del repo, que sí va a git):

```
{
  "mcpServers": {
    "postgres-ro": {
      "command": "npx",
      "args": ["-y", "@bytebase/dbhub@1.6.1", "--config", "${DBHUB_CONFIG}"],
      "env": {
        "MCP_PG_DSN": "${MCP_PG_DSN}"
      }
    }
  }
}
```

> **CUIDADO**
>
> Nunca escribas la contraseña en `.mcp.json`: solo referencias `${VAR}`. Con `claude mcp add`, `~/.claude.json` guarda el valor ya expandido; no va a git, pero trátalo como un secreto. Si una contraseña llegó a un commit, cámbiala con `\password`. Este comando y este archivo no los ejecutamos: los comprobamos con `claude mcp add --help` (Claude Code 2.1.238) y la documentación.

PASO 04

Apúntalo a una réplica

- En un standby, `transaction_read_only` siempre vale `true` y las escrituras, el DDL y `GRANT` dan error.
- Crea el rol en la primaria: con replicación física llega a la réplica.
- La réplica va con retraso: los datos de hace un segundo pueden no estar.
- Las consultas largas se cancelan al pasar `max_standby_streaming_delay`.
- Sin réplica, usa el mismo rol en la primaria con `statement_timeout` bajo y `max_rows`.

No probamos una réplica (PGlite no tiene replicación); este paso se apoya en la documentación de PostgreSQL.

PASO 05

La prueba de seguridad

Guarda `probe.sql`, cambia `orders` por una de tus tablas y córrelo una vez contra la primaria con `psql "$MCP_PG_DSN" -f probe.sql`:

```
SHOW default_transaction_read_only;
SET default_transaction_read_only = off;
SELECT current_user, session_user;
SELECT rolsuper, rolcreaterole, rolcreatedb, rolbypassrls FROM pg_roles WHERE rolname = current_user;
SELECT count(*) AS owned_tables FROM pg_tables WHERE tableowner = current_user;
DELETE FROM orders WHERE false;
UPDATE orders SET total = total WHERE false;
INSERT INTO orders SELECT * FROM orders WHERE false;
```

LÍNEA	RESULTADO ESPERADO
`SHOW`	`on` (el `ALTER ROLE` se aplicó al iniciar sesión)
`current_user`	`mcp_readonly`
`pg_roles`	cuatro `false`
`owned_tables`	`0`
Las tres escrituras	`permission denied for table orders`

- El `SET` apaga el modo de solo lectura en esta sesión; sin él, todo falla con `cannot execute DELETE in a read-only transaction` y no ves el permiso.

- `WHERE false` no toca ninguna fila aunque el rol estuviera mal, y Postgres revisa el permiso igual.
- Después pídele a Claude Code que ejecute `DELETE FROM orders WHERE false` con la herramienta de Postgres: con `readonly = true`, DBHub responde `READONLY_VIOLATION`.

En nuestra prueba con DBHub 1.6.1, `DELETE` y `COMMIT; DELETE` dieron `READONLY_VIOLATION` con `readonly = true`, y `permission denied for table orders` sin él. La tabla conservó sus filas.

> **CUIDADO**
>
> Revisa las funciones `SECURITY DEFINER` que escriben: PostgreSQL da `EXECUTE` a `PUBLIC` por defecto, y en nuestra prueba una de ellas borró una fila llamada desde `mcp_readonly`. Quítalo con `REVOKE EXECUTE ON FUNCTION ... FROM PUBLIC`. Nunca hagas pruebas de escritura sin `WHERE false` contra tu base de producción.

AL TERMINAR

Checklist

- [ ] El servidor MCP usa un rol propio, no el de la aplicación ni el de las migraciones.
- [ ] El rol tiene `USAGE` en el esquema y `SELECT` solo en lo que hace falta.
- [ ] `ALTER DEFAULT PRIVILEGES FOR ROLE` apunta al rol que crea tus tablas.
- [ ] La contraseña se puso con `\password` y vive solo en `MCP_PG_DSN`.
- [ ] `dbhub.toml` tiene `readonly = true` y `max_rows`; la versión está fijada.
- [ ] `.mcp.json` solo tiene referencias `${VAR}`.
- [ ] El DSN apunta a una réplica, o a la primaria con `statement_timeout` bajo.
- [ ] `probe.sql` da `permission denied` en las tres escrituras y `0` tablas propias.
- [ ] Ninguna función `SECURITY DEFINER` que escribe tiene `EXECUTE` para `PUBLIC`.