NeuraGrowth

Supabase MCP Field Guide: Claude, Cursor & RLS

Wire Supabase into your AI-native workflow without opening a security hole.

NEURAGROWTH.CO

2026-06-05

CONTENTS

01

What the Supabase MCP Server Actually Does

The Problem It Solves

AI coding assistants are great at writing SQL and TypeScript. They are terrible at knowing what is already in your database. The Supabase MCP server closes that gap: it gives Claude or Cursor a live, authenticated window into your Supabase project so the assistant can read your schema, inspect RLS policies, run queries, and propose migrations - all from inside the same chat session where you are writing application code.

This is not a generic database connector. It is a purpose-built Model Context Protocol server maintained by the Supabase team, published on npm as `@supabase/mcp-server-supabase`.

How It Authenticates

The server authenticates with Supabase through the **Management API**, not through your project's Postgres connection string. That distinction matters.

The Management API uses a **Personal Access Token (PAT)** scoped to your Supabase account. You generate one at `supabase.com/dashboard/account/tokens`. This token lets the MCP server:

- List your projects
- Read and write project settings
- Invoke management-level actions (create branches, run migrations)

It does NOT directly touch Postgres unless one of the exposed tools explicitly issues a SQL call through the Management API's SQL execution endpoint.

The 23 Exposed Tools (Grouped)

At last count the server exposes 23 tools, organised into four functional groups:

Group	Representative tools
Schema inspection	`list_tables`, `list_columns`, `list_extensions`, `list_migrations`
SQL execution	`execute_sql` (read-only by default), `apply_migration`
RLS & policies	`list_policies`, `create_policy`, `delete_policy`
Project management	`list_projects`, `get_project`, `create_branch`, `merge_branch`, `delete_branch`
Edge Functions	`list_edge_functions`, `deploy_edge_function`
Storage	`list_buckets`, `create_bucket`
Logs	`get_logs`

Each tool is a typed JSON-RPC call. Claude or Cursor decides which tools to invoke based on your prompt; you never call them directly.

Read-Only Mode: Why It Exists and When to Lift It

The server ships with a `--read-only` flag that restricts `execute_sql` to SELECT statements. This is the safe default because:

1. AI models make mistakes. A hallucinated DROP TABLE in read-only mode is a no-op.
2. The Management API token has broad account-level scope. If you paste a prompt like "clean up the users table" into an unrestricted session, the model might interpret that creatively.
3. Development environments rarely need write access through the MCP channel - application code and explicit migrations are the right path for writes.

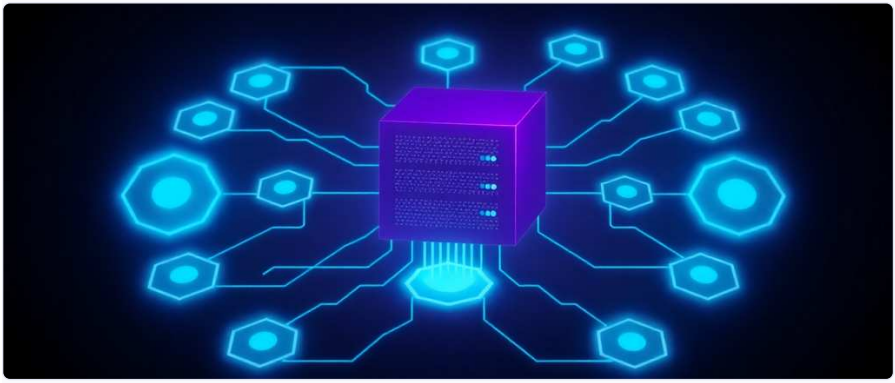
Read-only mode is enforced server-side by the MCP process, not by Postgres permissions. If you later lift it for branch-based work (covered in the final chapter), do so only for a branch project, never against your production project ID.

The Transport Layer

The server communicates over **stdio** - standard input/output. The MCP host (Claude Desktop or Cursor) spawns the server as a child process and exchanges JSON-RPC messages through stdin/stdout. No port is opened, no HTTP server runs locally. This means:

- No firewall rules needed
- No network exposure
- The server dies when the host process dies

This architecture is one reason MCP is safe to run on a developer laptop. The attack surface is limited to whoever can send messages to the host application.



WHAT THE SUPABASE MCP SERVER ACTUALLY DOES

02

Installing the MCP Server: Prerequisites and First Run

What You Need Before You Start

Before touching any config file, confirm the following:

- **Node.js 18+** installed (`node --version`)
- **npx** available (ships with npm 5.2+; verify with `npx --version`)
- A **Supabase project** already created at `supabase.com`
- A **Personal Access Token** generated at `supabase.com/dashboard/account/tokens` - copy it now, you cannot retrieve it later

You do NOT need to install the package globally. Both Claude Desktop and Cursor invoke it via `npx` , which downloads and caches the package on first run.

Generating Your Personal Access Token

1. Log into `supabase.com`
2. Click your avatar (top-right) -> Account
3. Navigate to **Access Tokens**
4. Click **Generate new token**
5. Label it something specific: `mcp-dev-laptop-2025`
6. Copy the token immediately - it will not be shown again

Store it in your OS keychain or a secrets manager. Do not store it in a `.env` file that lives inside a git repo.

Finding Your Project Reference ID

Every Supabase project has a reference ID - a short alphanumeric string visible in the dashboard URL:

```
https://supabase.com/dashboard/project/abcdefghijklmnop
                                       ^^^^^^^^^^^^^^^^
                                       this is your ref
```

You will need this ref when scoping the MCP server to a single project.

Verifying the Package Exists and Runs

Before wiring up any editor, confirm the server starts:

```
SUPABASE_ACCESS_TOKEN=your_pat_here npx @supabase/mcp-server-supabase --read-only
```

Expected output on stderr (stdio mode means nothing goes to stdout until the host connects):

```
Supabase MCP server running (stdio)
```

Press Ctrl+C to stop. If you see an authentication error, the PAT is wrong or expired. If you see a module resolution error, your Node version is too old.

A Note on Version Pinning

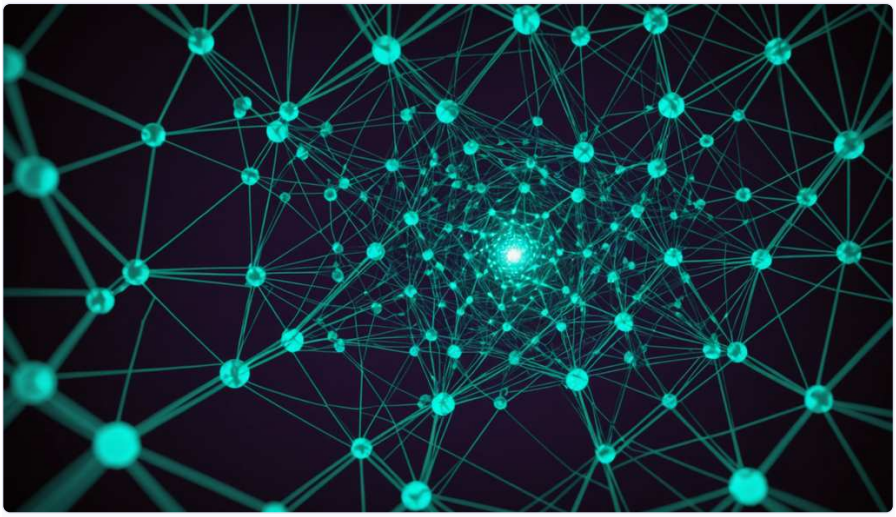
`npx` fetches the latest version by default. For a production-grade workflow, pin the version in your config:

```
"command": "npx",
"args": ["@supabase/mcp-server-supabase@0.4.2", "--read-only"]
```

Check the current stable version at `npmjs.com/package/@supabase/mcp-server-supabase` before pinning. Unpinned configs are fine for experimentation but become a debugging headache when the package releases a breaking change.

03

Configuring Claude Desktop: claude_desktop_config.json Explained



Where the Config File Lives

Claude Desktop reads MCP server configuration from a JSON file at a platform-specific path:

OS	Path
macOS	`~/Library/Application Support/Claude/claude_desktop_config.json`
Windows	`%APPDATA%\Claude\claude_desktop_config.json`
Linux	`~/.config/Claude/claude_desktop_config.json`

If the file does not exist, create it. Claude Desktop will pick it up on next launch.

Minimal Working Configuration

```
{
  "mcpServers": {
    "supabase": {
      "command": "npx",
      "args": [
        "-y",
        "@supabase/mcp-server-supabase",
        "--read-only",
        "--project-ref", "abcdefghijklmnop"
      ],
      "env": {
        "SUPABASE_ACCESS_TOKEN": "sbp_your_token_here"
      }
    }
  }
}
```

Key fields explained

- `command` : The executable to spawn. `npx` is correct for stdio-based MCP servers.
- `args` : Array of arguments passed to npx. `-y` skips the interactive install prompt.
- `--read-only` : Restricts SQL execution to SELECT. Remove only when working on a branch.
- `--project-ref` : Scopes the server to one project. Without this, the model can list and interact with ALL projects under your PAT. Always set it.

- `env`: Environment variables injected into the child process. This is where the PAT goes.

Using macOS Keychain Instead of Plaintext Token

If you are uncomfortable with the PAT sitting in a JSON file, you can pull it from Keychain at process spawn time using a wrapper script:

```
#!/bin/bash
# ~/scripts/supabase-mcp-wrapper.sh
export SUPABASE_ACCESS_TOKEN=$(security find-generic-password
-a "$USER" -s "supabase-mcp-pat" -w)
exec npx -y @supabase/mcp-server-supabase --read-only --
project-ref abcdefghijklmnop
```

Then reference the wrapper in the config:

```
{
  "mcpServers": {
    "supabase": {
      "command": "/bin/bash",
      "args": ["/Users/you/scripts/supabase-mcp-wrapper.sh"]
    }
  }
}
```

Add the secret to Keychain once:

```
security add-generic-password -a "$USER" -s "supabase-mcp-
pat" -w "sbp_your_token_here"
```