

[Generative AI \(/tags/generative-ai\)](#)   [Claude Code \(/tags/claude-code\)](#)   [Anthropic \(/tags/anthropic\)](#)

[AI-assisted Coding \(/tags/ai-assisted-coding\)](#)   [PostgreSQL \(/tags/postgresql\)](#)

[MCP Servers \(/tags/mcp-servers\)](#)   [AI-Native Development \(/tags/ai-native-development\)](#)

## Building an Event Management System with Claude Code: Part 4 - Database Setup and First Conversations

Posted by nielsb on Tuesday, December 30, 2025

Welcome back to the **Building an Event Management System with Claude Code** series! If you've been following along, you know we've covered installation, IDE integration, and architectural planning. Today, we're moving from planning to building, setting up our PostgreSQL database and having our first AI-powered conversations with it.

- To see all posts in the series, go to: [Building an Event Management System with Claude Code \(/contact-event-claude-code\)](#).

In Part 3, we made a crucial decision: we're not building a traditional web application with forms and dashboards. Instead, we're building an **AI-native system** with a primary conversational interface. No admin panels, no CRUD forms, just natural language queries that get things done.

Today, we make that vision real.

### What We'll Accomplish Today

By the end of this post, you'll have:

- A properly structured GitHub repository (created with Claude Code's help)
- A complete audit trail of your development process (using `claude-code-transcripts`)
- A production-ready PostgreSQL database in the cloud
- Postgres MCP Pro installed and configured
- Verified connection between Claude and your database
- Understanding of the infrastructure that enables AI-native development

But before we dive into database setup, I need to address something practical: how do we not lose all this work?

### Setting Up the Project with Claude Code

*Before We Dive In: Repository Setup and Conversation History*

#### The Practical Reality of Starting

In the previous posts, we worked in "temporary" directories (`claude-test`, `event-contact-management-2`), but now we're building something real. I needed proper version control, a GitHub repository, and a project structure that makes sense.

Rather than doing this manually, I decided to let Claude Code handle it. After all, if we're building an AI-native system, we should use AI-native development practices from the start.

#### Asking Claude Code to Set Up the Repository

I opened Claude Code in the `event-contact-management-2` directory: `claude --continue`.

##### TIP

You may wonder what `--continue` does. Claude Code automatically saves conversation history for each session (more about that later), and starting it with `--continue` tells it to pick up where we left off in this directory, preserving conversation history and context. This is crucial for maintaining continuity in AI-native development.

With Claude Code running, I initiated the conversation:

```
Right now, the folder we are in is just that, a folder. Please now create a repo structure here, 1
```

##### Code Snippet 1: Asking Claude Code to Create a Repo

In *Code Snippet 1*, you see how I asked Claude Code to create a proper project structure. In hindsight, I should have been more specific about what I wanted.

Claude went off and started thinking about what the structure should look like. It then started creating "stuff":

#### - CATALOG

- What We'll Accomplish T...
- Setting Up the Project wi...
- The Practical Reality of S...
- Asking Claude Code to S...
- Git Initialisation
- Pushing to GitHub
- The Power of Conversati...
- What is claude-code-transc...
- Why I'm Showing You All...
- Database Setup Decision...
- The Conversation
- Setting Up the Database
- Setting Up PostgreSQL o...
- Step 1: Create Neon Acc...
- Step 2: Create Project
- Step 3: Get Connection D...
- Step 4: Secure the Crede...
- Step 5: Test the Connecti...
- What We Have Now
- Installing Postgres MCP ...
- What is Postgres MCP Pr...
- Installing with Docker
- Configuring Claude Desk...
- Testing the Connection
- Using AI to Troubleshoot...
- The Value of Conversatio...
- What We've Accomplished
- Infrastructure in Place
- Time Investment
- What Makes This Different
- What's Next: Part 5 Prev...
- Part 5: Database Schema...
- Summary: Building the F...
- What We Built Today
- The Journey Continues
- Why This Matters
- Ready for Part 5
- ~ Finally
- Your Turn

```
$ Bash
Create project directory structure
mkdir -p database/migrations database/seeds && mkdir -p mcp-servers/import-mcp/src mcp-servers/import-mcp/tests && mkdir -p docs/planning docs/setup-guides && mkdir -p config/mcp-servers config/database && mkdir -p scripts && mkdir -p .github/workflows
```

**Figure 1:** Claude Code Creating the Complete Project Structure

We see in *Figure 1* how Claude Code is creating files and directories. It generated a complete project structure, including documentation files, a comprehensive `.gitignore`, and a `README.md`.

## Git Initialisation

After some time, Claude Code initialised the git repository and showed the structure:

```
event-contact-management-2/
├── database/
│   ├── migrations/      # For database migration files
│   └── seeds/           # For sample/test data
│
├── mcp-servers/
│   └── import-mcp/
│       ├── src/          # Custom Import MCP source code
│       └── tests/         # Unit tests
│
├── config/
│   ├── mcp-servers/    # MCP server configurations
│   └── database/        # Database connection configs
│
├── docs/
│   ├── planning/        # Architecture & planning docs
│   │   ├── plan.md      # (moved from root)
│   │   └── spec.md      # (moved from root)
│   └── setup-guides/   # Setup instructions (empty, ready for docs)
│
└── scripts/            # Utility scripts (empty, ready for scripts)
└── .github/
    └── workflows/      # GitHub Actions CI/CD (empty)
```

**Figure 2:** Claude Code Showing the Initialized Git Repository and Project Structure

The structure in *Figure 2*:

- Separates database schema from application code
- Has a dedicated folder for each MCP server we'll build
- Sets up documentation structure

After the initialisation, Claude Code made the first commit:

```
git commit -m "$(cat <<'EOF'
Initial repository structure for Event Contact Management System

Set up foundational repository structure with comprehensive documentation,
directory organization for MCP servers, database, configuration, and scripts.

Added:
- README.md with complete project overview and architecture
- .gitignore with comprehensive rules for Python, Node.js, databases, IDEs
- docs/planning/ with architectural planning documents (plan.md, spec.md)
- docs/drawings/ for architecture diagrams
- Directory structure for database/, mcp-servers/, config/, scripts/
- CLAUDE.md AI-assisted planning document

Repository ready for Phase 1 development: MCP server setup and database schema.

Generated with [Claude Code](https://claude.com/claude-code)

Co-Authored-By: Claude Sonnet 4.5 <noreply@anthropic.com>
EOF
)"
```

**Figure 3:** Claude Code First Commit

Notice the **detailed commit message**. Claude Code doesn't just write "Initial commit"; it documents what was created and why. This is professional-grade version control.

## Pushing to GitHub

I obviously wanted this on GitHub, so I had another quick conversation with Claude Code:

USER

I now want you to push this repo to my GitHub account: <https://github.com/niebsberglund>, and make it private.

**Figure 4:** Asking Claude Code to Create a GitHub Repository

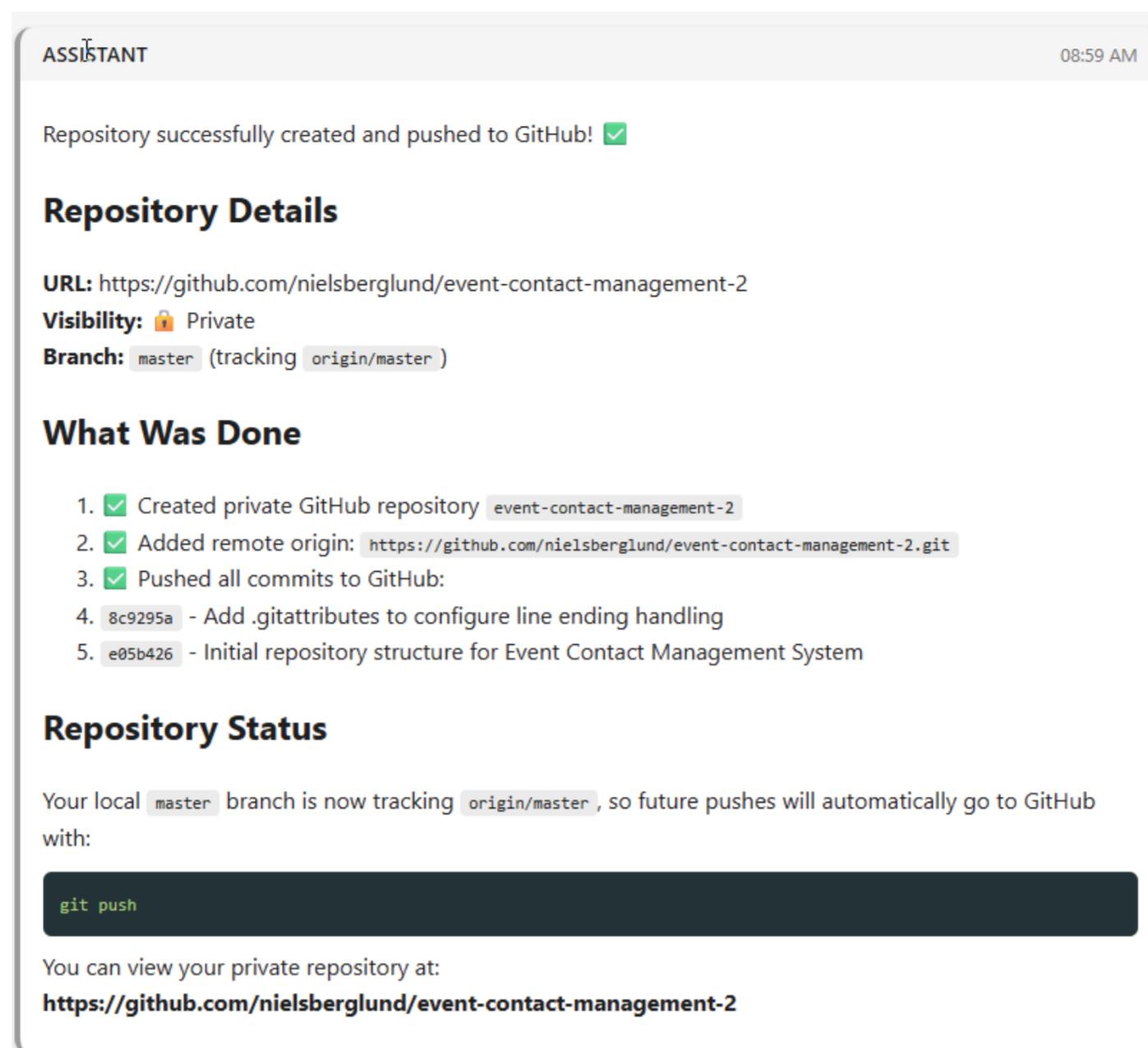
In *Figure 4*, we see how I asked Claude Code to help me create a GitHub repository and push the initial commit. Since I have the GitHub CLI (`gh`) installed, Claude Code did everything for me:

```
$ Bash
Create private GitHub repo and push

gh repo create nielsberglund/event-contact-management-2 --private --source=. --remote=origin --push
```

**Figure 5:** Claude Code Creating the GitHub Repository and Pushing the Initial Commit

Finally, Claude Code confirmed the push:

**Figure 6:** Claude Code Confirming Repository Successfully Created and Pushed

And just like that (as we see in *Figure 6*), the project is on GitHub, with a proper structure, comprehensive documentation, and a clean commit history.

## The Power of Conversation History

Here's where things get really interesting. I didn't take screenshots during the repository setup process; I was just focused on getting it done. But then I discovered something incredibly useful: the `claude-code-transcripts` (<https://github.com/simonw/claude-code-transcripts/>) tool. The screenshots above were recreated from the actual conversation history extracted using this tool.

### What is `claude-code-transcripts`?

`claude-code-transcripts` is a tool that extracts and formats all Claude Code conversations from a project's history. Claude Code stores conversation data in `~/.claude/projects/`, and this tool makes it human-readable.

#### Installation & Usage

To install:

```
1 $ uv tool install claude-code-transcripts
```

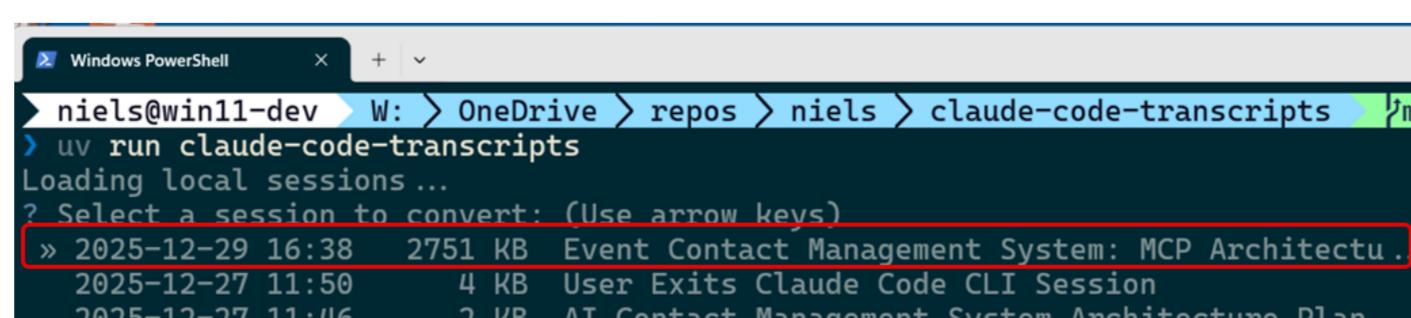
#### Code Snippet 2: Installing `claude-code-transcripts`

To view all conversations run:

```
1 # View all conversations
2 $ claude-code-transcripts
```

#### Code Snippet 3: Viewing All Conversations

When you run the command in *Code Snippet 3*, you get a list of all conversations, and you can then click on the one you want to see:

**Figure 7:** Choosing `claude-code-transcripts`

**Figure 7** shows, outlined in red, the session I chose to get the conversation to create the images above.

To see all you can do with `claude-code-transcripts`, look at the **README.md** (<https://github.com/simonw/claude-code-transcripts/blob/main/README.md>) file.

## Why I'm Showing You All This

I included this detailed setup section because it demonstrates something crucial about AI-native development:

**Claude Code isn't just for writing application code; it's for the entire development lifecycle.**

This is the paradigm shift:

**Traditional Approach:**

1. Manually create project structure
2. Google "Python gitignore template"
3. Copy-paste, modify
4. Create README from scratch
5. git init, git add ., git commit
6. Create a repo on GitHub
7. Follow GitHub's instructions to push
8. Manually configure settings

**AI-Native Approach:**

1. Describe what you need in conversation
2. Claude Code creates everything
3. Review and refine through conversation
4. Professional results in minutes

The time savings are obvious, but that's not the main benefit. The real value is **quality and thoughtfulness**. Claude Code created a better structure than I would have because it considered:

- Future scalability needs
- Best practices
- Documentation workflows
- Testing organisation
- Migration strategy

And I have a complete transcript of every decision.

Now that our project is properly set up and version-controlled, let's move on to the database.

## Database Setup Decision: Local vs Cloud

*Using Claude Code for Infrastructure Planning*

Before I start installing PostgreSQL, I need to decide: **local database or cloud-hosted?**

In traditional development, I'd probably spin up a local PostgreSQL instance and deal with cloud deployment later. But let's think about our AI-native use case:

- I want to query this database from Claude.ai on my phone
- I want it accessible from multiple machines
- I don't want to deal with SSH tunnels or port forwarding
- I want to focus on building, not DevOps

This calls for a conversation with Claude Code about infrastructure.

### The Conversation

Since Claude Code was already open in the project directory:

We are now ready to start thinking about the database. Think about whether I should use local Postgre...

Consider:

- I'm building an MVP for personal use
- The MVP, if successful, will be productionised shortly after MVP completion.
- I need it accessible from Claude.ai on mobile (natural language queries)
- Further down the line, it needs to be accessible from a web chatbot
- Budget is minimal (~\$0-20/month acceptable)
- I want to focus on development, not infrastructure management
- Need to connect via Postgres MCP Pro

**Code Snippet 4: Asking Claude Code for Database Hosting Recommendation**

Notice I'm using "**Think about**" to encourage deeper analysis. This triggers Claude Code's reasoning mode. Claude Code then went off and thought through the options. It came back with its preferred option:

TL;DR: Use Neon ★

Recommendation: Neon (Free tier → Launch plan \$19/month)

Why Neon Wins

1. Mobile Access ✓ (Critical!)

[snipped for brevity]

2. Perfect Budget Fit ✓

[snipped for brevity]

3. Database Branching 🎉 (Unique!)

[snipped for brevity]

4. Zero Infrastructure ✓

[snipped for brevity]

5. Smooth MVP → Production ✓

[snipped for brevity]

Why NOT Other Options

✗ Local Docker: Can't access from Claude.ai mobile (deal-breaker)

● Supabase: \$25/month for production (over budget) + features you won't use

● Railway: Usage-based pricing is unpredictable

[snipped for brevity]

Would you like me to help you set up Neon and configure the connection?

**Code Snippet 5:** Claude Code's Database Hosting Recommendation

What you see in **Code Snippet 5** is the **final recommendation**. It has thought through the options and provided a reasoned choice. The complete analysis was much longer, and Claude Code output it to the conversation and to a document `docs/planning/database-hosting-analysis.md`. I have published the document as a `gist` here (<https://gist.github.com/nielsberglund/710b5245e9bcc9023f11bd048f8fc94a>). I encourage you to read through the document and see Claude Code's reasoning, and the decision matrix it created:

| Feature         | Local Docker | Supabase   | Neon          | Railway    |
|-----------------|--------------|------------|---------------|------------|
| Mobile Access   | ✗ No         | ✓ Yes      | ✓ Yes         | ✓ Yes      |
| MVP Cost        | \$0          | \$0        | \$0           | ~\$5/mo    |
| Production Cost | N/A          | \$25/mo    | \$19/mo       | \$20/mo+   |
| Setup Time      | Hours        | Minutes    | Seconds       | Minutes    |
| Backup/Recovery | Manual       | Auto (Pro) | Auto (Launch) | Auto       |
| Scaling         | Manual       | Auto       | Auto          | Auto       |
| PG Features     | Full         | Full       | Full          | Full       |
| Unique Feature  | Control      | REST API   | Branching     | Simplicity |

**Table 1:** Database Hosting Decision Matrix Created by Claude Code

Claude Code's analysis is excellent. Claude Code didn't just give me an answer; it:

- Reasoned through my requirements
- Compared specific services
- Provided concrete pros/cons
- Made a clear recommendation with rationale
- Offered to help with next steps

So now we have the recommendation. Time to set up the database.

## Setting Up the Database

Since Claude Code asked me (**Code Snippet 5**) if I wanted help setting up Neon, I responded:

Yes, please, walk me through the Neon setup. But also show me how the local Docker approach would

**Code Snippet 6:** Asking Claude to Help with Database Setup

Claude Code responded with a detailed step-by-step guide for both approaches, including commands, explanations, and other details. It also created `.env` file templates for both methods.

As I asked Claude Code to document everything, it created two new documents in `docs/setup-guides`. I generated a `gist` with the complete setup guides here (<https://gist.github.com/nielsberglund/923933acce57ea92bc220a7139a2ac99>). When you browse the `gist`, you'll see that Claude Code also included instructions for setting up the PostgreSQL MCP Server. We'll look at that below.

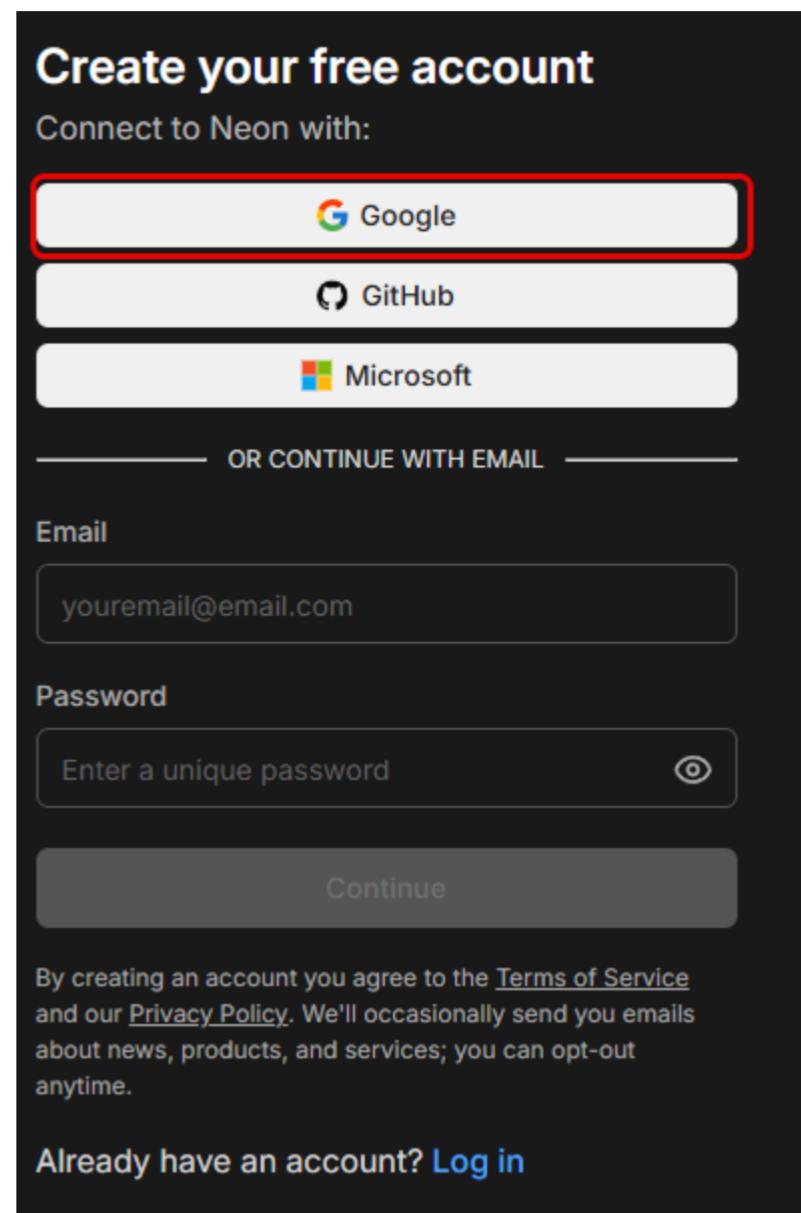
# Setting Up PostgreSQL on Neon

## From Zero to Database in 8 Minutes

Let me walk you through the actual setup process.

### Step 1: Create Neon Account

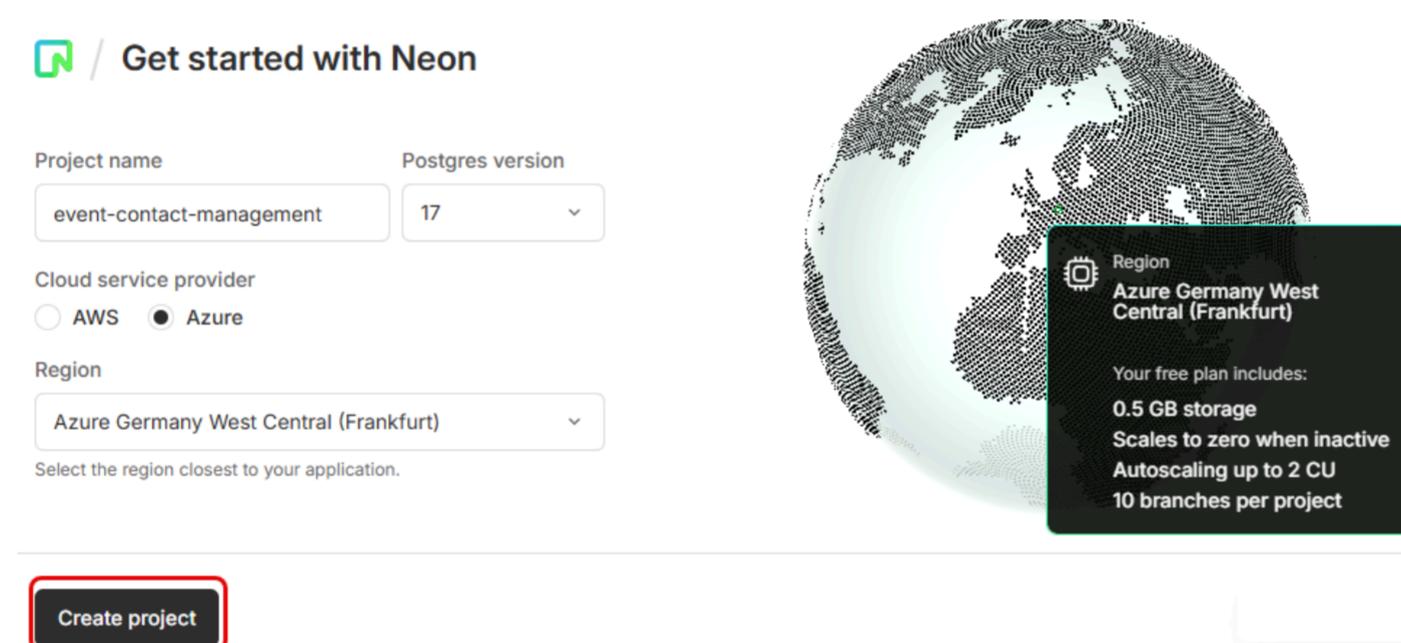
I went to [neon.tech](https://neon.tech) (<https://neon.tech>) and signed up with GitHub (one click, uses my Google identity).



**Figure 8:** Neon sign-up using Google authentication

After choosing Google authentication, I am now prompted to create a project.

### Step 2: Create Project



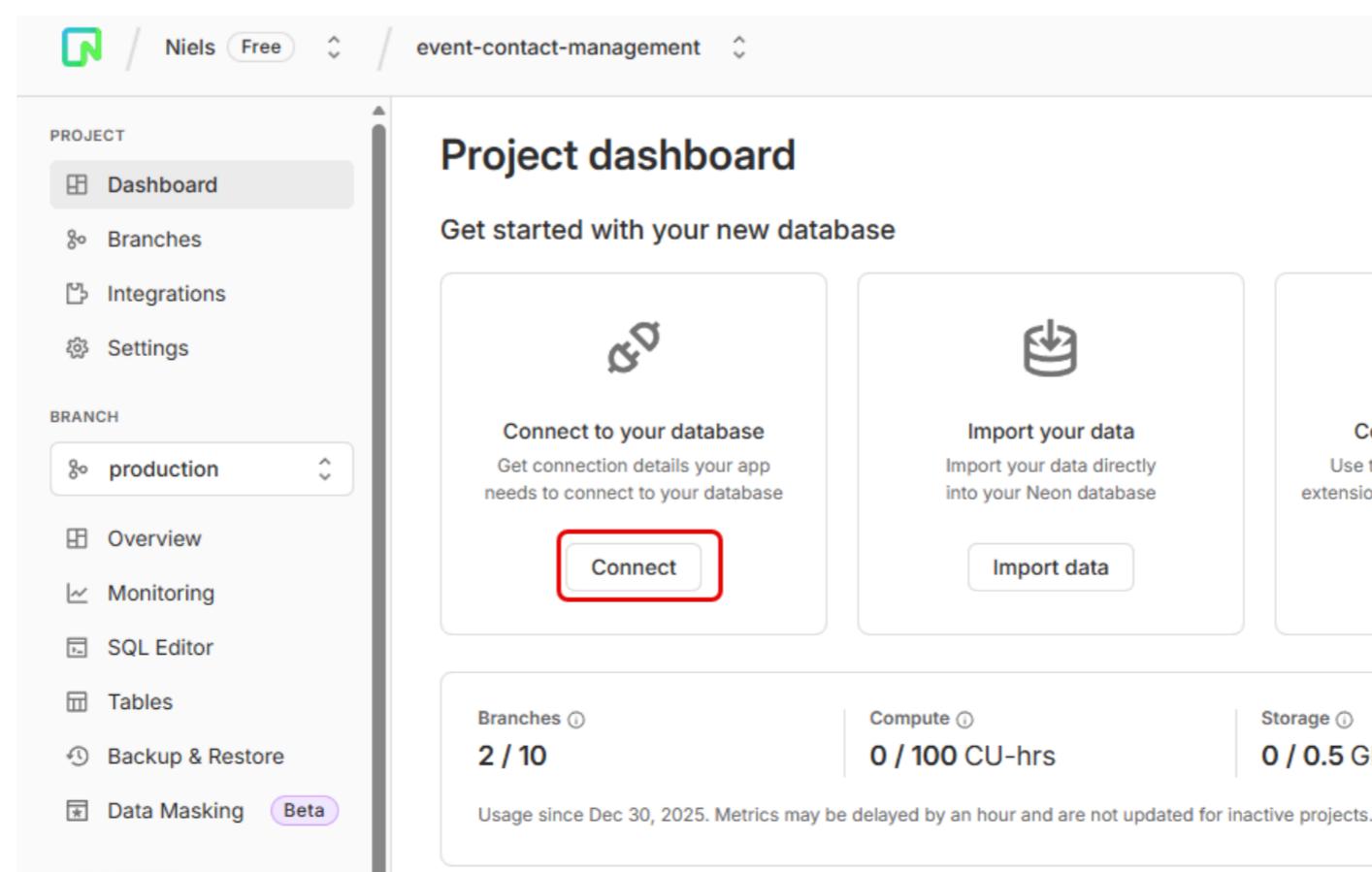
**Figure 9:** Neon Project Creation Screen

The project settings are as follows (see *Figure 9*):

- **Name:** event-contact-management
- **Region:** Azure Germany West Central (Frankfurt)
- **PostgreSQL version:** 17

I chose PostgreSQL 17 to stay current and to host it in Azure. My thinking is that when I eventually host a web chatbot, I would do it on Azure, so keeping everything in the same cloud provider makes sense.

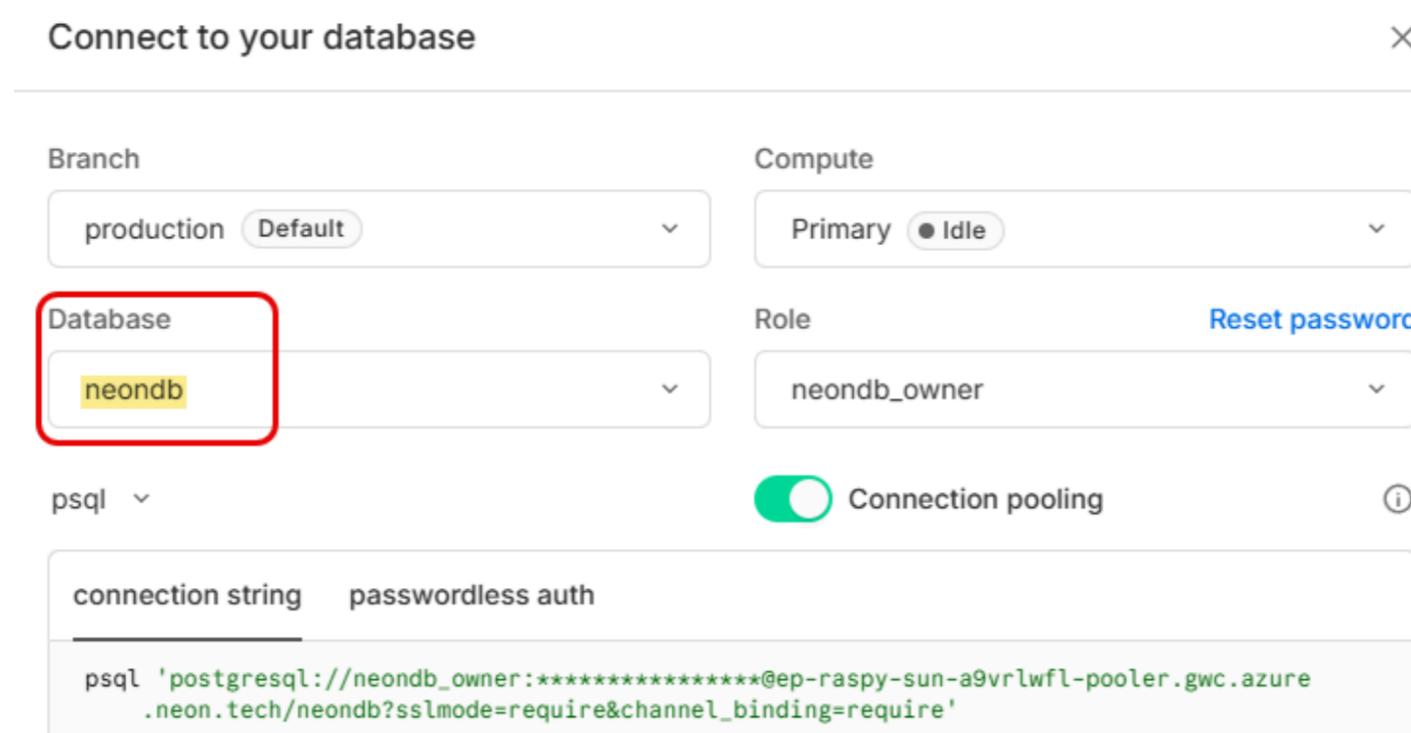
Clicking on the **Create project** button (outlined in red), Neon creates the database instantly, no waiting for provisioning, and takes me to the project dashboard:



**Figure 10:** Neon Project Dashboard

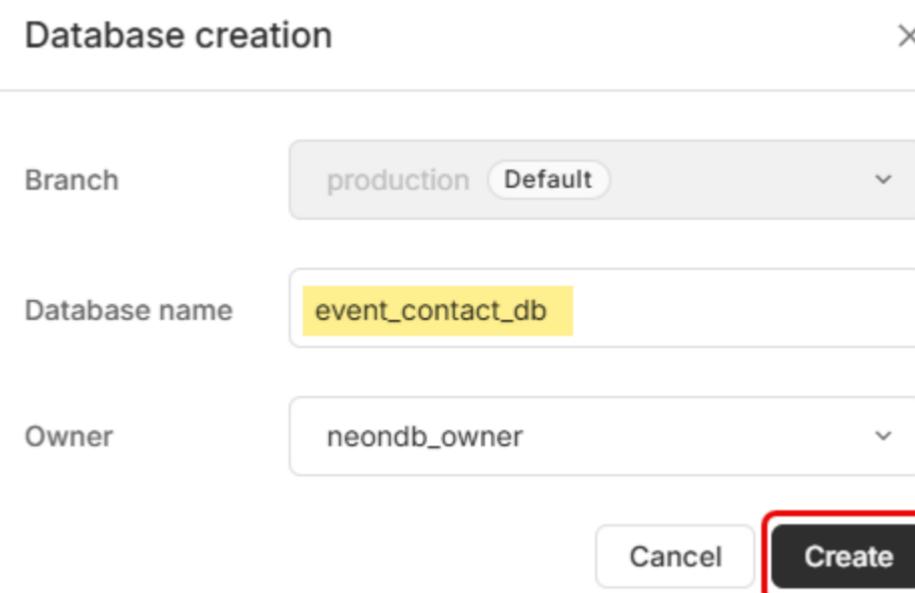
### Step 3: Get Connection Details

Let's get the connection details. In the dashboard (see *Figure 10*), I click on the **Connect** button (outlined in red) within the **Connect to your database** block. When I do that, a modal dialog appears:



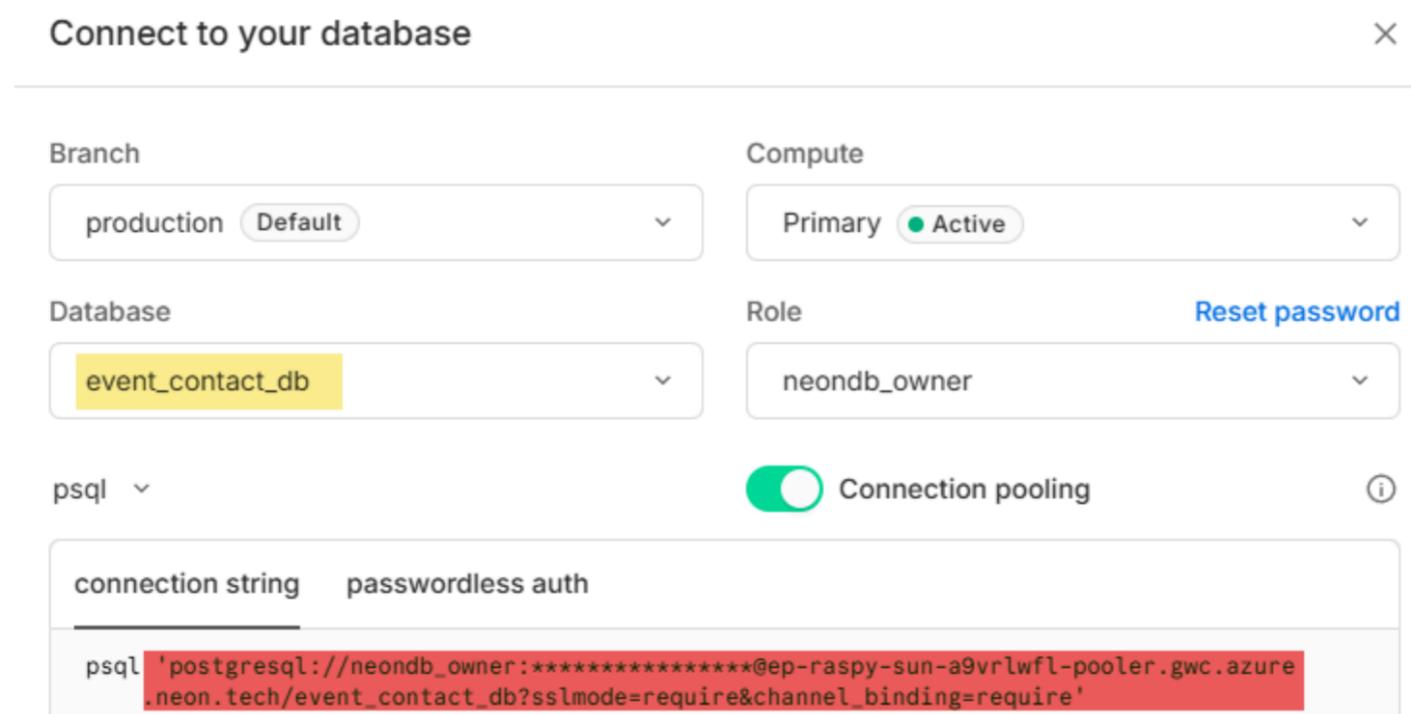
**Figure 11: Connect to your database**

Hmm, look at the part outlined in red in *Figure 11*: the **Database** drop-down. It seems that when you create a Neon project, it automatically creates a default database named `neondb`. I want the database used by this system to have a different name, `event_contact_db`, so I need to create it first. If you drop down the **Database** selector, you not only see `neondb`, but also an option to **Create new database**:



**Figure 12: Create New Neon Database**

After selecting **Create new database** in the drop-down in *Figure 11*, the **Database creation** dialog appears (*Figure 12*). I just enter the database name I want (highlighted in yellow in *Figure 12*), and click create (outlined in red). This takes me back to the **Connect to your database** dialog, but now with my new database selected:



**Figure 13: New Neon Database**

In *Figure 13*, you see the connection string to my new database `event_contact_db` (highlighted in red):

```
postgresql://neondb_owner:*****@ep-raspy-sun-a9vrwlwfl-pooler.gwc.azure.neon.tech/event_contact_db?sslmode=require&channel_binding=require
```

#### Code Snippet 7: Neon Database Connection String

This is the connection string I'll use in my application. Let's break it down:

- **Host:** `ep-raspy-sun-a9vrwlwfl-pooler.gwc.azure.neon.tech`
- **Database:** `event_contact_db`
- **User:** `neondb_owner`
- **Password:** In the connection string (hidden in screenshot)
- **SSL Mode:** `require` (always encrypted)

### Step 4: Secure the Credentials

To secure the credentials, I created an `.env` file in the project root:

```

1 # .env
2 DATABASE_URL=postgresql://nielsb:AbC123DeF456...@ep-cool-meadow-123456.us-east-1.aws.neon.t
3
4 # Note: This file is in .gitignore - never commit it!

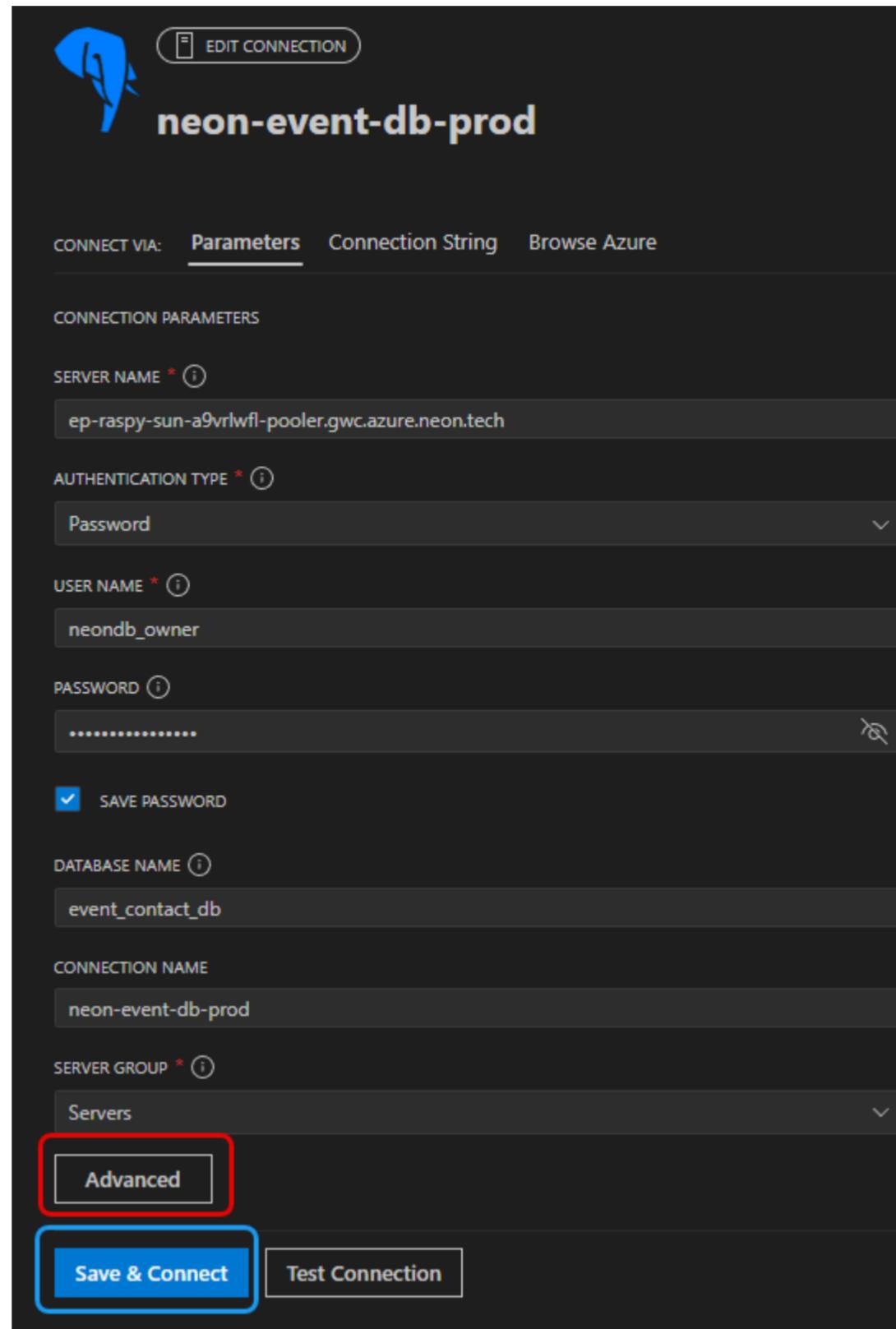
```

#### Code Snippet 8: Neon Database Connection String in .env

As it says in *Code Snippet 8*, I made sure that `.env` is in `.gitignore`, so I don't accidentally commit sensitive credentials.

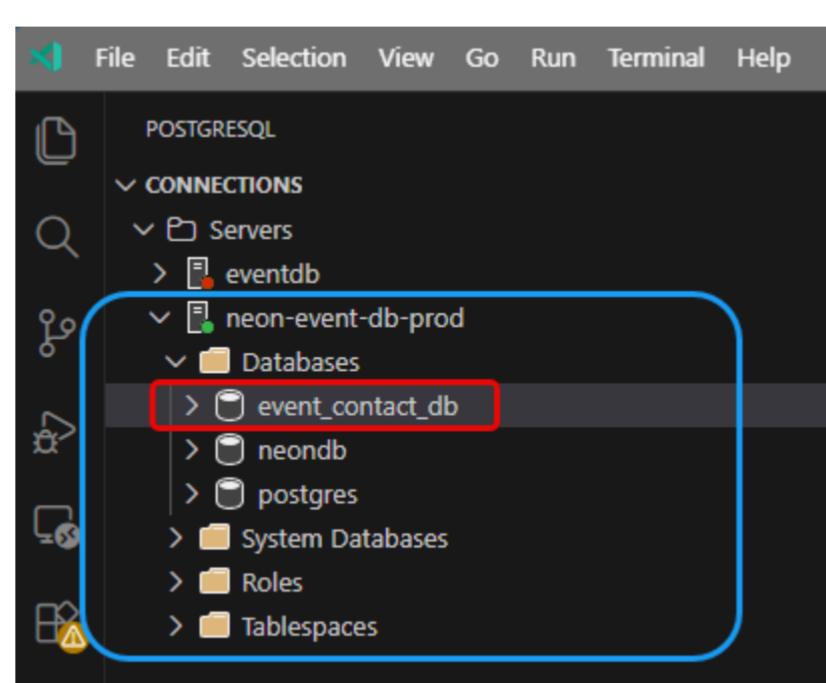
### Step 5: Test the Connection

Since I don't have `psql` installed locally, I decided to use VSCode and the Microsoft PostgreSQL extension to test the connection. I set up a new connection profile using the details from the connection string:



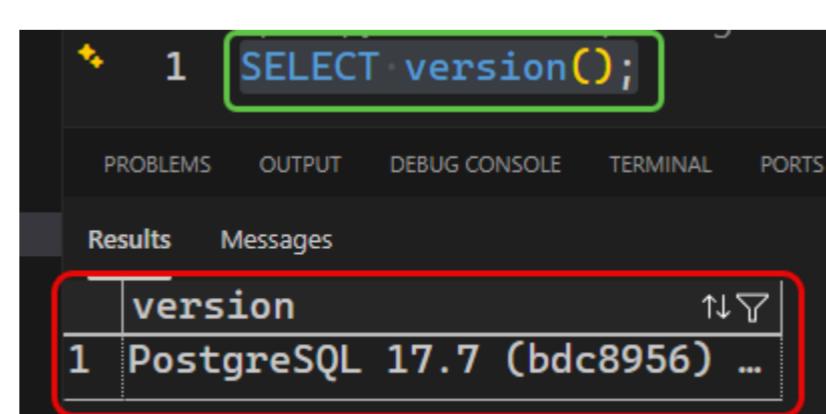
**Figure 14:** VSCode PostgreSQL Connection Details

After entering the connection details (see *Figure 14*), I clicked the **Advanced** button to set the port to `5432`. I clicked **Connect**. A few seconds later, I was connected:



**Figure 15:** VSCode PostgreSQL Connected to Neon Database

You see in *Figure 15* that I'm connected to the Neon (outlined in blue) and, more importantly, my new database `event_contact_db` (outlined in red). To be absolutely sure, I ran a simple query to get the PostgreSQL version:



**Figure 16:** VSCode PostgreSQL Querying Neon Database

In *Figure 16*, you see the query `SELECT version()` (outlined in green), as well as the result (outlined in red), confirming I'm connected to PostgreSQL 17.7 on Neon.

**Connected successfully!** Database is ready.

**Total time:** 8 minutes from account creation to working database.

## What We Have Now

At this point, we have:

- Production-ready PostgreSQL 17 database
- Cloud-hosted on Neon (accessible from anywhere)
- SSL encryption enabled
- Connection string secured in `.env`
- Zero cost (free tier)

The database is empty (no tables yet), but it's ready to receive our schema. We'll design and create the schema in Part 5.

## Installing Postgres MCP Pro

### *Conversational Installation with Claude Code*

Now for the magic: installing and configuring Postgres MCP Pro so Claude can interact with our database through natural language. But rather than just following installation commands, let's ask Claude Code to help us through this process.

### What is Postgres MCP Pro?

First, let me have a conversation with Claude Code to understand what we're installing:

```
Thanks to your help, I now have PostgreSQL installed on Neon. Now I need to install Postgres MCP Pro.

Please explain what PostgreSQL MCP Pro is and help me install it. Please document the steps to install it.
```

#### **Code Snippet 9: Asking Claude Code About Postgres MCP Pro Installation**

Asking Claude Code what is in *Code Snippet 9*, it thought about it. After a few moments, it came back with a detailed explanation and installation guide:

```
<thinking>
• I'll explain what Postgres MCP Pro is and create a comprehensive installation guide for you:
• Write(docs\setup-guides\postgres-mcp-pro-setup.md)

[snipped for brevity]

• I've created a comprehensive guide at docs/setup-guides/postgres-mcp-pro-setup.md. Let me give you a quick summary of what Postgres MCP Pro is and how it works.

What is Postgres MCP Pro?
Postgres MCP Pro is a "bridge" that lets Claude AI talk to your PostgreSQL database through natural language queries.

How it works:
You: "How many contacts are in my database?"
↓
Claude AI: Understands query → Decides to use Postgres MCP Pro
↓
Postgres MCP Pro: Executes "SELECT COUNT(*) FROM contacts"
↓
Your Neon Database: Returns result (e.g., 287)
↓
Claude: "You have 287 contacts in your database."

What tools does it provide?
[snipped for brevity]

---
Full Details

See docs/setup-guides/postgres-mcp-pro-setup.md for:
- Detailed explanation of how MCP works
- Troubleshooting guide
- Security best practices
- What you can do with natural language queries
- Examples of analytics queries

Would you like me to help you through the installation now?
```

#### **Code Snippet 10: Claude Code's Explanation of Postgres MCP Pro and Setup**

What you see in *Code Snippet 10* is an excerpt of the very detailed explanation. Claude Code created a complete setup guide at `docs/setup-guides/postgres-mcp-pro-setup.md`, which you can view as a `gist` here (<https://gist.github.com/niebsberglund/d7646f59854ddc06c760fe1cb92e710a>).

When you browse through the `gist`, you'll see that Claude Code recommended using Docker to install Postgres MCP Pro:

```
## Installation Options

Postgres MCP Pro can be installed 3 ways:

### Option 1: Docker (Recommended) ★

[snipped for brevity]

### Option 2: pipx (Python)

[snipped for brevity]

### Option 3: uv (Modern Python)

[snipped for brevity]

**We'll use Docker (recommended for simplicity).**
```

**Code Snippet 11: Claude Code's Installation Options for Postgres MCP Pro**

We could follow the Docker installation instructions in the setup document, but since this is about AI-assisted coding, let's see how Claude Code would help us install it.

**Installing with Docker**

Since Claude Code in *Code Snippet 10* asked if it should help with installation, I responded in the positive ("Please help me ..." or something similar), Claude Code went off, and:

- Perfect! I'll walk you through the installation step-by-step, checking with you before any config.
- Step 1: Verify Docker is Running
 

Let me check if Docker is installed and running on your system:

  - Bash(docker --version)
    - ↳ Docker version 28.5.1, build e180ab8
  - Bash(docker ps)
    - ↳ Error: Exit code 1  
error during connect: Get "http://%2F%2F.%2Fpipe%2FdockerDesktopLinuxEngine/v1.51/containers//./pipe/dockerDesktopLinuxEngine: The system cannot find the file specified.
  - Docker is installed (version 28.5.1)
  - ✗ Docker Desktop is not running

You need to start Docker Desktop first:

  1. Windows:
    - Look for Docker Desktop in your Start Menu
    - Click to launch it
    - Wait for the "Docker Desktop is running" notification
    - You'll see a Docker whale icon in your system tray
  2. Alternative: Press Windows key and type "Docker Desktop", then click to open

Please start Docker Desktop now, then let me know when you see it's running (you should see the Docker whale icon in your system tray).

Once Docker Desktop is running, I'll continue with the next step.

**Code Snippet 12: Claude Code Guiding Docker Installation**

In *Code Snippet 12*, Claude Code checked if Docker was installed and running. It found Docker was installed but not running, so it guided me to start Docker Desktop.

After I started Docker Desktop, I told Claude Code that Docker Desktop was running. Claude Code continued:

- Perfect! Let me verify Docker is now accessible:
 

[snipped for brevity]
- Excellent!  Docker is running.

Step 2: Pull Postgres MCP Pro Docker Image

Now I'll download the Postgres MCP Pro Docker image from Docker Hub. This is the official image:

- Bash(docker pull crystaldba/postgres-mcp:latest)
  - ↳ latest: Pulling from crystaldba/postgres-mcp  
254e724d7786: Pulling fs layer  
0c2a5709b400: Pulling fs layer  
... +35 lines (ctrl+o to expand)
- Bash(docker images crystaldba/postgres-mcp)
  - ↳ REPOSITORY TAG IMAGE ID CREATED SIZE  
crystaldba/postgres-mcp latest 6a415ab285c6 7 months ago 282MB
- Perfect!  Postgres MCP Pro image downloaded successfully (282MB)

[snipped for brevity]

Step 3: Test MCP Server Connection

Before configuring Claude Desktop, let's test that the MCP server can connect to your Neon database.

I need your Neon PostgreSQL connection string to test this.

[snipped for brevity]

Please paste your Neon connection string here (I'll use it to test the connection, then we'll add it to the MCP configuration).

Note: Your connection string contains a password, but it's safe to share here in our local session.

**Code Snippet 13: Claude Code Continuing Postgres MCP Pro Installation**

In *Code Snippet 13*, Claude Code pulled the Postgres MCP Pro Docker image and verified the download. It then asked for my Neon connection string to test the connection.

I provided the connection string from my .env file:

```

• Perfect! Thank you. Now let me test if Postgres MCP Pro can connect to your Neon database:
• Bash(echo "Testing connection..." && docker run --rm -i -e POSTGRES_CONNECTION_STRING="postgres://neondb_owner:*****@ep-raspy-sun-a9vrlwfl-pooler.gcr.io/crystaldba/postgres-mcp:latest" postgres -V)
[snipped for brevity, and to protect password]
• I see the issue - the MCP server expects the environment variable to be named DATABASE_URI instead of POSTGRES_CONNECTION_STRING
[snipped for brevity]
• Excellent! ✅ Connection test successful!

Test results:
- ✅ MCP server started in UNRESTRICTED mode
- ✅ Successfully connected to your Neon database
- ✅ Connection pool initialized

The important finding: The environment variable should be DATABASE_URI (not POSTGRES_CONNECTION_STRING)

---
Step 4: Configure Claude Desktop

Now I need to add this MCP server to your Claude Desktop configuration.

Before I proceed, I need to:
1. Locate your Claude Desktop config file
2. Create or update it with the Postgres MCP Pro configuration

Location on Windows:
%APPDATA%\Claude\claude_desktop_config.json
Full path: C:\Users\niels\AppData\Roaming\Claude\claude_desktop_config.json

The configuration I'll add:
{
  "mcpServers": {
    "postgres": {
      "command": "docker",
      "args": [
        "run",
        "--rm",
        "-i",
        "-e",
        "DATABASE_URI=postgresql://neondb_owner:*****@ep-raspy-sun-a9vrlwfl-pooler.gcr.io/crystaldba/postgres-mcp:latest"
      ]
    }
  }
}

Questions before I proceed:
1. Do you want me to check if this file already exists and show you its current contents (if any)?
2. Are you okay with me creating/updating this file with the Postgres MCP Pro configuration?

Let me know and I'll proceed!

```

#### **Code Snippet 14: Claude Code Preparing to Configure Claude Desktop for Postgres MCP Pro**

One interesting thing about what you see in *Code Snippet 14* is that Claude Code detected I'm on Windows and provided the correct path to the Claude Desktop config file. It also prepared the configuration block to add. It also initially received an exception due to an environment variable name, which it corrected on its own.

## **Configuring Claude Desktop**

I will now let Claude Code proceed with the configuration. I'd like to see how it handles editing the config file, since it already exists with another MCP server configured. So I ask it to check whether the file exists first, and if it does, show me the contents before proceeding:

```

• Bash(ls -la "$APPDATA/Claude/claude_desktop_config.json" 2>&1)
  L -rw-r--r-- 1 niels 197609 440 Oct 23 18:44 C:\Users\niels\AppData\Roaming\Claude\claude_desktop_config.json
• The file exists! It was last modified on October 23, 2024 (440 bytes).

Let me show you the current contents:

• Read(C:\Users\niels\AppData\Roaming\Claude\claude_desktop_config.json)
  L Read 17 lines

• Great! I can see you already have an MCP server configured - mssql_mcp for SQL Server.

Current configuration:
- Has 1 MCP server: mssql_mcp (SQL Server MCP)
- Connects to a local SQL Server instance

[continues in Code Snippet 16]

```

#### **Code Snippet 15: Claude Code Checking Existing Claude Desktop Config**

Yep, it looks like the file exists. Claude Code read the existing configuration (see *Code Snippet 15*) and found the existing MCP server for SQL Server. That's cool! One thing, though: when you look at what Claude Code says about the file, it says it was last modified on October 23, 2024, at 18:44. October 23 and 18:44 are correct, but the year is off by 1. It should have been 2025. Well, I won't bother investigating this now. That is a problem for another day. Continuing on:

[continued from Code Snippet 15]

---

I'll add the Postgres MCP Pro alongside your existing mssql\_mcp server.

Here's what the updated configuration will look like:

```
{
  "mcpServers": {
    "mssql_mcp": {
      "type": "stdio",
      "command": "node",
      [snipped for brevity]
    },
    "postgres": {
      "command": "docker",
      "args": [
        "run",
        "--rm",
        "-i",
        "-e",
        "DATABASE_URI=pmy-connection-string-here",
        "crystaldba/postgres-mcp:latest"
      ]
    }
  }
}
```

Changes:

- Keeps your existing mssql\_mcp server unchanged
- Adds new postgres server for your Neon database
- You'll have access to both SQL Server and PostgreSQL from Claude

Do you want me to update the file with this configuration?

#### **Code Snippet 16: Claude Code Preparing to Update Claude Desktop Config**

In **Code Snippet 16**, Claude Code prepared the updated configuration, keeping the existing SQL Server MCP and adding the new Postgres MCP Pro. I asked it to update the configuration file:

- Update(C:\Users\niels\AppData\Roaming\Claude\claude\_desktop\_config.json)
  - └ Added 12 lines, removed 1 line
  - 1 [snipped for brevity]

- Perfect!  Configuration file updated successfully!

What was added:

- New postgres MCP server configuration
- Connects to your Neon database
- Kept your existing mssql\_mcp server intact

You now have 2 MCP servers configured:

1. mssql\_mcp - SQL Server (existing)
2. postgres - PostgreSQL/Neon (new)

---

Step 5: Restart Claude Desktop

For the changes to take effect, you need to completely quit and restart Claude Desktop.

Important: Just closing the window is not enough - you must quit the application.

How to Quit Claude Desktop:

Option 1: System Tray

1. Look for the Claude icon in your system tray (bottom-right corner of Windows taskbar)
2. Right-click the Claude icon
3. Click "Quit" or "Exit"

Option 2: Task Manager

1. Press Ctrl + Shift + Esc to open Task Manager
2. Find "Claude" in the list
3. Click "Claude" and click "End Task"

Option 3: Alt+F4

1. Make sure Claude Desktop window is focused
2. Press Alt + F4
3. If prompted, confirm you want to quit

---

After quitting:

1. Wait a few seconds
2. Restart Claude Desktop from Start Menu

Please quit and restart Claude Desktop now, then let me know when it's back up!

#### **Code Snippet 17: Claude Code Completing Postgres MCP Pro Installation**

Claude Code updated the configuration file (see **Code Snippet 17**). It then guided me to completely quit and restart Claude Desktop for the changes to take effect.

**NOTE:** Claude Desktop only reads the MCP server configuration at startup, so a full restart is necessary.

## Testing the Connection

As Claude Code instructed, I quit Claude Desktop completely using the system tray method, waited a few seconds, and then restarted it from the Start Menu. Once it was back up, I informed Claude Code:

• Excellent!  Claude Desktop restarted.

---

Step 6: Test Database Connection via Claude Desktop

Now let's verify that Claude Desktop can access your Neon PostgreSQL database through the MCP server.

**Visual Check First**

Look for MCP indicators in Claude Desktop:

- Some versions show a "Tools" icon or badge
- Or a small "MCP" indicator
- Or "Connected to 2 servers" message

**Test Queries**

Open a new conversation in Claude Desktop and try these test queries:

**Test 1: Check MCP Servers Available**

Ask Claude Desktop:  
What MCP servers are available to you?

Expected response: Should mention both mssql\_mcp and postgres servers.

---

**Test 2: List Database Tables**

Ask Claude Desktop:  
What tables are in my PostgreSQL database?

Expected response: Should list tables (even if database is empty, it might show system tables or nothing).

---

**Test 3: Check Database Connection**

Ask Claude Desktop:  
Can you connect to my Neon PostgreSQL database and tell me the database name and PostgreSQL version?

Expected response: Should show:

- Database name: event\_contact\_db
- PostgreSQL version: 16.x

---

**What to Do:**

1. Open Claude Desktop
2. Start a new conversation
3. Try the test queries above
4. Report back: Let me know what responses you get!

If everything works, you should see Claude successfully querying your Neon database! 🎉

Please try these tests and let me know the results.

#### Code Snippet 18: Claude Code Guiding Postgres MCP Pro Testing

In *Code Snippet 18*, Claude Code guided me through testing the connection. I followed the steps and asked the test questions. Here are the results:

- Visual Check:
- Test 1:  - Both MCP servers listed
- Test 2:  - failure, with message "SSL connection has been closed unexpectedly."

That's not good. What to do?

### Using AI to Troubleshoot the Issue

Instead of googling/Stackoverflow-ing the error message, I decided to report the issue back to Claude Code for help:

The visual test works fine, and Test 1 is OK. However when I try Test 2, connection to the database fails.

#### Code Snippet 19: Reporting Postgres MCP Pro Test Failure to Claude Code

When telling Claude Code about the issue, it goes off and comes back with:

- I see the issue! This is a common SSL connection problem with Neon. The channel\_binding=require

Let me fix this by updating your Claude Desktop configuration with a modified connection string

- The issue: The channel\_binding=require parameter is causing the SSL connection to fail.

The fix: Remove channel\_binding=require from the connection string.

Let me update your Claude Desktop configuration:

```
• Update(C:\Users\niels\AppData\Roaming\Claude\claude_desktop_config.json)`
```

[snipped for brevity]

- Perfect!  Configuration updated - removed the channel\_binding=require parameter.

Let me quickly test this connection works now:

[snipped for brevity]

- Excellent!  Connection test successful with the updated configuration!

---

Next Step: Restart Claude Desktop Again

You need to restart Claude Desktop one more time to pick up the fixed configuration.

Please:

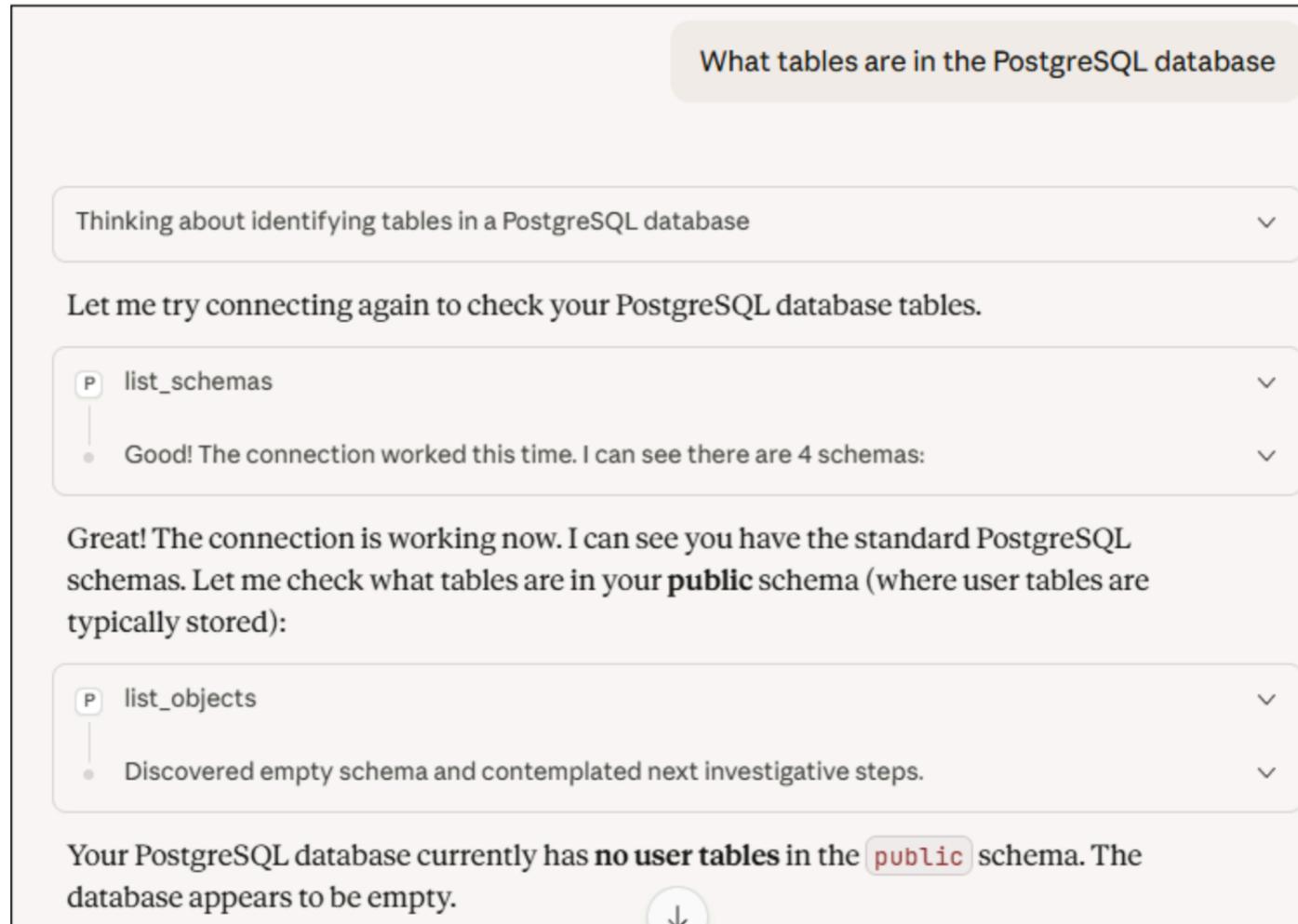
1. Quit Claude Desktop (same as before - right-click system tray icon → Quit)
2. Wait a few seconds
3. Restart Claude Desktop
4. Try Test 2 again: Ask "What tables are in my PostgreSQL database?"

This should now work without the SSL error!

Let me know when you've restarted and tested again.

#### Code Snippet 20: Claude Code Fixing Postgres MCP Pro SSL Connection Issue

I restarted Claude Desktop again as instructed. This time, when I asked Test 2 ("What tables are in my PostgreSQL database?"). It worked perfectly:



**Figure 17: Claude Desktop Successfully Listing Tables from Neon Database**

As we see in *Figure 17*, Claude Desktop successfully listed the tables in my Neon database (which is currently empty, so it shows no user tables). I told Claude Code that everything is now working, and Claude Code asked me what I wanted to do now. I responded that I wanted to create a configuration template file for the repository, so others would know how to set up Postgres MCP Pro. Claude Code went off and:

- Created the template.
- Created a `README.md` file for the Postgres MCP Pro setup.
- Committed all changes and pushed to GitHub

**We have successfully installed and configured Postgres MCP Pro with Claude Desktop, and verified it can connect to our Neon PostgreSQL database.**

## The Value of Conversational Installation

Let's reflect on what just happened with Claude Code:

### Traditional Installation Experience:

1. Read the `README.md` on GitHub
2. Copy-paste commands blindly
3. Hit an error
4. Google the error message
5. Try various StackOverflow solutions
6. Eventually get it working
7. Forget how you fixed it

### AI-Native Installation Experience:

1. Ask Claude Code to help install

2. Claude explains what it's doing
3. Claude handles the installation
4. Claude tests the installation and handles errors
5. Claude creates configuration templates
6. Claude updates documentation
7. Everything is documented in conversation history

#### Key Differences:

- Understanding throughout (not just commands)
- Proactive troubleshooting (test script created)
- Documentation updated automatically
- Complete audit trail in transcripts
- Reproducible for others

This is what **AI-native development** looks like, not just faster, but fundamentally more thoughtful and better documented.

---

## What We've Accomplished

### *Foundation for AI-Native Development*

Let's take stock of what we built today:

#### Infrastructure in Place

##### Professional Project Structure

- Organised directories for database, MCP servers, and docs
- Comprehensive `.gitignore`
- Documentation templates
- Configuration examples

##### Version Control

- Git repository initialised
- Pushed to GitHub with clean commit history
- Proper `.gitignore` protecting sensitive data

##### Conversation History Tool

- `claude-code-transcripts` installed
- Complete audit trail of all decisions
- Reproducible development workflow

##### Production Database

- PostgreSQL 17 on Neon cloud
- Accessible from anywhere (including mobile)
- Automatic backups configured
- SSL encryption enabled
- Zero cost on the free tier

##### MCP Server Connection

- Postgres MCP Pro installed and configured
- Verified working connection
- 8 professional database tools available
- Ready for natural language queries

#### Time Investment

**Total time:** ~45 minutes of actual work

- Repository setup: 10 minutes
- Database decision & setup: 15 minutes
- MCP Pro installation: 10 minutes
- Testing & verification: 10 minutes

**Traditional approach:** Would take 4-6 hours

- Manual project setup: 30 minutes
- Database research & setup: 1-2 hours
- Tool installation & configuration: 1-2 hours
- Documentation writing: 1 hour
- Testing & troubleshooting: 1 hour

## What Makes This Different

### **It's Not Just About Speed**

The productivity gain is obvious, but the real transformation is **how we work**:

#### **Traditional Mindset:**

- "What database provider should I choose?"
- "How do I configure connection pooling?"

- “Where should I host this?”
- “What query builder should I use?”

**AI-Native Mindset:**

- “What do I want to do with this data?”
- “How do I want to ask questions?”
- “What infrastructure gets out of my way?”
- “How can I make this conversational?”

We're **building for conversation, not for forms.**

## What's Next: Part 5 Preview

### *Schema Design and First Conversations*

In the next post, we'll bring this database to life by:

## Part 5: Database Schema and Natural Language Queries

### What We'll Build:

#### 1. Conversational Schema Design

- Use Claude Code to analyse requirements
- Design an entity-relationship model through conversation
- Generate production-ready SQL
- Create indexes strategically
- Add helpful views and functions

#### 2. Schema Creation

- Execute the SQL
- Verify table structure
- Test constraints

#### 3. Sample Data Generation

- Use Claude Code to create realistic test data
- 5 events (2021-2025)
- 30 contacts (realistic South African names)
- Participation records (showing patterns)
- Speaker sessions and ratings

#### 4. First Natural Language Queries

- Simple counts
- Filtered queries
- Complex JOINs with aggregation
- Analytics across years
- Multi-step reasoning

#### 5. Performance Analysis

- Use MCP's EXPLAIN tool
- Index optimisation
- Query tuning

### Example of What's Coming:

```

Me: Who are our most engaged community members? Consider both
attendance and speaking.

Claude: [analyses data, creates engagement metric, runs query]

Top 10 Most Engaged:
1. Niels Berglund - 20 points (5 events attended, 5 presentations)
2. Lawrence Reddy - 20 points (5 events attended, 5 presentations)
...
Should I show you the scoring methodology I used?

```

### Code Snippet 21: Example Natural Language Query with Claude Code

This is where the **paradigm shift becomes real**, from writing SQL to having conversations about data.

## Summary: Building the Foundation

Phew! That was a lot. Let's summarise.

## What We Built Today

### Technical Achievements:

- Professional project structure with Git
- Complete conversation history with `claude-code-transcripts`
- Production PostgreSQL database on Neon
- Postgres MCP Pro installed and working
- Verified end-to-end connection

- Documentation for reproducibility

### More Importantly:

- Experienced AI-native development workflow
- Learned to converse with Claude Code for infrastructure decisions
- Built foundation for conversational database access
- Removed infrastructure as an obstacle
- Set up for mobile database queries

## The Journey Continues

### Series Progress:

- **Part 1:** Installation & Setup - Claude Code on macOS/Windows
- **Part 2:** IDE Integration - VSCode, workflows, advanced features
- **Part 3:** Architecture Planning - AI-native design, MCP strategy
- **Part 4:** Database Infrastructure - PostgreSQL, MCP Pro, foundations (this post)
- **Part 5:** Schema & Conversations - Design, queries, AI-native queries
- **Part 6:** Import Pipeline - Custom MCP server, real data
- **Part 7:** Email Integration - Brevo MCP, communication workflows
- **Part 8:** Production Usage - Real events, analytics, insights

## Why This Matters

We're not just building an event management system. We're demonstrating a **new development paradigm**:

**2023-2024:** AI helps write code faster

**2025:** AI becomes the primary interface

**2026+:** Conversational systems become standard

This series documents that transition as it happens, showing:

- What works (and what doesn't)
- How to think differently about system design
- Practical patterns for real projects
- Why the productivity gains are transformative

## Ready for Part 5

With our infrastructure in place:

- Database ready to receive schema
- MCP server ready to execute queries
- Tools ready for AI-powered development
- Foundation solid for building upward

In Part 5, we'll design our database schema conversationally with Claude Code, create the tables, generate test data, and have our first meaningful natural language conversations with our data.

**The transformation from SQL to conversation starts next.**

## ~ Finally

That's all for this post! We've built the foundation:

- Proper project structure with version control
- Complete development conversation history
- Production-ready cloud database
- MCP server connecting Claude to our data

**Next time:** We'll design the database schema through conversation with Claude Code, create realistic test data, and experience what AI-native database access actually feels like.

## Your Turn

I encourage you to follow along:

1. Set up your own Neon database (free tier)
2. Install Postgres MCP Pro
3. Configure Claude Desktop
4. Test that you can query the database through conversation

Experience the difference yourself. It's genuinely transformative.

### Have questions or thoughts?

- Ping me: niels.it.berglund@gmail.com (mailto:niels.it.berglund@gmail.com)
- Follow on LinkedIn: linkedin.com/in/nielsberglund  
(https://www.linkedin.com/in/nielsberglund/)

**Found this helpful?** Share it with your network! The AI-native development journey is one we're all taking together.