Call a Postgres function | WeWeb Documentation

|  |  |
| --- | --- |
| URL source | <https://docs.weweb.io/workflows/actions/supabase/call-postgres-function.html> |
| Date de scraping | 2025-04-08T15:56:57.053119+00:00 |

## PostgreSQL functions ​

PostgreSQL or database functions are built-in database tools that handle data tasks directly inside your database. Think of them like automatic helpers that can:  
Add up all your monthly sales Fix customer names that were entered with typos Create reports showing which products sold best last week And much more!  
For example, when a customer clicks "View Sales Report" on your web app, it could call a PostgreSQL function named get\_monthly\_sales that instantly calculates and returns total sales for each month, instead of having to write complex code in your frontend to do these calculations.

### Code

Langage: unknown

SELECT CustomerName, City FROM Customers;

## Beyond basic queries ​

Let's imagine you’re building an app that helps track user activity by calculating important metrics based on three connected tables:  
logins : tracks when users log in. posts : tracks posts created by users. comments : tracks comments written by users.  
While you could use SELECT with multiple table joins to gather this information, it creates lengthy, complex queries that need to be repeated everywhere you need these metrics. Using a database function instead lets you write the logic once and reuse it by simply calling get\_user\_activity\_stats() . This makes your code cleaner and easier to maintain, since any changes only need to be made in one place.  
Like having a reusable report template, database functions give you:  
Single source of truth for your query logic Consistent data calculations and formatting Simpler code maintenance  
This is what our function would look like:  
If the code above seems a bit intimidating, don't fret, let's go through it line by line:  
If you call the function with user\_id = 123 , an example output could be:  
If we tried to achieve the same for three users using only SELECT statements, our code would look like this:  
We are repeating SELECT many times. This approach is prone to errors and hard to maintain, hence the usefulness of database functions.  
Learn more about how to create database functions here

### Code

Langage: unknown

CREATE OR REPLACE FUNCTION get\_user\_activity\_stats(user\_id INT)  
RETURNS TABLE(logins INT, posts INT, comments INT) AS $$  
BEGIN  
 RETURN QUERY  
 SELECT  
 (SELECT COUNT(\*) FROM logins WHERE user\_id = user\_id AND login\_date >= NOW() - INTERVAL '30 days') AS logins,  
 (SELECT COUNT(\*) FROM posts WHERE user\_id = user\_id AND post\_date >= NOW() - INTERVAL '30 days') AS posts,  
 (SELECT COUNT(\*) FROM comments WHERE user\_id = user\_id AND comment\_date >= NOW() - INTERVAL '30 days') AS comments;  
END;  
$$ LANGUAGE plpgsq

Langage: unknown

CREATE OR REPLACE FUNCTION get\_user\_activity\_stats(user\_id INT)

Langage: unknown

RETURNS TABLE(logins INT, posts INT, comments INT) AS $$

Langage: unknown

BEGIN  
 RETURN QUERY

Langage: unknown

(SELECT COUNT(\*) FROM logins WHERE user\_id = user\_id AND login\_date >= NOW() - INTERVAL '30 days') AS logins,

Langage: unknown

(SELECT COUNT(\*) FROM posts WHERE user\_id = user\_id AND post\_date >= NOW() - INTERVAL '30 days') AS posts,

Langage: unknown

(SELECT COUNT(\*) FROM comments WHERE user\_id = user\_id AND comment\_date >= NOW() - INTERVAL '30 days') AS comments;

Langage: unknown

END;  
$$ LANGUAGE plpgsql;

Langage: unknown

-- Stats for User 123  
SELECT  
 (SELECT COUNT(\*) FROM logins WHERE user\_id = 123 AND login\_date >= NOW() - INTERVAL '30 days') AS logins,  
 (SELECT COUNT(\*) FROM posts WHERE user\_id = 123 AND post\_date >= NOW() - INTERVAL '30 days') AS posts,  
 (SELECT COUNT(\*) FROM comments WHERE user\_id = 123 AND comment\_date >= NOW() - INTERVAL '30 days') AS comments;  
  
-- Stats for User 456  
SELECT  
 (SELECT COUNT(\*) FROM logins WHERE user\_id = 456 AND login\_date >= NOW() - INTERVAL '30 days') AS logins,  
 (SELECT COUNT(\*) FROM posts WHERE user\_id = 456 AND post\_date >= NOW() - INTERVAL '30 days') AS posts,  
 (SELECT COUNT(\*) FROM comments WHERE user\_id = 456 AND comment\_date >= NOW() - INTERVAL '30 days') AS comments;  
  
-- Stats for User 789  
SELECT  
 (SELECT COUNT(\*) FROM logins WHERE user\_id = 789 AND login\_date >= NOW() - INTERVAL '30 days') AS logins,  
 (SELECT COUNT(\*) FROM posts WHERE user\_id = 789 AND post\_date >= NOW() - INTERVAL '30 days') AS posts,  
 (SELECT COUNT(\*) FROM comments WHERE user\_id = 789 AND comment\_date >= NOW() - INTERVAL '30 days') AS comments;

## Call a Postgres function ​

The Call a Postgres function action in WeWeb, available after installing the Supabase plugin, lets you execute database functions directly from your application.  
  
Consider a customer service ticket management app with these connected tables:  
tickets : basic ticket info. ticket\_status\_history : how many times the ticket was reopened. ticket\_comments : how many comments are waiting for a response. agents : how many tickets the agent is handling. departments : SLA rules for the ticket.  
We could create a function to return ticket stats. For example, you might want to know:  
The total number of tickets in the system How many tickets are currently open vs closed The number of high-priority tickets that need attention The average time it takes to resolve tickets  
Once you are done writing the code for the function, you can add the Call a Postgres function action to call that function.  
To verify your Postgres function execution, use the Logs tab to inspect the returned values:

### Images

<https://docs.weweb.io/assets/postgres.DdTZpToY.png>

<https://docs.weweb.io/assets/postgres-example.ByU36avQ.png>

## PostgreSQL functions vs. Edge Functions ​

It's easy to mix up database functions with Edge Functions . Database functions operate directly within your database, working on your data at its source. On the other hand, edge functions run on distributed servers located closer to users, managing tasks such as authentication, data transformations, and integrations with external services like payment processors.  
Here is a comparison table: