PostgreSQL Recipes: Logging of Statements

Josef Machytka
4 min read · Just now

It might sound like a joke to write yet another article on logging all statements in PostgreSQL — there are already so many of them out there. But you know how it goes: you think it will be easy to quickly find a solution online when you need it, only to discover it's harder than expected.

I have already lost a lot of time trying to "quickly Google" for small solutions. So, I decided to write a series of simple yet comprehensive recipes for various PostgreSQL operations, mainly for my ad-hoc usage. But they could also be useful for others.

Why Log All Statements?

In my case, I often need to debug applications and connection libraries from different programming languages to understand what they're actually sending to the database. Additionally, some problems, especially locking

issues, depend on the sequence of commands sent to the database. Assumptions, as we know, can be — and often are — wrong.

Not surprisingly, in such situations, developers tend to theorize about catastrophic, zombie-like, or even deliberately evil behaviors of PostgreSQL (we all "know" that relational databases are old and bad, right?) instead of considering potential issues in their apps or libraries. In such cases, a complete log of all statements usually helps to uncover flaws in app logic, bad type casting, and more.

Step-by-Step Configuration

1. Enable the Logging Collector

The most important requirement is enabling PostgreSQL's logging collector. By default, it's set to `off`, and changing this setting requires a PostgreSQL restart. So, it's best to enable it right after the installation so we can fine-tune logging later as needed.

```
logging_collector = "on"
```

2. Specify a Logging Directory

The `log_directory` setting specifies where log files will be stored. It can either contain:

- A relative path (default value is `log`), treated as a subdirectory under PostgreSQL's `data_directory`. On Debian, for example, this defaults to `/var/lib/postgresql/16/main` for PostgreSQL 16.

- Or an absolute path, e.g. `/path/to/logs`.

We must of course ensure that the PostgreSQL user can write to the specified directory. No restart is needed for change in this setting.

```
log_directory = "...."
```

3. Adjust the Logging Level

The `log_min_messages` setting controls the verbosity of logs. By default, it's set to `warning`. For debugging, setting it to `info` is usually sufficient. I generally needed more granular settings only for handful of very special cases. Change does not require restart.

```
log_min_messages = "info"
```

4. Configure Log Line Prefix

This is actually a very important setting. it adds contextual information to each log entry. We shall include as much detail as possible. Change does not require restart.

```
log_line_prefix = '%t [%p]: [%l] user=%u,db=%d,app=%a,client=%h,xid=%x %v '
```

This setting will log:

- `%t` : Timestamp
- `%p` : Process ID
- `%l` : Log line number for each session or process
- `%u` : Database user
- `%d` : Database name
- `%a` : Application name
- `%h` : Client hostname
- `%x` : Transaction ID (0 if none assigned)
- `%v` : Virtual transaction ID (procNumber/localXID)

Note: `%x` is always `0` for standalone `SELECT` statements. However, for connection libraries (many of which encapsulate actions in `BEGIN-COMMIT` blocks), this value can be critical for understanding in some situations.

5. Log All Statements

To log all SQL statements, we shall use the `log_statement` setting. It can be changed without restarting PostgreSQL.

```
log_statement = "all"
```

Setting `log_statement` is by default set to `none`, for debugging purposes I strongly recommend to set it to the highest option `all`. In case of complicated issues we really need to see all statements and understand sequence of actions.

6. Set Statement Duration Threshold

The `log_min_duration_statement` setting specifies the minimum execution time (in milliseconds) for a query to be logged. Setting it to `0` logs all statements regardless of duration. Change does not require restart.

```
log_min_duration_statement = "0"
```

Global vs. Local Scope of Settings

Global Changes

It's usually best to apply these settings globally. Even if the problem seems to be limited to a specific user or application. Comprehensive logging often reveals unexpected insights about the entire traffic flow.

Global changes can be made in the `postgresql.conf` file, or in override files included via:

- `include`
- `include_dir`
- `include_if_exists`

These settings for overrides are not visible in PostgreSQL's `pg_settings` table but are part of the main configuration file.

After editing configuration files, we must reload them without restarting PostgreSQL using:

```
SELECT pg_reload_conf();
```

PostgreSQL will log into the main log file any parameter changes during the reload.

User-Specific Settings

In very specific cases, we can configure logging for a specific user with commands like this one. These changes must be done using a superuser account:

```
ALTER USER <username> SET log_statement='all';
```

Monitor Log Directory for File Sizes

And last but not least , allowing this extended logging of all statements can cause skyrocketing growth of log files. It’s very important to monitor disk usage and keep this verbose logging enabled only as long as necessary to catch the problem.

Summary

Logging all SQL statements processed by PostgreSQL is very important for debugging complex issues, especially those related to external applications and their connection libraries. While it requires careful configuration and can produce very large large log files, it often uncovers the root cause of problems that would otherwise remain elusive.

Image created by the author using DeepDreamGenerator

Postgresql Debugging Connection Logging And Monitoring Logging

Written by Josef Machytka

41 Followers · 7 Following

Edit profile

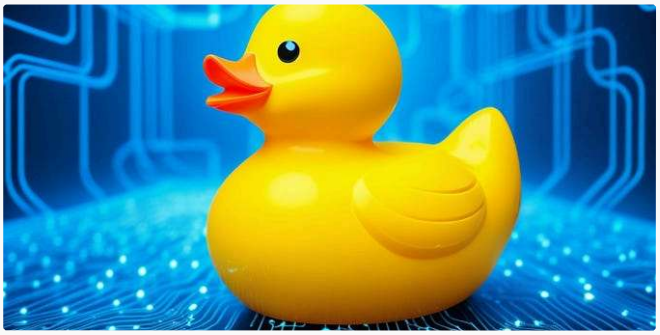
I work as Professional Service Consultant - PostgreSQL specialist in NetApp Deutschland GmbH, Open Source Services division.

No responses yet

What are your thoughts?

Respond

More from Josef Machytka



Josef Machytka

Extending DuckDB ETL Capabilities with Python

```
ate the table
 TABLE special_data_types (
 INT AUTO_INCREMENT PRIMARY KEY,
me VARCHAR(50) NOT NULL,
atus ENUM('active', 'inactive', 'pending') NOT NULL,
ermissions SET('read', 'write', 'execute') NOT NULL,
all_number TINYINT NOT NULL,
dium_number MEDIUMINT NOT NULL,
scription TEXT,
ta BLOB,
eated_at DATE NOT NULL

ert 10 rows of data
 INTO special_data_types (name, status, permissions, small_number, medium_number, description, data, created_at)
e', 'active', 'read,write', 5, 1000, 'Alice description', 'Alice data', '2023-01-01'),
, 'inactive', 'read', 10, 2000, 'Bob description', 'Bob data', '2023-02-01'),
lie', 'pending', 'write,execute', 15, 3000, 'Charlie description', 'Charlie data', '2023-03-01'),
d', 'active', 'read,write,execute', 20, 4000, 'David description', 'David data', '2023-04-01'),
, 'inactive', 'execute', 25, 5000, 'Eve description', 'Eve data', '2023-05-01'),
k', 'pending', 'read,write', 30, 6000, 'Frank description', 'Frank data', '2023-06-01'),
e', 'active', 'read', 35, 7000, 'Grace description', 'Grace data', '2023-07-01'),
', 'inactive', 'write,execute', 40, 8000, 'Hank description', 'Hank data', '2023-08-01'),
, 'pending', 'read,write,execute', 45, 9000, 'Ivy description', 'Ivy data', '2023-09-01'),
, 'active', 'execute', 50, 10000, 'Jack description', 'Jack data', '2023-10-01');
```

Josef Machytka

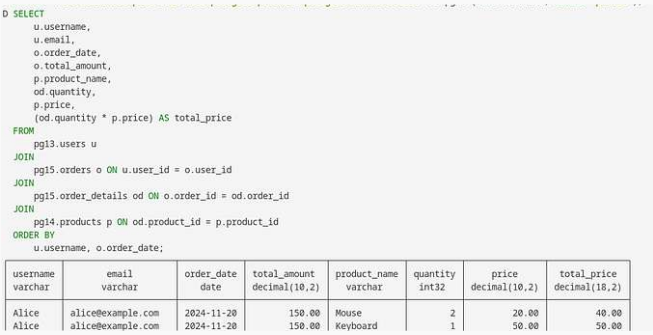
DuckDB as a Rudimentary Data Migration Tool

DuckDB has recently become my go-to solution for small ETL tasks. It is an...

Nov 24 20

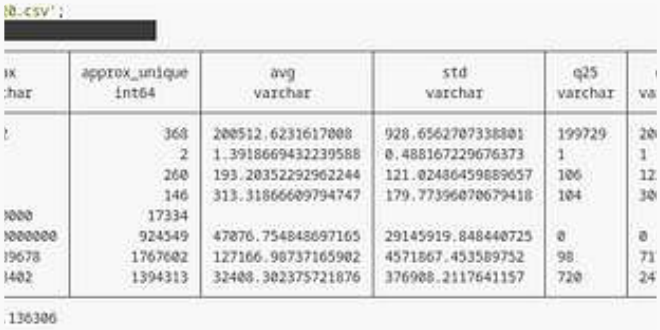
After exploring how to use DuckDB as an intelligent ETL tool for PostgreSQL, and how...

Nov 30 1



```
D SELECT
      u.username,
      u.email,
      o.order_date,
      o.total_amount,
      p.product_name,
      od.quantity,
      p.price,
      (od.quantity * p.price) AS total_price
  FROM
      pg13.users u
  JOIN
      pg15.orders o ON u.user_id = o.user_id
  JOIN
      pg15.order_details od ON o.order_id = od.order_id
  JOIN
      pg14.products p ON od.product_id = p.product_id
  ORDER BY
      u.username, o.order_date;
```

username varchar	email varchar	order_date date	total_amount decimal(10,2)	product_name varchar	quantity int32	price decimal(10,2)	total_price decimal(18,2)
Alice	alice@example.com	2024-11-20	150.00	Mouse	2	20.00	40.00
Alice	alice@example.com	2024-11-20	150.00	Keyboard	1	50.00	50.00



approx_unique int64	avg varchar	std varchar	q25 varchar
368	200512.6231617008	928.6562707338801	199729
2	1.3918669432239588	0.488167229676373	1
260	193.20352292962244	121.02486459889657	106
146	313.31866609794747	179.77396070679418	104
17334			
924549	47076.754848697165	29145919.848440725	0
1767602	127166.98737165902	4571867.453589752	98
1394313	32408.302375721876	376908.2117641157	720

Josef Machytka

Easy Cross-Database Selects with DuckDB

DuckDB was created with simplicity and ease of use in mind. In my previous article, I...

Nov 26 3

Josef Machytka

Using DuckDB as an Intelligent ETL tool for PostgreSQL

There is a lot of hype around DuckDB these days. At one PostgreSQL conference, I even...

Nov 2 7

See all from Josef Machytka

Recommended from Medium

Paul Ammann

Python Design Patterns: Best Practices and Anti-Patterns

Understanding not just what design patterns to use, but also how to use them correctly—...

Dec 11 101

In The Pythoneers by Abhay Parashar

40 Python One-Liners You Must Try

Why Code More, When You Can Do It In One Line ??

6d ago 256 9

Lists

Stories to Help You Grow as a Software Developer

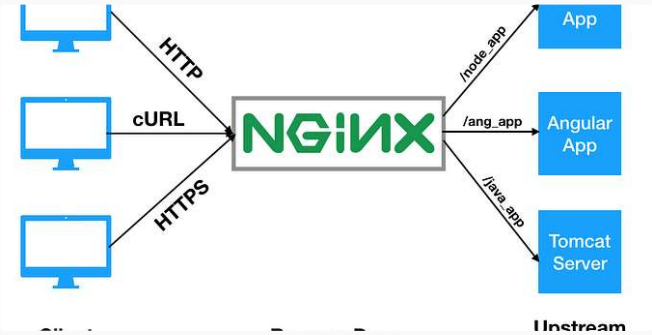
19 stories · 1522 saves

Visual Storytellers Playlist

61 stories · 564 saves

Staff picks

789 stories · 1514 saves



Mr.PlanB

Rob Blackbourn

Home Hosting 101: Reverse Proxies, WAFs, and Network...

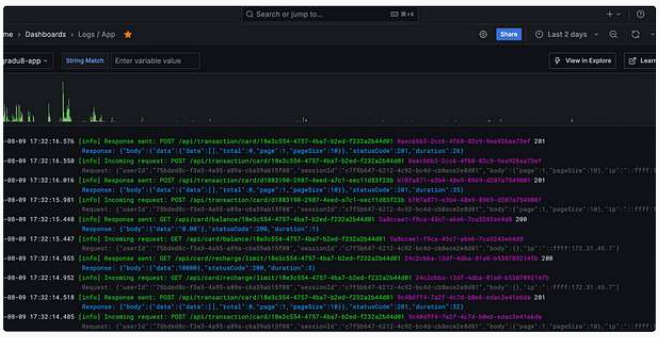
If you’re running a homelab or hosting services at home, you’ve probably heard...

Dec 11 40

How to use Python logging QueueHandler with dictConfig in...

Support was added for configuring the logging queue handler with dictConfig in...

Aug 31



Ahmad Bilal

Integrating NestJS with Loki via Winston and Prometheus: Part 2 ...

In this article, we’ll integrate NestJS with Loki using Winston for logging and Prometheus...

Aug 12 58

In Level Up Coding by Rahul Beniwal

Don’t Let Celery Fail in Production: The Ultimate Guide to...

Ensure Scalability and Reliability with Celery in Python for Production-Ready Task...

Dec 9 51 1

See more recommendations