




Let's make it better, now, together!

 [Home](#) [About](#) [Download](#) [Documentation](#) [Community](#) [Developers](#) [Support](#) [Donate](#) [Your account](#)

Search for... 

13th October 2022: PostgreSQL 15 Released!

[Documentation](#) → [PostgreSQL 15](#)
Supported Versions: [Current \(15\)](#) / [14](#) / [13](#) / [12](#) / [11](#) / [10](#)
Development Versions: [devel](#)
Unsupported versions: [9.6](#) / [9.5](#) / [9.4](#) / [9.3](#) / [9.2](#) / [9.1](#) / [9.0](#) / [8.4](#) / [8.3](#) / [8.2](#) / [8.1](#) / [8.0](#) / [7.4](#) / [7.3](#) / [7.2](#)

Search the documentation for... 

PostgreSQL 15.0 Documentation

[Next](#)

PostgreSQL 15.0 Documentation

The PostgreSQL Global Development Group

Copyright © 1996–2022 The PostgreSQL Global Development Group

[Legal Notice](#)

Table of Contents

[Preface](#)

- [1. What Is PostgreSQL?](#)
- [2. A Brief History of PostgreSQL](#)
- [3. Conventions](#)
- [4. Further Information](#)
- [5. Bug Reporting Guidelines](#)

[I. Tutorial](#)

- [1. Getting Started](#)
- [2. The SQL Language](#)
- [3. Advanced Features](#)

[II. The SQL Language](#)

- [4. SQL Syntax](#)
- [5. Data Definition](#)

Sarah Haïm-Lubczanski

- Documentation Architect
 - (previously Technical Writer, previously PHP Developer)
 - I like Monty Python, riding my bike, eating pizzas, and learn new things everyday.
- Twitter addict : @sarahhaim



Twitter is my addiction

- Too often relying on tweets
- This time...
Used it to learn more about PostgreSQL
- What does Twitter say?

Twitter user said...



Andrew Meredith

@a4w_m6h



Postgres is incredible documentation on how to build an RDBMS that happens to come with a production-ready reference implementation and extension framework.

#PostgreSQL

6:48 PM · Sep 17, 2021 · Twitter for Android

Amazing doc / seen on Twitter



Andrew Meredith

@a4w_m6h



Yep - it's just me. Still over here being amazed with how good the documentation and community for Postgres are.

12:23 AM · Sep 28, 2021 · Twitter Web App

How I got here

I read many documentations
Training my colleagues

Disclaimer about PostgreSQL



A trigger is a specification that the database should automatically execute a particular function whenever a certain type of operation is performed. Triggers can be attached to tables (partitioned or not), views, and foreign tables.

On tables and foreign tables, triggers can be defined to execute either before or after any `INSERT`, `UPDATE`, or `DELETE` operation, either once per modified row, or once per SQL statement. `UPDATE` triggers can moreover be set to fire only if certain columns are mentioned in the `SET` clause of the `UPDATE` statement. Triggers can also fire for `TRUNCATE` statements. If a trigger event occurs, the trigger's function is called at the appropriate time to handle the event.

On views, triggers can be defined to execute instead of `INSERT`, `UPDATE`, or `DELETE` operations. Such `INSTEAD OF` triggers are fired once for each row that needs to be modified in the view. It is the responsibility of the trigger's function to perform the necessary modifications to the view's underlying base table(s) and, where appropriate, return the modified row as it will appear in the view. Triggers on views can also be defined to execute once per SQL statement, before or after `INSERT`, `UPDATE`, or `DELETE` operations. However, such triggers are fired only if there is also an `INSTEAD OF` trigger on the view. Otherwise, any statement targeting the view must be rewritten into a statement affecting its underlying base table(s), and then the triggers that will be fired are the ones attached to the base table(s).

The trigger function must be defined before the trigger itself can be created. The trigger function must be declared as a function taking no arguments and returning type `trigger`. (The trigger function receives its input through a specially-passed `TriggerData` structure, not in the form of ordinary function arguments.)

Once a suitable trigger function has been created, the trigger is established with `CREATE TRIGGER`. The same trigger function can be used for multiple triggers.

PostgreSQL offers both *per-row* triggers and *per-statement* triggers. With a per-row trigger, the trigger function is invoked once for each row that is affected by the statement that fired the trigger. In contrast, a per-statement trigger is invoked only once when an appropriate statement is executed, regardless of the number of rows affected by that statement. In particular, a statement that affects zero rows will still result in the execution of any applicable per-statement triggers. These two types of triggers are sometimes called *row-level* triggers and *statement-level* triggers, respectively. Triggers on `TRUNCATE` may only be defined at statement level, not per-row.

Triggers are also classified according to whether they fire *before*, *after*, or *instead of* the operation. These are referred to as `BEFORE` triggers, `AFTER`

It's open source, I know



Agenda

#0 : get the documentation

#1 : architecture of the content

#2 : the content

#3 : user experience with the doc

- What is already good ?
 - What can we improve ?
-

Doc is a tool

- Some of you can build Ikea furniture without notice
- Can you screw without a screwdriver ?



0 / Contributing

Contribute : how do I get the doc?

- Online version
- With PostgreSQL
- Then I read Appendix J

Note that for historical reasons the documentation source files are named with an extension `.sgml` even though they are now XML files. So you might need to adjust your editor configuration to set the correct mode.

The Wiki / TODO

- There is a [Documentation part](#) : good
- Why is this not an issue?

Documentation

- ☐ Provide a manpage for postgresql.conf
 - [A smaller default postgresql.conf](#)
 - [A smaller default postgresql.conf](#)
- ☐ Document support for N' ' national character string literals, if it matches the SQL standard
 - <http://archives.postgresql.org/message-id/1275895438.1849.1.camel@fsopti579.F-Secure.com>

- Improving: delete it

Versions

- Many versions
- Latest is visible
-
-
- **Why** so many versions available?
-
- Delivered with the software
- Archived versions are accessible

Manuals

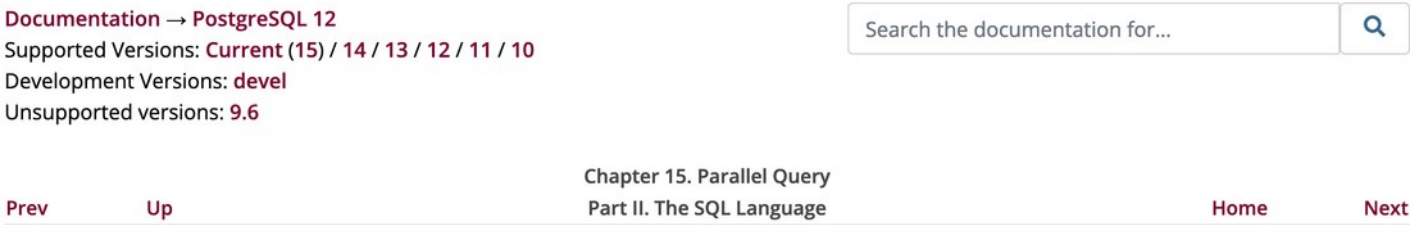
You can view the manual for an older version or download a PDF of a manual from the below table.

Online Version	PDF Version
15 / Current	A4 PDF (13.5 MB) • US PDF (13.4 MB)
14	A4 PDF (13.3 MB) • US PDF (13.2 MB)
13	A4 PDF (12.9 MB) • US PDF (12.8 MB)
12	A4 PDF (12.6 MB) • US PDF (12.5 MB)
11	A4 PDF (12.3 MB) • US PDF (12.2 MB)
10	A4 PDF (12.0 MB) • US PDF (11.9 MB)
Development snapshot	PDF version not available

Looking for documentation for an older, unsupported, version? Check the **archive** of older manuals.

Improvement / Versioning


- Not so clear : am I viewing an old version of the doc?

- 

The screenshot shows the top navigation bar of the PostgreSQL documentation. On the left, it includes links for 'Documentation', 'PostgreSQL 12', 'Supported Versions' (listing Current (15), 14, 13, 12, 11, and 10), 'Development Versions' (listing devel), and 'Unsupported versions' (listing 9.6). On the right, there is a search bar with the placeholder text 'Search the documentation for...' and a magnifying glass icon. Below this bar, a breadcrumb trail shows 'Chapter 15. Parallel Query' and 'Part II. The SQL Language'. At the bottom of the header, there are navigation links: 'Prev', 'Up', 'Home', and 'Next'.
- Chapter 15. Parallel Query

Coming from Google or StackOverflow

Example: Docusaurus shows version

 **Docusaurus** Docs [API](#) [Blog](#) [Showcase](#) [Community](#)

2.1.0 ▼  English ▼

Introduction

Getting Started ▼

Installation

Configuration

Playground

TypeScript Support

Guides >


Advanced Guides ▼

Architecture

Plugins

Routing

Static site generation

 > Advanced Guides > Routing

Version: 2.1.0

Routing

Docusaurus' routing system follows single-page application conventions: one route, one component. In this section, we will begin by talking about routing within the three content plugins (docs, blog, and pages), and then go beyond to talk about the underlying routing system.

Routing in content plugins

Every content plugin provides a `routeBasePath` option. It defines where the plugins append their routes to. By default, the docs plugin puts its routes under `/docs`; the blog plugin, `/blog`; and the pages plugin, `/`. You can think about the route structure like this:

Appendix J : documentation

Appendix J. Documentation

Table of Contents

- J.1. DocBook
- J.2. Tool Sets
 - J.2.1. Installation on Fedora, RHEL, and Derivatives
 - J.2.2. Installation on FreeBSD
 - J.2.3. Debian Packages
 - J.2.4. macOS
 - J.2.5. Detection by `configure`
- J.3. Building the Documentation
 - J.3.1. HTML
 - J.3.2. Manpages
 - J.3.3. PDF
 - J.3.4. Plain Text Files
 - J.3.5. Syntax Check
- J.4. Documentation Authoring
 - J.4.1. Emacs
- J.5. Style Guide
 - J.5.1. Reference Pages

PostgreSQL has four primary documentation formats:

- Plain text, for pre-installation information
- HTML, for on-line browsing and reference
- PDF, for printing
- man pages, for quick reference.

Additionally, a number of plain-text README files can be found throughout the PostgreSQL source tree, documenting various implementation issues.

HTML documentation and man pages are part of a standard distribution and are installed by default. PDF format documentation is available separately for download.

How we could participate in doc ?

- Fill a form, if I spot a typo

[Submit correction](#)

If you see anything in the documentation that is not correct, does not match your experience with the particular feature or requires further clarification, please use [this form](#) to report a documentation issue.

- Hacktoberfest ?

SGML/Docbook

Improvement/ contribute to the doc

- Process ?
- Make a Pull/Merge Request
 - Who is reviewing it?
- What is the update cycle for the documentation?



Documentation Review

- Governance is not clear
 - from an external point of view



Improvement / contribution to doc

Do you want to lower the entry bar?

External resources / the good

- Links towards tutorials or how-tos
 - <https://www.postgresql.org/docs/online-resources>
- Blogs of the community
 - <https://planet.postgresql.org>




External resources:more ?


- Not distributed with the doc
- Videos tutorials, maybe?
 - It takes time to maintain.



Example : Airtable uses video

 [Product](#) > [Solutions](#) > [Pricing](#) [Enterprise](#) > [Resources](#) >




Home > [Guides](#) > [Build your workflow](#) > [Add data with records](#)

WRITTEN BY
 [Airtable](#)

FILED UNDER
[Build your workflow](#)

TOPICS

- [Add data with records](#)
- [What is a record?](#)
- [Same record, different perspective](#)
- [Take action: Add records to your table](#)

SHARE
  

[← Guides](#)

[← Build your workflow](#)


Add data with records

With your base's structure set up, you're ready to start filling in all the items you need to track for your workflow.

Records are the core unit of your Airtable base. Each one might look like a simple item in a list, but they're actually the most important building blocks for a dynamic workflow.

Basics

What is a record?



What is a record?

A record is an individual item in a table, along with all of its relevant details. You can think of it as the individual unit of the table—if your table is organizing events at a conference, each record would be a presentation, or if you're producing a television series, each record would be an individual episode.

airtable.com

Build your workflow > Add data with records > What is a record?

What is a record?

1 min

Part 1 / Architecture

« Home » link

- Always visible
- Leads to the all the content

[Documentation](#) → [PostgreSQL 15](#)

Supported Versions: [Current \(15\)](#) / [14](#) / [13](#) / [12](#) / [11](#) / [10](#)

Development Versions: [devel](#)

Unsupported versions: [9.6](#) / [9.5](#) / [9.4](#) / [9.3](#) / [9.2](#) / [9.1](#) / [9.0](#) / [8.4](#) / [8.3](#) / [8.2](#) / [8.1](#) / [8.0](#) / [7.4](#) / [7.3](#)



[Prev](#)

[Up](#)

5.4. Constraints
Chapter 5. Data Definition

[Home](#)

[Next](#)

5.4. Constraints
Chapter 5. Data Definition

[Home](#)

[Next](#)



Cedit : Luke Rauscher, CC BY 2.0

Many many chapters

Table of Contents

Preface

1. What Is PostgreSQL?
2. A Brief History of PostgreSQL
3. Conventions
4. Further Information
5. Bug Reporting Guidelines

I. Tutorial

1. Getting Started
2. The SQL Language
3. Advanced Features

II. The SQL Language

4. SQL Syntax
5. Data Definition
6. Data Manipulation
7. Queries
8. Data Types
9. Functions and Operators
10. Type Conversion
11. Indexes
12. Full Text Search
13. Concurrency Control
14. Performance Tips
15. Parallel Query

III. Server Administration

16. Installation from Binaries
17. Installation from Source Code
18. Installation from Source Code on Windows
19. Server Setup and Operation
20. Server Configuration
21. Client Authentication
22. Database Roles
23. Managing Databases
24. Localization
25. Routine Database Maintenance Tasks

25. Routine Database Maintenance Tasks

26. Backup and Restore
27. High Availability, Load Balancing, and Replication
28. Monitoring Database Activity
29. Monitoring Disk Usage
30. Reliability and the Write-Ahead Log
31. Logical Replication
32. Just-in-Time Compilation (JIT)
33. Regression Tests

IV. Client Interfaces

34. libpq — C Library
35. Large Objects
36. ECPG — Embedded SQL in C
37. The Information Schema

V. Server Programming

38. Extending SQL
39. Triggers
40. Event Triggers
41. The Rule System
42. Procedural Languages
43. PL/pgSQL — SQL Procedural Language
44. PL/Tcl — Tcl Procedural Language
45. PL/Perl — Perl Procedural Language
46. PL/Python — Python Procedural Language
47. Server Programming Interface
48. Background Worker Processes
49. Logical Decoding
50. Replication Progress Tracking
51. Archive Modules

VI. Reference

- I. SQL Commands
- II. PostgreSQL Client Applications
- III. PostgreSQL Server Applications

VII. Internals

52. Overview of PostgreSQL Internals
53. System Catalogs

II. PostgreSQL Client Applications

III. PostgreSQL Server Applications

VII. Internals

52. Overview of PostgreSQL Internals
53. System Catalogs
54. System Views
55. Frontend/Backend Protocol
56. PostgreSQL Coding Conventions
57. Native Language Support
58. Writing a Procedural Language Handler
59. Writing a Foreign Data Wrapper
60. Writing a Table Sampling Method
61. Writing a Custom Scan Provider
62. Genetic Query Optimizer
63. Table Access Method Interface Definition
64. Index Access Method Interface Definition
65. Generic WAL Records
66. Custom WAL Resource Managers
67. B-Tree Indexes
68. GIST Indexes
69. SP-GiST Indexes
70. GiN Indexes
71. BRIN Indexes
72. Hash Indexes
73. Database Physical Storage
74. System Catalog Declarations and Initial Contents
75. How the Planner Uses Statistics
76. Backup Manifest Format

VIII. Appendixes

- A. PostgreSQL Error Codes
- B. Date/Time Support
- C. SQL Key Words
- D. SQL Conformance
- E. Release Notes
- F. Additional Supplied Modules
- G. Additional Supplied Programs
- H. External Projects

d ?

VIII. Appendixes

- A. PostgreSQL Error Codes
- B. Date/Time Support
- C. SQL Key Words
- D. SQL Conformance
- E. Release Notes
- F. Additional Supplied Modules
- G. Additional Supplied Programs
- H. External Projects
- I. The Source Code Repository
- J. Documentation
- K. PostgreSQL Limits
- L. Acronyms
- M. Glossary
- N. Color Support
- O. Obsolete or Renamed Features

Bibliography Index

Table of Contents

Preface

1. What Is PostgreSQL?
2. A Brief History of PostgreSQL
3. Conventions
4. Further Information
5. Bug Reporting Guidelines

I. Tutorial

1. Getting Started
2. The SQL Language
3. Advanced Features

II. The SQL Language

4. SQL Syntax
5. Data Definition
6. Data Manipulation
7. Queries
8. Data Types
9. Functions and Operators
10. Type Conversion
11. Indexes
12. Full Text Search
13. Concurrency Control
14. Performance Tips
15. Parallel Query

III. Server Administration

16. Installation from Binaries
17. Installation from Source Code
18. Installation from Source Code on Windows
19. Server Setup and Operation
20. Server Configuration
21. Client Authentication
22. Database Roles
23. Managing Databases
24. Localization
25. Routine Database Maintenance Tasks
26. Backup and Restore
27. High Availability, Load Balancing, and Replication
28. Monitoring Database Activity
29. Monitoring Disk Usage
30. Reliability and the Write-Ahead Log
31. Logical Replication
32. Just-in-Time Compilation (JIT)
33. Regression Tests

IV. Client Interfaces

34. libpq — C Library
35. Large Objects
36. ECPG — Embedded SQL in C
37. The Information Schema

V. Server Programming

38. Extending SQL
39. Triggers
40. Event Triggers
41. The Rule System
42. Procedural Languages
43. PL/pgSQL — SQL Procedural Language
44. PL/Tcl — Tcl Procedural Language
45. PL/Perl — Perl Procedural Language
46. PL/Python — Python Procedural Language
47. Server Programming Interface
48. Background Worker Processes
49. Logical Decoding
50. Replication Progress Tracking
51. Archive Modules

VI. Reference

- I. SQL Commands
- II. PostgreSQL Client Applications
- III. PostgreSQL Server Applications

VII. Internals

52. Overview of PostgreSQL Internals
53. System Catalogs
54. System Views
55. Frontend/Backend Protocol
56. PostgreSQL Coding Conventions
57. Native Language Support
58. Writing a Procedural Language Handler
59. Writing a Foreign Data Wrapper
60. Writing a Table Sampling Method
61. Writing a Custom Scan Provider
62. Genetic Query Optimizer
63. Table Access Method Interface Definition
64. Index Access Method Interface Definition
65. Generic WAL Records
66. Custom WAL Resource Managers
67. B-Tree Indexes
68. GIST Indexes
69. SP-GiST Indexes
70. GiN Indexes
71. BRIN Indexes
72. Hash Indexes
73. Database Physical Storage
74. System Catalog Declarations and Initial Contents
75. How the Planner Uses Statistics
76. Backup Manifest Format

VIII. Appendixes

- A. PostgreSQL Error Codes
- B. Date/Time Support
- C. SQL Key Words
- D. SQL Conformance
- E. Release Notes
- F. Additional Supplied Modules
- G. Additional Supplied Programs
- H. External Projects
- I. The Source Code Repository
- J. Documentation
- K. PostgreSQL Limits
- L. Acronyms
- M. Glossary
- N. Color Support
- O. Obsolete or Renamed Features

Bibliography

Index

Index

Sections

- Part I. Tutorial
- Part II. The SQL Language
- Part III. Server Administration
- Part IV. Client Interfaces
- Part V. Server Programming
- Part VI. Reference
- Part VII. Internals



Part VIII. Appendixes

Some are scary

- Part V. Server Programming
- What will I program?
- Naming



MySQL 8.0 Reference Manual

- Preface and Legal Notices
- ▼ General Information
 - About This Manual
 - › Overview of the MySQL Database Management System
 - What Is New in MySQL 8.0
 - Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 8.0
 - MySQL Information Sources
 - How to Report Bugs or Problems
 - › MySQL Standards Compliance
 - › Credits
- › Installing and Upgrading MySQL
- › Tutorial
- › MySQL Programs
- › MySQL Server Administration
- › Security
- › Backup and Recovery
- › Optimization
- › Language Structure
- › Character Sets, Collations, Unicode
- › Data Types
- › Functions and Operators
- › SQL Statements
- › MySQL Data Dictionary
- › The InnoDB Storage Engine
- › Alternative Storage Engines
- › Replication

MySQL 8.0 Reference Manual

version 8.0 ▼

Including MySQL NDB Cluster 8.0

Abstract

This is the MySQL Reference Manual. It documents MySQL 8.0 through 8.0.32, as well as NDB Cluster releases based on version 8.0 of [NDB](#) through 8.0.32-ndb-8.0.32, respectively. It may include documentation of features of MySQL versions that have not yet been released. For information about which versions have been released, see the [MySQL 8.0 Release Notes](#).

MySQL 8.0 features. This manual describes features that are not included in every edition of MySQL 8.0; such features may not be included in the edition of MySQL 8.0 licensed to you. If you have any questions about the features included in your edition of MySQL 8.0, refer to your MySQL 8.0 license agreement or contact your Oracle sales representative.

For notes detailing the changes in each release, see the [MySQL 8.0 Release Notes](#).

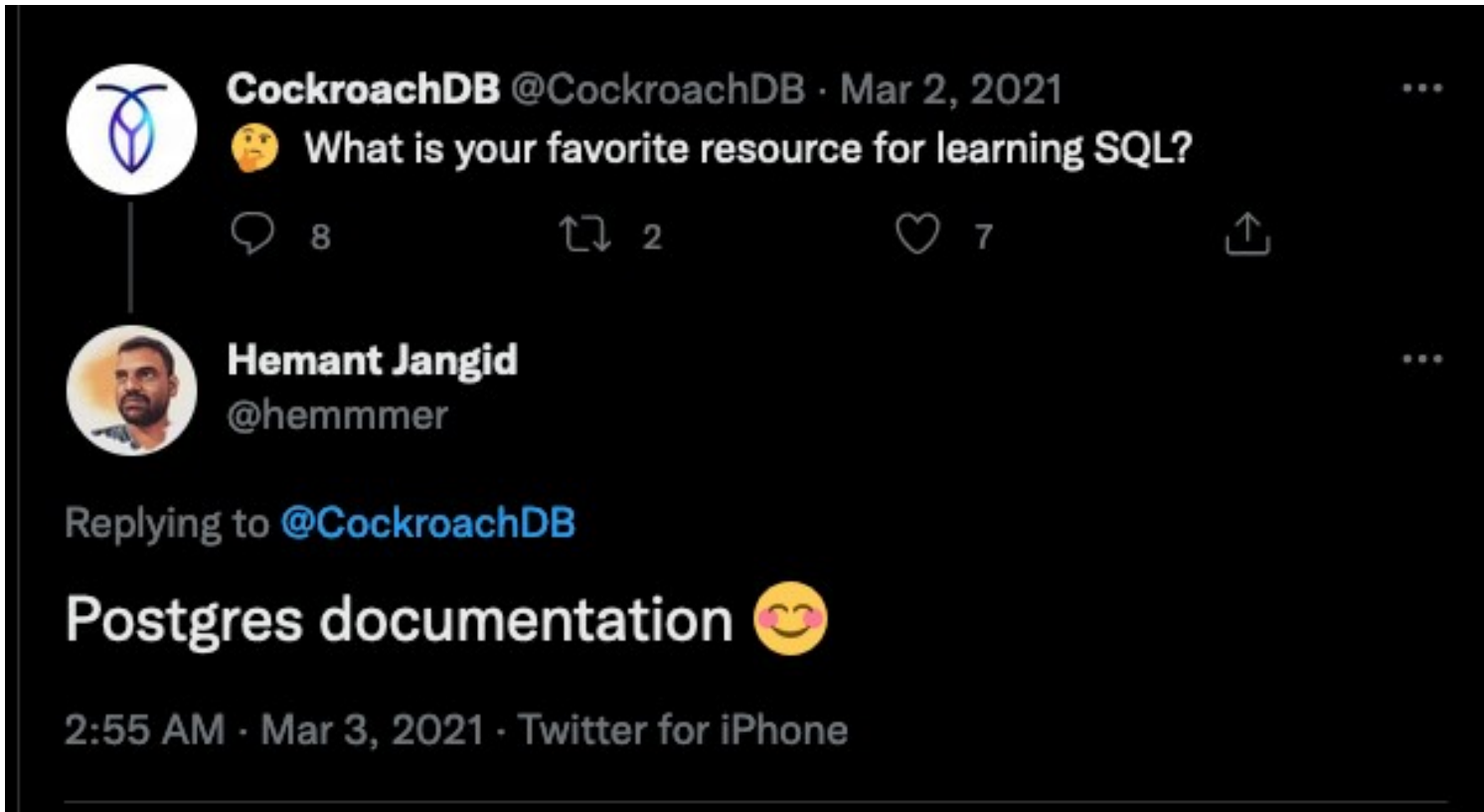
For legal information, including licensing information, see the [Preface and Legal Notices](#).

For help with using MySQL, please visit the [MySQL Forums](#), where you can discuss your issues with other MySQL users.

Document generated on: 2022-10-26 (revision: 74410)

[HOME](#) [NEXT](#) ▶

Learning SQL / seen on Twitter



SQL, it's basic

II. The SQL Language

- 4. SQL Syntax
- 5. Data Definition
- 6. Data Manipulation
- 7. Queries
- 8. Data Types
- 9. Functions and Operators
- 10. Type Conversion
- 11. Indexes
- 12. Full Text Search
- 13. Concurrency Control
- 14. Performance Tips
- 15. Parallel Query

31. Archive Modules

VI. Reference

I. SQL Commands

37. The Information Schema

V. Server Programming

38. Extending SQL

39. Tutorial

Improving : renaming sections



26 years

- 26 years of this content organisation
- for this content
- you don't change it so quickly



Twitter people are quite happy



typedfemale
@typedfemale

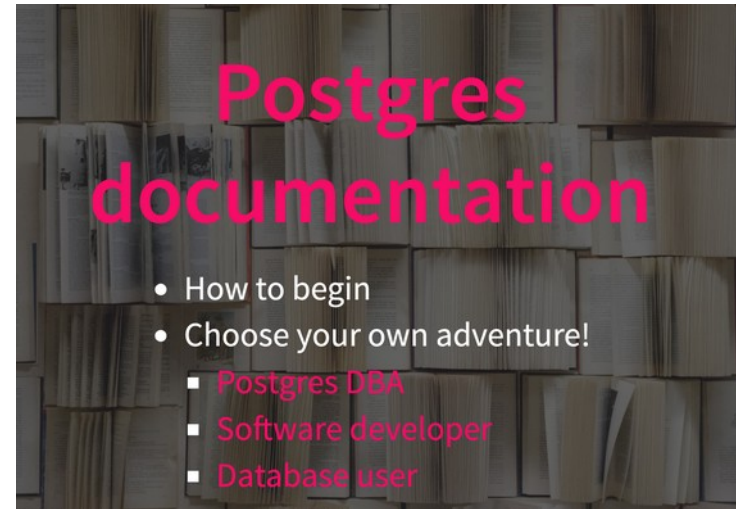


i live my life in accordance with the postgres
documentation

1:55 AM · Jun 6, 2022 · Twitter for iPhone


Improvement : specific paths

- Have an orientation screen/page
 - Path for developers
 - Path for DBAs
 - Path for users
- cf [conference by Laetitia Avrot](#)



And in 2000, a re-org was discussed

Documentation organization

From: Peter Eisentraut <peter_e(at)gmx(dot)net>
To: pgsql-docs(at)postgresql(dot)org
Subject: Documentation organization
Date: 2000-06-29 17:28:13
Message-ID: [Pine.LNX.4.21.0006291616400.397-100000@localhost.localdomain](#)
Views: [Raw Message](#) | [Whole Thread](#) | [Download mbox](#) | [Resend email](#)
Thread: 2000-06-29 17:28:13 from Peter Eisentraut <peter_e(at)gmx(dot)net> 
Lists: [pgsql-docs](#)

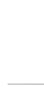
I've been looking into ways to handle the User/Admin/Programmer vs Integrated dichotomy a little better. I think the intergrated document as we know it needs to go. There's too much weirdness in the order (e.g., the tutorial coming after the FE/BE protocol, release notes somewhere in the middle, etc.) that is created by just pasting together the "guides". Also, the chapters are numbered differently, which makes referring to them by number impossible.

Looking Stackoverflow



Improvement / more summaries

- Maybe some « shortcuts » pages are missing
- A TL;DR ?



Example : Excerpts of MySQL



Excerpts from this Manual

[MySQL Backup and Recovery](#)
[MySQL Globalization](#)
[MySQL Information Schema](#)
[MySQL Installation Guide](#)
[Security in MySQL](#)
[Starting and Stopping MySQL](#)
[MySQL and Linux/Unix](#)
[MySQL and Windows](#)
[MySQL and macOS](#)
[MySQL and Solaris](#)
[Building MySQL from Source](#)
[MySQL Restrictions and Limitations](#)
[MySQL Partitioning](#)
[MySQL Tutorial](#)
[MySQL Performance Schema](#)
[MySQL Replication](#)
[Using the MySQL Yum Repository](#)
[MySQL NDB Cluster 8.0](#)

- Short content
- Links to the doc
- PDF available

Improvement/ Clarity

- Where am I?
- What is this feature?
 - repeat the desc of the section somewhere?
- Too much clutter ?



A doc about the doc?

Preface

Table of Contents


- 1. What Is PostgreSQL?
- 2. A Brief History of PostgreSQL
 - 2.1. The Berkeley POSTGRES Project
 - 2.2. Postgres95
 - 2.3. PostgreSQL
- 3. Conventions
- 4. Further Information
- 5. Bug Reporting Guidelines
 - 5.1. Identifying Bugs
 - 5.2. What to Report
 - 5.3. Where to Report Bugs

This book is the official documentation of PostgreSQL. It has been written by the PostgreSQL developers and other volunteers in parallel to the development of the PostgreSQL software. It describes all the functionality that the current version of PostgreSQL officially supports.

To make the large amount of information about PostgreSQL manageable, this book has been organized in several parts. Each part is targeted at a different class of users, or at users in different stages of their PostgreSQL experience:

- **Part I** is an informal introduction for new users.
 - **Part II** documents the SQL query language environment, including data types and functions, as well as user-level performance tuning. Every PostgreSQL user should read this.
 - **Part III** describes the installation and administration of the server. Everyone who runs a PostgreSQL server, be it for private use or for others, should read this part.
 - **Part IV** describes the programming interfaces for PostgreSQL client programs.
 - **Part V** contains information for advanced users about the extensibility capabilities of the server. Topics include user-defined data types and functions.
 - **Part VI** contains reference information about SQL commands, client and server programs. This part supports the other parts with structured information sorted by command or program.
 - **Part VII** contains assorted information that might be of use to PostgreSQL developers.
-

You are not alone

 Microsoft | [Learn](#) [Documentation](#) [Training](#) [Certifications](#) [Q&A](#) [Code Samples](#) [Shows](#) [Events](#)

[Sign in](#)

[SQL Docs](#) [Overview](#) [Install](#) [Secure](#) [Develop](#) [Administer](#) [Analyze](#) [Reference](#) [Download SQL Server](#)

Version

SQL Server 2022 Preview

Filter by title

SQL Server

Docs navigation tips

Previous versions 2005-2014

> Overview

> Business continuity

> Database design

> Development

> Internals & Architecture

> Installation

> Migrate & load data

> Manage, monitor, & tune

> Query data

> Reporting & Analytics

> Security

> Tools

> Tutorials

Download PDF

Learn / SQL /

SQL Server docs navigation guide


Article • 08/11/2022 • 2 minutes to read • 5 contributors [Feedback](#)

This topic provides some tips and tricks for navigating the SQL Server technical documentation space.

Hub page

The SQL Server hub page can be found at <https://aka.ms/sqldocs> and is the entry point for finding relevant SQL Server content.

You can always navigate back to this page by selecting **SQL Docs** from the header at the top of every page within the SQL Server technical documentation set:



Offline documentation

If you would like to view the SQL Server documentation on an offline system, you have two options to do so. You can either create a PDF wherever you are in the SQL Server technical documentation, or you can download the offline content using [SQL Server offline Help Viewer](#).

In this article

[Hub page](#)

[Offline documentation](#)

[TOC symbols](#)

[TOC search](#)

Show more

An architecture I saw before...

Version

SQL Server 2019



Filter by title

Azure Data Studio documentation

Download Azure Data Studio

Release notes

> Overview

> Quickstarts

> Tutorials

> Concepts

> How-to guides

> References

> Resources

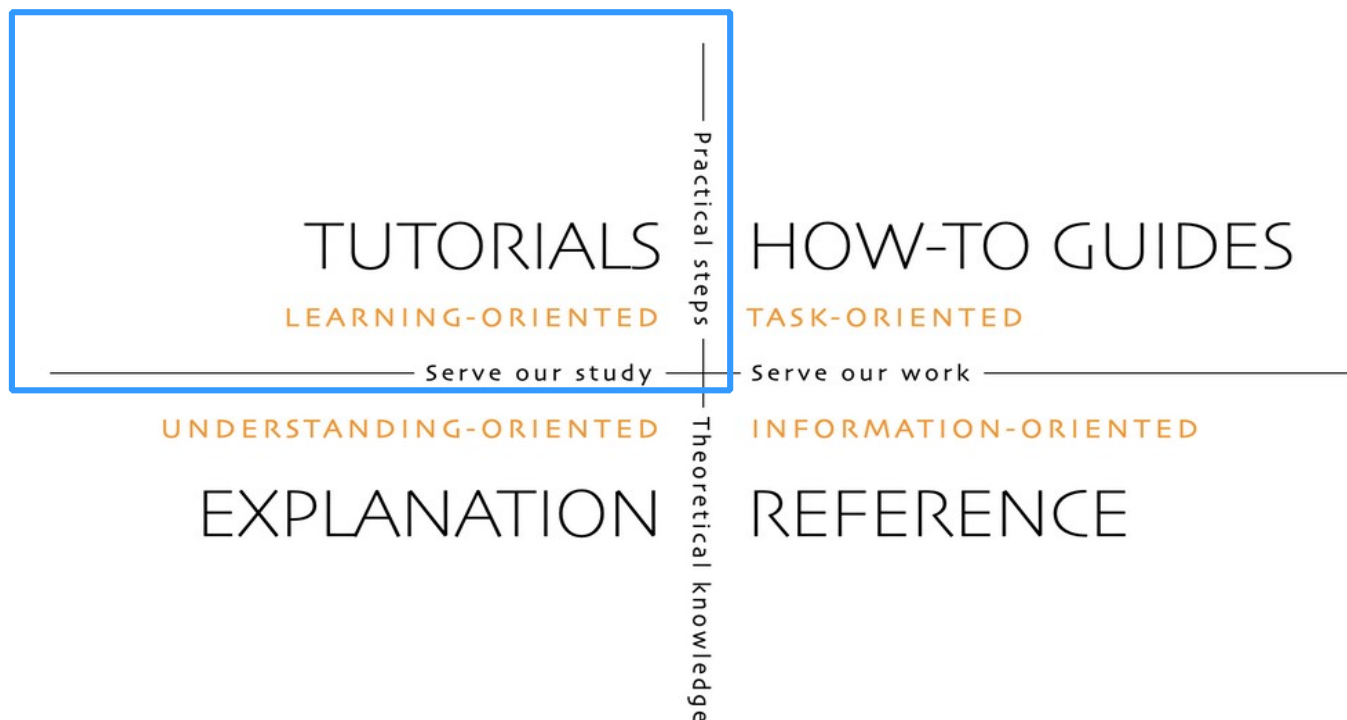
Diataxis

www.diataxis.fr



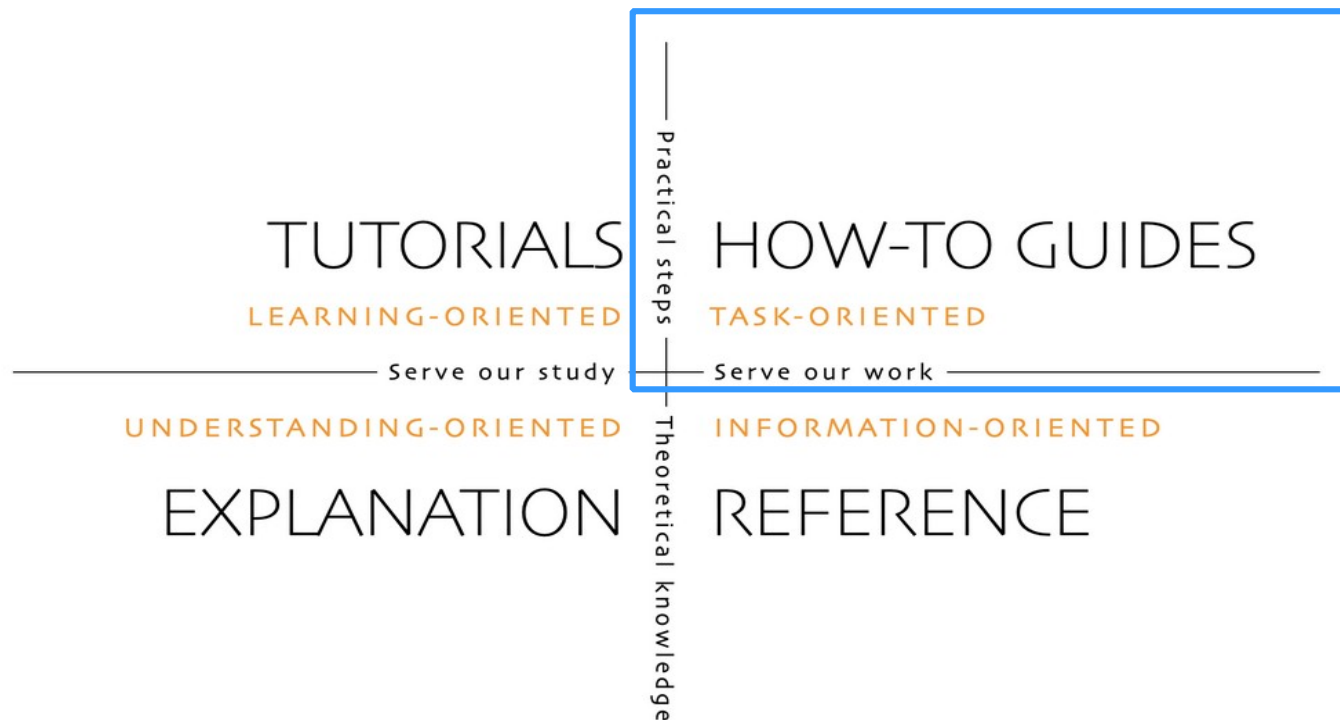
Diataxis

www.diataxis.fr



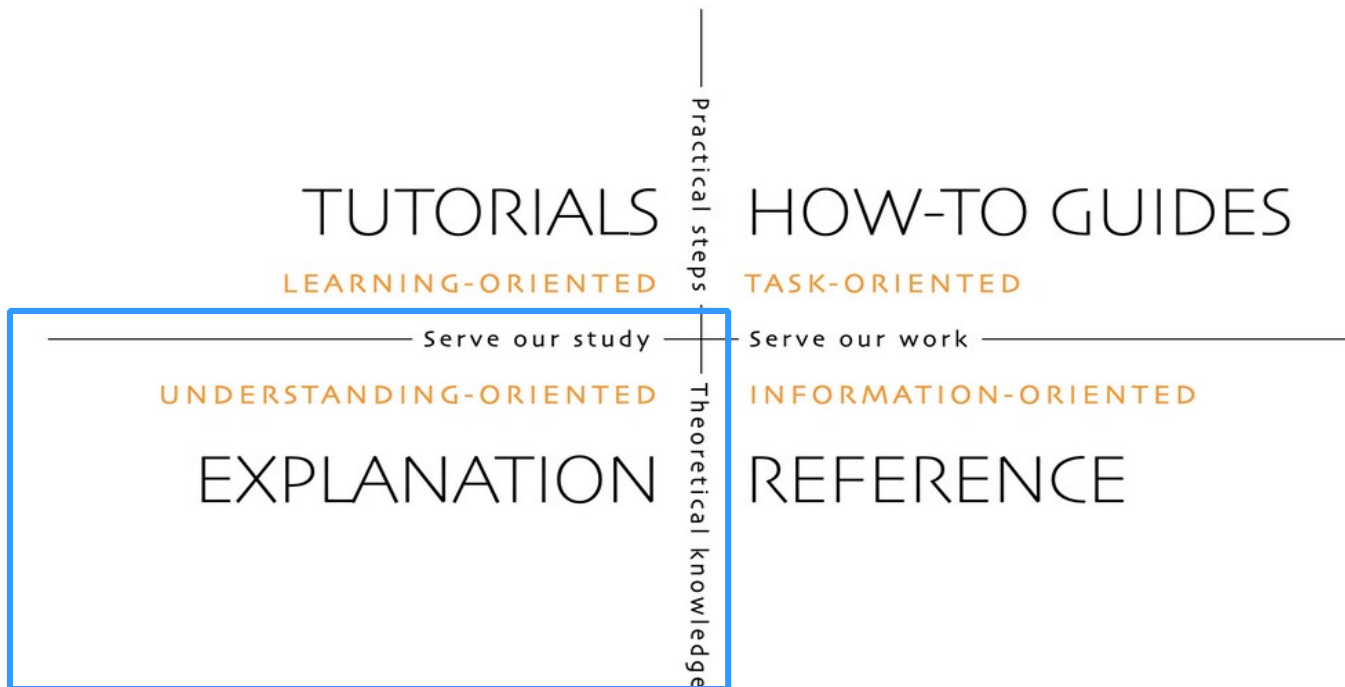
Diataxis

www.diataxis.fr



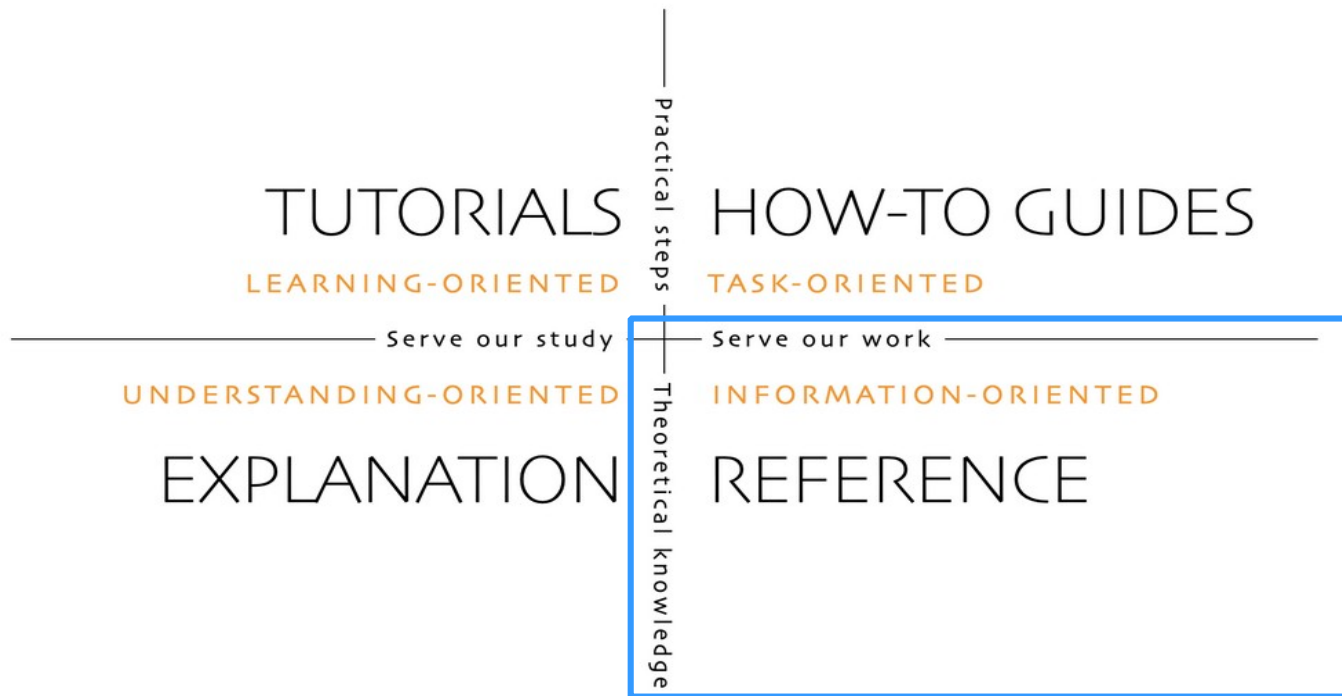
Diataxis

www.diataxis.fr



Diataxis

www.diataxis.fr



Part 2 The content and UX

UX

- User Experience
- Developer Experience even!

Tone and jokes

- The content is clear and very readable.
- Jokes are well done and non-intrusive.



Twitter

- http

The screenshot shows a web browser window with the PostgreSQL website. The browser's address bar displays the URL `https://www.postgresql.org/search/?q=postgresqlco.nf`. The page features a dark blue header with the PostgreSQL logo and a notification banner that reads "13th October 2022: PostgreSQL 15 Released!". Below the header, the "Site Search" section is visible, with the search term "postgresqlco.nf" entered in the search bar. The search results indicate that no hits were found for this term. At the bottom of the page, there are social media links for Twitter and YouTube, and a footer containing links to "Policies", "Code of Conduct", "About PostgreSQL", and "Contact", along with the copyright notice "Copyright © 1996-2022 The PostgreSQL Global Development Group".

PostgreSQL: Search results

13th October 2022: PostgreSQL 15 Released!

Site Search

postgresqlco.nf

Your search for **postgresqlco.nf** returned no hits.

[Policies](#) | [Code of Conduct](#) | [About PostgreSQL](#) | [Contact](#)

Copyright © 1996-2022 The PostgreSQL Global Development Group



MANAGE

YOUR postgresql.conf CONFIGURATIONS

- Upload your **postgresql.conf** files and manage them from your web browser.
- Edit, delete, validate parameters.
- Name and store different configurations.
- Download them in several formats. Or [via an API](#).
- Share publicly the configurations you want.
- **This is a free service, forever!**

[SIGN UP / SIGN IN](#)[TEST IT! UPLOAD FILE](#)

NAME	VALUE	COMMENT
autovacuum_work_mem	-1	Sets the maximum memory to be used by each autovacuum worker process.
checkpoint_completion_target	0.9	Time spent flushing dirty buffers during checkpoint, as fraction of total checkpoint time.
checkpoint_warning	30s	Enables warnings if checkpoint segments are filled more frequently than this.
default_statistics_target	100	Sets the default statistics target.
effective_cache_size	4GB	Sets the planner's assumption about the total size of the data in the database.

DOCUMENTATION

BROWSE PARAMETERS DOCUMENTATION

- All the documentation and help you need about all the **postgresql.conf** parameters.
- Covering the nine most recent versions of Postgres
- Available in 5 languages: [English](#), [Japanese](#), [Russian](#), [Chinese](#) and [French](#).
- Check other user's recommendations, or share yours.

[BROWSE ALL PARAMETERS](#)

shared_buffers

Type: `string`

Default: `384K (25%)`

Min: `32 (256KB)`

Max: `1073741824 (256MB)`

Unit: `B`

Context: `shared_buffers`

Restart: `no`

DESCRIPTION

Sets the number of shared memory buffers used by the server.

Sets the amount of memory the database server uses for shared memory buffers. The default is typically 128 megabytes (25%), but might be less if your kernel settings will not support it (as determined during install). This setting must be at least 128 kilobytes. However, settings significantly higher than the minimum are usually needed for good performance. If this value is specified without units, it is taken as blocks, that is 8192 bytes, typically 8KB. (Non-default values of 8192 change the minimum value.) This parameter can only be set at server start.

If you have a dedicated database server with 1GB or more of RAM, a reasonable starting value for `shared_buffers` is 25% of the memory in your system. There are some workloads where even larger settings for `shared_buffers` are effective, but because PostgreSQL also relies on the operating system cache, it is unlikely that an allocation of more than 40% of RAM to `shared_buffers` will work better than a smaller amount. Lower settings for `shared_buffers` usually require a corresponding increase in

Code examples : SQL

- Clear
- Useful

The `point` type requires a coordinate pair as input, as shown here:

```
INSERT INTO cities VALUES ('San Francisco', '(-194.0, 53.0)');
```

The syntax used so far requires you to remember the order of the columns. An alternative syntax allows you to list the columns explicitly:

```
INSERT INTO weather (city, temp_lo, temp_hi, prcp, date)
VALUES ('San Francisco', 43, 57, 0.0, '1994-11-29');
```

You can list the columns in a different order if you wish or even omit some columns, e.g., if the precipitation is unknown:

```
INSERT INTO weather (date, city, temp_hi, temp_lo)
VALUES ('1994-11-29', 'Hayward', 54, 37);
```

<https://www.postgresql.org/docs/13/tutorial-populate.html>

Improvement / Code examples

- Highlighting in HTML

 [Documentation Home](#)

MySQL 8.0 Reference Manual

- [Preface and Legal Notices](#)
- › [General Information](#)
- › [Installing and Upgrading MySQL](#)
- › [Tutorial](#)
- › [MySQL Programs](#)
- › [MySQL Server Administration](#)
- › [Security](#)
- › [Backup and Recovery](#)
- › [Optimization](#)
- › [Language Structure](#)
- › [Character Sets, Collations, Unicode](#)
- › [Data Types](#)
- › [Functions and Operators](#)
- ▼ [SQL Statements](#)
 - › [Data Definition Statements](#)
 - ▼ [Data Manipulation Statements](#)
 - [CALL Statement](#)
 - [DELETE Statement](#)
 - [DO Statement](#)

`INSERT` statements using `VALUES ROW()` syntax can also insert multiple rows. In this case, each value list must be contained within a `ROW()` (row constructor), like this:

```
INSERT INTO tbl_name (a,b,c)
VALUES ROW(1,2,3), ROW(4,5,6), ROW(7,8,9);
```

The affected-rows value for an `INSERT` can be obtained using the `ROW_COUNT()` SQL function or the `mysql_affected_rows()` C API function. See [Section 12.16, “Information Functions”](#), and [mysql_affected_rows\(\)](#).

If you use `INSERT ... VALUES` or `INSERT ... VALUES ROW()` with multiple value lists, or `INSERT ... SELECT` or `INSERT ... TABLE`, the statement returns an information string in this format:

```
Records: N1 Duplicates: N2 Warnings: N3
```

If you are using the C API, the information string can be obtained by invoking the `mysql_info()` function. See [mysql_info\(\)](#).

`Records` indicates the number of rows processed by the statement. (This is not necessarily the number of rows actually inserted because `Duplicates` can be nonzero.) `Duplicates` indicates the number of rows that could not be inserted because they would duplicate some existing unique index value. `Warnings` indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting `NULL` into a column that has been declared `NOT NULL`. For multiple-row `INSERT` statements or `INSERT INTO ... SELECT` statements, the column is set to the implicit default value for the column data type. This is 0 for numeric types, the empty string (‘’) for

Callouts are existing

Tip

When you create many interrelated tables it is wise to choose a consistent naming pattern for the tables and columns. For instance, there is a choice of using singular or plural nouns for table names, both of which are favored by some theorist or other.

<https://www.postgresql.org/docs/15/ddl-basics.html>

Note

`<>` is the standard SQL notation for “not equal”. `!=` is an alias, which is converted to `<>` at a very early stage of parsing. Hence, it is not possible to implement `!=` and `<>` operators that do different things.

<https://www.postgresql.org/docs/15/functions-comparison.html>

Callouts are planned in CSS

CSS

/* Put these here instead of inside
the HTML (see unsetting of
admon.style in XSL) so that the
web site stylesheet can set its own
style. */

```
.tip,  
.note,  
.important,  
.caution,  
.warning {  
    margin-left: 0.5in;  
    margin-right:  
0.5in;  
}
```

Callout in the version offline

13.4.1. Enforcing Consistency with Serializable Transactions

If the Serializable transaction isolation level is used for all writes and for all reads which need a consistent view of the data, no other effort is required to ensure consistency. Software from other environments which is written to use serializable transactions to ensure consistency should “just work” in this regard in PostgreSQL.

When using this technique, it will avoid creating an unnecessary burden for application programmers if the application software goes through a framework which automatically retries transactions which are rolled back with a serialization failure. It may be a good idea to set `default_transaction_isolation` to `serializable`. It would also be wise to take some action to ensure that no other transaction isolation level is used, either inadvertently or to subvert integrity checks, through checks of the transaction isolation level in triggers.

See [Section 13.2.3](#) for performance suggestions.

Warning


This level of integrity protection using Serializable transactions does not yet extend to hot standby mode ([Section 27.4](#)). Because of that, those using hot standby may want to use Repeatable Read and explicit locking on the primary.

13.4.2. Enforcing Consistency with Explicit Blocking Locks

When non-serializable writes are possible, to ensure the current validity of a row and protect it against concurrent updates one must use `SELECT FOR UPDATE`, `SELECT FOR SHARE`, or an appropriate `LOCK TABLE` statement. (`SELECT FOR UPDATE` and `SELECT FOR SHARE` lock just the returned rows against concurrent updates, while `LOCK TABLE` locks the whole table.) This should be taken into account when porting applications to PostgreSQL from other environments.

Improvement / more callouts

MongoDB

 MongoDB

ProductsSolutionsResourcesCompanyPricing

MongoDB Documentation

[← Back To Develop Applications](#)

MongoDB Manual

4.4

[Introduction](#)

[Installation](#)

[The mongo Shell](#)

[MongoDB CRUD Operations](#)

[Aggregation](#)

[Data Models](#)

▼ **Transactions**

[Drivers API](#)

[Production Considerations](#)

[Production Considerations \(Sharded Clusters\)](#)

[Transactions and Operations](#)

[Indexes](#)

[Security](#)

[Change Streams](#)

[Replication](#)

[Sharding](#)

[Administration](#)

[Storage](#)

[Frequently Asked Questions](#)

[Reference](#)

[Release Notes](#)

[Technical Support](#)

IMPORTANT

In most cases, multi-document transaction incurs a greater performance cost over single document writes, and the availability of multi-document transactions should not be a replacement for effective schema design. For many scenarios, the [denormalized data model \(embedded documents and arrays\)](#) will continue to be optimal for your data and use cases. That is, for many scenarios, modeling your data appropriately will minimize the need for multi-document transactions.

For additional transactions usage considerations (such as runtime limit and oplog size limit), see also [Production Considerations](#).

TIP

See also:

[Outside Reads During Commit](#)

Transactions and Operations

Distributed transactions can be used across multiple operations, collections, databases, documents, and, starting in MongoDB 4.2, shards.

For transactions:

- You can specify read/write (CRUD) operations on **existing** collections. For a list of CRUD operations, see [CRUD Operations](#).
- When using [feature compatibility version \(fcv\)](#) "4.4" or greater, you can create collections and indexes in transactions. For details, see [Create Collections and Indexes In a Transaction](#)
- The collections used in a transaction can be in different databases.

NOTE

You cannot create new collections in cross-shard write transactions. For example, if you write to an existing collection in one shard and implicitly create a collection in a different shard, MongoDB cannot perform both operations in the same transaction.

Select your language

C C

On this page

[Transactions API](#)

[Transactions and Atomicity](#)

[Transactions and Operations](#)

[Transactions and Sessions](#)

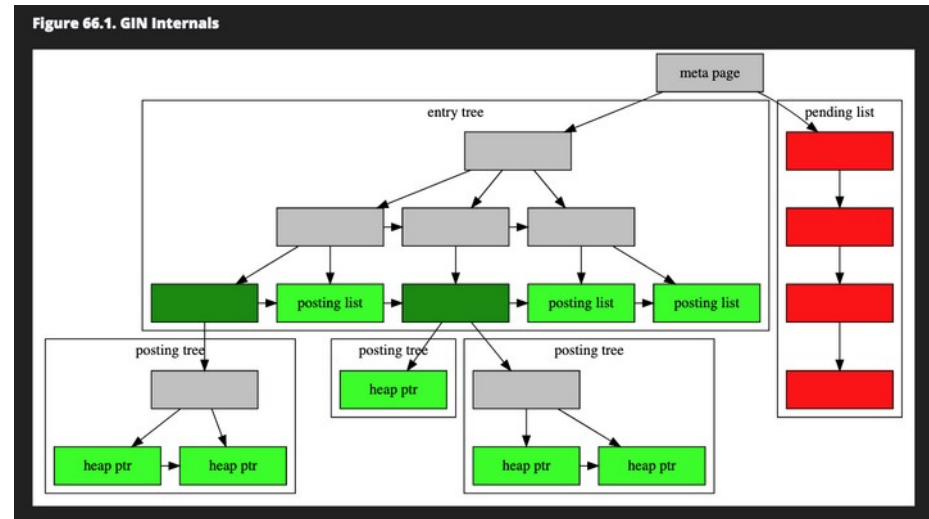
[Read Concern/Write Concern/Read Preference](#)

[General Information](#)

[Additional Transactions Topics](#)

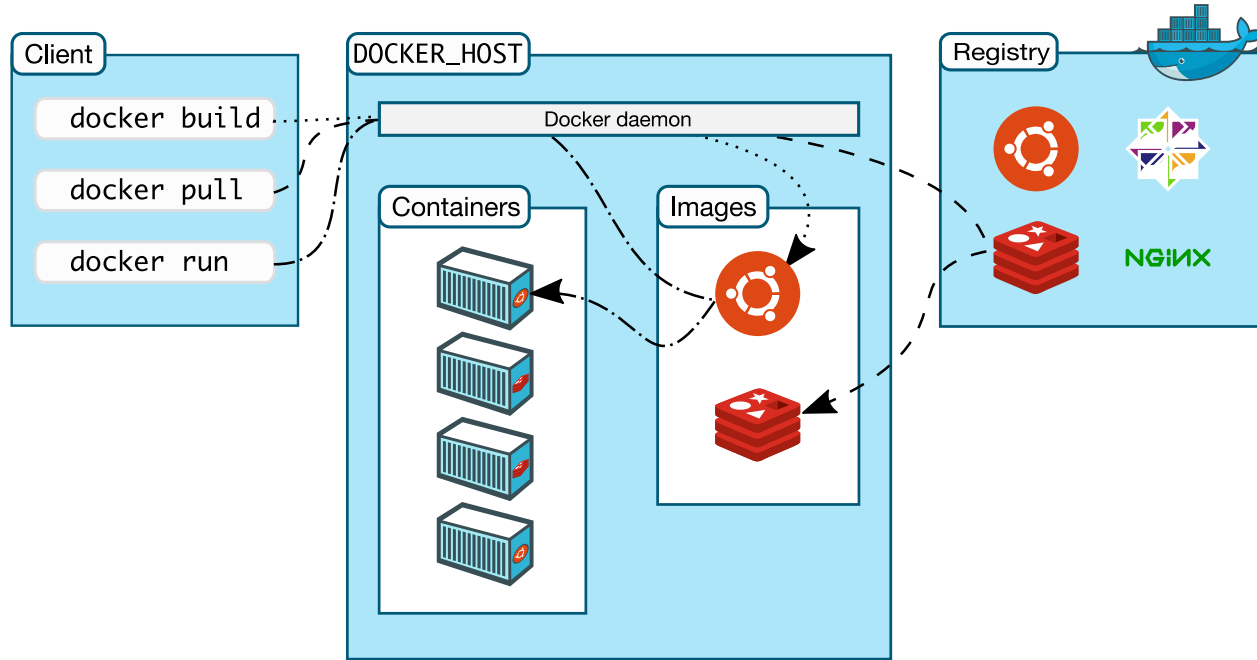
Diagrams, wherever they are

- Doc is lightweight but...
- Is it only GIN internals that needs a diagram?



- Color blindness ?

In another docs : diagrams



In another doc : icons

- Heroku

The screenshot shows the Heroku Dev Center interface. The top navigation bar is dark purple with the Heroku logo, 'Heroku Dev Center', and links for 'Get Started', 'Documentation', 'Changelog', and 'More'. A left sidebar lists various categories, with 'Patterns & Best Practices' highlighted in blue. The main content area is titled 'Application Load Testing' and includes a table of contents with eight items: 'Set up a load testing environment', 'Add seed data', 'Install logging and monitoring add-ons', 'Select a load testing tool', 'Configure your load testing tool', 'Run your load test', and 'Use results to improve app performance'. The page also shows a breadcrumb trail 'Patterns & Best Practices / Application Load Testing' and a language selector set to 'English'.

Heroku Dev Center ⚡ Get Started 📖 Documentation 🕒 Changelog More ▾

CATEGORIES

- Heroku Architecture ▶
- Command Line
- Deployment ▶
- Continuous Delivery ▶
- Language Support ▶
- Databases & Data Management ▶
- Monitoring & Metrics ▶
- App Performance
- Add-ons ▶
- Collaboration
- Security ▶
- Heroku Enterprise ▶
- Patterns & Best Practices**
- Extending Heroku ▶
- Accounts & Billing

Patterns & Best Practices / Application Load Testing

Application Load Testing

🕒 Last updated May 19, 2021

📖 Table of Contents

- Set up a load testing environment
- Add seed data
- Install logging and monitoring add-ons
- Select a load testing tool
- Configure your load testing tool
- Run your load test
- Use results to improve app performance

Load tests allow you to see how your application performs under real-world traffic. testing to test a new feature at scale before it launches, or to prepare your app for t

Improvement / Icons

- Guidance
- UX



Diagrams and images

[projects](#) / [postgresql.git](#) / blob

[summary](#) | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#)
[blame](#) | [history](#) | [raw](#) | [HEAD](#)

Allow nodeSort to perform Datum sorts for byref types

[\[postgresql.git\]](#) / [doc](#) / [src](#) / [sgml](#) / [images](#) / [README](#)

```
1 Images
2 =====
3
4 This directory contains images for use in the documentation.
5
6 Creating an image
7 -----
8
9 A variety of tools can be used to create an image. The appropriate
10 choice depends on the nature of the image. We prefer workflows that
11 involve diffable source files.
12
13 These tools are acceptable:
14
15 - Graphviz (https://graphviz.org/)
16 - Ditaa (http://ditaa.sourceforge.net/)
17
```

Already discussed...in 2018

Re: Images in the official documentation

From: Pavel Golub <pavel(at)microolap(dot)com>
To: Jürgen Purtz <juergen(at)purtz(dot)de>, pgsql-docs(at)lists(dot)postgresql(dot)org
Subject: Re: Images in the official documentation
Date: 2018-07-19 12:06:08
Message-ID: [1554502154.20180719150608@gf.microolap.com](#)
Views: [Raw Message](#) | [Whole Thread](#) | [Download mbox](#) | [Resend email](#)
Thread: 2018-07-19 12:06:08 from Pavel Golub <pavel(at)microolap(dot)com> 
Lists: [pgsql-docs](#)

Hello, Jürgen.

You wrote:

```
JP> Our discussion about grafics in the documentation reached to the
JP> conclusion that we shall use SVG, the importance to 'diff-ability'
JP> is rated differently, and there is no consensus about tools.
```

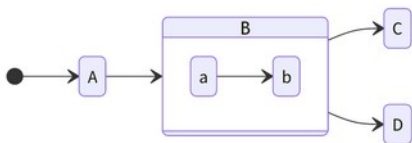
MermaidJS?

- <https://mermaid-js.github.io>



With state diagrams you can use the direction statement to set the direction which the diagram will render like in this example.

```
stateDiagram
    direction LR
    [*] --> A
    A --> B
    B --> C
    state B {
        direction LR
        a --> b
    }
    B --> D
```



classDiagram

direction RL

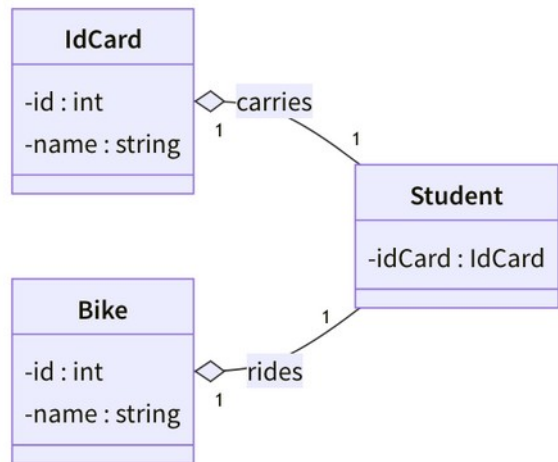
```
class Student {
    -idCard : IdCard
}
```

```
class IdCard{
    -id : int
    -name : string
}
```

```
class Bike{
    -id : int
    -name : string
}
```

Student "1" --o "1" IdCard : carries

Student "1" --o "1" Bike : rides

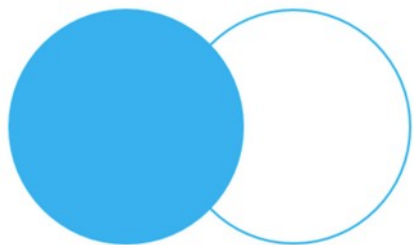


Joining the movement?

- Why is this tutorial illustrated?
 - <https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-joins/>
- And not the official documentation?
 - <https://www.postgresql.org/docs/15/queries-table-expressions.html>



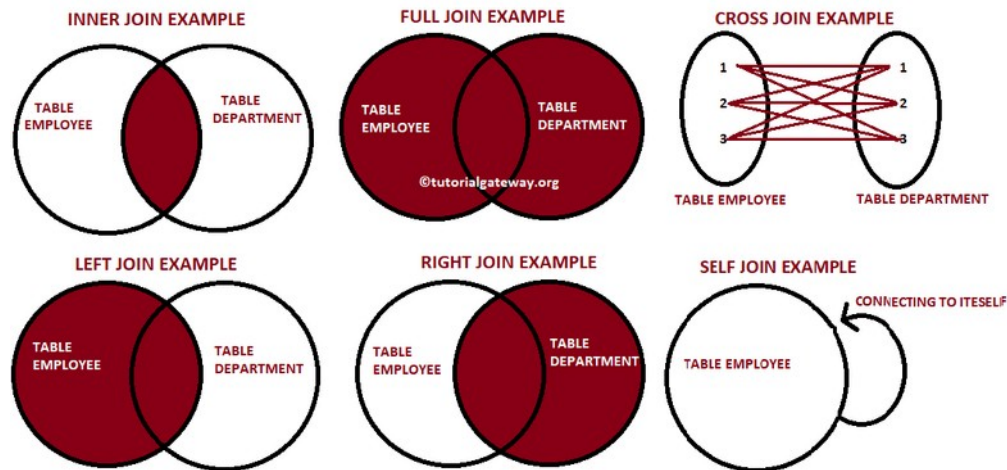
Tutorial: with images



LEFT OUTER JOIN

To select rows from the left table that do not have matching rows in the right table, you use the left join with a `WHERE` clause. For example:

```
SELECT
  a,
  fruit_a,
  b,
  fruit_b
FROM
  basket_a
LEFT JOIN basket_b
  ON fruit_a = fruit_b
WHERE b IS NULL;
```



Source: <https://theartofpostgresql.com/blog/2019-09-sql-joins/>

Official doc: text only

Qualified joins

```
T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2 ON boolean_expression  
T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2 USING ( join column list )  
T1 NATURAL { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2
```

The words **INNER** and **OUTER** are optional in all forms. **INNER** is the default; **LEFT**, **RIGHT**, and **FULL** imply an outer join.

The *join condition* is specified in the **ON** or **USING** clause, or implicitly by the word **NATURAL**. The join condition determines which rows from the two source tables are considered to “match”, as explained in detail below.

The possible types of qualified join are:

INNER JOIN

For each row **R1** of **T1**, the joined table has a row for each row in **T2** that satisfies the join condition with **R1**.

LEFT OUTER JOIN

First, an inner join is performed. Then, for each row in **T1** that does not satisfy the join condition with any row in **T2**, a joined row is added with null values in columns of **T2**. Thus, the joined table always has at least one row for each row in **T1**.

RIGHT OUTER JOIN

First, an inner join is performed. Then, for each row in **T2** that does not satisfy the join condition with any row in **T1**, a joined row is added with null values in columns of **T1**. This is the converse of a left join: the result table will always have a row for each row in **T2**.

FULL OUTER JOIN

First, an inner join is performed. Then, for each row in **T1** that does not satisfy the join condition with any row in **T2**, a joined row is added with null values in columns of **T2**. Also, for each row of **T2** that does not satisfy the join condition with any row in **T1**, a joined row with null values in the columns of **T1** is added.

7.2.3. The GROUP BY and HAVING Clauses

After passing the WHERE filter, the derived input table might be subject to grouping, using the GROUP BY clause, and elimination of group rows using the HAVING clause.

```
SELECT select_list
  FROM ...
  [WHERE ...]
  GROUP BY grouping_column_reference [, grouping_column_reference]...
```

The GROUP BY clause is used to group together those rows in a table that have the same values in all the columns listed. The order in which the columns are listed does not matter. The effect is to combine each set of rows having common values into one group row that represents all rows in the group. This is done to eliminate redundancy in the output and/or compute aggregates that apply to these groups. For instance:

```
=> SELECT * FROM test1;
```

x	y
a	3
c	2
b	5
a	1

(4 rows)

```
=> SELECT x FROM test1 GROUP BY x;
```

x
a
b
c

(3 rows)

In the second query, we could not have written SELECT * FROM test1 GROUP BY x, because there is no single value for the column y that could be associated with each group. The grouped-by columns can be referenced in the select list since they have a single value in each group.

The `GROUP BY` clause specifies which grouping columns should be used to perform any aggregations in the `SELECT` clause. If the `GROUP BY` clause is specified, the query is always an aggregate query, even if no aggregations are present in the `SELECT` clause.

The content is quite good

- The formatting is old fashioned.



Crédit : Chris huh, Public domain

Walls of texts, plural

- 1/ Trigger Overview
<https://www.postgresql.org/docs/15/trigger-definition.html>
- 2/ GIN extensibility
<https://www.postgresql.org/docs/13/gin-extensibility.html>
- 3/ TOAST Database Physical Storage
<https://www.postgresql.org/docs/13/storage-toast.html#STORAGE-TOAST-ONDISK>
-



Breakin the wall

- Why is this page more clear than the official doc ?

https://www.tutorialspoint.com/postgresql/postgresql_triggers.htm



- PostgreSQL - Data Types
- PostgreSQL - Create Database
- PostgreSQL - Select Database
- PostgreSQL - Drop Database
- PostgreSQL - Create Table
- PostgreSQL - Drop Table
- PostgreSQL - Schema
- PostgreSQL - Insert Query
- PostgreSQL - Select Query
- PostgreSQL - Operators
- PostgreSQL - Expressions
- PostgreSQL - Where Clause
- PostgreSQL - AND & OR Clauses
- PostgreSQL - Update Query
- PostgreSQL - Delete Query
- PostgreSQL - Like Clause
- PostgreSQL - Limit Clause
- PostgreSQL - Order By Clause
- PostgreSQL - Group By
- PostgreSQL - With Clause
- PostgreSQL - Having Clause
- PostgreSQL - Distinct Keyword

Advanced PostgreSQL

PostgreSQL **Triggers** are database callback functions, which are automatically performed/invoked when a specified database event occurs.

The following are important points about PostgreSQL triggers –

- PostgreSQL trigger can be specified to fire
 - Before the operation is attempted on a row (before constraints are checked and the INSERT, UPDATE or DELETE is attempted)
 - After the operation has completed (after constraints are checked and the INSERT, UPDATE, or DELETE has completed)
 - Instead of the operation (in the case of inserts, updates or deletes on a view)
- A trigger that is marked FOR EACH ROW is called once for every row that the operation modifies. In contrast, a trigger that is marked FOR EACH STATEMENT only executes once for any given operation, regardless of how many rows it modifies.
- Both, the WHEN clause and the trigger actions, may access elements of the row being inserted, deleted or updated using references of the form **NEW.column-name** and **OLD.column-name**, where column-name is the name of a column from the table that the trigger is associated with.
- If a WHEN clause is supplied, the PostgreSQL statements specified are only executed for rows for which the WHEN clause is true. If no WHEN clause is supplied, the PostgreSQL statements are executed for all rows.
- If multiple triggers of the same kind are defined for the same event, they will be fired in alphabetical order by name.
- The BEFORE, AFTER or INSTEAD OF keyword determines when the trigger actions will be executed relative to the insertion, modification or removal of the associated row.
- Triggers are automatically dropped when the table that they are associated with is dropped.
- The table to be modified must exist in the same database as the table or view to which the trigger is attached and one must use just **tablename**, not **database.tablename**.
- A CONSTRAINT option when specified creates a *constraint trigger*. This is the same as a regular trigger except that the timing of the trigger firing can be adjusted using SET CONSTRAINTS. Constraint triggers are expected to raise an exception when the constraints they implement are violated.

Improvement/ Be bold

- Sometimes, being bold is **helpful** for the reader
- **Scanning** the content
- Looking for **information**



Doc is useful / seen on Twitter



Richard Michael

@richardkmichael



It is difficult to overstate what a great experience interacting with the [#postgres](#) OSS community is.. mailing list is friendly & helpful, source code is beautiful & understandable, documentation is thorough & useful. Very motivating.

6:13 PM · Sep 9, 2021 · Twitter Web App

A good doc

- **Gain** of time
- Really **helpful**
- **Communicates** your choices
- **Learn** your best practices
- However: **written by experts for experts**

Conclusion

- 25+ years old and it shows
- Improvements
 - Orientation page by persona
 - Renaming sections (BC break)
 - UX: more diagrams
 - Welcoming doc contributors



Merci beaucoup !

- Thanks !
 - Thanks to Lætitia Avrot and the Postgres Conf team
 - Images, except screenshots, are from [Wikimedia Commons](#)



Questions?

- Tweet me [@sarahhaim](#)

