

Full Stack Python

[All topics](#) | [Blog](#) | [Supporter's Edition](#) | [@fullstackpython](#) | [Facebook](#) | [What's new?](#)

PostgreSQL

PostgreSQL, often written as "Postgres" and pronounced "Poss-gres", is an open source relational database implementation frequently used by Python applications as a backend for data storage and retrieval.



How does PostgreSQL fit within the Python stack?

PostgreSQL is the default database choice for many Python developers, including the Django team when testing the [Django ORM](#). PostgreSQL is often viewed as more feature robust and stable when compared to MySQL, SQLServer and Oracle. All of those databases are reasonable choices. However, because PostgreSQL tends to be used by Python developers the drivers and example code for using the database tend to be better documented and contain fewer bugs for typical usage scenarios. If you try to use an Oracle database with Django, you'll see there is far less example code for that setup compared to PostgreSQL backend setups.

PostgreSQL is an implementation of the relational database concept. Learn more in the data chapter or view the table of contents for all topics.

Why is PostgreSQL a good database choice?





PostgreSQL's open source license allows developers to operate one or more databases without licensing cost in their applications. The open source license operating model is much less expensive compared to Oracle or other proprietary databases, especially as replication and sharding become necessary at large scale. In addition, because so many people ranging from independent developers to multinational organizations use PostgreSQL, it's often easier to find developers with PostgreSQL experience than other relational databases. There is also anecdotal evidence that PostgreSQL fixes bugs faster than MySQL, although to be fair there has not been a comprehensive study comparing how the two projects handle defect resolution.

The PostgreSQL core team also releases frequent updates that greatly enhance the database's capabilities. For example, in the PostgreSQL 9.4 release the jsonb type was added to enhance JavaScript Object Notation (JSON) storage capabilities so that in many cases a separate NoSQL database is not required in an application's architecture.

Connecting to PostgreSQL with Python

To work with relational databases in Python you need to use a database driver, which is also referred to as a database connector. The most common driver library for working with PostgreSQL is psycopg2. There is a list of all drivers on the PostgreSQL wiki, including several libraries that are no longer maintained. If you're working with the asyncio Python stdlib module you should also take a look at the aiopg library which wraps psycopg2's asynchronous features together.

To abstract the connection between tables and objects, many Python developers use an object-relational mapper (ORM) with to turn relational data from PostgreSQL into objects that can be used in their Python application. For example, while PostgreSQL provides a relational database and psycopg is the common database connector, there are many ORMs that can be used with varying web frameworks, as shown in the table below.

web framework	Bottle	Flask	Flask	Django
ORM	Peewee	Pony ORM	SQLAlchemy	Django ORM
database connector	psycopg	psycopg	psycopg	psycopg
relational database	 PostgreSQL	 PostgreSQL	 PostgreSQL	 PostgreSQL

Learn more about [Python ORMs on that dedicated topic page.](#)

PostgreSQL data safety

If you're on Linux it's easy to get PostgreSQL installed using a package manager. However, once the database is installed and running your responsibility is just beginning. Before you go live with a production application, make sure to:

1. Lock down access with [a whitelist](#) in the `pg_hba.conf` file
2. Enable [replication](#) to another database that's preferably on different infrastructure in a separate location
3. Perform regular [backups](#) and [test the restoration process](#)
4. Ensure your application prevents [SQL injection attacks](#)

When possible have someone qualified do a [PostgreSQL security audit](#) to identify the biggest risks to your database. Small applications and bootstrapped companies often cannot afford a full audit in the beginning but as an application grows over time it becomes a bigger target.

The data stored in your database is the lifeblood of your application. If you have ever [accidentally dropped a production database](#) or been the victim of malicious activity such as SQL injection attacks, you'll know it's far easier to recover when a bit of work has been performed beforehand on backups, replication and security measures.

Python-specific PostgreSQL resources

Many quickstarts and tutorials exist specifically for Django, Flask and other web application frameworks. The ones below are some of the best walkthroughs I've read.

- [Setting up PostgreSQL with Python 3 and psycopg on Ubuntu 16.04](#) provides instructions for getting a fresh Ubuntu install working with PostgreSQL and Python 3.
 - This post on [using PostgreSQL with Django or Flask](#) is a great quickstart guide for either framework.
 - This article explains how and why PostgreSQL can handle [full text searching](#) for many use cases. If you're going down this route, read [this blog post](#) that explains how one developer implemented PostgreSQL full text search with SQLAlchemy.
 - [django-postgres-copy](#) is a tool for bulk loading data into a PostgreSQL database based on Django models. [Say hello to our new open-source software for loading bulk data into PostgreSQL](#) is an introduction to using the tool in your own projects.
 - [How to speed up tests in Django and PostgreSQL](#) explains some hacks for making your schema migration-backed run quicker.
 - [Thinking psycopg3](#) is written by a developer who has worked on this critical Python library for interacting with PostgreSQL since 2005. The author writes up thoughts on what should change if backwards-incompatible changes are ever introduced in a new hypothetical future version.
 - [Records](#) is a wrapper around the psycopg2 driver that allows easy access to direct SQL access. It's worth a look if you prefer writing SQL over using an [ORM](#) like SQLAlchemy.
- o [Postgres Joins and Django Querysets](#) is a well done post with a specific example of how a standard Django ORM query can lead to degraded performance due when obtaining data from many related tables. The `prefetch_related` command and database performance monitoring tools can help analyze and alleviate some of the issues in these unoptimized queries.
- [Loading Google Analytics data to PostgreSQL using Python](#) is a quality tutorial that combines [API calls](#) with psycopg and PostgreSQL to take data from Google Analytics and save it in a PostgreSQL database.
 - [1M rows/s from Postgres to Python](#) shows some benchmarks for the performance of the [asyncpg](#) Python database client and why you may want to consider using it for data transfers.

General PostgreSQL resources

PostgreSQL tutorials not specific to Python are also really helpful for properly handling your data.

- [Why PostgreSQL? \(5 years later\)](#) covers the improvements that have been made to PostgreSQL over the past five years. It's amazing to see how far this project has come and how it continues to evolve.
- [The Internals of PostgreSQL](#) is a book that goes into how PostgreSQL works, including core topics such as [query processing](#), [concurrency control](#) and the [layout of heap table files](#).
- [PostgreSQL Weekly](#) is a weekly newsletter of PostgreSQL content from around the web.
- [My Favorite PostgreSQL Extensions - Part One](#) and [part two](#) are roundups of useful PostgreSQL extensions that augment the standard PostgreSQL functionality.
- [An introduction to PostgreSQL physical storage](#) provides a solid walkthrough of where PostgreSQL files are located on disk, how the files store your data and what mappings are important for the underlying database structure. This post is an easy read and well worth your time.
- Braintree wrote about their experiences [scaling PostgreSQL](#). The post is an inside look at the evolution of Braintree's usage of the database.
- There is no such thing as total security but this IBM article covers [hardening a PostgreSQL database](#).
- [Handling growth with Postgres](#) provides 5 specific tips from Instagram's engineering team on how to scale the design of your PostgreSQL database.
- [Inserting And Using A New Record In Postgres](#) shows some SQL equivalents to what many developers just do in their ORM of choice.
- [Following a Select Statement Through Postgres Internals](#) provides a fascinating look into the internal workings of PostgreSQL during a query.
- [Locating the recovery point just before a dropped table](#) and [logging transactions that dropped tables](#) are two posts that show you how to recover from an accidentally dropped table. In the first post the author shows how recovery is possible with recovery points while the second post shows how to put logging in place to assist in future recoveries.

- [awesome-postgres](#) is a list of code libraries, tutorials and newsletters focused specifically on PostgreSQL.
- While you can use a graphical interface for working with PostgreSQL, it's best to spend some time getting [comfortable with the command-line interface](#).
- Backing up databases is important because data loss can and does happen. This article explains [how to back up a PostgreSQL database hosted on an Amazon Web Services EC2 instance](#) if managing your own database on a cloud server is your preferred setup.
- [Is bi-directional replication \(BDR\) in PostgreSQL transactional?](#) explores a relatively obscure topic with the final result that BDR is similar to data stores with eventual consistency rather than consistency as a requirement.
- [PostgreSQL-metrics](#) is a tool built by Spotify's engineers that extracts and outputs metrics from an existing PostgreSQL database. There's also a way to extend the tools to pull custom metrics as well.
- [Creating a Document-Store Hybrid in Postgres 9.5](#) explains how to store and query JSON data, similar to how [NoSQL data stores](#) operate.
- This [slideshow on high availability for web applications](#) has a good overview of various database setups common in production web applications.
- The [JSONB data type](#) was introduced in PostgreSQL 9.4 to make it easier to store semi-structured data that previously [NoSQL databases](#) such as MongoDB covered. However, there are times when using JSONB isn't a good idea and [this blog post](#) covers [when to avoid the column type](#).

PostgreSQL monitoring and performance

Monitoring one or more PostgreSQL instances and trying to performance tune them is a rare skillset. Here are some resources to get you started if you have to handle these issues in your applications.

- This [guide to PostgreSQL monitoring](#) is handy for knowing what to measure and how to do it.
- Craig Kerstiens wrote a detailed post about [understanding PostgreSQL performance](#).
- The [Practical Guide to PostgreSQL Optimizations](#) covers using cache sizes, restore configurations and shared buffers to improve database performance.

- [This article on performance tuning PostgreSQL](#) shows how to find slow queries, tune indexes and modify your queries to run faster.
- [What PostgreSQL tells you about its performance](#) explains how to gather general performance metrics and provides the exact queries you should run to get them. The article also covers performance monitoring and how to analyze trigger functions.
- [PostgreSQL monitoring queries](#) is a simple GitHub repository of SQL queries that can be run against a PostgreSQL instance to determine usage, caching and bloat.
- [PgSQL Indexes and "LIKE"](#) examines why LIKE queries do not take advantage of PostgreSQL indexes when the locale is set to something other than the default "C", which is for the North American UNIX default. The gist is that you need to build a special index to support LIKE whenever you use a locale other than "C".
- [The PostgreSQL page on PopSQL](#) has a ton of useful syntax snippets categorized by type of action you want to perform using SQL.

Do you want to learn more about data or web apps?

Tell me about standard relational databases.

What're these NoSQL data stores hipster developers keep talking about?

I want to learn how to code a Python web application using a framework.

Sponsored By



Software errors are inevitable. Chaos is not. Try Sentry for free.



The most accurate speech-to-text API. Built for Python developers.



Adobe Creative Cloud for Teams starting at \$33.99 per month.

ADS VIA CARBON

Full Stack Python

Full Stack Python is an open book that explains concepts in plain language and provides helpful resources for those topics.

Updates via Twitter & Facebook.

Chapters

1. Introduction

2. Development Environments

3. Data

» PostgreSQL

4. Web Development

5. Deployment

6. DevOps

Changelog

What Full Stack Means

About the Author

Future Directions

Page Statuses

...or view the full table of contents.

Matt Makai 2012-2021