Full Stack Python

Databases

A database is an abstraction over an operating system's file system that makes it easier for developers to build applications that create, read, update and delete persistent data.

Why are databases necessary?

At a high level web applications store data and present it to users in a useful way. For example, Google stores data about roads and provides directions to get from one location to another by driving through the Maps application. Driving directions are possible because the data is stored in a structured format.

Databases make structured storage reliable and fast. They also give you a mental framework for how the data should be saved and retrieved instead of having to figure out what to do with the data every time you build a new application.

Databases are a concept with many implementations, including PostgreSQL, MySQL and SQLite. Non-relational databases called NoSQL data stores also exist. Learn more in the data chapter or view the table of contents for all topics.

Relational databases

The database storage abstraction most commonly used in Python web development is sets of relational tables. Alternative storage abstractions are explained on the NoSQL page.

Relational databases store data in a series of tables. Interconnections between the tables are specified as *foreign keys*. A foreign key is a unique reference from one row in a

relational table to another row in a table, which can be the same table but is most commonly a different table.

Databases storage implementations vary in complexity. SQLite, a database included with Python, creates a single file for all data per database. Other databases such as PostgreSQL, MySQL, Oracle and Microsoft SQL Server have more complicated persistence schemes while offering additional advanced features that are useful for web application data storage. These advanced features include but are not limited to:

1. data replication between a master database and one or more read-only slave instances
2. advanced column types that can efficiently store semi-structured data such as JavaScript Object Notation (JSON)
3. sharding, which allows horizontal scaling of multiple databases that each serve as read-write instances at the cost of latency in data consistency
4. monitoring, statistics and other useful runtime information for database schemas and tables

Typically web applications start with a single database instance such as PostgreSQL with a straightforward schema. Over time the database schema evolves to a more complex structure using schema migrations and advanced features such as replication, sharding and monitoring become more useful as database utilization increases based on the application users' needs.

Most common databases for Python web apps

PostgreSQL and MySQL are two of the most common open source databases for storing Python web applications' data.

SQLite is a database that is stored in a single file on disk. SQLite is built into Python but is only built for access by a single connection at a time. Therefore is highly recommended to not run a production web application with SQLite.

PostgreSQL database

PostgreSQL is the recommended relational database for working with Python web applications. PostgreSQL's feature set, active development and stability contribute to its usage as the backend for millions of applications live on the Web today.

Learn more about using PostgreSQL with Python on the PostgreSQL page.

MySQL database

MySQL is another viable open source database implementation for Python applications. MySQL has a slightly easier initial learning curve than PostgreSQL but is not as feature rich.

Find out about Python applications with a MySQL backed on the dedicated MySQL page.

Connecting to a database with Python

To work with a relational database using Python, you need to use a code library. The most common libraries for relational databases are:

- psycopg2 (source code) for PostgreSQL.
- MySQLdb (source code) for MySQL. Note that this driver's development is mostly frozen so evaluating alternative drivers is wise if you are using MySQL as a backend.
- cx_Oracle for Oracle Database (source code). Oracle moved their open source driver code from SourceForge to GitHub in 2017.

SQLite support is built into Python 2.7+ and therefore a separate library is not necessary. Simply "import sqlite3" to begin interfacing with the single file-based database.

Object-relational Mapping

Object-relational mappers (ORMs) allow developers to access data from a backend by writing Python code instead of SQL queries. Each web application framework handles integrating ORMs differently. There's an entire page on object-relational mapping (ORMs) that you should read to get a handle on this subject.

Database third-party services

Numerous companies run scalable database servers as a hosted service. Hosted databases can often provide automated backups and recovery, tightened security configurations and easy vertical scaling, depending on the provider.

- Amazon Relational Database Service (RDS) provides pre-configured MySQL and PostgreSQL instances. The instances can be scaled to larger or smaller configurations based on storage and performance needs.
- Google Cloud SQL is a service with managed, backed up, replicated, and auto-patched MySQL instances. Cloud SQL integrates with Google App Engine but can be used independently as well.

- BitCan provides both MySQL and MongoDB hosted databases with extensive backup services.
- ElephantSQL is a software-as-a-service company that hosts PostgreSQL databases and handles the server configuration, backups and data connections on top of Amazon Web Services instances.

SQL resources

You may plan to use an object-relational mapper (ORM) as your main way of interacting with a database, but you should still learn the basics of SQL to create schemas and understand the SQL code generated by the ORM. The following resources can help you get up to speed on SQL if you have never previously used it.

- Select Star SQL is an interactive book for learning SQL. Highly recommended even if you feel you will only be working with an object-relational mapper on your projects because you never know when you will need to drop into SQL to improve a generated query's slow performance.
- A beginners guide to SQL does a good job explaining the main keywords used in SQL statements such as `SELECT`, `WHERE`, `FROM`, `UPDATE` and `DELETE`.
- SQL Tutorial teaches the SQL basics that can be used in all major relational database implementations.
- Life of a SQL query explains what happens both conceptually and technically within a database when a SQL query is run. The author uses PostgreSQL as the example database and SQL syntax throughout the post.
- A Probably Incomplete, Comprehensive Guide to the Many Different Ways to JOIN Tables in SQL elaborates on one of the trickiest parts of writing SQL statements that bridge one or more tables: the `JOIN`.
- Writing better SQL is a short code styling guide to make your queries easier to read.
- SQL Intermediate is a beyond-the-basics tutorial that uses open data from the US Consumer Financial Protection Bureau as examples for counting, querying and using views in PostgreSQL.

General database resources

- How does a relational database work? is a detailed longform post on the sorting, searching, merging and other operations we often take for granted when using an established relational database such as PostgreSQL.

- Databases 101 gives a great overview of the main relational database concepts that is relevant to even non-developers as an introduction.
- Five Mistakes Beginners Make When Working With Databases explains why you should not store images in databases as well as why to be cautious with how you normalize your schema.
- DB-Engines ranks the most popular database management systems.
- DB Weekly is a weekly roundup of general database articles and resources.
- Designing Highly Scalable Database Architectures covers horizontal and vertical scaling, replication and caching in relational database architectures.
- Online migrations at scale is a great read on breaking down the complexity of a database schema migration for an operational database. The approach the author's team used was a 4-step dual writing pattern to carefully evolved the way data for subscriptions were stored so they could move to a new, more efficient storage model.
- A one size fits all database doesn't fit anyone explains Amazon Web Services' specific rationale for having so many types of relational and non-relational databases on its platform but the article is also a good overview of various database models and their use cases.
- SQL is 43 years old - here's 8 reasons we still use it today lists why SQL is commonly used by almost all developers even as the language approaches its fiftieth anniversary.
- SQL keys in depth provides a great explanation for what primary keys are and how you should use them.
- Exploring a data set in SQL is a good example of how SQL alone can be used for data analysis. This tutorial uses Spotify data to show how to extract what you are looking to learn from a data set.
- Databases integration testing strategies covers a difficult topic that comes up on every real world project.
- GitLab provided their postmortem of a database outage on January 31 as a way to be transparent to customers and help other development teams learn how they screwed up their database systems then found a way to recover.
- Asynchronous Python and Databases is an in-depth article covering why many Python database drivers cannot be used without modification due to the differences

in blocking versus asychronous event models. Definitely worth a read if you are using WebSockets via Tornado or gevent.

- PostgreSQL vs. MS SQL Server is one perspective on the differences between the two database servers from a data analyst.

Databases learning checklist

1. Install PostgreSQL on your server. Assuming you went with Ubuntu run `sudo apt-get install postgresql .`
2. Make sure the psycopg2 library is in your application's dependencies.
3. Configure your web application to connect to the PostgreSQL instance.
4. Create models in your ORM, either with Django's built-in ORM or SQLAlchemy with Flask.
5. Build your database tables or sync the ORM models with the PostgreSQL instance, if you're using an ORM.
6. Start creating, reading, updating and deleting data in the database from your web application.

What's next to get your app running?

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

My app runs but looks awful. How do I style the user interface?

How do I use JavaScript with my Python web application?

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
» Relational Databases
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.