Techguide: 6 things a developer should know about Postgres

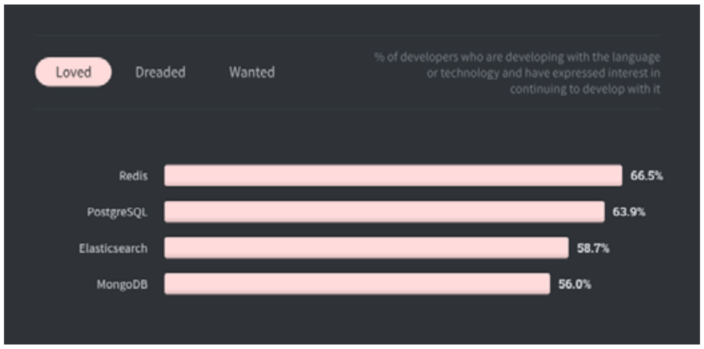
Contents

Introduction

PostgreSQL has long been one of the most frequently deployed relational database management systems. According to DB-Engines, PostgreSQL (or Postgres) and IBM® Db2® were two of the top five most popular relational database management systems in June 2020, as scored by search engine results, technical discussions, social and professional network mentions, and job offers.

DMS	**Rank**		
	June 2020	May 2020	June 2019
Oracle	1	1	1
MySQL	2	2	2
Microsoft SQL Server	3	3	3
PostgreSQL	4	4	4
MongoDB	5	5	5
IBM Db2	6	6	6
Elasticsearch	7	7	7
Redis	8	8	8
SQLite	9	9	11
Cassandra	10	11	10

Over the past few years, PostgreSQL has seen a sharp rise in popularity with developers—the 2020 Stack Overflow Developer Survey ranks it as the second "most loved" database.



In short, Postgres is a popular and powerful open-source option for relational database management. In this tech guide, IBM Business Partner® EDB describes six things that a developer should know about Postgres, focusing on its installation, use, deployment, capabilities and features.

Postgres has come a long way since its origins as a research project in the mid-1980s at the University of California, Berkeley. Its first step towards the mainstream was in 1995 when it replaced its query language (PostQUEL) with SQL, the de facto industry standard. The very next year, PostgreSQL left academia with the establishment of the PostgreSQL Global Development Group, which adopted the open-source Postgres codebase. Since the release of version 9.5 in 2016 (the oldest version still currently supported), its popularity has risen steadily.[1]

There are a number of factors that have contributed to PostgreSQL's popularity, including its highly active open-source community that is not controlled by any single sponsor or company. Its feature set is continually being extended, with the community testing each new feature thoroughly before integration, so that the highest level of reliability is upheld. Despite its wealth of features, Postgres is lightweight (the source code is less than 20 MB), easy to install, and easy to move around compared to bulky proprietary RDBMS. It is also highly flexible, supporting relational and non-relational models in the same DB, and can be deployed across a wide range of infrastructures from bare metal to VMs, containers, and DBaaS.

01. No need to learn a new IDE

When you opt for a proprietary DBMS, you will likely need to adopt their development and administrative tooling as well. If you choose to work with PostgreSQL, however, it is unlikely that you will need to install and learn a new integrated development environment (IDE), because chances are that the framework you are used to working with is already integrated with PostgreSQL. The Community Guide to PostgreSQL GUI Tools on Wiki currently lists no fewer than 18 freeware and 63 proprietary IDEs that support PostgreSQL, including:

– **pgAdmin**, a GUI maintained by members of the PostgreSQL community for developing and administering PostgreSQL databases on Linux®, Unix, Mac OS X, and Windows
– **TOra**, a free Oracle DBA interface with some PostgreSQL support
– **DBeaver**, a multi-platform database tool that supports the most popular SQL and NoSQL databases, including Postgres (Community Edition is free; Enterprise Edition is subject to a license fee)
– **dbForge Studio for Postgres**, a user-friendly interface for creating, developing, and executing queries, editing and adjusting code, importing and exporting data, reporting and editing data, and more
– **EMS SQL Manager for PostgreSQL**, with visual and text tools to build queries, a visual database designer, rapid database management, navigation and maintenance, and more
– **JetBrains' DataGrip**, a multi-engine database IDE for writing SQL code, exploring databases, running queries, importing/exporting data, and more

02. Simple local setup and configuration

Many traditional RDBMS interact in proprietary ways with storage and operating systems. For example, they require special kernels or raw devices, they use low-level operating system functionality, their backup tools and appliances are highly invasive, and so on. As a result, setting up the local environment and configuring proprietary database management systems are complex and time-consuming tasks that must be carried out by specialized and experienced DBAs before developers can get started on the project at hand.

By contrast, one of the design principles of PostgreSQL from the very start was that it would interact seamlessly and predictably with operating systems and storage frameworks. It is remarkably easy to install, set up, and configure a PostgreSQL deployment. Developers often can handle simpler deployments themselves, with DBAs only needing to get involved in larger-scale production deployments. PostgreSQL also integrates easily with standard operating system backup and monitoring tooling. It is lightweight and easy to move around, and surprises are rare. For all these reasons, PostgreSQL is a great enabler for developers across the entire application lifecycle—from prototyping to production.

And if you would like to avoid having to set up and configure a local environment for your PostgreSQL deployment, all of the major cloud providers offer PostgreSQL in a Database as a Service (DBaaS) model, including Amazon RDS for PostgreSQL, Azure Database for PostgreSQL, (Google) Cloud SQL for PostgreSQL, and IBM Cloud® Databases for PostgreSQL. There are also third-party vendors that offer fully managed PostgreSQL cloud services, including the EDB Cloud Database Service on the AWS cloud.

03. Flexible multi-model architecture

The Postgres research project that ultimately spawned PostgreSQL was a pioneer of the object-relational DBMS (ORDBMS): a relational database that is extended to support some object-oriented programming features. These include extensibility of data types, access methods and functions, inheritance of table properties and data types, polymorphism (with an operator having different meanings within the same database), and encapsulation of tables. When storing and retrieving data in an ORDBMS, an object-relational mapping (ORM) tool automatically translates between non-scalar object values (attributes and fields) and scalar relational values (integers and strings organized within tables). Today, all of the leading ORM tools support PostgreSQL.

The multi-model ORDBMS approach is important for today's advanced apps that typically have to query and manipulate both structured and semi-structured data, which can require implementing both SQL and NoSQL databases in a single application. PostgreSQL, for example, has built-in support for storing JSON objects in a compact JSON binary format. This feature is especially convenient for applications that pass around semi-structured documents in JSON format and want to store them directly in the database in order to ensure high fidelity. This semi-structured data can be stored and queried alongside more structured data in tables as well, and all of it can be accessed using standard SQL and tooling.

In short, the power of the PostgreSQL multi-model architecture is that you do not have to embrace a new development and administrative tool chain to get the benefits of an ORDBMS, such as accelerated development cycles and enhanced runtime performance.

04. Agile and DevOps-friendly

Perhaps one of the biggest attractions of PostgreSQL for developers is that it lends itself well to dynamic DevOps environments and requirements.

First of all, getting started with PostgreSQL is frictionless. There are no lengthy procurement cycles, since the license for any kind of use, including commercial, is free and permissive. As discussed above, setting up and configuring the PostgreSQL environment is straightforward. And because PostgreSQL is readily available as a cloud service, it aligns well with the cloud-based nature of DevOps methods.

Secondly, PostgreSQL's multi-model architecture and its support for JSON and ORM mean that systems can be quickly prototyped without the need to fully design a detailed schema. In general, PostgreSQL fits well into continuous development, integration, and deployment processes, because it is easy to make and test small updates frequently. Developers can ensure consistent infrastructure configurations across the entire application lifecycle by introducing Infrastructure-as-Code recipes using Chef, Ansible, Puppet, or any other infrastructure automation platform. PostgreSQL also fully supports the use of containers and microservices that are at the heart of today's modular, distributed app architectures.

And finally, PostgreSQL is highly portable, making it extremely easy to replicate instances for development, testing, and staging purposes. In PostgreSQL, there are clear boundaries between the database and underlying operating and storage systems. This is unlike traditional database management systems that are inextricably tied into the DBMS owner's operating system. Thus, it is easy to move PostgreSQL databases around, run them locally or in the cloud, or run multiple versions that do not interfere with each other. Whatever you can do in your standard operating system or storage framework (copy directory, change permissions, etc.) you can do seamlessly in PostgreSQL. With everything, including data storage, installed in one directory and virtually no connections to external resources, all you have to do is copy the file system in order to copy the database.

05. Cool things you can do with PostgreSQL

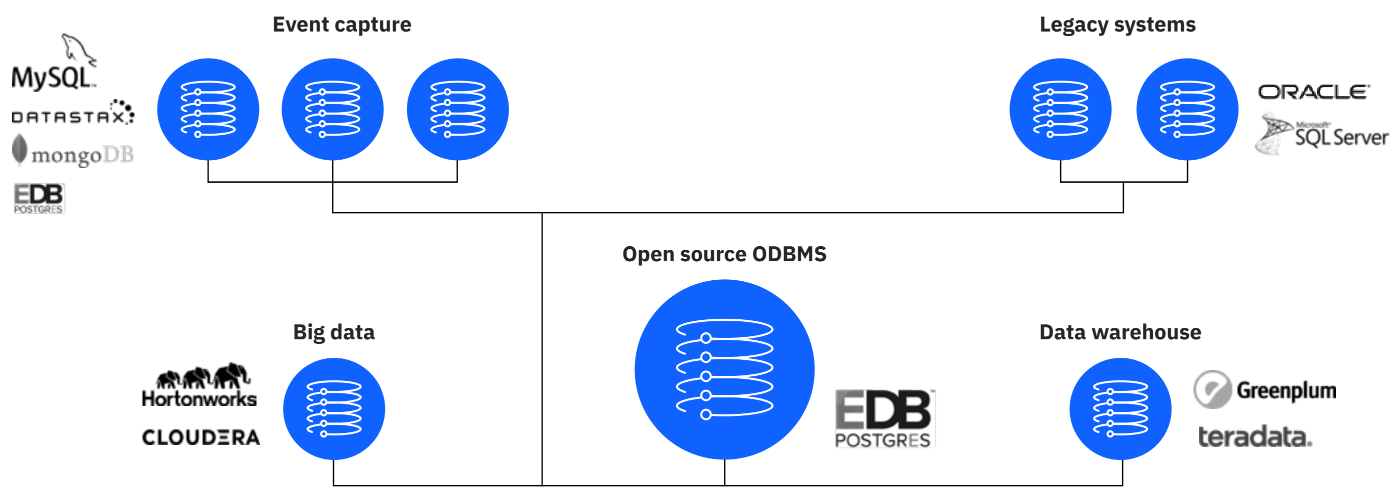
Before considering the really cool stuff, you should remember that PostgreSQL is highly ANSI SQL-compliant. Thus, developers who already have SQL skills will feel right at home in PostgreSQL. Those who are new to SQL can take comfort in the fact that SQL is considered easy to learn and that there's a lot of PostgreSQL tooling for building and testing SQL. EDB's Postgres Enterprise Manager (PEM), for example, extends the community's administration and management tool (pgAdmin) to meet the needs of large-scale enterprise PostgreSQL deployments, including a powerful SQL editor and query builder.

The real benefits and innovative opportunities, however, arise because PostgreSQL has built-in support for many non-relational NoSQL features without the drawbacks of NoSQL such as client-side data analysis, lack of a powerful query language and optimizer, and data retention becoming an admin responsibility. Some of these supported features include:

- **Arrays:** A table column can be defined as a multidimensional array of any built-in or user-defined base type, enum type, or composite type.
- **JSON/JSONB data type:** This validates stored values against the JSON rules, with built-in JSON- specific functions and operators available. JSON stores an exact copy of the input text, while JSONB decomposes the input text into a binary format that is faster to process and supports indexing.
- **Character strings:** PostgreSQL has all the support that you would expect for storing, manipulating, and querying character strings. More advanced capabilities are available through extensions, such as full-text indexing using ZomboDB, an Elasticsearch PostgreSQL extension.
- **Full-text search:** This feature is fast and optimized, with support for fuzzy matching, ranking, phrase search, and multiple languages. It also supports GiST and GIN index types to speed up full-text searches.
- **Range types:** Built-in range types include integer, bigint, numeric, timestamp (with and without time zone), and date. PostgreSQL also lets you define custom range types.
- **Geometric support:** Data types for representing two-dimensional spatial objects as well as a set of functions and operators to perform geometric operations such as scaling, translation, rotation, and determining intersections.
- **XML support:** The XML data type checks input values for well-formedness and provides functions to perform type-safe operations.
- **Many specialized index types:** In addition to the GiST and GIN indexing infrastructures already mentioned in the context of full-text search, PostgreSQL supports:
  - **B-Tree:** Used for most data types and queries, can also be helpful in retrieving data in sorted order, and supports multicolumn indexes
  - **SP-GiST:** Space-partitioned GiST, used for larger datasets with natural but uneven clustering
  - **BRIN:** "Block Range Index," a form of indexing for large datasets that line up sequentially

There are also numerous third-party plugins—many of them open source—that extend Postgres capabilities. These extensions are so tightly integrated that they behave the same way as built-in features. Some good open source examples are: TimescaleDB and PipelineDB, for time-series data aggregation and graphs; AgensGraph, a transactional graph database; and PostGIS, which adds support for geographic objects and location queries, turning PostgreSQL into a fast, robust, and feature-rich spatial database management system.

PostgreSQL is a highly communicative and versatile DBMS that uses foreign data wrappers (FDW) to seamlessly read from and write to foreign data sources. Some notable FDWs include CouchDB, Informix®, MongoDB, MySQL, Neo4j, Oracle, and Redis. You can see a long list of currently available FDWs in the Postgres Wiki (although not all of them are officially supported by the PostgreSQL Global Development Group). With FDWs, you can use a single PostgreSQL database to federate at scale a diverse range of data sources and formats as well as app requirements, as illustrated below:



06. Smooth road to production

PostgreSQL's flexibility and ease of use also apply to how it is deployed in production. PostgreSQL lives comfortably on all of today's popular deployment platforms, from bare metal to VMs (on-prem or IaaS), containers, and DBaaS. The right choice of platform will depend on many factors, including the app's scalability and availability requirements, its architecture, and the extent to which the organization wants or needs to have direct control over infrastructure and orchestration issues.

For example, PostgreSQL is the third-most-popular technology being run by enterprises on Docker containers, according to Datadog. Containerized PostgreSQL will behave consistently throughout the application lifecycle (development, staging, production) and is well-suited to microservice-based app architectures. It also delivers high availability in failover situations and optimizes compute-storage costs through elastic on-demand scaling rather than upfront over-provisioning.

Fully managed PostgreSQL DBaaS offerings, such as EDB's Cloud Database Service or other cloud service providers, simplify administration and can potentially boost developer productivity. A middle ground would be EDB Ark Platform, which lets organizations set up their own DBaaS for flexible self-service database provisioning and also benefit from the enterprise-grade high availability, monitoring, backup, and load balancing of the EDB Postgres Advanced Server.

Final note

Bruce Momjian is co-founder of the PostgreSQL Global Development Group, a tireless PostgreSQL evangelist, and a prolific blogger. His article Making Postgres Central in Your Data Center notes that all the reasons discussed above for developers to embrace PostgreSQL also perfectly position PostgreSQL to play a central role in the enterprise data center.

PostgreSQL's inherent reliability, flexibility, and extensibility, together with the support of its active open source community, help ensure that it will stay up to date as DBMS methods and technology stacks evolve.

EDB employs a team of PostgreSQL experts and is proud to offer a cost-effective, enterprise-ready Postgres platform. And thanks to the partnership between IBM and EDB, IBM customers can benefit from EDB solutions that are fully integrated with the larger IBM data management and AI platform, including a dedicated Data Management Platform for EDB and the IBM Cloud Pak® for Data.

Learn more about IBM solutions for EDB and PostgreSQL here.

1 https://db-engines.com/en/ranking_trend/system/PostgreSQL

N58LN9VQ