What is a Relational Database (RDBMS)?

A relational database is a type of database that stores and provides access to data points that are related to one another. Relational [databases](https://www.oracle.com/database/what-is-database/) are based on the relational model, an intuitive, straightforward way of representing data in tables. In a relational database, each row in the table is a record with a unique ID called the key. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.

## A relational database example

Here’s a simple example of two tables a small business might use to process orders for its products. The first table is a customer info table, so each record includes a customer’s name, address, shipping and billing information, phone number, and other contact information. Each bit of information (each attribute) is in its own column, and the database assigns a unique ID (a key) to each row. In the second table—a customer order table—each record includes the ID of the customer that placed the order, the product ordered, the quantity, the selected size and color, and so on—but not the customer’s name or contact information.

These two tables have only one thing in common: the ID column (the key). But because of that common column, the relational database can create a relationship between the two tables. Then, when the company’s order processing application submits an order to the database, the database can go to the customer order table, pull the correct information about the product order, and use the customer ID from that table to look up the customer’s billing and shipping information in the customer info table. The warehouse can then pull the correct product, the customer can receive timely delivery of the order, and the company can get paid.

## How relational databases are structured

The relational model means that the logical data structures—the data tables, views, and indexes—are separate from the physical storage structures. This separation means that database administrators can manage physical data storage without affecting access to that data as a logical structure. For example, renaming a database file does not rename the tables stored within it.

The distinction between logical and physical also applies to database operations, which are clearly defined actions that enable applications to [manipulate the data](https://www.oracle.com/database/what-is-data-management/) and structures of the database. Logical operations allow an application to specify the content it needs, and physical operations determine how that data should be accessed and then carries out the task.

To ensure that data is always accurate and accessible, relational databases follow certain integrity rules. For example, an integrity rule can specify that duplicate rows are not allowed in a table in order to eliminate the potential for erroneous information entering the database.

# Sqlite vs. MySQL vs. PostgreSQL: A Comparison of Relational Databases

Relational Database Management Systems (RDBMS) are one of the most widely used database management systems in the world. Based on the[relational model](https://en.wikipedia.org/wiki/Relational_model) invented by Edgar F. Codd, these databases store data in the form of tables, and allow the data to be linked by establishing a relationship between the tables. This results in an efficient data storage mechanism where the data can be referenced from elsewhere in the database.

**More on the subject:**

* [**Shipping AWS Lambda Metrics to Logz.io**](https://logz.io/blog/aws-lambda-metrics/)
* [**Can Distributed Tracing Replace Logging?**](https://logz.io/blog/can-distributed-tracing-replace-logging/)
* [**Full Observability with Your Node.js App**](https://logz.io/blog/observability-nodejs-logs-metrics-traces/)

In this article, we compare three of the[most popular](https://db-engines.com/en/ranking/relational+dbms) open-source RDBMS on the market. The comparison delves into differences in architecture, business continuity, use cases, and support that help you make an informed decision while pursuing database systems for your application.

## **SQLite**

## 

### **Overview and features**

SQLite is an embedded, file-based RDBMS that does not require any installation or setup. This, in turn, means that the application does not run under a separate server process that needs to be started, stopped, or configured. This serverless architecture enables the database to be cross-platform compatible.

The complete SQL database is contained within a single disk file and all reads and writes take place directly on this disk file. As the data is directly written back to the disk file, SQLite adheres to the ACID properties to safeguard transactions against memory allocation failures and disk I/O errors that can result from unexpected system crashes or power failures.

### **Advantages and use cases**

The SQLite library is one of the most compact libraries in this list where the size of the library can easily be under 600 KB. Due to its very small footprint and the nature of the RDBMS, it is a very good fit for IoT and embedded devices.

Some other good use cases include low-to-medium traffic websites (~ 100K requests a day), testing and internal development purposes, data analysis using Tcl or Python and educational purposes (this is simple to set up and can be used to teach SQL concepts to students).

One major advantage of SQLite is how it can act as a complementary solution for client/server enterprise RDBMS. For example, it can cache data from client/server RDBMS locally and thereby reduce the latency for queries and keep the end application alive in case of enterprise RDBMS outages.

### **Disadvantages**

One of the main drawbacks of the SQLite system is its lack of multi-user capabilities which can be found in full-fledged RDBMS systems like MySQL and PostgreSQL. This translates to a lack of granular access control, a friendly user management system, and security capabilities beyond encrypting the database file itself. This is a major drawback when designing multi-user applications like CRM and SaaS applications and is normally not favored when building multi-user or multi-tenant applications.

Another big disadvantage of SQLite is its handling of writes operations which are serialized. This can be a major bottleneck for applications that require concurrency. As SQLite is a file-based DBMS, it can cause performance issues with larger datasets because of file system limitations. It would be appropriate to choose client/server databases like MySQL and PostgreSQL in these instances, especially when dealing with large datasets like Big Data.

Also worth noting is the lack of any Database as a Service (DBaaS) offering from any major cloud provider. With the advent of the public cloud, use of PaaS services (like DBaaS) by developers and DevOps teams have become common place. Lack of a managed service offering from top public cloud providers means that the common management tasks must be the responsibility of the DB Admin thus increasing OPEX costs.

Ownership, support, and key customers

SQLite is an open-source database maintained by a group of[developers](https://www.sqlite.org/crew.html). For community support, it offers a[public mailing list](http://sqlite.org:8080/cgi-bin/mailman/listinfo/sqlite-users) and also offers paid[professional support](https://www.sqlite.org/prosupport.html). Some of the key customers of SQLite are Facebook, Google, and Apple.

## **MySQL**

### **Overview and features**

MySQL is [one of the most popular](https://db-engines.com/en/ranking) open-source and large-scale RDBMS systems out there. Unlike SQLite, it employs a server/client architecture that consists of a multi-threaded SQL server. This multi-threaded nature of MySQL allows for greater performance as the kernel threads can easily utilize multiple CPUs. The database is written in C and C++ and supports[various platforms](https://www.mysql.com/support/supportedplatforms/database.html) like Windows Server Operating Systems and Linux distros like RHEL 7 and Ubuntu. It also adheres to the ACID system for transaction consistency and provides various Connectors and APIs like C, C++, Java, PHP, etc.

Scalability, security, and replication are some of the key features that make MySQL one of the most popular choices in enterprise applications:

* Security features include MySQL Access Privilege System which provides user authentication,[user account management system](https://dev.mysql.com/doc/refman/8.0/en/creating-accounts.html#creating-accounts-granting-privileges) and encrypted connections using SSL.
* MySQL offers replication from master to slave and master to master which can prove useful while scaling out reads, useful as a backup solution or even failover scenarios in case of downtime. MySQL also has[commercial products](https://www.mysql.com/products/) which have more extensive features.
* The MySQL Enterprise edition, for example, has additional features like MySQL Transparent Data Encryption (TDE), MySQL Enterprise Backup, and MySQL document store.
* MySQL also offers an embedded multi-threaded library which provides a smaller footprint for use in embedded and IoT systems.

### **Advantages and use cases**

Apart from having several enterprise features, another major differentiation between MySQL and SQLite is MySQL’s support for multi-user features. This, along with the enterprise features and scalability, makes it a perfect candidate for distributed applications.

MySQL holds an edge over PostgreSQL for simple read-heavy operations when it comes to throughput and performance. It is also much simpler to install and use and has a broader community compared to PostgreSQL.

The ease of use and manageability makes it a great tool for websites, web applications, OLTP applications and is an important part of the[LAMP stack](https://en.wikipedia.org/wiki/LAMP_(software_bundle)). MySQL 8 has also introduced NoSQL features which brings it on par with PostgreSQL.

### **Disadvantages**

As MySQL moves old data to a separate area called rollback segments, bulk INSERTs can have an adverse impact on performance. This is where PostgreSQL shines. It also does not work well with long-running SELECTs and is best suited to smaller SELECTs especially the ones covering clustered index. Some of the other disadvantages include a lack of full-text search and slow concurrent read-writes.

### **Ownership, support, and key customers**

MySQL is owned and maintained by Oracle. Community support is maintained through forums and premier support can be obtained by[purchasing](https://www.mysql.com/products/) commercial products. Some of the major customers of MySQL are Facebook, GitHub, and YouTube.

## **PostgreSQL**

### **Overview and features**

PostgreSQL is an open-source object RDBMS with special emphasis on extensibility and standards compliance. Like MySQL, PostgreSQL uses a client/server database model and the server process that handles the client communications, manages the database files and operations, is known as the *postgres* process.

PostgreSQL handles concurrent client sessions by creating (“forking”) a new process for each connection. This process is separate from the master *postgres* process and is created and destroyed during the lifetime of the client connection. Written in C, Postgres is ACID compliant and supports functions and stored procedures. Unlike MySQL, PostgreSQL supports materialized views (cached views) resulting in faster frequent access to big and active tables.

Like MySQL, PostgreSQL also has several advanced features like security and replication. PostgreSQL relies on synchronous replication between the master and the slave database. Apart from providing user access control, host-based access control, and user authentication, PostgreSQL also natively provides the capability to encrypt client/server communications using SSL. Full ACID compliance is native to PostgreSQL whereas it is present in InnoDB and NDB Cluster programs for MySQL.

PostgreSQL uses a technology known as Multiversion Concurrency Control or MVCC for maintaining data consistency during concurrent access of data. This technology is superior to just using locks for concurrency as it minimizes lock contention in multi-user environments thereby significantly improving performance. For backward compatibility or applications which want the classic lock technology, PostgreSQL also allows table and row locking technologies to provide concurrency. On the contrary, MySQL only supports MVCC in InnoDB instances.

### **Advantages and use cases**

Because of its superior parallel processing capability, PostgreSQL comes out on top (compared to MySQL) when running long SELECTs like in analytical applications. PostgreSQL has always been considered the best for analytical processes such as data warehousing. A prime example is[Timescale DB](https://www.timescale.com/), which can allow you to INSERT 1 million records per second. Its use case also includes OLTP applications and applications in the financial industry due to strong adherence to ACID compliance. The extensibility of the PostgreSQL database also makes it a perfect candidate for research and scientific projects.

### **Disadvantages**

Although the differences between PostgreSQL and MySQL are not as great as they once were, it is worth noting that the storage engine of PostgreSQL still needs extensive work and[as per PostgreSQL](https://wiki.postgresql.org/wiki/Future_of_storage), it is suboptimal in certain use cases and may even need to adopt features from engines like InnoDB.

When compared to MySQL, PostgreSQL is more power hungry. As PostgreSQL forks off a process for new client connections, it can[take up to 10 MB per connection](https://www.citusdata.com/blog/2017/05/10/scaling-connections-in-postgres/). This model can take up a lot of memory as concurrent client connection goes when compared to the thread-per-connection model of MySQL.

Another big disadvantage can be seen during frequent UPDATEs, where due to no support for clustered indexes, PostgreSQL can have a huge adverse impact on performance compared to MySQL databases.

### **Ownership, support, and key customers**

PostgreSQL is open source with the source code published on[GitHub](https://github.com/postgres/postgres) and is maintained by the PostgreSQL Global Development Group. PostgreSQL has both options of community and commercial support. Community Support is in the form of[mailing lists](https://www.postgresql.org/list/) and[IRC](https://www.postgresql.org/community/irc/) while the list of commercial support providers can be found[here](https://www.postgresql.org/support/professional_support/). Although PostgreSQL has a smaller market share compared to MySQL, it has an impressive list of clients like AWS RedShift, Instagram, ViaSat, and Cloudera.

## **Summary**

In terms of popularity, MySQL is way ahead of PostgreSQL and SQLite, but one must consider the use case and features before making it the de-facto choice.

Features like strong ACID compliance and concurrency would make PostgreSQL a strong contender. SQLite, on the other hand would excel in embedded and IoT applications where the small footprint of the SQLite library would give it a huge advantage over its competitors. If you are looking at a multi-user application, then SQLite would be a poor choice and you must hash out the pros and cons of PostgreSQL and MySQL.

To make this choice easier for you, we have compiled a list of comparisons between SQLite, MySQL, and PostgreSQL that might help you make your decision.