DigitalOcean

Subscribe Share Contents

SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems

94

Posted February 21, 2014 1.1m MYSQL POSTGRESQL CONCEPTUAL

Introduction

Relational databases have been in use for a long time. They became popular thanks to management systems that implement the relational model extremely well, which has proven to be a great way to work with data [especially for mission-critical applications].

In this DigitalOcean article, we are going to try to understand the core differences of some of the most commonly used and popular relational database management systems (RDBMS). We will explore their fundamental differences in terms of features and functionality, how they work, and when one excels over the other in order to help developers with choosing a RDBMS.

Glossary

1. Database Management Systems

1. Relational Database Management Systems
2. Relations And Data Types
3. Popular And Important Relational Databases

2. SQLite

1. SQLite's Supported Data Types
2. Advantages of SQLite
3. Disadvantages of SQLite
4. When To Use SQLite

3. MySQL

4. PostgreSQL

Database Management Systems

Databases are logically modelled storage spaces for all kinds of different information (data). Each database, other than schema-less ones, have a model, which provide structure for the data being dealt with. Database management systems are applications (or libraries) which manage databases of various shapes, sizes, and sorts.

Note: To learn more about Database Management Systems, check out our article: Understanding Databases.

Relational Database Management Systems

Relational Database Systems implement the relational model to work with the data. Relational model shapes whatever information to be stored by defining them as related entities with attributes across tables (i.e. schemas).

These type of database management systems require structures (e.g. a table) to be defined in order to contain and work with the data. With tables, each column (e.g. attribute) holds a

different type (e.g. data type) of information. Each record in the database, uniquely identified with *keys*, translates to a row that belongs to a table, with each row's series of attributes being represented as the columns of a table -- all related together, as defined within the relational model.

Relations And Data Types

Relations can be considered as mathematical sets that contain series of attributes which collectively represent the database and information being kept. This type of identification and collection method allow relational databases to work the way they do.

When defining a table to insert records, each element forming a record (i.e. attribute) must match the defined *data type* (e.g. an integer, a date etc.). Different relational database management systems implement different data types -- which are not always directly interchangeable.

Working with and through constraints, like the one we have just explained, is common with relational databases. In fact, constraints form the core of the relations.

Note: If you need to work with truly unrelated, randomly represented information (e.g. a document), you might be interested in using a NoSQL (schema-less database). If you would like to learn more about them, check out our article A Comparison Of NoSQL Database Management Systems.

Popular And Important Relational Databases

In this article, we are going to introduce three major and important open-source relational database management systems that have helped to shape the world of application development.

- **SQLite:**

  A very powerful, embedded relational database management system.

- **MySQL:**

  The most popular and commonly used RDBMS.

- **PostgreSQL:**

  The most advanced, SQL-compliant and open-source objective-RDBMS.

Note: Open-source applications almost always come with the freedom to use any way desired. Most of the time freedom to fork the project (therefore use the code) to create something new is also permitted. If you are interested in DBMSs, you might want to check out some forked projects, based on these popular ones, such as the MariaDB.

SQLite

SQLite is an amazing library that gets embedded inside the application that makes use of. As a self-contained, file-based database, SQLite offers an amazing set of tools to handle all sorts of data with much less constraint and ease compared to hosted, process based (server) relational databases.

When an application uses SQLite, the integration works with functional and direct calls made to a file holding the data (i.e. SQLite database) instead of communicating through an interface of sorts (i.e. ports, sockets). This makes SQLite extremely fast and efficient, and also powerful thanks to the library's underlying technology.

SQLite's Supported Data Types

- **NULL:**

  NULL value.

- **INTEGER:**

  Signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.

- **REAL:**

  Floating point value, stored as an 8-byte IEEE floating point number.

- **TEXT:**

  Text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).

- **BLOB:**

  A blob of data, stored exactly as it was input.

Note: To learn more about SQLite's data types and SQLite type affinity, check out the official documentation on the subject.

Advantages of SQLite

- **File based:**

  The entire database consists of a single file on the disk, which makes it extremely portable.

- **Standards-aware:**

  Although it might appear like a "simple" DB implementation, SQLite uses SQL. It has some features omitted (`RIGHT OUTER JOIN` or `FOR EACH STATEMENT`), however, some additional ones are baked in.

- **Great for developing and even testing:**

  During the development phase of most applications, for a majority of people it is extremely likely to need a solution that can scale for concurrency. SQLite, with its rich feature base, can offer more than what is needed for development with the simplicity of working with a single file and a linked C based library.

Disadvantages of SQLite

- **No user management:**

  Advanced databases come with the support for users, i.e. managed connections with set access privileges to the database and tables. Given the purpose and nature of SQLite (no higher-levels of multi-client concurrency), this feature does not exist.

- **Lack of possibility to tinker with for additional performance:**

  Again by design, SQLite is not possible to tinker with to obtain a great deal of additional performance. The library is simple to tune and simple to use. Since it is not complicated, it is technically not possible to make it more performant than it already, amazingly is.

When To Use SQLite

- **Embedded applications:**

All applications that need portability, that do not require expansion, e.g. single-user local applications, mobile applications or games.

- **Disk access replacement:**

  In many cases, applications that need to read/write files to disk directly can benefit from switching to SQLite for additional functionality and simplicity that comes from using the *Structured Query Language* (SQL).

- **Testing:**

  It is an overkill for a large portion of applications to use an additional process for testing the business-logic (i.e. the application's main purpose: functionality).

When Not To Use SQLite

- **Multi-user applications:**

  If you are working on an application whereby multiple clients need to access and use the same database, a fully-featured RDBM (e.g. MySQL) is probably better to choose over SQLite.

- **Applications requiring high write volumes:**

  One of the limitations of SQLite is the *write* operations. This DBMS allows only one single write*operating to take place at any given time, hence allowing a limited throughput.

MySQL

MySQL is the most popular one of all the large-scale database servers. It is a feature rich, open-source product that powers a lot of web-sites and applications online. Getting started with MySQL is relatively easy and developers have access to a massive array of information regarding the database on the internet.

Note: It should be stated that given the popularity of the product, there are a lot of third-party applications, tools and integrated libraries which help greatly with many aspects of working with this RDBMS.

Despite not trying to implement the full SQL standard, MySQL offers a lot of functionality to the users. As a stand-alone database server, applications talk to MySQL daemon process to access the database itself -- unlike SQLite.

MySQL's Supported Data Types

- **TINYINT:**

  A very small integer.

- **SMALLINT:**

  A small integer.

- **MEDIUMINT:**

  A medium-size integer.

- **INT or INTEGER:**

  A normal-size integer.

- **BIGINT:**

  A large integer.

- **FLOAT:**

  A small (single-precision) floating-point number. Cannot be unsigned.

- **DOUBLE, DOUBLE PRECISION, REAL:**

  A normal-size (double-precision) floating-point number. Cannot be unsigned.

- **DECIMAL, NUMERIC:**

  An unpacked floating-point number. Cannot be unsigned.

- **DATE:**

  A date.

- **DATETIME:**

A date and time combination.

- **TIMESTAMP:**

  A timestamp.

- **TIME:**

  A time.

- **YEAR:**

  A year in 2- or 4- digit formats (default is 4-digit).

- **CHAR:**

  A fixed-length string that is always right-padded with spaces to the specified length when stored.

- **VARCHAR:**

  A variable-length string.

- **TINYBLOB, TINYTEXT:**

  A BLOB or TEXT column with a maximum length of 255 (2^8 - 1) characters.

- **BLOB, TEXT:**

  A BLOB or TEXT column with a maximum length of 65535 (2^16 - 1) characters.

- **MEDIUMBLOB, MEDIUMTEXT:**

  A BLOB or TEXT column with a maximum length of 16777215 (2^24 - 1) characters.

- **LONGBLOB, LONGTEXT:**

  A BLOB or TEXT column with a maximum length of 4294967295 (2^32 - 1) characters.

- **ENUM:**

An enumeration.

- **SET:**

  A set.

Advantages of MySQL

- **Easy to work with:**

  MySQL can be installed very easily. Third-party tools, including visual ones (i.e. GUIs) make it extremely simple to get started with the database.

- **Feature rich:**

  MySQL supports a lot of the SQL functionality that is expected from a RDBMS -- either directly or indirectly.

- **Secure:**

  A lot of security features, some rather advanced, are built in MySQL.

- **Scalable and powerful:**

  MySQL can handle *a lot* of data and furthermore it can be used "at scale", if needed be.

- **Speedy:**

  Giving up some standards allows MySQL to work very efficiently and cut corners, thus providing speed gains.

Disadvantages of MySQL

- **Known limitations:**

  By design, MySQL does not intend to do everything and it comes with functional limitations that some state-of-the-art applications might require.

- **Reliability issues:**

The way certain functionality gets handled with MySQL (e.g. references, transactions, auditing etc.) renders it a little-less reliable compared to some other RDBMSs.

- **Stagnated development:**

  Although MySQL is still technical an open-source product, there are complaints regarding the development process since its acquisition. However, it should be noted that there are some MySQL-based, fully-integrated databases that add value on top of the standard MySQL installations (e.g. MariaDB).

When To Use MySQL

- **Distributed operations:**

  When you need more than what SQLite can offer, including MySQL to your deployment stack, just like any stand-alone database server, brings a lot of operational freedom together with some advanced features.

- **High security:**

  MySQL's security features provide reliable protection for data-access (and use) in a simple way.

- **Web-sites and web-applications:**

  A great majority of web-sites (and web-applications) can simply work on MySQL despite the constraints. This flexible and somewhat scalable tool is easy to use and easy to manage -- which proves very helpful in the long run.

- **Custom solutions:**

  If you are working on a highly specific and extremely custom solution, MySQL can tag along easily and go by your rules thanks to its rich configuration settings and operation modes.

When Not To Use MySQL

- **SQL compliance:**

  Since MySQL does not [try to] implement the full SQL standard, this tool is not completely SQL compliant. If you might need integration with such RDBMSs, switching from MySQL will not be easy.

- **Concurrency:**

  Even though MySQL and some storage engines perform really well with *read* operations, concurrent *read-writes* can be problematic.

- **Lack of features:**

  Again, depending on the choice of the database-engine, MySQL can lack certain features, such as the full-text search.

PostgreSQL

PostgreSQL is *the* advanced, open-source [object]-relational database management system which has the main goal of being standards-compliant and extensible. PostgreSQL, or Postgres, tries to adopt the ANSI/ISO SQL standards together with the revisions.

Compared to other RDBMSs, PostgreSQL differs itself with its support for highly required and integral object-oriented and/or relational database functionality, such as the complete support for reliable transactions, i.e. Atomicity, Consistency, Isolation, Durability (ACID).

Due to the powerful underlying technology, Postgres is extremely capable of handling many tasks very efficiently. Support for concurrency is achieved without read locks thanks to the implementation of Multiversion Concurrency Control (MVCC), which also ensures the ACID compliance.

PostgreSQL is highly programmable, and therefore extendible, with custom procedures that are called "stored procedures". These functions can be created to simplify the execution of repeated, complex and often required database operations.

Although this DBMS does not have the popularity of MySQL, there are many amazing third-party tools and libraries that are designed to make working with PostgreSQL simple, despite this database's powerful nature. Nowadays it is possible to get PostgreSQL as an application package through many operating-system's default package manager with ease.

PostgreSQL's Supported Data Types

- **bigint:**

  signed eight-byte integer

- **bigserial:**

autoincrementing eight-byte integer

- **bit [(n)]:**

  fixed-length bit string

- **bit varying [(n)]:**

  variable-length bit string

- **boolean:**

  logical Boolean (true/false)

- **box:**

  rectangular box on a plane

- **bytea:**

  binary data ("byte array")

- **character varying [(n)]:**

  variable-length character string

- **character [(n)]:**

  fixed-length character string

- **cidr:**

  IPv4 or IPv6 network address

- **circle:**

  circle on a plane

- **date:**

calendar date (year, month, day)

- **double precision:**

  double precision floating-point number (8 bytes)

- **inet:**

  IPv4 or IPv6 host address

- **integer:**

  signed four-byte integer

- **interval [fields] [(p)]:**

  time span

- **line:**

  infinite line on a plane

- **lseg:**

  line segment on a plane

- **macaddr:**

  MAC (Media Access Control) address

- **money:**

  currency amount

- **numeric [(p, s)]:**

  exact numeric of selectable precision

- **path:**

geometric path on a plane

- **point:**

  geometric point on a plane

- **polygon:**

  closed geometric path on a plane

- **real:**

  single precision floating-point number (4 bytes)

- **smallint:**

  signed two-byte integer

- **serial:**

  autoincrementing four-byte integer

- **text:**

  variable-length character string

- **time [(p)] [without time zone]:**

  time of day (no time zone)

- **time [(p)] with time zone:**

  time of day, including time zone

- **timestamp [(p)] [without time zone]:**

  date and time (no time zone)

- **timestamp [(p)] with time zone:**

date and time, including time zone

- **tsquery:**

  text search query

- **tsvector:**

  text search document

- **txid_snapshot:**

  user-level transaction ID snapshot

- **uuid:**

  universally unique identifier

- **xml:**

  XML data

Advantages of PostgreSQL

- **An open-source SQL standard compliant RDBMS:**

  PostgreSQL is open-source and free, yet a very powerful relational database management system.

- **Strong community:**

  PostgreSQL is supported by a devoted and experienced community which can be accessed through knowledge-bases and Q&A sites 24/7 for free.

- **Strong third-party support:**

  Regardless of the extremely advanced features, PostgreSQL is adorned with many great and open-source third-party tools for designing, managing and using the management system.

- **Extensible:**

It is possible to extend PostgreSQL programmatically with stored procedures, like an advanced RDBMS should be.

- **Objective:**

  PostgreSQL is not just a relational database management system but an objective one - with support for nesting, and more.

Disadvantages of PostgreSQL

- **Performance:**

  For simple *read*-heavy operations, PostgreSQL can be an over-kill and might appear less performant than the counterparts, such as MySQL.

- **Popularity:**

  Given the nature of this tool, it lacks behind in terms of popularity, despite the very large amount of deployments - which might affect how easy it might be possible to get support.

- **Hosting:**

  Due to above mentioned factors, it is harder to come by hosts or service providers that offer managed PostgreSQL instances.

When To Use PostgreSQL

- **Data integrity:**

  When reliability and data integrity are an absolute necessity without excuses, PostgreSQL is the better choice.

- **Complex, custom procedures:**

  If you require your database to perform custom procedures, PostgreSQL, being extensible, is the better choice.

- **Integration:**

In the future, if there is a chance of necessity arising for migrating the entire database system to a propriety (e.g. Oracle) solution, PostgreSQL will be the most compliant and easy to handle base for the switch.

- **Complex designs:**

  Compared to other open-source and free RDBMS implementations, for complex database designs, PostgreSQL offers the most in terms of functionality and possibilities without giving up on other valuable assets.

When Not To Use PostgreSQL

- **Speed:**

  If all you require is fast *read* operations, PostgreSQL is not the tool to go for.

- **Simple set ups:**

  Unless you require absolute data integrity, ACID compliance or complex designs, PostgreSQL can be an over-kill for simple set-ups.

- **Replication:**

  Unless you are willing to spend the time, energy and resources, achieving replication with MySQL might be simpler for those who lack the database and system administration experience.

Submitted by: O.S. Tezer

Related Tutorials

How to Deploy Elixir-Phoenix Applications with MySQL on Ubuntu 16.04

How To Install and Use PostgreSQL on Ubuntu 18.04

How To Install MySQL on Ubuntu 18.04

How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 18.04

How To Install and Secure phpMyAdmin with Nginx on Ubuntu 16.04

37 Comments

Leave a comment...

Log In to Comment

adsfas *March 20, 2014*

0 nosql comparison link is broken

Foo *March 20, 2014*

0 I question the claim that MySQL is "The most popular and commonly used RDBMS". It depends on what you mean by "popular and commonly used", but SQLite is used almost everywhere these days. I honestly don't know how to do anything on my computer or phone today that doesn't use SQLite. It's in iOS and Android, it's in Firefox and Chrome, it's in OS X (and used by every OS X application, indirectly, and several of them directly) and Linux. It's used by Dropbox and Skype and Lightroom and Airbus and pretty much every major software company in the world.

MySQL is very popular for web apps, but SQLite is very popular for basically everything else.

---

divinity76 *August 20, 2015*

1 Yes. SQLite is much more popular than any other database. every Mac OS X and iPhone and iPod and iPad and Android and Skype and Dropbox and Firefox and Chrome and Windows 10 installation comes with SQLite built in. and they try to say MySQL is more popular? MySQL is not even close. most people have several SQLite databases. MySQL by comparison, is just installed on a couple of servers. ;)

---

mounir1003 *November 15, 2015*

1 We're talking web apps here, MySql is a mix of complexity and speed, the best for web apps, not the fat slow complex pgsql and not the very simple/fast SQLite, but of corse it depends on apps. almost every website/CMS/Framework today uses MySql.

---

lucas586955 *March 20, 2014*

1 I would like to point out that the JSON and hstore datatypes in PostgreSQL aren't mentioned, and are extremely useful.

---

anon587193 *March 20, 2014*

3 Please rename "Glossary" to "Table of Contents"

---

msx *February 21, 2015*

0 Thank you for pointing that.

---

lauris *March 20, 2014*

0 There was an interesting poll recently. It appears that Postgres is currently more popular among developers than MySQL http://www.databasefriends.co/2014/03/favorite-relational-database.html

---

valdis.veidelis+digitalocean *March 20, 2014*

2 Postgres JSON data type is a huge advantage over MySQL.

---

rickyseltzer *March 20, 2014*

3 The date-time type of MySQL doesn't store the time zone. In PostrgeSQL, it does.

---

kiliankoe *March 20, 2014*

0 It seems you forgot the link where it says

Note: To learn more about Database Management Systems, check out our article: [Understanding Databases](link?).

Otherwise thanks for the great overview!

---

joe587723 *March 20, 2014*

0 I'd love to see this comparison when using a hosted solution like Amazon's RDS. The reason I say this is that the complexity issues with postgres go away, and I believe you're left with all of the benefits of postgres, and very few of the downsides.

---

patrick.hetu *March 20, 2014*

0 "understanding databases" link is broken

---

milan *March 20, 2014*

0 Regarding PostgreSQL read performance,
you just need to put some connection pooling on front, and you will solve that problem.

---

dghaegtrdgasf *March 20, 2014*

2 "Unless you require absolute data integrity, ACID compliance or complex designs, PostgreSQL can be an over-kill for simple set-ups."

People who don't need those things probably have no business using a database at all. It's shocking how many people think that the average self-hosted Wordpress site is actually dynamic and thus requires a database. The vast majority of things out there using a database are static sites with a silly, over-engineered backend.

---

trent.lloyd *March 21, 2014*

0 "Lack of features: Again, depending on the choice of the database-engine, MySQL can lack certain features, such as the full-text search."

MySQL has supported Full-Text search since the dark ages in MyISAM, and full-text search is available for the much more commonly used InnoDB in the current MySQL 5.6 "GA"/stable version (First released 14 months ago in February 2013)

http://dev.mysql.com/doc/refman/5.6/en/fulltext-search.html

---

kamaln7 MOD *March 30, 2014*

0 Thanks! I have updated the broken links.

---

jordan.burke *April 28, 2014*

2 As other's have said JSON support in PostgreSQL is a huge benefit over other RDBMS. The fact that I can throw in a simple JSON object and then query against its property fields using a SQL query makes PostgreSQL immensely powerful.

---

rbmovingforward *July 8, 2014*

1 Just joined D.O. because of this post. Great information for someone looking to get more familiar with databases. Thanks to everyone who commented as well.

---

barry244139 *July 18, 2014*

6 There's a whole lot of things missing from the PostgreSQL benefits section. Just off the top of my head:

Full Text Search

PG has full text search capabilities that rival standalone dedicated search engines like Solr, Sphinx, and Elastic Search. The full text search that exists in MySQL is scary by comparison. If your are only searching data in the database, its the way do go. You don't have to setup and manage another system. You don't have to worry about data syncing issues either or encoding issues transitioning your data from PG to another search engine. It has multiple dictionaries that you can implement all over the place. You can even add your own, weight searchable parts, etc.

Multi Index Queries

This is big and ties into the previous one. By default, PostgreSQL does multi-index queries so when you're creating an index you just add them for individual columns that you will need to search on. You don't need to create an index over multiple fields unless it's there to maintain uniqueness across multiple fields.

Combined with the built in full text search, that means that you can just add the full text search to any where clause and all of the indexes for other items will be hit as well.

Custom data types

PG is extensible and allows the creation of custom data types as well as having 2 types of indexes built for working with those custom data types. JSON and hstore were mentioned earlier and they are the result of this. There are many other custom data types (XML, ISBN, etc) to handle just about any need that you could have...and since PG handles multi-index queries...well, that's awesome.

Functions are amazing

Database functions are great for use in queries or manipulating data that comes back, but PostgreSQL takes them to another level. You can you create an index with the results of a function that will automatically get used when the matching function is used in a where clause. Things like a unique index on a lowercase username come to mind. This is extremely helpful if you're working with XML too because you can't index an XML file...but you can index the result of an XPATH function on the XML datatype.

Wasted space is TOASTed

TOAST is the compression layer behind PG's TEXT field types. It automatically zip's large data. I dropped a 2.2 mb XML file in and it stored in as 81 kb.

Stored procedures aren't painful

You can write stored procedures in other languages including Python and Javascript. This makes putting logic that belongs in your database in there a whole lot simpler.

LISTEN/NOTIFY will change your life

Never have a long running process poll for changes to data again. Long running server processes can now hold a connection and let the database tell them when a change happened. These remove your dependence on framework "AFTER SAVE" hooks, which really comes in handy when you need to connect to your database with another language. If it changes in the database, it changes everywhere.

A few quick use cases for this:

- A Postgres based queuing system that listens for new tasks. Heroku actually built one for Ruby called queue_classic that's available on Github (which they use in production). Also uses logic to find and update the record simultaneously to ensure no 2 workers grab the same job.
- A cache updating system that listens for data changes and pushes them to memcached / redis
- A process that listens for data changes and pushes them to a 3rd party service, statsd, or standalone search engine if needed

PostGIS for Geospatial Data

Adds a bunch of custom datatypes and logic for dealing with geospatial data. Because you can do that with PostgreSQL.

Because of multi-index queries, this means that in one query you can search for basic where conditions, full text, and filter by geospatial parameters like distance. In ONE. DANG. QUERY.

This is off the top of my head. There's a lot more.

---

barry244139 *July 22, 2014*

1 In fact, here's a good write up of Why you should learn PostgresSQL

---

iphoon *August 21, 2014*

0 There is one topic not normally address: using PostgreSQL for analytic or data warehouse. Has anyone experiences to share?

---

barry244139 *August 21, 2014*

0 For Big Data or Analytics with PostgreSQL there are a few options:

PostgreSQL + Cassandra and Hadoop
http://www.bigsql.org/se/

MySQL AND PostgreSQL options here:
https://www.infobright.com/index.php/introducing-iee-postgres-edition/

Many of the columnar databases that you'd use for analytics are built to take data from any sources. If you have enough data (pay per TB) to warrant an specialized analytic system it won't be all that big of a concern. That's generally what ETL is for (moving data from it's origin into an analytic system).

For more normalized datasets, PostgreSQL queries are pretty impressive thanks to it's query optimizer. The ability to do multi-index queries is fairly huge in this regard, but every "big data" solution is totally dependent on your use case.

Load More Comments