10

Optimizing MySQL for Big Data

Once you've optimized your database instance and some time has passed, you may start to notice that the measures you've used to optimize your data for the SQL queries behind it are no longer sufficient. They may start to be insufficient because your data is growing – and as your data is growing, the measures to deal with that growth should grow together with it.

As your data is growing, you will have a lot of things to consider: from disk space and hosting to index and data fragmentation. None of those things are something that cannot be worked on, and all of them can be amended and changed as time goes on. The point here is that amending them is likely to require a lot of time and effort, and that's not even taking into account the monetary costs. Optimizing your database can be a tedious task – and as more and more data is introduced into that database, the task is likely to grow out of proportion if you're not ready for what's to come.

Not everyone has to optimize their database for big data but it's always a good idea to keep optimization techniques at bay should you need them. When problems knock at your door, many engineers and DBAs turn to StackOverflow to ask questions and I know of DBAs who use Secure Notes within password managers such as 1Password to remind themselves of database optimization secrets they find on the web too: doing so might be a great idea if you find yourself making use of password managers to keep your passwords safe; if not, keep this book somewhere within reach and come back to this chapter as your data grows.

Can MySQL Deal with Big Data?

There are a lot of misconceptions about the state of databases – and one of the primary ones is related to relational databases and bigger data sets. Many say that MySQL and its counterparts (MariaDB, Percona Server) aren't a fit for big data sets at all due to them being based on a relational database model, but that's not the problem – improperly optimizing databases for big data is.

Optimizing MySQL for big data is not that different from optimizing MySQL for data sets of an ordinary size; data is still data, there's just a lot of it. Granted, when data sets get bigger, one may have to employ different methods to deal with it, but that doesn't negate the effectiveness of database management systems as such.

The core of the saying "MySQL is not fit for bigger data sets" may stem from the premise that non-relational database management systems don't follow the relational database model, and, as this model is swapped for a relational model followed by MySQL and its counterparts like MariaDB or Percona Server, a pattern emerges – you will hear people say:

- MySQL is only a fit for applications where the ratio between read/write operations is no bigger than 80%/20%.
- NoSQL databases are meant to serve big data by default!
- Only NoSQL databases provide you with a performance benefit for bigger data sets, they're the only option!

While non-relational databases like MongoDB and its counterparts may be of great use for bigger data sets due to their flexible data model and ability to scale out when necessary, the bottom line is that MySQL is an extremely capable database management system and MySQL *can* deal with big data just as well as other database management systems can – it's just our configuration or use case that is the problem. Once your data gets bigger, irrespective of the database vendor, it's all about your server, database configuration, schema, and query design. Don't get me wrong – your database still has to be *optimized* to make use

of the setup of your server – but that doesn't mean that MySQL is the worst option you can pick.

<...>

MariaDB and Big Data – Operations with Big Data Sets

After you've made a choice to use MySQL or MariaDB to support bigger data sets, inevitably you will have to optimize the database management system for big data too. Contrary to popular belief, optimizing MariaDB for bigger data sets is not at all different from optimizing your database management system as a whole. That's why I've already walked you through the methods you can use to optimize your database for performance – once you're working with big data, you're going to have to make use of the settings you've just optimized. There's nothing new here – really, you don't have to invent a bike or a rocket. With that being said, ordinary `INSERT`s won't cut it and there are a few things you must know for your work with bigger data sets to not become an absolute nightmare – that's why we're jumping into methods to help facilitate your work with big data.

Before working with big data, give your data a specific definition: when do you consider your application/database to cross the realm into big data? This question may seem weird, but people have so many definitions for big data that it's close to absurd. For these examples, we will consider big data "big" once it crosses the mark of 100 million records.

Inserting Big Data Into MariaDB

Before you go ahead and insert bigger data sets into MySQL, there are a couple of things you should keep in mind:

1. Most of the time, your data insertion will have nothing to do with `INSERT` queries. This is because `INSERT` queries come with unnecessary overhead.
2. The files you will insert data into your database from will most likely be text files, thus having a smaller size than logical/physical backups.

3. The columns within your data set(s) are most likely to be terminated by the ":", "|", "-" signs, or the comma sign (",") if Excel (CSV) files are in use.
4. You will be able to use your database as your data is inserted into it – your storage engine is unlikely to show the row count at first – the row count will show up after all of the data is inserted.
5. If you need partitions, partition your table before inserting data into it. If you need indexes, it is advisable you index after the data is inserted.

When inserting big data sets into your database, the query you will work with will likely concern the `secure-file-priv` variable too. Make sure the variable is set to a directory that's not readily accessible (i.e. outside `public_html`) and that the directory resides in a hard drive with plenty of space. Space concerns both the file you will be loading the data into your database from, the copies of the table/data your database will possibly make (I'm referring to the internal functionality of `ALTER` queries) and the amount of space required to store all of your precious data in the first place.

Note: The file you will be loading data into your database from will most likely be a text (.txt) or an Excel (.csv) file – not all of the columns in the file may be necessary for your use case either, but the size of text or excel files will be smaller than the size of the table containing your data regardless for various reasons starting from the collation and the character set of your choice and ending up with the partitions and indexes you may elect to use. Make sure you're equipped and ready to sacrifice disk space for speed on big data. *It is going to hurt.*

Summary

Optimizing MySQL or MariaDB for bigger data sets isn't that different of a task than optimizing MariaDB for ordinary use cases. Those who run bigger data sets

within MariaDB have to be aware that it's sometimes not feasible to run certain operations through the database and operations may complete faster if the equivalents of those commands are run on a Linux architecture (the same commands can be run on Windows if we use an appliance like Cygwin.)

Regardless, there are certain things we should be aware of when working with bigger data sets within a relational database management system – but none of those includes throwing MariaDB to the curb in favor of a non-relational database management system. MariaDB, MySQL, and Percona Server are all perfectly capable of crunching big data if we follow best practices. Indexes are one of those best practices – and the bigger your data sets get, the more prevalent they will become. That's what we're getting into now.