Introduction to Databases Checkpoint

wrote by :
Bahia Khaled

If you work with databases, you need to understand the differences between SQL, MySQL, and NoSQL. By knowing how they differ, you can then ensure you use each one effectively at the right time.

Get started by checking out the following helpful guide to SQL, MySQL, and NoSQL.

What Are SQL, MySQL, and NoSQL ?

Structured Query Language, commonly known by the abbreviation SQL, is a programming language that is used to manage data that is held in a relational database management system or for stream processing in a relational database management system.

While SQL is still hugely popular, some argue it is not worth using in the age of IoT.

MySQL is a relational database management system that uses SQL. Whereas SQL is primarily used to query and operate database systems, MySQL enables you to store, handle, delete, and modify data in an organized way.

As for NoSQL, it is a non-relational database that does not use SQL. NoSQL databases use a simpler data structure, meaning they are quicker, more responsive, and more scalable than traditional relational databases.

The Key Differences Between MySQL and NoSQL

By knowing the precise differences between MySQL and NoSQL, you can use each effectively. Both are popular market choices, so it is important that you learn about the differences to find the right option for your needs.

Here are some of the key differences between MySQL and NoSQL.

First off, MySQL is a relational database that is based on a tabular design. NoSQL is non-relational and has a document-based design.

A MySQL database is currently more popular in the market than NoSQL because the latter is still fairly new. That means, at present, MySQL encompasses a large community while NoSQL has a comparatively small community.

While MySQL is not easily scalable due to rigid schema restrictions, NoSQL can easily be scaled because of its dynamic schema nature.

Another key difference is MySQL requires a detailed database model before the creation of the database while NoSQL requires no detailed modeling.

Also, unlike MySQL, which is a type of relational database, NoSQL is more design-based, with examples like CouchDB and MongoDB. Furthermore, NoSQL is much more flexible than MySQL.

One of the good things about MySQL database management is that it is available with a broad range of reporting tools that can help the validity of an application. On the other hand, NoSQL databases do not have reporting tools for performance testing and analysis.

How to Monitor and Analyze Database Performance

To make monitoring and analyzing database performance easier, you should use a database performance analyzer.

It can be utilized for simply monitoring multiple DBMS platforms You can then quickly identify bottlenecks, pinpoint the root causes of those bottlenecks, and prioritize your actions.

You can also highlight issues that are difficult to find by proactively optimizing poor-performing applications. Thus, you can solve issues before they become major problems.

The Takeaway

Whether you use MySQL or NoSQL, or a combination of both, is ultimately up to you. Weigh up the pros and cons of each and identify your own needs to determine which option is right for you.

NoSQL has many advantages, such as being able to perform at a big data level, having flexible designs, and being great for scalability. So, NoSQL is starting to be a real game-changer for the IT market.

However, NoSQL is still a young technology and does not provide the set of standards that MySQL databases have to offer. With MySQL, you can easily access and modify databases and there is a large community that can assist when problems arise.

At the end of the day, the choice of using MySQL or NoSQL is up to you.

Comparison between MongoDB vs MySQL

MySQL is a traditional *Relational Database Management System* (RDBMS) using the *Standard Query Language* (SQL) for data querying and manipulation of the database schema. *MongoDB*, on the other hand, is a document-oriented database, part of the wider class of *NoSQL* non-relational databases, which also includes key-value stores.

NoSQL database systems, such as *MongoDB*, arose from the necessity of handling huge quantities of data with which conventional RDBMS solutions, such as *MySQL*, could no cope. *NoSQL* database systems are used to manage large volumes of data that do not necessarily follow a fixed schema, and will frequently be found in hybrid configurations, alongside a RDBMS, for the portion of the data for which the relational model is a better fit. One commonly found limitation in *NoSQL* database systems versus its SQL counterparts, is that their design limitations, including the fact that the data is partitioned between different servers (for improved performance and due to size limitations), prevent relation style JOIN operations from being used.

While a RDBMS, such as *MySQL* using a suitable storage engine such as InnoDB, will offer ACID (atomicity, consistency, isolation, durability) guarantees, *NoSQL* databases, such as *MongoDB* won't, in general, be able to give full ACID guarantees due to their design. *MongoDB*, however, supports high-performance atomic update operations on single documents, and given a sufficiently long period of time, if no further modifications occur, eventually all updates can be expected to propagate through the system.

On the other hand, trading ACID guarantees for performance and real-time behaviour, allows *MongoDB*, and *NoSQL* systems in general, a degree of freedom which makes possible the implementation of a distributed, fault-tolerant and (horizontally) scalable architecture

(using sharding). Scaling usually is as simple as adding more servers to the mix, and the failure of a server is tolerated. Typical uses cases include: indexing large numbers of documents, serving web pages with dynamic content on high-traffic web sites and streaming media delivery. *MySQL*, and most RDMBSes in general, also offer clustering solutions for fault-tolerance and load-balancing. These however, won't be able to scale or perform to the level of the *NoSQL* solutions.

In terms of how the data is structured, collections in *MongoDB* could be considered akin to tables in *MySQL* and documents could be considered as records (or rows). However, they have some fundamental differences: every record in a relational table has the same sequence of fields, while documents in a collection may have fields that are completely different. Where rows in *MySQL* are simply represented by a tuple of column values, *MongoDB* uses *BSON* (a binary *JSON*-style representation) for its documents. *MongoDB* requires that **all** documents have a **unique** key that represents the document in the database and is used to address it. Contrast this with *MySQL*, and relational databases in general, where a primary key is not mandatory for a table.

Beyond the simple key-document lookup, *MongoDB* also provides a query API, which allows searches by field, range queries and regular expression searches. These queries can also return specific fields of documents instead of the whole document. In contrast with *MySQL* and relational databases, which were designed based on relational algebra, and where SQL allows expressive queries and relationships between tables, *MongoDB* queries are limited to a single collection, and any SQL JOIN like functionality has to be handled by the application (though ORM layers exist to handle this automatically). *MongoDB* offers full index support on any attribute, very similarly to what *MySQL* and relational databases offer for improved query performance.

There are two important features of *MongoDB* which don't have an equivalent in *MySQL*. The first of these is the possibility of using Google's *Map/Reduce* API for batch processing of data and aggregation operations, enabling efficient processing of large data sets. The second

is *GridFS* which allow *MongoDB* to be used as a distributed file system, taking advantage of its load balancing and data replication features over multiple machines, for storing files of any size, without requiring additional components in the application stack.

Last, but not least, both *MongoDB* and *MySQL* are open source products.

Both *MySQL* and *MongoDB*, representing different database paradigms, have use cases where they excel. When choosing between the two, it is important to correctly analyse the problem at hand, and decide which of the two better fits the requirements. As was previously mentioned, many times this will consist in an hybrid approach using both *MongoDB* and *MySQL* for different partitions of the data domain.