SQL vs NoSQL

What is vertical scaling?

**Vertical scaling** refers to increasing the power of a single machine or single server through a more powerful CPU, increased RAM, or increased storage capacity.

What is horizontal scaling?

**Horizontal scaling** allows for near-limitless scalability to handle big data and intense workloads.

Horizontal scaling, also known as scale-out, refers to adding machines to share the data set and load.

This allows for larger datasets to be split into smaller chunks and stored in multiple data nodes, increasing the total storage capacity of the system.

What is nosql?

NoSQL databases (aka "not only SQL") are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model.

The main types:

* document
* key-value
* wide-column
* graph

What are the benefits of NoSQL databases?

NoSQL databases offer many benefits over relational databases. NoSQL databases have flexible data models, scale horizontally, have incredibly fast queries, and are easy for developers to work with.

|  |  |
| --- | --- |
| **Flexible data models** | NoSQL databases typically have very flexible schemas. A flexible schema allows you to easily make changes to your database as requirements change. You can iterate quickly and continuously integrate new application features to provide value to your users faster. |
| **Horizontal scaling** | Most SQL databases require you to scale-up vertically (migrate to a larger, more expensive server) when you exceed the capacity requirements of your current server. Conversely, most NoSQL databases allow you to scale-out horizontally, meaning you can add cheaper commodity servers whenever you need to. **LOWER COST IN THE LONG RUN.** |
| **Fast queries** | Queries in NoSQL databases can be faster than SQL databases. Why? Data in SQL databases is typically normalized, so queries for a single object or entity require you to join data from multiple tables. As your tables grow in size, the joins can become expensive. However, data in NoSQL databases is typically stored in a way that is optimized for queries. The rule of thumb when you use MongoDB is data that is accessed together should be stored together. Queries typically do not require joins, so the queries are very fast. |
| **Easy for developers** | Some NoSQL databases like MongoDB map their data structures to those of popular programming languages. This mapping allows developers to store their data in the same way that they use it in their application code. While it may seem like a trivial advantage, this mapping can allow developers to write less code, leading to faster development time and fewer bugs. |

What are the main 4 types of nosql databases?

**Document** databases store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types including things like strings, numbers, booleans, arrays, or objects.

**Key-value** databases are a simpler type of database where each item contains keys and values.

**Wide-column** stores store data in tables, rows, and dynamic columns.

**Graph** databases store data in nodes and edges. Nodes typically store information about people, places, and things, while edges store information about the relationships between the nodes.

What is mongodb?

MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need.

What are the differences between sql and nosql?

NoSQL (“non SQL” or “not only SQL”) databases focus on scaling, fast queries, allowing for frequent application changes, and making programming simpler for developers. (2000s)

Relational databases accessed with SQL (Structured Query Language) focus on reducing data duplication. SQL databases tend to have rigid, complex, tabular schemas and typically require expensive vertical scaling. (1970s)

What are replica sets?

**Replication** refers to creating copies of a database or database node.

**Replication adds fault-tolerance to a system.** Each node in a cluster contains a copy of the data.

If one of the nodes goes down, the cluster is still able to serve client requests because the other nodes in the cluster can respond to the requests.

Replication is also a form of scaling because client requests can be spread across all the nodes in the cluster instead of overwhelming a single node.

One of the nodes in a replica set is the **primary node**, and the other nodes are secondary nodes. Read requests are distributed between each of the nodes. However, **only the primary node can be written to,** and updates made to the primary node are then replicated to the other nodes.

What is sharding in mongodb?

**Sharding** is a method for distributing or partitioning data across multiple machines.

It is useful when no single machine can handle large modern-day workloads, by allowing you to scale horizontally.

Benefits

|  |  |
| --- | --- |
| **Increased read/write throughput**: | You can take advantage of parallelism by distributing the data set across multiple shards. Let’s say one shard can process one thousand operations per second. For each additional shard, you would gain an additional one thousand operations per second in throughput. |
| **Increased storage capacity**: | Similarly, by increasing the number of shards, you can also increase overall total storage capacity. Let’s say one shard can hold 4TB of data. Each additional would increase your total storage by 4TB. This allows near-infinite storage capacity. |
| **Data Locality**: | Zone Sharding allows you to easily create distributed databases to support geographically distributed apps, with policies enforcing data residency within specific regions. Each zone can have one or more shards. |

What are the two different ways of modelling relationships within mongodb?

MongoDB Relationships are the representation of how the multiple documents are logically connected to each other in MongoDB. The **Embedded and Referenced methods** are two ways to create such relationships.

|  |  |
| --- | --- |
| **Embedded Document Model:** | **Reference Model:** |
| The documents are embedded within one another in this model. For instance, we have two documents: one is a student document (which contains the student's basic information such as id and branch) and the other is an address document (which contains the address of the student). As a result, rather than creating two separate documents, the address documents are embedded within the student document. It will assist a user in retrieving data with a single query rather than a series of queries. | We keep the documents separate in this model, but one document contains the references to the others. For instance, we have two documents: one is a student document (which contains the student's basic information such as id and branch) and the other is an address document (which contains the address of the student). As a result, the id field of the address document is referenced in the student document. We can now query the address and get the student's address using this reference id. The normalised relationships are usually designed using this model. |

What situations is MongoDB good for?

|  |  |
| --- | --- |
| **Integrating large amounts of diverse data** | If you are bringing together tens or hundreds of data sources, the flexibility and power of the document model can create a single unified view in ways that other databases cannot. MongoDB has succeeded in bringing such projects to life when approaches using other databases have failed. |
| **Describing complex data structures that evolve** | Document databases allow embedding of documents to describe nested structures and easily tolerate variations in data in generations of documents. Specialized data formats like geospatial are efficiently supported. This results in a resilient repository that doesn’t break or need to be redesigned every time something changes. |
| **Delivering data in high-performance applications** | MongoDB’s scale-out architecture can support huge numbers of transactions on humongous databases. Unlike other databases that either cannot support such scale or can only do so with massive amounts of engineering and additional components, MongoDB has a clear path to scalability because of the way it was designed. MongoDB is scalable out of the box. |
| **Supporting hybrid and multi-cloud applications** | MongoDB can be deployed and run on a desktop, a massive cluster of computers in a data center, or in a public cloud, either as installed software or through MongoDB Atlas, a database-as-a-service product. If you have applications that need to run wherever they make sense, MongoDB supports any configuration now and in the future. |
| **Supporting agile development and collaboration** | Document databases put developers in charge of the data. Data becomes like code that is friendly to developers. This is far different from making developers use a strange system that requires a specialist. Document databases also allow the evolution of the structure of the data as needs are better understood. Collaboration and governance can allow one team to control one part of a document and another team to control another part. |

What situations is MongoDB not suitable for?

|  |  |
| --- | --- |
| **Transactions** | One of the downsides of MongoDB is that it doesn’t support transactions. Though fewer and fewer applications are requiring transactions, there are still some that need transactions in order to update multiple documents/collections. If that’s a necessary function for your team, MongoDB should not be used. There’s potential for data corruption. |
| **Joins** | Joining documents in MongoDB is no easy task and though 3.2 introduced left-outer joins, developers are still working on the function and it’s not quite mature yet. Pulling data from several collections requires a number of queries, which will inevitably lead to messy code and long turn-around times. |
| **Indexing** | As mentioned earlier, enjoying MongoDB’s quick speeds and high performance is only possible with the right indexes. With shoddily implemented indexes and out of order composite indexes, MongoDB will operate at a shockingly slow speed. |
| **Duplicates** | Some of these downsides could ultimately lead to duplicate data (as has happened for many users in the past). The relationships in MongoDB are not typically well-defined and the resulting duplicate data sets can be hard to handle. That, along with not being ACID compliant, would lead to corrupted data as well. |

What are the differences between the two count methods in mongodb - .count() and .estimatedDocumentCount()?

How does the aggregate method work in mongodb? - include syntax example and pipeline description

In MongoDB, aggregation operations **process the data records/documents and return computed results**. It collects values from various documents and groups them together and then performs different types of operations on that grouped data like sum, average, minimum, maximum, etc to return a computed result.

What are some examples of aggregate pipeline methods?

How do you access embedded fields?

How can we manipulate what fields are returned in a simple mongodb query?

What are some examples of single purpose aggregation methods?

# NoSQL pros cons:

**Advantages of NoSQL**

NoSQL provides the following databases:

1. **Unstructured Schema:** NoSQL provides an easy way to store the data as the structure of the database is not restricted. It can be modified as needed with no limitation. For the applications where the data structure is not final, such as network data, voice, or images, NoSQL provides an efficient way to store/retrieve and process the data.
2. **Scalability:** Need for NoSQL was needed because the volume by which data generated was larger than ever. Moreover, data today does not compromise only of text, rather binary data such as images, videos, sounds were becoming common. NoSQL was introduced to improve the scalability of the databases. NoSQL allows the scaling of the database horizontally by adding more servers.
3. **Economical to Use:** NoSQL databases are relatively cheaper to install and manage rather than traditional SQL servers. This is because NoSQL does not require licensed servers and data centers to operate bringing down the cost of the application and maintenance.

**Disadvantages of NoSQL**

While NoSQL provides benefits, it is necessary to consider the disadvantages of adopting it. No solution is final or perfect rather it depends on scenario.

1. **Immature and Less Support:** NoSQL is recently introduced. NoSQL does not have as vast a community as SQL. Similarly, it lacks features needed, which are present in traditional SQL databases.
2. **Lack of Analytics:** NoSQL is designed to tackle large amounts of data having varying nature. However, when it comes to analytics and drawing business insights, NoSQL is not as efficient as SQL. This again revives the SQL. Today applications usually adopt both the models for varying nature of the use case.
3. **Lack of Consistency:** NoSQL suffers from consistency when it comes to storing large amounts of data. SQL databases provide something called ACID transactions (**A**tomic, **C**onsistent, **I**solated, **D**urability) meaning that guarantee of the reliable transactions is present. NoSQL lacks this, compelling programmers to write their custom code for this.

|  | **SQL Databases** | **NoSQL Databases** |
| --- | --- | --- |
| Data Storage Model | Tables with fixed rows and columns | Document: JSON documents, Key-value: key-value pairs, Wide-column: tables with rows and dynamic columns, Graph: nodes and edges |
| Development History | Developed in the 1970s with a focus on reducing data duplication | Developed in the late 2000s with a focus on scaling and allowing for rapid application change driven by agile and DevOps practices. |
| Examples | Oracle, MySQL, Microsoft SQL Server, and PostgreSQL | Document: MongoDB and CouchDB, Key-value: Redis and DynamoDB, Wide-column: Cassandra and HBase, Graph: Neo4j and Amazon Neptune |
| Primary Purpose | General purpose | Document: general purpose, Key-value: large amounts of data with simple lookup queries, Wide-column: large amounts of data with predictable query patterns, Graph: analysing and traversing relationships between connected data |
| Schemas | Rigid | Flexible |
| Scaling | Vertical (scale-up with a larger server) | Horizontal (scale-out across commodity servers) |
| Multi-Record ACID Transactions | Supported | Most do not support multi-record ACID transactions. However, some — like MongoDB — do. |
| Joins | Typically required | Typically not required |
| Data to Object Mapping | Requires ORM (object-relational mapping) | Many do not require ORMs. MongoDB documents map directly to data structures in most popular programming languages. |