**NoSQL and MongoDB**

What is nosql?

What are the main 4 types of nosql databases?

What is mongodb?

What are the differences between sql and nosql?

What are replica sets?

A replica set is **a group of mongod instances that maintain the same data set.** A replica set contains several data bearing nodes and optionally one arbiter node. Of the data bearing nodes, one and only one member is deemed the primary node, while the other nodes are deemed secondary nodes.

What is sharding in mongodb?

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?