|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Atomicity** | **Sharding** | **Indexes** | **Large number of**  **collections** | **Collection contains large number of small documents** |
| Arrays of  Ancestors | x |  | x | x |  |
| Materialized  Paths |  | x |  | x |  |
| Nested  Sets |  |  | x |  | x |

Array of Ancestors - Atomicity: The atomicity factor is applicable in this case since each of the nodes refer to their immediate parent and atomicity is a series of database operations where all of them occur or none of them occur, so we always want to update the whole tree rather than just the single node because then we might have ambiguous database relationships between parents/children.

Array of Ancestors - Indexes: The index factor allows us to give an unique field to each of the nodes by which we can reference it. MongoDB will limit the number of nodes it must inspect if we specify the exact id we want to query by.

Array of Ancestors - Large number of collections – Since the array of ancestors basically looks like a tree with nodes as well as an array with all of the node's ancestors, it would be optimal to use in a case where we have large number of collections because they would be easily accessible by going through their ancestors field.

Materialized Paths - Sharding: The reason why sharding is useful in the Materialized Paths model is because we can spread out our data onto multiple different machines/servers and still query for specific results because except the id, Materialized Paths model stores node's ancestors or path which we can then use to find nodes by partial paths. We don't have to have the full overview of the dataset in order to be able to make queries, the only thing we need is the node's path.

Materialized Paths - Large number of collections: Large number of collections do not really matter in Materialized Paths since they are all closely related with each other by having their paths stored. It is still relatively easy to look through a collection and get a quick query response.

Nested Sets - Indexes: If you do not have have an index for every node in Nested Sets model, you won't be able to really know which part of the tree was visited and which part of the tree has yet to be visited since the whole model is based on a round-trip traversal of the tree.

Nested Sets – Collection contains large number of small documents: This model is very static, not good when it comes to mutability and also has an easier time for finding subtrees.