Session 13 &14: Introduction to MongoDB

MongoDB is an open-source database that uses a document-oriented data model and a non-structured query language. It is one of the most powerful NoSQL systems and databases around, today.

- it does not use the usual rows and columns that you so much associate with relational database management.
- It is an architecture that is built on collections and documents.

Topic 1: What are MongoDB Internals

a. Every document that you save must have an "_ID" field. (Spelled exactly '_ID'). If you don't specify an ID, Mongo will assign you one.
b. The rest of the fields or document structure is up-to you. Mongo imposes no other rules
c. The size of a document is limited to 16 MB. If you need more size the document will have to be split across as multiple documents.

key aspects of MongoDB internals:

1. **Storage Engine**:

   MongoDB uses a pluggable storage engine architecture. The default storage engine used to be MMAPv1, but as of MongoDB 4.0, the default is WiredTiger. WiredTiger offers better performance, compression, and concurrency control.

   The storage engine is responsible for managing how data is stored on disk, memory management, and handling read/write operations efficiently.

2. **Data Representation**:

   Data in MongoDB is stored in collections, which are analogous to tables in relational databases. Each collection contains documents, which are JSON-like data structures.

   BSON (Binary JSON) is the binary-encoded format used by MongoDB to serialize and store documents. BSON extends the JSON model to include additional data types and optimizations for efficient storage and retrieval.

3. **Indexes**:

   MongoDB uses indexes to support efficient query execution. Indexes are data structures that store a subset of the data in a more organized format to speed up data retrieval operations.

Common types of indexes in MongoDB include single-field indexes, compound indexes (indexes on multiple fields), and text indexes.

4. **Concurrency Control**:

   MongoDB employs concurrency control mechanisms to ensure data consistency in multi-user environments. WiredTiger uses a combination of optimistic concurrency control and multiversion concurrency control (MVCC) to manage concurrent read and write operations.

5. **Replication**:

   MongoDB supports replica sets, which are clusters of MongoDB instances that maintain copies of the same data. Replica sets provide high availability and fault tolerance by automatically electing a primary node for writes and handling failover scenarios.

6. **Sharding**:

   Sharding is a horizontal scaling technique used by MongoDB to distribute data across multiple servers. It involves partitioning data into shards based on a shard key and distributing these shards across different nodes in a cluster. Sharding allows MongoDB to handle large volumes of data and high write loads.

7. **Journaling and Durability**:

   MongoDB uses a write-ahead logging (WAL) mechanism called journaling to ensure data durability. Journaling records write operations in a journal file before applying them to the data files, which helps recover data in case of crashes or failures.

8. **Query Execution**:

   MongoDB's query execution engine processes queries and selects an optimal query plan based on the query's criteria, indexes, and available resources. The query optimizer plays a crucial role in generating efficient query plans to minimize response times.

Topic 2: Essential Concepts behind a Database Index

The process of creating additional data structures that enable efficient data retrieval.

Indexes are data structures that support the efficient execution of queries in MongoDB. They contain copies of parts of the data in documents to make queries more efficient. Without indexes, MongoDB must scan every document in a collection to find the documents that match each query.

There are many indexing data structures used in NoSQL databases.

- **B-Tree indexing**
- **T-Tree indexing**
- **O2-Tree indexing**

primary indexes are automatically created on the primary key or identifier of each record and are used for direct lookups by key. Secondary indexes are manually created on other attributes or fields of the records and can be used for filtering or sorting.

Creating an Index :

MongoDB provides a method called createIndex() that allows user to create an index.

Syntax: db.COLLECTION_NAME.createIndex({KEY:1})

The key determines the field on the basis of which you want to create an index and 1 for ascending or -1 for descending.

Example –

db.cdac.createIndex({“age”:1})

{

“createdCollectionAutomatically” : false,

“numIndexesBefore” : 1,

“numIndexesAfter” : 2,

“ok” : 1

}

1 simple Index

db.college.createIndex({ age: 1 }, { unique: true })

If you wanna check your index whether it has been created successfully.

db.college.getIndexes()

2 Compound Index

If you want to create an index on multiple fields at once. suppose age and gender.

db.teachers.createIndex({ age: 1, gender: 1 },{ unique: true })

3 Partial Index

If you want to create indexing on certain conditions like if the age is above 40 then create an index.

db.collection.createIndex(

{ age: 1 },

{ partialFilterExpression: { age: { $gt: 40 } } }

)

We should avoid creating indexing when our data is.

1 when data is small

2 Frequent write updates: If your database is write heavy we should try to avoid index as it will slow down the query because those index needs to be updated too.

3 Make sure the field you are creating is indexed. should be a proper field which can make your query faster.

Drop an index:

In order to drop an index, MongoDB provides the dropIndex() method.

A.dropIndex()

Syntax – db.NAME_OF_COLLECTION.dropIndex({KEY:1})

The dropIndex() methods can only delete one index at a time. In order to delete (or drop) multiple indexes from the collection, **dropIndexes()** will be used.

B.dropIndexes()

Syntax – db.NAME_OF_COLLECTION.dropIndexes({KEY1:1, KEY2: 1})

C.Get description of all indexes:

The getIndexes() method in MongoDB gives a description of all the indexes that exists in the given collection.

Syntax – db.NAME_OF_COLLECTION.getIndexes()

MongoDB – Index Types

MongoDB is a NoSQL document type database that follows indexing. Indexes make searching in a collection is easier with the limited number of documents. A **binary tree** is the data structure used by an index.

In documents, the **_id** field is a default index which is automatically created by MongoDB and we are not allowed to drop this index.

Types of index

MongoDB provides different types of indexes that are used according to the data type or queries. The indexes supported by MongoDB is are:

1. Single field Index: A single field index means index on a single field of a document. This index is helpful for fetching data in ascending as well as descending

Syntax: db.students.createIndex({"<fieldName>" : <1 or -1>});

- 1 represents the field is specified in ascending order and -1 for descending order.

1. Indexing and Ordering in MongoDB
2. Creating and Using Indexes in MongoDB

2. Compound Index: We can combine multiple fields for compound indexing and that will help for searching or filtering documents in that way.The compound index is an index where a single index structure holds multiple references.

Syntax: db.<collection>.createIndex({ <field1>: <type>, <field2>: <type2>, … })

Example: create a compound index on studentAge: 1, studentName:1

db.students.createIndex({studentAge: 1, studentName:1})

3. Multikey Index: MongoDB uses the multikey indexes to index the values stored in arrays. When we index a field that holds an array value then MongoDB automatically creates a separate index of each and every value present in that array.

4. Geospatial Indexes: MongoDB provides two geospatial indexes:

- **2d indexes:** The 2d indexes support queries that are used to find data that is stored in a two-dimensional plane. It only supports data that is stored in legacy coordinate pairs.
- **2d sphere indexes**: 2d sphere indexes support queries that are used to find the data that is stored in spherical geometry. It supports data that is stored in legacy coordinate pairs as well as GeoJSON objects.

  Syntax of 2d sphere indexes:

  db.<collection>.createIndex({ <Locationfield>: "2dsphere"})

5. Text Index: Text index allows us to find the string content in the specified collection. It can include any field that contains string content or an array of string items. A collection can contain at most one text index.

Syntax: db.<collection>.createIndex({ <field>: "text"})

6. Hash Index: This kind of index is mainly required in the even distribution of data via sharding. Hashed keys are helpful to partition the data across the sharded cluster.

Syntax: db.<collection>.createIndex({ _id: "hashed" })

Note: From Version 4.4 onwards, the compound Hashed Index is applicable

7. Wildcard Index: MongoDB supports creating indexes either on a field or set of fields and if the set of fields are mentioned, it is called as Wildcard Index. Generally, the wildcard index does not include _id field but if you what to include _id field in the wildcard index then you

have to define it explicitly. MongoDB allows you to create multiple wildcard indexes in the given collection. Wildcard indexes support queries for unknown or arbitrary fields.

Syntax: To create a wild card index on the specified field:

db.<collection>.createIndex({ “field.$**”:1 })