Mongo DB

# Querying Data from MongoDB

BSON - MongoDB represents JSON documents in binary form called BSON internally. It enriches JSON with additional data types that Mongo uses like Object Id, date, etc.

Mongo DB is schemaless. Mongo does not enforces a schema, but documents inside the same collection should have a similar structure for consistency.

mongo "localhost:27017"

db, show dbs, use db\_name, show collections, db.aircraft.count(), db.aircraft.find(), db.aircraft.pretty()

Mongo GUI - Mongo Compass, Robo 3T

The find() method returns a cursor to the documents that match the query criteria. In projection, 1 means inclusion and 0 for exclusion.

db.aircraft.find({}, {model: 1, range:1, \_id: 0})

Cursor - A virtual object where MongoDB stores the documents returned by the find method. It can have any methods like below -

* db.aircraft.find().pretty()
* db.aircraft.find().count()
* db.aircraft.find().skip(3)
* db.aircraft.find().limit(5)
* db.aircraft.find().sort({model: 1})

Mongo DB does not guarantee the order of the returned documents unless sort() method is used. 1 for the ascending order and -1 for descending order.

The find() method returns a cursor, however the findOne() method returns an actual document. If no document match the criteria, the method returns null.

Best Practice - We should pass an empty object to find() method when we want to return all the documents instead of using blank find() method -

db.aircraft.find({})

Comparison query operations - $eq, $ne, $in, $nin, $lt, $lte, $gt, $gte

* db.aircraft.find({model: 'Boeing'})
* db.aircraft.findOne({model: 'Boeing'})
* db.aircraft.findOne({model: {$ne: 'Boeing'}})
* db.aircraft.findOne({model: {$in: ['Boeing', 'Airbus]}})
* db.aircraft.findOne({model: {$in: [/^A/]}}) // using regular expression

Geo Spatial data types - GeoJSON, Legacy

Geospatial queries - $near - filters documents where a location field is between a min and max value in meters from a specified geometry.

Logical Query Operators - $and and $or.

* db.aircraft.find({$and:[{capacity: 124}, {range: {$gt: 6000}}]})
* db.aircraft.find({range: {$lt: 600, $gt: 6000}}) // short-hand syntax if same field

Nested document query -

db.crew.find({"address.city": "berlin"})

Querying null or missing fields, and field types -

* db.crew.find({address: null}).pretty();
* db.crew.find({address: {$exists: false}}).pretty();
* db.crew.find({address: {$type: "object"}}).pretty();

BSON types - double, string, object, array, boolean, date, 64-bit integer

Free text search - the Text index support fast text searches on strings and arrays of strings fields. We cannot perform free text searches without a text index.

db.crew.createIndex({name: "text", skills: "text"})

db.crew.find({$text: {$search: "technical Anna"}})

Sorting by relevance - we can aggregate results by score using the $meta projection operator.

db.crew.find({$text: {$search: "technical Anna"}}, {score: {$meta: "textScore"}}).sort({score: {$meta: "textScore"}})

Array query operators - $all, $size, $elemMatch

* db.crew.find({skills: {$all: ["technical", "sales"]}}) // order does not matter
* db.crew.find({skills: {$size: 2}}) // which have skills array size as 2
* db.crew.find({skills: {$elemMatch: {name: "flying", lvl: {$gt: 7}}}}) // supports multiple conditions

Array projection operators - use to determine which elements of an array should be returned or hidden from a query. Operators - $slice, $, $elemMatch

* db.crew.find({}, {skills: {$slice: 1}})
* db.crew.find({}, {skills: {$slice: [1,1]}}) // skip n element and pick m element
* db.crew.find({skills: "management"}, {skills.$: 1}) // show first element that matches an array query
* db.crew.find({}, {skills: {$elemMatch: {lvl: {$gt: 7}}}}) // display only the first element to match the projection condition

# Foundations of Document Databases with MongoDB

NoSQL - Not Only SQL

CAP Theoram - consistency, availability, partition tolerance (system won't fail). MongoDB supports consistency and partition tolerance.

Types of NoSQL databases - relational database, document database, key-value databases (Redis DB), white-column stores (Cassandra DB), Graph Database

MongoDB is a document database.

Why use MongoDB - open source, document database, high performance, rich query language, high availability, horizon scalability.

JSON is a UTF-8 String but BSON is a Binary. JSON is human and machine readable but BSON is a machine readable only.

Create operation - if the collection does not currently exist, insert operation will create the collection.

* db.createCollection("collection\_name")
* db.newUsers.insertOne({name: "dummy user", email: "dummy@abc.com})
* db.newUsers.insertMany([{name: "dummy user", email: "dummy@abc.com}, {name: "dummy2 user", email: "dummy2@abc.com}])
* db.newUsers.find({})

Read Concern for find() operator - allows to control the consistency and isolation properties of the data read from replicate sets and replica set shards. Types - Local, Available, Majority (default), Linearizable, Snapshot.

Write concern - level of acknowledgement requested from MongoDB for write operations. Types - w:1 (Ack from primary), w:0 (No ack), w:n (primary + (n-1) secondary), w: (majority), wtimeout (time limit to prevent write operations from blocking indefinitely)

We cannot replace the \_id value with another value.

$set creates field if not already existing.

By using upsert as true, if no document matches the query to update, then new document will be inserted.

* db.collection.updateOne({title: {$eq: "ABC"}}, {$set: {title: "XYZ"}})
* db.collection.updateMany({year: {$eq: 1988}}, {$set: {"year": "2025"}}, {upsert: true}) // if no match found, then insert 1988 and then update it with 2025
* db.collection.replaceOne({runtime: {$eq: 1122}, {runtime: 1122, title: "no title"}}) // replace whole document unlike just the mentioned fields
* db.collection.deleteOne({runtime: 25})
* db.collection.deleteMany({runtime: 25})
* db.collection.remove({}) // remove entire documents form the collection
* db.collection.remove({runtime: 35}, true) // remove first matched document, if we do not pass true then all matched document will be removed

MongoDB is a case-sensitive language. It supports MongoDB query language (MQL). It has dynamic JSON based schema unlike predefined in MySQL. It has no foreign keys or joins or triggers. It follows CAP theorem not ACID properties. It is horizontal scalable unlike SQL which have vertical scalable.

Use $lookup operator to get documents from other collection.

Creating index -

db.collection.createIndex({name: 1, age: -1}) // 1 for ascending

Which is best database - is your data structured or unstructured? preferred scalability strategy? No need for ORM (object relational mappings) with MongoDB.

# Introduction to MongoDB 2

There is no schema enforcement from MongoDB, it is a application responsibility.

Replicate sets - Minimum replica sets in MongoDB - primary db (only writable instance), secondary db (read-only instances, data is replicated from primary db) and arbiter db (no data, provides additional vote to elect the db in-case if primary gets failed).

By using the BSON and memory mapped files concept to store the data, makes the MongoDB very fast and efficient.

The \_id can contains any type of data except array.

The ObjectId() will return a new object id and it also contains a timestamp ObjectId().getTimestamp() in ISODate format. We can use it for sorting in created by format.

Operators - $set, $unset, $rename, $push, $pull, $pop

MongoDB uses a cursor to support the efficient retrieval of the document as documents might be in huge number which might support the memory to load. By this, it will give you a batch of documents and close the connection.

We can use index to speed up the queries. The scanning of each location on the disk is bad for performance which finding a record, the solution is to use an index, it basically holds mapping to those locations from field values. By this, we can jump directly to the disk location where document is stored is good for performance and less i/o operations. It also good for sorting performance purpose as-well. Without index it will show n number of objects has been scanned, after index it will show 1 number od object has been scanned. Index on the \_id field can not be dropped.

Index types - Regular (B-tree), Geo (sort nearby locations), Text, Hashed, TTL (expiration date for documents)

* db.foo.ensureIndex(keys, options)
* db.system.indexes.find({ns: 'test.animals'}, {key:1}) // to find if index exists
* db.animals.dropIndex("name\_1")

To know about how MongoDB will find the document, we can use explain() method -

db.animals.find({name: 'cat}).explain()

Covering index - we use query using index and all the information is with index itself then there is no need to go to the dist to get the actual document, we can use the index itself to the actual result. In the explain() it will say indexOnly as true. Suppose we have applied index on name field and returning only the name field, not even \_id, then the covering index will be used.

When our database is large, we can create index in the background so that read and write action won't get blocked.

db.animals.ensureIndex({tags: 1}, {background: true})

We can also create Unique index type, \_id is already have a unique index. It won't allow to insert the same value in the document.

# Using MongoDB with Node.js

Loading the data

Graphical user interface, text, application, email

Description automatically generated