## MongoDB Study Notes

MongoDB: General purpose document database. Structures data as documents which are similar to JSON objects.

### Structure

Document: Basic unit of data in MongoDB. They are displayed in JSON format but stored internally in a BSON (Binary JSON) format. BSON supports more data types like Dates. The values in a document can be any data type, including strings, objects, arrays, booleans, nulls, dates, ObjectIds, and more. Documents look like following

{

\_id:< ObjectID >  
<key>:<value>  
<key>:<value>  
<key>:<value>

}

ObjectID: Special data type used in MongoDB to create unique identifiers. Every document has a unique ID that acts as a primary key, called “\_id”. ObjectId values are 12 bytes in length, consisting of:

* A 4-byte timestamp, representing the ObjectId's creation, measured in seconds since the Unix epoch. We can access creation time of the ObjectID by using ObjectId.getTimestamp()
* A 5-byte random value generated once per process. Random value is unique to machine & process.
* A 3-byte incrementing counter, initialized to a random value.

Collection: Grouping of documents. Documents in a collection mostly have the same format but they need not to exactly similar as MongoDB has a flexible Schema model (polymorphism). Documents in a collection can have different fields and fields can have different data types.

Database: A container of many collections.

Optional Schema Validation: Set constraints on the structure of documents. But this is optional. This is for more control on the schema.

### Data

BSON Date is a 64-bit integer that represents the number of milliseconds since the Unix epoch (Jan 1, 1970). This results in a date range of about 290 million years into the past and future. It uses UTC date time.

Timestamps: internal timestamp type is a 64 bit value. The most significant 32 bits are a time\_t value (seconds since the Unix epoch), the least significant 32 bits are an incrementing ordinal for operations within a given second.

Arrays are an important data type. We need to be careful to avoid bloating of documents due to big arrays.

### Ecosystem

MongoDB Shell (mongosh), is a Node.js REPL (Read Evaluate Print Loop) environment that allows users to interact with MongoDB deployments. As it's built on top of Node.js REPL, so it gives users access to the Node.js API and npm packages. So it gives access to JS variables, functions, conditionals, loops and control flow inside the shell.

Atlas: A multi cloud database service that helps build deploying, managing applications on data (a developer data platform). Replication and sharding is handled by Atlas.

Compass: is a GUI that allows querying, composing pipelines and analyzing data

Charts: a data visualization tool offered by MongoDB

Data Lake: to query, transform, and move data across Amazon S3 and Atlas clusters

Organization: Allows to group and define users and teams.

Projects: Help to organize and define resources. Create separate projects for development, testing and production environments.

Cluster: a group of connected MongoDB instances, or nodes, that work together to provide redundancy, high availability, and scalability for handling large volumes of data

Atlas CLI is a unified command line interface to manage MongoDB Atlas, including Atlas Search and Vector Search

MongoDB drivers are used to connect, query and update MongoDB databases from programming languages such as Python, Node.JS, C#, Java and others. They run database operations on behalf of client applications. For Python drivers are called

1. Pymongo: For synchronous applications
2. Motor: For asynchronous applications
3. PyMongoArrow is a PyMongo extension for loading MongoDB query result sets as Numpy array or Panda Dataframes.

### Connect to a Database

Connection String: Allows us to connect to a cluster on Atlas from Mongo Shell, Mongo Compass or any other application. We need only one MongoClient instance per Atlas cluster for an application. Having more than one MongoClient instance for a single Atlas cluster will cause unnecessary connections to the database, increasing costs and impacting performance. It comes in 2 types (standard and DNS)

1. Standard format: connect to standard clusters, replica sets or sharded clusters. For localhost servers we use this.
2. DNS seed format: A DNS server list, gives flexibility in deployment and server rotation without reconfiguring clients
3. Format for Atlas deployment is mongodb+srv://<userName>:<password>@<clusterName>.<portNumber>.mongodb.net/
4. In above, mongodb is the required prefix that identifies this as a standard connection string
5. we can specify connection options. Each option is separated by a ?, followed by the <name>=<value> pair option

### Data Operations

#### Inserting

db.collection.insertOne(oneDocument)

In case collection is not already created, Mongo will automatically create a collection. In case –id field is not mentioned in document, Mongo will create a unique \_id

db.collection .insertMany([array of comma seperated documents])

#### Finding

db.collection.find({field:”value”} / { field: {$operator:[value or array of values]} })

$in operator will check if field is equal to any one of the inputs in the array provided.

Comparison operators are $gt, $gte, $lt, $lte, $ne

Nested fields are accessed using dot – “field.nestedField” within quotations

$elemMatch() is used to find documents that have a particular value in a array, that matches all the specified query criteria. { <field>: { $elemMatch: { <query1>, <query2>, ... } } }

db.collection.find({field: {$elemMatch: value }})

Logical operators are $or and $and; they are mentioned as { operator:[ array of {field: value}]} }   
e.g. {$or: [{"item.name":"pens"}, {tag: "writing"}]}

We can also have a $and query implicitly by having comma separated conditions without {}.   
E.g. find({name:"pens", "tag": "writing"})

#### Replacing

Replace an entire document. The replaceOne() method takes the following parameters:

1. filter: A query that matches the document to replace.
2. replacement: The new document to replace the old one with.
3. options: An object that specifies options for the update.

This method returns a document containing an acknowledgement of the operation, a matched count, modified count, and an upserted ID

#### Updating

The updateOne(filter, updates, options) method accepts a filter document, an update document, and an optional options object. findAndModify({query: {}, update: {}, new = true,options) method updates and returns the updated document to us.

The $set operator replaces the value of a field with the specified value. If the field is not there, it will add this new field to the document. $unset removes a field from document.

The $push operator adds a new value to an array field. If the array is not there it will create a new one. Use the $each modifier with the $push operator to append multiple elements to an array, such as { $push: {field: { $each: ["element1", "element2"]}}

$pop Removes the first or last item from an array. You can specify 1 to remove the last item or -1 to remove the first item. { $pop: { arrayField: 1 } }

$addToSet Adds a value to an array only if the value does not already exist in the array (avoids duplicates).

$rename renames a field

A number can be incremented by the $inc operator { $inc: { counter: 1 }}. $mul multiples,

{upsert: true} is an option that creates a new document if no documents match the filtered criteria.

### Data Modeling

How data is stored and the relationships between different entities in the data. Schema is the organization of data inside a DB. These can be decided on the basis of

1. what the application does: Structure data such that it matches the way the application queries and updates data.
2. what data is to be stored: Data relationships between sets are 1:1, 1:many (modeled as Array) and many:many.
3. how will users access data (access patterns): Data that’s accessed together is to be stored together.
4. what data is most valuable.

Data Relationships are modeled using

1. Embedding: Related data are inserted into a document. Avoids joins and one single query to update and access data into/from a record. This creates larger documents (unbounded documents maybe created as array is a valid data type) which can slow performance as document is fully loaded to memory while accessing. BSON document threshold is 16 MB. Embedding many:many relationships result in data duplication.
2. Referencing (linking or data normalization): Reference to a document in another collection. References store the \_id field of one document in another document creating a unidirectional link between the two. This reduces data duplication and makes documents smaller but reduces read performance and uses more resources as we need to join data from multiple collections.

Common data modeling errors (Schema anti-patterns)

1. Massive arrays
2. Large number of collections
3. Bloated Documents
4. Unnecessary indexes
5. Queries without indexes
6. Data accessed together but kept in different collections

**ObjectId('6286809e2f3fa87b7d86dccd')**

## Resources

### Lesson 01: Introduction to Data Modeling

* [Data Modeling Introduction](https://www.mongodb.com/docs/manual/core/data-modeling-introduction/?_ga=2.60935781.810066485.1665291537-836515500.1666025886)
* [Separating Data That is Accessed Together](https://www.mongodb.com/developer/products/mongodb/schema-design-anti-pattern-separating-data/?_ga=2.60935781.810066485.1665291537-836515500.1666025886)

### Lesson 02: Types of Data Relationships

* [Data Model Design](https://www.mongodb.com/docs/manual/data-modeling/schema-design-process/)
* [Model Relationships Between Documents](https://www.mongodb.com/docs/v4.2/applications/data-models-relationships/?_ga=2.19332209.810066485.1665291537-836515500.1666025886)
* [Embedding MongoDB](https://www.mongodb.com/basics/embedded-mongodb?_ga=2.19332209.810066485.1665291537-836515500.1666025886)
* [MongoDB Schema Design Best Practices](https://www.mongodb.com/developer/products/mongodb/mongodb-schema-design-best-practices/?_ga=2.19332209.810066485.1665291537-836515500.1666025886)

### Lesson 03: Modeling Data Relationships

* [Data Model Design](https://www.mongodb.com/docs/manual/data-modeling/schema-design-process/)
* [Model Relationships Between Documents](https://www.mongodb.com/docs/v4.2/applications/data-models-relationships/?_ga=2.19332209.810066485.1665291537-836515500.1666025886)

### Lesson 04: Embedding Data in Documents

* [Embedding MongoDB](https://www.mongodb.com/basics/embedded-mongodb?_ga=2.19332209.810066485.1665291537-836515500.1666025886)
* [Model One-to-One Relationships with Embedded Documents](https://www.mongodb.com/docs/manual/tutorial/model-embedded-one-to-one-relationships-between-documents/?_ga=2.19332209.810066485.1665291537-836515500.1666025886)
* [Model One-to-Many Relationships with Embedded Documents](https://www.mongodb.com/docs/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/?_ga=2.19332209.810066485.1665291537-836515500.1666025886)

### Lesson 05: Referencing Data in Documents

* [Normalized Data Models](https://www.mongodb.com/docs/manual/data-modeling/#references)
* [Model One-to-Many Relationships with Document References](https://www.mongodb.com/docs/manual/tutorial/model-referenced-one-to-many-relationships-between-documents/?_ga=2.64006886.810066485.1665291537-836515500.1666025886)

### Lesson 06: Scaling a Data Model

* [Operational Factors and Data Models](https://www.mongodb.com/docs/manual/core/data-model-operations/?_ga=2.64006886.810066485.1665291537-836515500.1666025886)
* [Performance Best Practices: MongoDB Data Modeling and Memory Sizing](https://www.mongodb.com/blog/post/performance-best-practices-mongodb-data-modeling-and-memory-sizing?_ga=2.64006886.810066485.1665291537-836515500.1666025886)

### Lesson 07: Using Atlas Tools for Schema Help

* [A Summary of Schema Design Anti-Patterns and How to Spot Them](https://www.mongodb.com/developer/products/mongodb/schema-design-anti-pattern-summary/?_ga=2.64006886.810066485.1665291537-836515500.1666025886)

### Lesson 01: Using MongoDB Python Client Libraries

* [MongoDB Drivers](https://www.mongodb.com/docs/drivers/)
* [PyMongo Documentation](https://www.mongodb.com/docs/drivers/pymongo/)

### Lesson 02: Connecting to an Atlas Cluster in Python Applications

* [Using PyMongo with MongoDB Atlas](https://pymongo.readthedocs.io/en/stable/atlas.html)
* [Get Connection String](https://www.mongodb.com/docs/guides/atlas/connection-string/)
* [Connection String URI Format](https://www.mongodb.com/docs/manual/reference/connection-string/)
* [Set Up Your Environment](https://www.mongodb.com/developer/languages/python/python-quickstart-crud/?_ga=2.148314039.1652794197.1722098604-695756192.1721270238&_gac=1.183249364.1721754794.CjwKCAjwqf20BhBwEiwAt7dtdRru60NBhLTaHYBd2Jbbg__2WKxt7TEqM6-tpyWrrg-p3L_KsQ_kjRoCcqAQAvD_BwE#set-up-your-environment)

### Lesson 03: Troubleshooting a MongoDB Connection in Python Applications

* [Troubleshoot Connection Issues](https://www.mongodb.com/docs/atlas/troubleshoot-connection/)

#### Lesson 1 - Inserting Documents

* [MongoDB Docs: insertOne()](https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/)
* [MongoDB Docs: insertMany()](https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/)

#### Lesson 2 - Finding Documents

* [MongoDB Docs: find()](https://docs.mongodb.com/manual/reference/method/db.collection.find/)
* [MongoDB Docs: $in](https://docs.mongodb.com/manual/reference/operator/query/in/)

#### Lesson 3 - Finding Documents Using Comparison Operators

* [MongoDB Docs: Comparison Operators](https://docs.mongodb.com/manual/reference/operator/query-comparison/)

#### Lesson 4 - Querying on Array Elements

* [MongoDB Docs: $elemMatch](https://docs.mongodb.com/manual/reference/operator/query/elemMatch/)
* [MongoDB Docs: Querying Arrays](https://docs.mongodb.com/manual/tutorial/query-array-of-documents/#combination-of-elements-satisfies-the-criteria)

#### Lesson 5 - Finding Documents Using Logical Operators

* [MongoDB Docs: Logical Operators](https://docs.mongodb.com/manual/reference/operator/query-logical/)