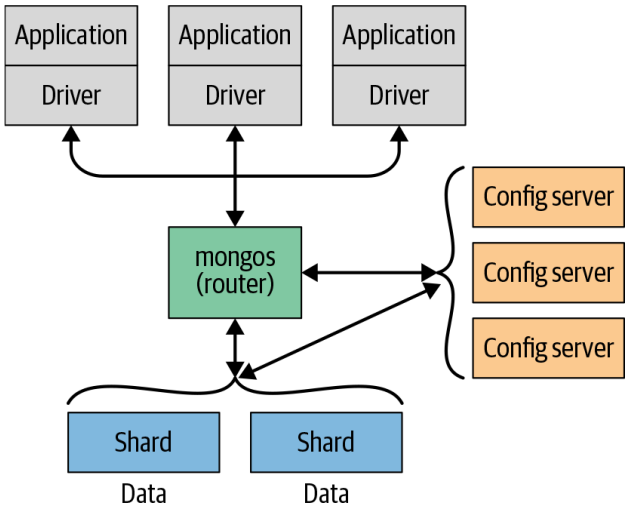
MongoDB Notes

MongoDB is a No-SQL (Not Only SQL) database that is document-oriented without a pre-defined schema.

Main structure of MongoDB:

1. Database: Organized group of collections
2. Collection: Organized group of documents with common fields (like tables)
3. Document: Method to organize and store data as a set of field-value pairs (like rows)



Unlike SQL, MongoDB is built for scaling-out by adding on additional nodes onto a cluster for adding storage and increasing read-write throughput.

Advantages and disadvantages of using MongoDB:

Advantages	Disadvantages
Saves time from pre-defining schema	High memory required
Easy to scale	Document size has limit of 16mb.
Easy to store arrays and objects	
Higher performance than SQL due to use of internal memory for storage	
High availability and fault-tolerant	

There are 3 ways for connecting with MongoDB:

1. MongoDB Shell (Local Command Line Interface – Main focus)
2. MongoDB Compass (GUI on local drive)
3. MongoDB Atlas (GUI on cloud)

Documents

Documents in MongoDB have the following structure:

{field1: value1, field2: value2, …}

Caveats of MongoDB documents:

1. MongoDB is type-sensitive and case-sensitive
2. Duplicate fields are not allowed
3. Avoid using reserved characters like "." and "$" for naming fields

Note that every document has special field "_id" that is unique within a collection, and it is generated by default.

Collections

While documents can be placed into any collection, it is best practice to store similar document schema under the same collection for the following reasons:

1. Ensure query of documents only returns results that follow a particular schema
2. Grouping similar documents allow for data locality
3. Better efficiency of indexing collections

When naming collections, the following restrictions apply:

1. Empty string is not valid
2. Collection names should not be reserved keywords like "system"
3. Collection names should not contain reserved characters like "$" and "."

Note that multiple collections can be created in a single database by using name-spaced subcollections separated by "." sign.

Example: mydb.coll1 and mydb.coll2 indicates that there are two collections named coll1 and coll2 under a database named as "mydb".

Databases

Databases in MongoDB can also consist of multiple collections.

When naming databases, the following restrictions apply:

1. Empty string is not valid
2. Database names should not have any special characters (Only alphanumeric allowed)
3. Database names are case-insensitive

MongoDB has three reserved database names as the following:

1. **Admin**: Used for database administration process
2. **Local**: Stores data used during replication process
3. **Config**: Stores information about each shard within a MongoDB cluster

Data Types of MongoDB

For MongoDB, data types do not need to be defined as MongoDB will automatically identify the data types of inputs.

The following are the lists of data types available for MongoDB:

String	Null
Integer	Date (Represent milliseconds since Unix Epoch)
Boolean	Object ID (Used in "_id" field)
Double	Binary
Arrays (Data contained in arrays do not have to be same data type – Behaves like lists in Python)	Code (Stores Java-Script code)
Timestamp	Regular Expression
Object (Used for embedded documents and creating dates in proper format)	

While JSON only supports a few data types, Binary JSON (BSON) extends the list of supported data types as shown above for efficient encoding and decoding within different languages.

Basic Commands for MongoDB Shell

1. Running MongoDB on CLI (On directory with mongo.exe)
   mongo

2. Viewing current database used
   db

3. Switch to using another database
   use new_db

4. Showing list of all databases
   show dbs

5. Showing list of all collections for a specific database
   show collections

6. Checking for database properties
   db.stats()

7. Importing JSON format
   mongoimport --uri "connection-string " **--drop** json_file_name
   mongoimport --db "db_name " **--drop** json_file_name

8. Exporting JSON format
   mongoexport --uri "connection-string " **--collection** coll_name **--out** json_file_name
   mongoexport --db "db_name" **--collection** coll_name **--out** json_file_name

9. Importing BSON format
   mongorestore --uri "connection-string" **--drop** bson_file_directory
   mongorestore --db "db_name " **--drop** bson_file_directory

10. Exporting BSON format
    mongodump --uri "connection-string" **--out** bson_file_directory
    mongodump --db "db_name" **--out** bson_file_directory

CRUD Operations of MongoDB

Creating Operations

When inserting single/multiple documents into a collection, **MongoDB checks for duplicate entry of "_id" key if exists that will result in error.** Note that new collections will be created when the collection has at least one document.

1. Inserting single document
   db.collection_name.insertOne({"key":value})

2. Inserting multiple documents (normal insert)
   db.collection_name.insertMany([{"key1":value1}, {"key2":value2}, …], {"ordered":true})

   Note that default value of ordered is true, indicating that documents are inserted in the order specified in the array of documents and operation will return an error as soon as one of the documents could not be inserted into the collection.

   Setting "ordered" to false will result in attempt to insert all documents into the collection when possible.

3. Inserting multiple documents (bulk insert)
   db.collection_name.insert([{"key1":value1}, {"key2":value2}, …], {"ordered":true})

4. Creating new collection
   db.createCollection("coll_name")

Deleting Operations

Note that delete operations require exercise of caution, as once the data is removed it could not be retrieved unless if there is a backup available.

1. Removing single document based on condition
   db.collection_name.deleteOne({condition})

2. Removing multiple documents based on condition
   db.collection_name.deleteMany({condition})

3. Removing entire collection
   db.collection_name.drop()

Updating Operations

1. Replace entire single document based on condition
   db.collection_name.replaceOne({condition},{new document})

2. Update part of single document based on condition
   db.collection_name.updateOne({condition}, {update_operator: {new key: new value}}, {upsert: false})

3. Update part of multiple documents based on condition
   db.collection_name.updateMany({condition}, {update_operator: {new key: new value}}, {upsert: false})

Note that update operations of documents will fail if the document does not exist based on condition by default (upsert is False). **Setting "upsert" to true result in new documents to be inserted if there is no existing document that satisfies specified condition.**

The following are the list of most common use of update operators:

1. **$inc**: Increments/Decrements by specified value (+ for increment and – for decrement)

2. **$mul**: Multiplies by specified value of field

3. **$set**: Sets the value of field/create new field if not exist

4. **$unset**: Removes the field from document by indicating "field name" and "1" as new key and value.

5. **$push**: Adds an item to array/create new array field if not exist

6. **$each**: Uses together with $push operator to add multiple items to array

7. **$addToSet**: Adds only unique items to existing array/create new array if not exist

8. **$currentDate**: Sets the value of field to current date or timestamp

For more details of other available update operators and its uses, please refer to the following website below for reference:

https://docs.mongodb.com/manual/reference/operator/update/

Querying Operations

Querying data in MongoDB returns results by using a cursor that acts as iterators.

By default, MongoDB automatically iterates through the results and display first couple of documents and viewing subsequent documents requires users to input "**it**" for more results in the CLI.

The following are the common query operations available

1. Query a single document
   db.collection_name.findOne({condition})

2. Query multiple documents
   db.collection_name.find({condition})

3. Specifying specific keys to return from query
   db.collection_name.find({condition}, {key1: 1, key2: 1, …})

   Note that only either specify keys to include (1) or keys to exclude (0) from query, with exception of "_id" key that can be excluded with other keys to include. **"_id" key is always included by default unless explicitly specified to exclude from query**

4. Limiting number of documents from query
   db.collection_name.find({condition}).limit(n)

5. Skipping number of documents and return remaining documents from query
   db.collection_name.find({condition}).skip(n)

   Note that it is not recommended to skip large number of documents due to lower performance.

6. Order documents from query by certain fields
   db.collection_name.find({condition}).sort({'key1': 1, 'key2': -1})

   Note that 1 represents ascending order and -1 represents descending order.

Querying operations for specific types will be covered in more detail in later section.

Common MongoDB Operators

Conditional Operators

The basic syntax for using conditional operators is as follows:

{key: {"operator": value}}

The following conditional operators can be used in MongoDB when querying documents:

$lt	Less than
$lte	Less than or equal to
$gt	Greater than
$gte	Greater than or equal to
$eq	Equal to
$ne	Not equal to

Logical Operators

By default, MongoDB uses $and operator without explicit mention when querying documents based on multiple conditions as follows:

{key1: value1, key2: value2}

The table below shows the list of logical operators available for MongoDB:

Operator	Explanation	Basic Syntax
$and	Returns true if multiple conditions are true	**{"$and": [{key1: value1}, {key2: value2}]}**
$or	Returns true if one of the conditions is true	**{"$or": [{key1: value1}, {key2: value2}]}**
$nor	Returns true if neither condition is true	**{"$nor": [{key1: value1}, {key2: value2}]}**
$not	Returns true if condition is false	**{key1: {$not: value1}**
$in	Returns true if value is in an array	**{key1: {$in: [value1, value2, …]}}**
$nin	Returns true if value is not in an array	**{key1: {$nin: [value1, value2, …]}}**

Type-Specific Queries

Null values

Querying for documents with null values in MongoDB behaves differently, compared to relational databases.

"null" keyword returns documents that have null values and documents where the field does not exist.

The following syntax is required for returning only existing fields with null values:

db.collection_name.find({key1: {'$eq': null, '$exists': true}})

Regular Expressions

$regex operator is used for pattern matching on string related queries

db.collection_name.find({key1: {'$regex': <pattern syntax>}})

Note that MongoDB uses Perl Compatible Regular Expression (PCRE) library to match regular expression syntax. Refer to the following website for syntax details of PCRE library:

https://learnxinyminutes.com/docs/pcre/

Arrays

The following are the main array query operations in MongoDB:

1. Single array element query
   db.collection_name.find({key1: value1})

   The syntax above returns documents that contain at least the specified single array element in the document.

2. Multiple array element query (Exact match)
   db.collection_name.find({key1: [value1, value2, …]})

   The syntax above returns documents that contain the exact array elements specified in the query. Note that this will not work for nested arrays.

3. Multiple array element query (Non-exact match)
   db.collection_name.find({key1: {'$all': [value1, value2, …]} })

   The syntax above returns documents that at least contain array elements specified in the query.

4. Array size query
   db.collection_name.find({key1: {$size: n}})

   The syntax above returns documents that contain specified size of array. Note that $size operator does not accept range of values.

5. Subset array query
   db.collection_name.find({condition}, {key1: {$slice: n}})

   Note that positive n value returns first n elements of array, while negative n value returns last n elements of array.

   By default, all fields (arrays and non-arrays) are returned from the query when using $slice operator.

6. Range of values array query
   db.collection_name.find({key1: {$elemMatch: {condition1, condition2}}})

   Note that $elemMatch operator forces both condition clauses to compare with a single array element. However, $elemMatch operator will only match array elements.

   Without $elemMatch operator, either condition clauses may match array elements instead (not desired behavior).

Embedded Documents

Querying for documents using embedded keys is preferred when querying for embedded documents.

Dot notation is used to indicate embedded fields for querying: **outer_key.inner_key**

Like querying range of values for arrays, querying for embedded documents based on multiple conditions require use of **$elemMatch** operator for non-exact matches.