**Mongo DB Interview Questions**

**List of Questions**

**✅ Core MongoDB Concepts**

1. **What is the difference between MongoDB and a relational database?**
   * (Follow-up: When would you choose one over the other?)
2. **What is a document in MongoDB? How is it different from a row in SQL?**
3. **What are collections in MongoDB?**
4. **Explain schema-less nature of MongoDB. How do you enforce structure?**
   * (Hint: Use of Mongoose schemas or schema validation)

**✅ CRUD Operations**

1. **How do you insert, read, update, and delete a document in MongoDB?**
   * Example-based questions:
     + "Write a query to find all users aged between 20 and 30."
     + "Update the status field to active for users who haven’t logged in for 6 months."
2. **What’s the difference between updateOne and updateMany?**
3. **What’s the difference between findOne() and find()?**

**✅ Indexes & Performance**

1. **What is an index in MongoDB? How does it improve performance?**
2. **What are different types of indexes in MongoDB?**
   * Single field, compound, text, hashed, geospatial
3. **What happens if a collection has no index and you run a complex query?**
4. **How would you identify slow queries in MongoDB?**

* (Hint: Use of explain() and MongoDB Atlas monitoring)

**✅ Aggregation Framework**

1. **What is the aggregation pipeline in MongoDB?**

* Can you explain a few stages like $match, $group, $sort, $project?

1. **Write an aggregation to get the average order amount per customer.**
2. **How is aggregation different from simple queries with .find()?**

**✅ Mongoose (ODM for Node.js)**

1. **What is Mongoose? Why do we use it in Node.js projects?**
2. **How do you define a schema and a model in Mongoose?**
3. **What are middleware in Mongoose?**

* (e.g., pre, post hooks)

1. **How do you handle relationships in MongoDB with Mongoose?**

* (e.g., population using ref, or embedding subdocuments)

**✅ Transactions & ACID**

1. **Are MongoDB operations ACID-compliant?**

* (Bonus: Explain transactions in MongoDB 4.0+)

1. **How would you implement a transaction in MongoDB using Mongoose?**

**✅ Real-world Scenarios / Best Practices**

1. **How do you handle large datasets in MongoDB?**
2. **How do you perform pagination efficiently in MongoDB?**

* (e.g., skip/limit vs. using \_id based pagination)

1. **What is the difference between embedding and referencing documents? When would you use each?**
2. **How do you ensure data validation in MongoDB?**
3. **What’s the use of MongoDB’s TTL index? Give a practical use case.**

### Additional MongoDB Questions:

1. **What is the difference between embedded documents and references in MongoDB? When would you use each?**
2. **How does MongoDB handle transactions? Explain multi-document transactions.**
3. **What is sharding in MongoDB? How does it help with scaling?**
4. **Explain replication in MongoDB. What is a replica set?**
5. **How do you enforce data validation in MongoDB?**
6. **What are Aggregation Pipelines in MongoDB? Give an example of a common use case.**
7. **Explain the difference between updateOne(), updateMany(), and replaceOne() in MongoDB.**
8. **What is the difference between $set, $unset, and $inc update operators?**
9. **How can you optimize query performance besides using indexes?**
10. **Explain the use of capped collections in MongoDB.**
11. **How do you perform text search in MongoDB? What are some limitations?**
12. **What are the advantages and disadvantages of MongoDB’s schema-less design?**
13. **How do you handle data backup and restore in MongoDB?**
14. **What is the significance of the WiredTiger storage engine in MongoDB?**
15. **How does MongoDB handle concurrency and locking?**

**🔁 Bonus: Behavioral MongoDB Qs**

* “Describe a time when you had to debug a performance issue in a MongoDB query.”
* “Have you worked with MongoDB Atlas? What monitoring or backup features have you used?”

**📄 Pro Tip:**

You can back your answers with **real-world examples** from your projects — this adds credibility and shows practical understanding.

If you’d like, I can also give you:

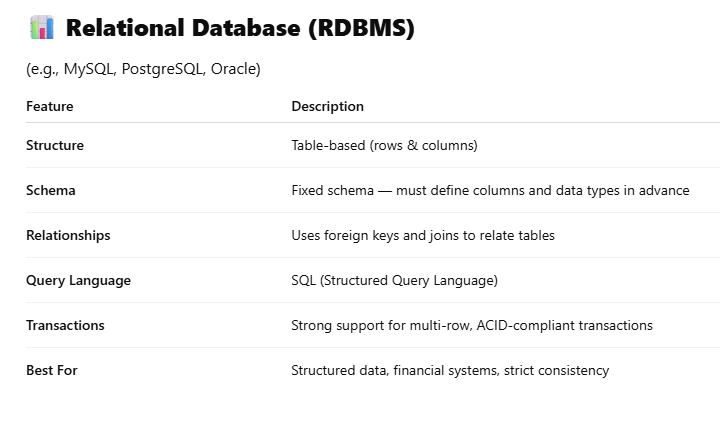
* **Coding-based MongoDB questions**
* **MongoDB questions in Node.js context (e.g., with Express/Mongoose)**
* Or even help you **mock interview** for MongoDB

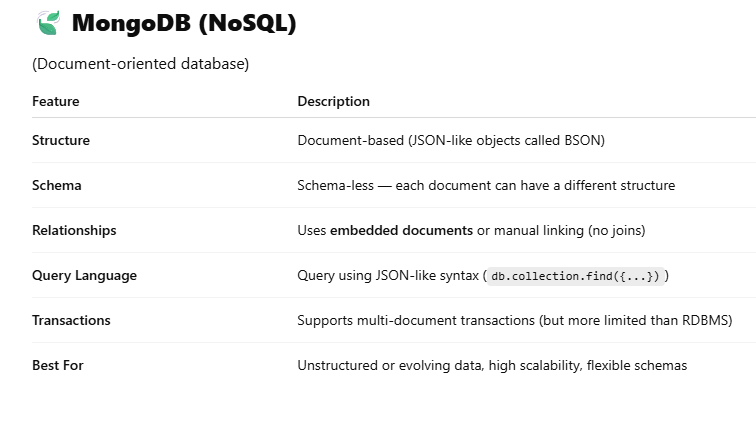
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Q1. What is mongo DB?**

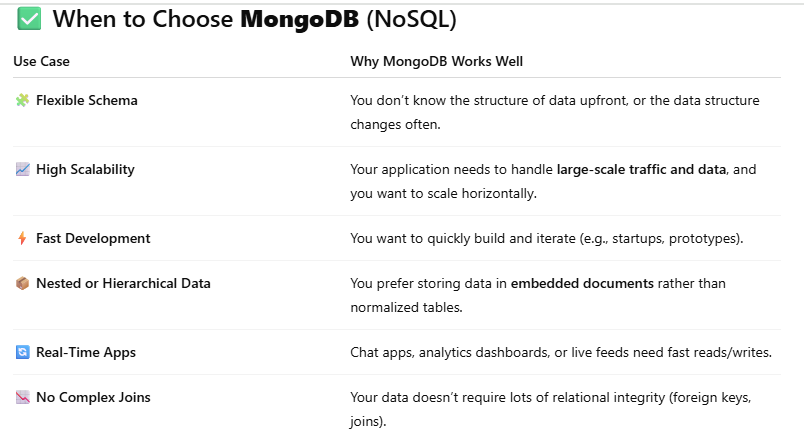
* **MongoDB** is an **open-source document-oriented,** **NoSQL database** designed for high performance, scalability, and flexibility.
* It stores data in **JSON-like documents** (**BSON format**), not tables and rows like SQL databases.
* Developed and manage by **MongoDB Inc.** under SSPL**(Server Side Public License**) and initially released in February 2009.
* It's an open-source and widely used modern database.

**Q2. What is the difference between MongoDB and a relational database?**

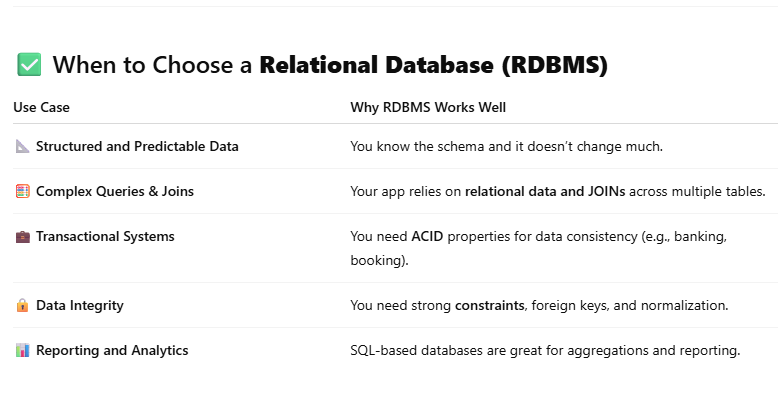




**Q3. When would you choose one over the other?**

****

**Examples**: Content management systems (CMS), product catalogs, user profiles, logging systems, IoT data, social media apps.

****

**Examples**: Banking apps, inventory systems, HR systems, CRM platforms, ERP software.

**Q4. What Are Collections in MongoDB?**

* A **collection** is a **group of MongoDB documents**.
* It holds **multiple documents**, just like a SQL table holds multiple rows.
* Unlike SQL tables, **collections do not enforce a strict schema** — documents can have **different structures**.
* Collections are created **automatically** when you insert the first document, or manually using db.createCollection().

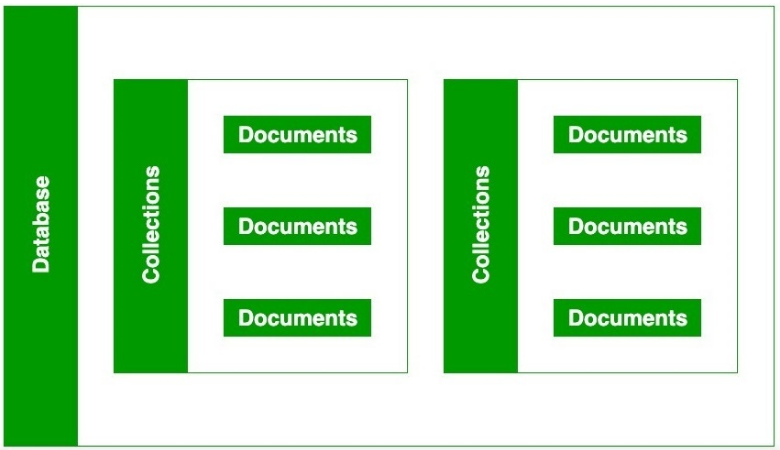
****

**Q5. What is a Document in MongoDB?**

* A **document** in MongoDB is a **single record** in a collection.
* It is stored in **BSON** format (Binary JSON), which supports rich data types.
* A document is similar to a **JavaScript object** or **JSON**.

****

**Image of Database, Collection and Document**

****

**Q6. Explain schema-less nature of MongoDB. How do you enforce structure?**

**(Hint: Use of Mongoose schemas or schema validation)**

📌 **Schema-less Nature of MongoDB**

**What Does "Schema-less" Mean?**

* MongoDB is **schema-less**, meaning:
  + Documents in the same **collection** can have **different structures**.
  + There’s **no need to define columns or data types** ahead of time.
  + This gives developers **flexibility** to store varied data in one collection.

****

**You can enforce structure using one of the following**

**1. Mongoose**

**2. MongoDB Native Schema Validation**

**-> MongoDB Native Schema Validation**

**Starting from MongoDB 3.2+, you can use JSON Schema validation directly in the database.**

****

**-> Mongoose (for Node.js)**

* **Mongoose** is an ODM (Object Data Modeling) library for MongoDB and Node.js.
* It allows you to define **schemas** with data types, validations, defaults, etc.

****

**Q7. What is the role of \_id in MongoDB documents? Can it be customized?**

**Role of** \_id**:**

* In MongoDB, every **document** must have a **unique \_id field**.
* This field acts as the **primary key** for the document within a collection.
* It is used to **uniquely identify and access** each document efficiently.

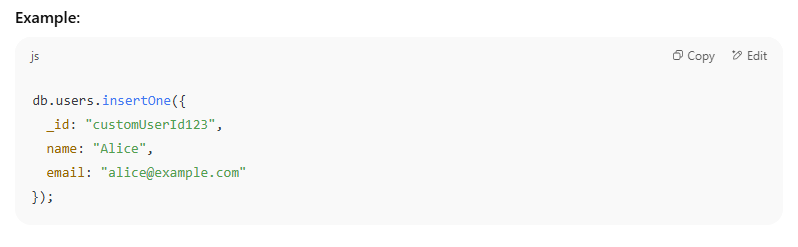
**By Default:**

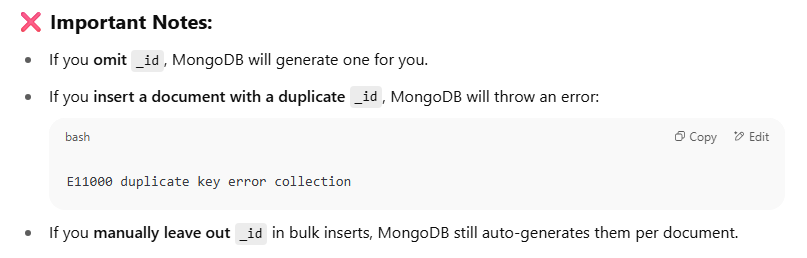
* MongoDB **automatically creates** an \_id field if you don’t specify one.
* The default value is an **ObjectId**, a 12-byte unique identifier that includes:
  + Timestamp
  + Machine ID
  + Process ID
  + Incrementing counter



**✏️ Can It Be Customized?**

Yes — you **can manually set your own \_id** when inserting a document.





**Q8.** **What is an index in MongoDB? How does it improve performance?**

An **index** in MongoDB is a special data structure (**B-Tree-like data structure**) that **improves the speed** of queries on a collection.

* It’s similar to an index in a book — it helps MongoDB **quickly locate documents** without scanning every single one.
* By default, MongoDB creates an **index on the \_id field** of every collection.

**Why is Indexing Important in MongoDB?**

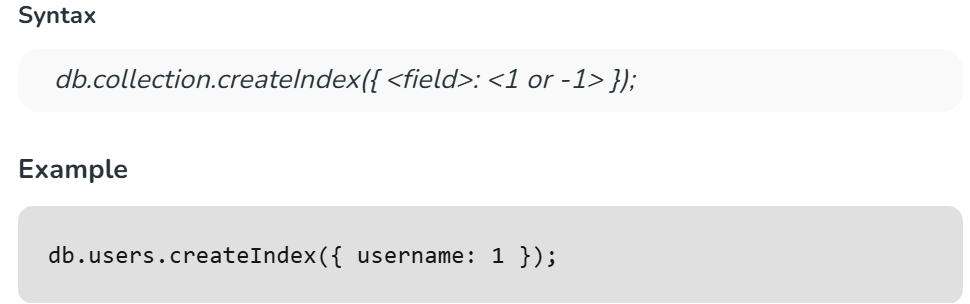
MongoDB provides a method called [createIndex()](https://www.geeksforgeeks.org/mongodb-db-collection-createindex-method/" \t "_blank) that allows users to create an index.

The **key determines the field on the basis of which we want to create an index** and **1 (or -1) determines the order in which these indexes will be arranged(ascending or descending).**

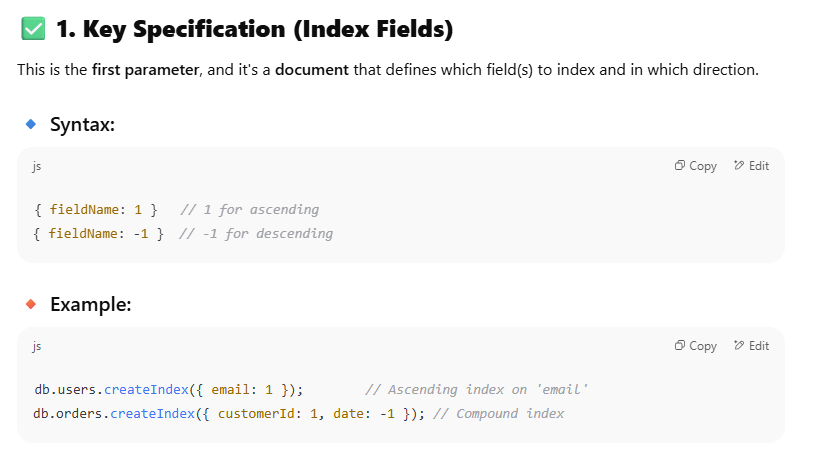
Indexing improves the performance of:

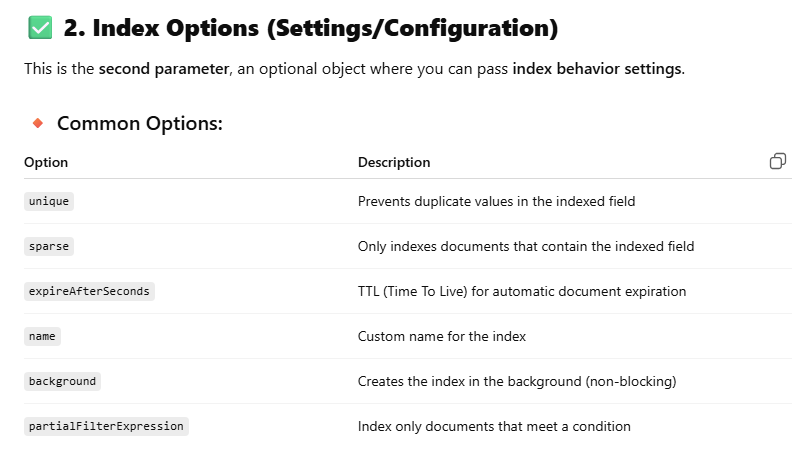
* **Find queries** (db.collection.find()).
* **Range queries** (e.g., queries with <, >, <=, >= operators).
* **Sorting** (e.g., db.collection.find().sort()).
* **Aggregation operations** involving filtering, grouping, and sorting.
* **Reduce Resource** Usage Lowers CPU and memory usage for operations.

**How to Create an Index in MongoDB?**



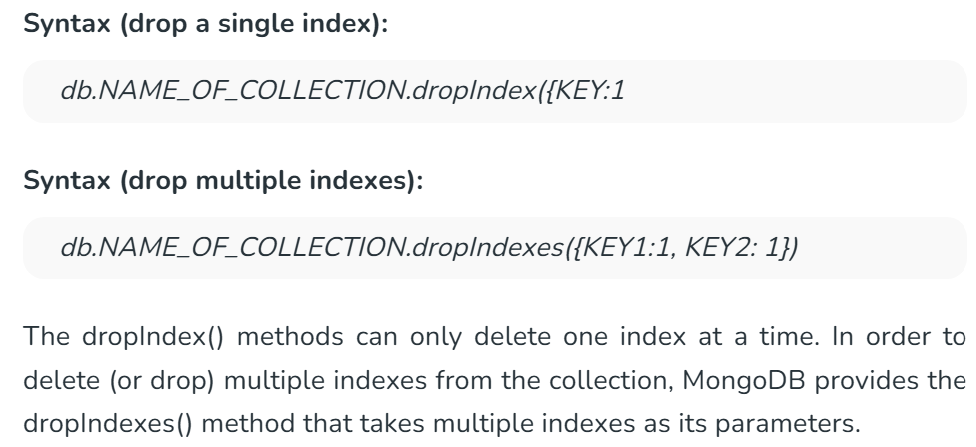
**What are the parameters to create an index?**

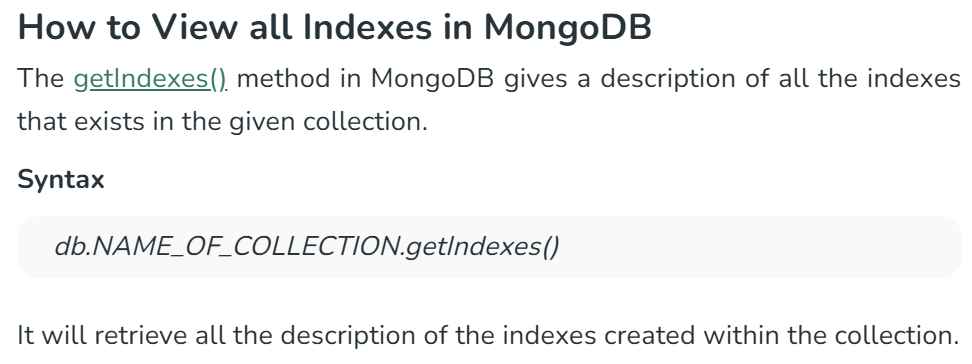
****

****

**How to Drop an Index in MongoDB ?**

* We can drop an index using the **dropIndex()** method.
* To drop multiple indexes, use **dropIndexes().**



****

**How does it improve performance?**

Without indexes:

* MongoDB performs a **collection scan**, checking each document one by one — slow for large datasets.

With indexes:

* MongoDB uses the index to **jump directly to matching documents**, skipping irrelevant ones — **much faster**.



**Why Do Inserts, Updates, and Deletes Affect Indexes?**

📌 1. **Insert Operations**

****

MongoDB must:

1. Insert the document into the users collection.
2. **Insert the email field into the index tree**.
   * The index structure (like a B-tree) must be updated to include "alice@example.com" at the correct position.

Note: This extra step takes **CPU, memory, and disk I/O**, especially when there are **multiple indexes**.

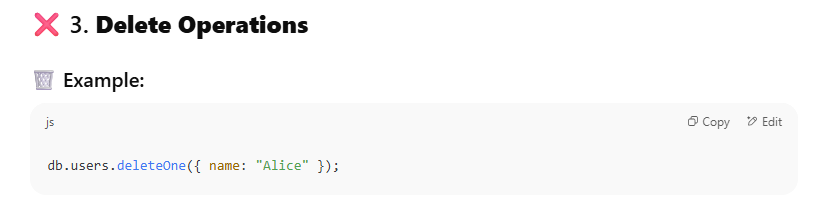


If email is indexed:

* MongoDB must **remove** the old email from the index.
* Then **insert** the new email into the correct position in the index.

If the update modifies a field that is **not indexed**, then indexes don’t change.  
But if **indexed fields are updated**, each affected index must be **rebalanced**.

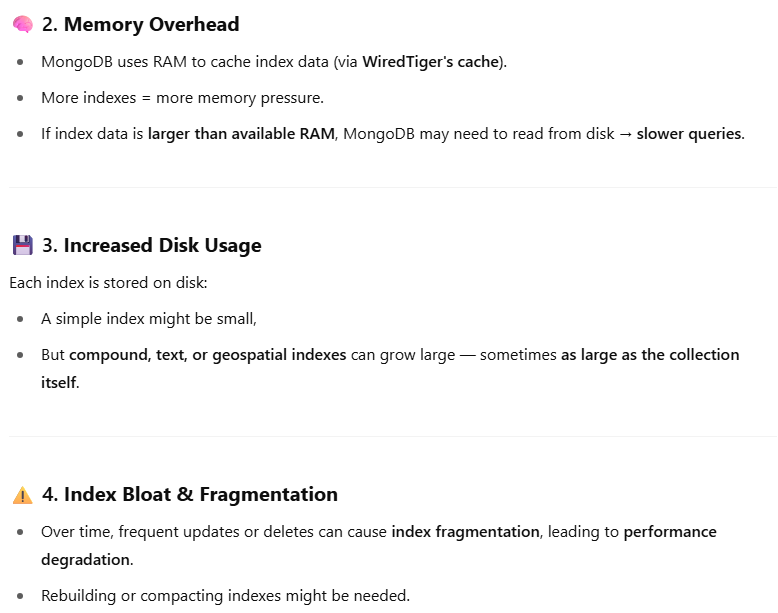
This can slow down update-heavy applications if too many indexes are involved.

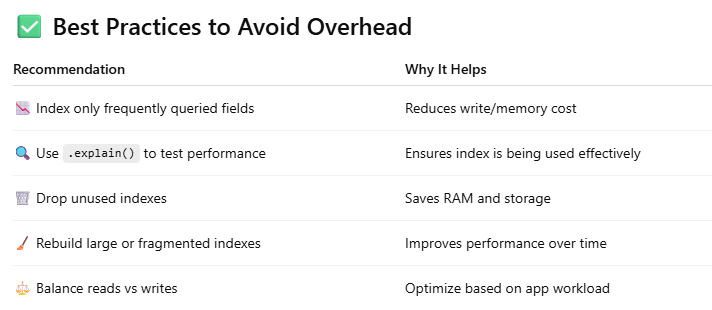
****

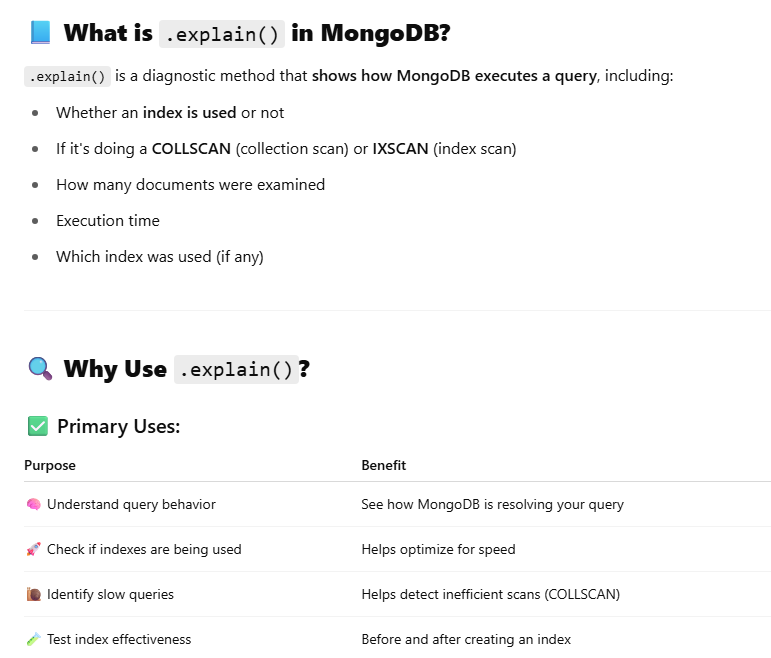
* MongoDB deletes the document from the collection.
* But it also must **find and remove** all corresponding index entries for that document.

This involves locating the document’s entry in **each index** and removing it — which takes additional processing time.

**Other Overhead of using Index**



****

****

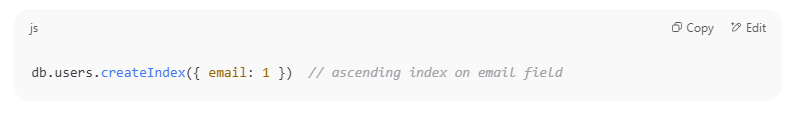
****

**Q9. What are different types of indexes in MongoDB?**

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

**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 order.



* **1** represents ascending order, meaning that MongoDB will store the values in increasing order.
* **-1** can be used for descending order.
* Ideal for querying or sorting based on a single field, such as searching for a student's studentId.

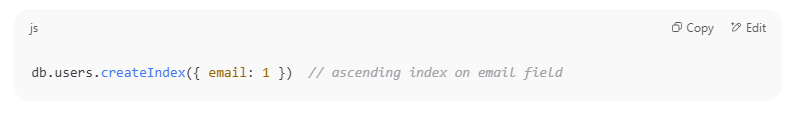
**2. Compound Index**

Indexes multiple fields to optimize queries filtering/sorting by those fields.



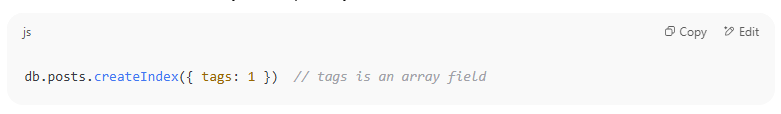
**3. Unique Indexing**

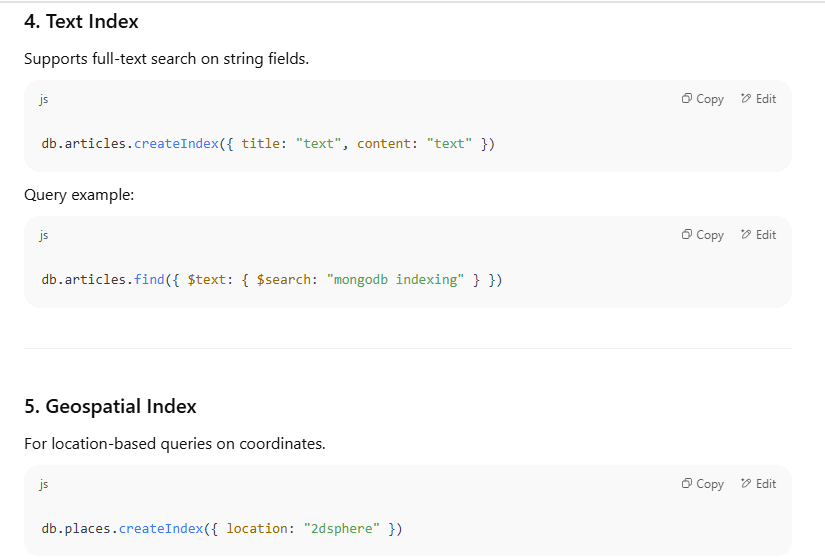
Indexes on unique filed like email, etc.



**4. Multikey Index**

**Indexes each element of an array field separately**.





**6. Hash Indexing**

**7. TTL (Time-To-Live) Index**

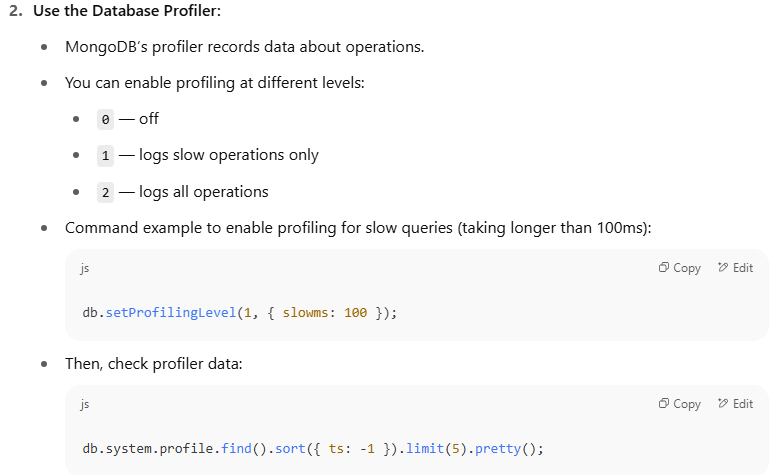
**Q10. What happens if a collection has no index and you run a complex query?**

If a collection has no indexes and you run a complex query, here’s what happens:

1. **Full Collection Scan:**  
   MongoDB will scan every document in the collection to find matches — this is called a collection scan.
2. **Slow Performance:**Since it checks every document, the query will be much slower, especially as the collection grows larger.
3. **High Resource Usage:**  
   Full scans use more CPU, memory, and I/O, which can impact overall database performance.
4. **No Sorting Optimization:**  
   Without indexes, sorting operations also become slower because MongoDB cannot use an index to quickly order results.

**Q** **11. How to Identify Slow Queries in MongoDB**

1. Enable the MongoDB Slow Query Log

****

**3. Use the explain() Method:**

* Run .explain("executionStats") on your queries to see detailed info about execution time and index usage.



**4. MongoDB Atlas Performance Advisor (If using Atlas):**

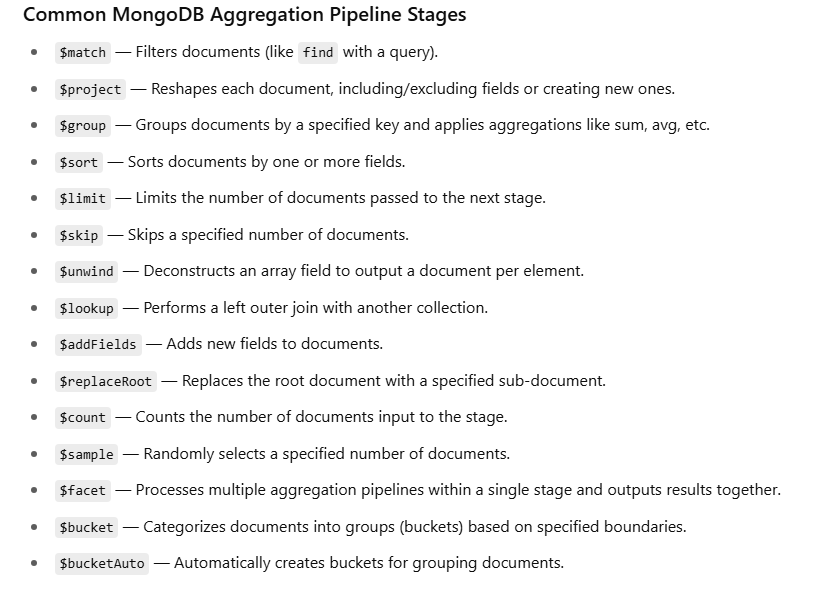
* It automatically identifies slow queries and recommends indexes.

**Q12.** **What is the aggregation pipeline in MongoDB?**

The **MongoDB Aggregation Pipeline** allows you to perform a sequence of data processing operations on the documents in a collection.

The aggregation pipeline consists of a series of stages, where each stage represents a specific operation or transformation.

**The output of each stage serves as the input for the next stage**, allowing you to create complex data processing workflows.



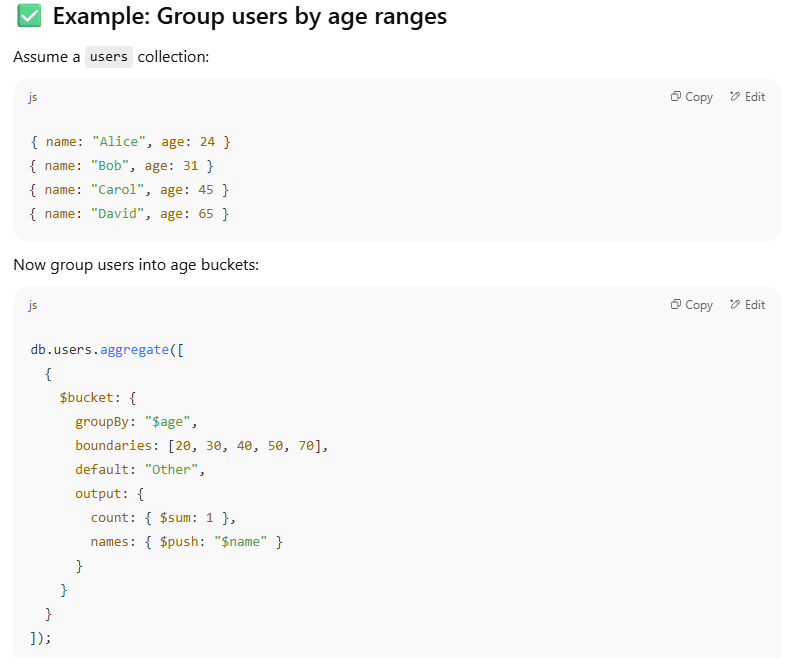


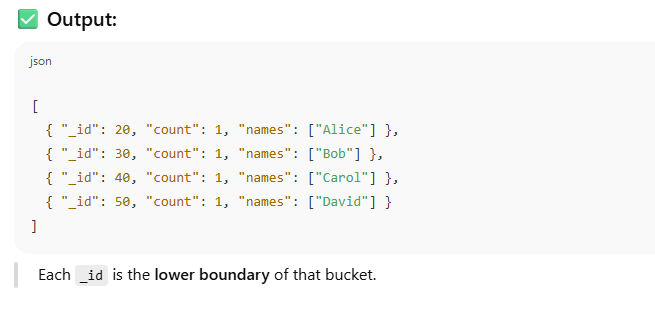
Some more example

**$bucket :**

The $bucket operator in MongoDB is used in **aggregation pipelines** to group documents into **buckets (ranges)** based on a specified **field's value**



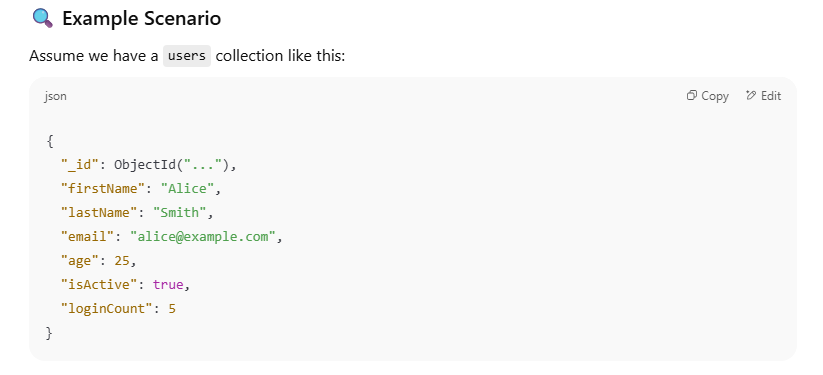


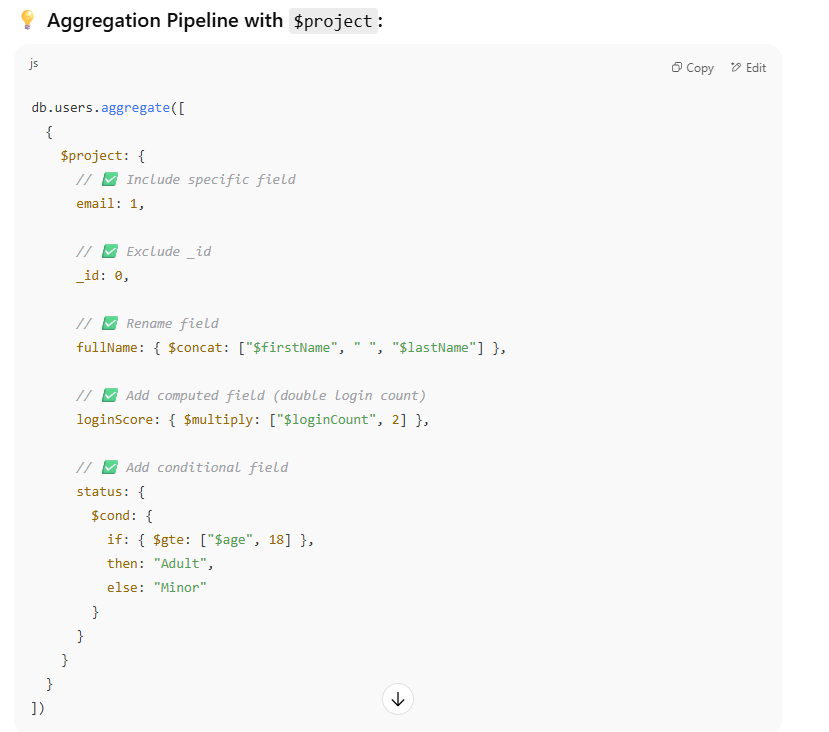


**$project:**

The $project stage in MongoDB's **aggregation pipeline** is used to **reshape documents**, i.e., to:

* Include or exclude specific fields
* Add new computed fields
* Rename fields
* Reformat or transform field values

****

****

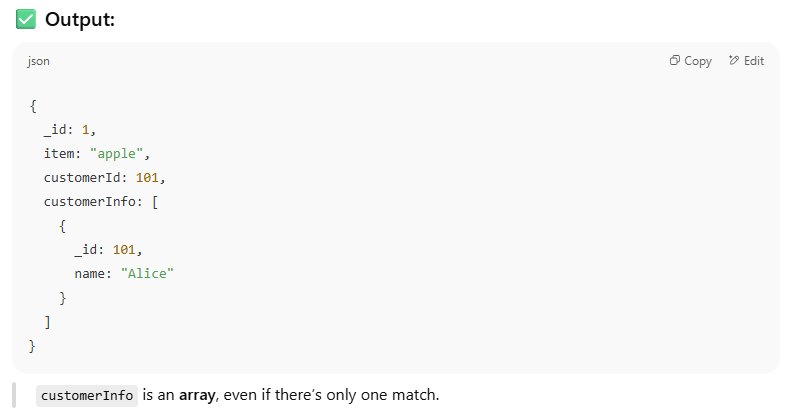
**$lookup:**

The $lookup stage in MongoDB's **aggregation pipeline** performs a **left outer join** between documents in two collections — kind of like an SQL JOIN.

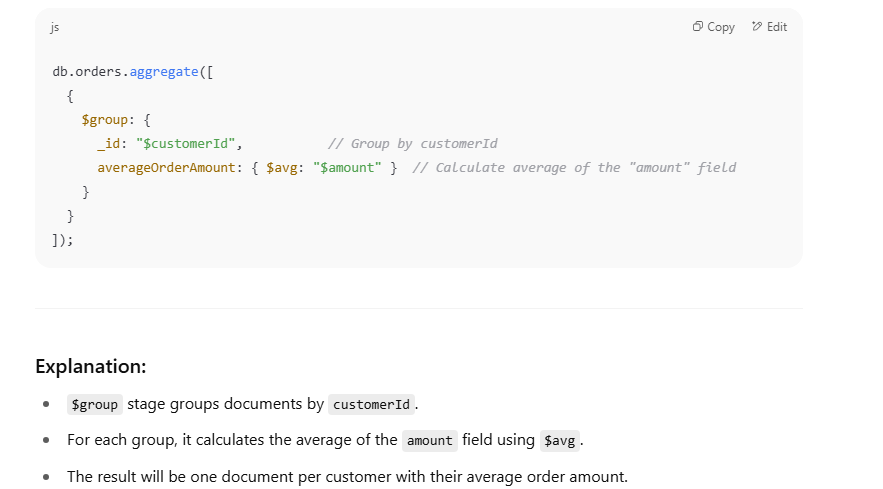








**Q13. Write an aggregation to get the average order amount per customer.**

****

**Q14. What is Mongoose? Why do we use it in Node.js projects?**

* **Mongoose** is **an Object Data Modeling (ODM) library for MongoDB and Node.js**.
* It **provides a straightforward, schema-based solution to model** your application data.

**Why do we use Mongoose in Node.js projects?**

* **Schema definition:** Unlike MongoDB which is schema-less, Mongoose lets you define schemas to enforce structure and data types on your documents.
* **Data validation:** It helps validate data before saving it to the database.
* **Middleware support:** You can add hooks (pre/post) for operations like save, update, delete, etc.
* **Query building:** Simplifies querying MongoDB with a fluent, chainable API.
* **Population:** Makes it easy to work with references between collections (like JOINs).
* **Better error handling:** Provides consistent and meaningful errors.

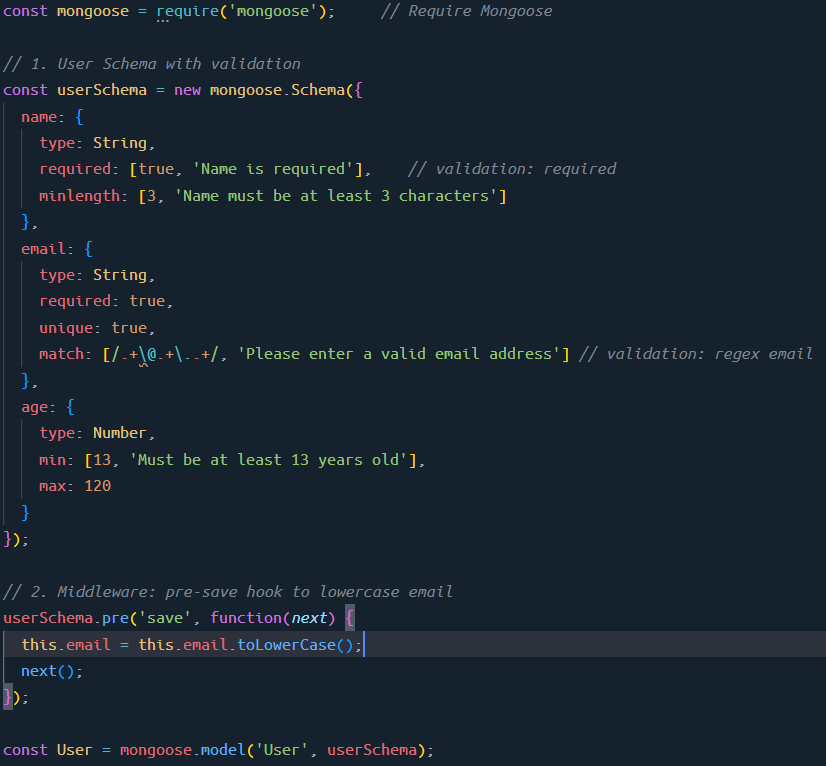
**Q15. How do you define a schema and a model in Mongoose?**

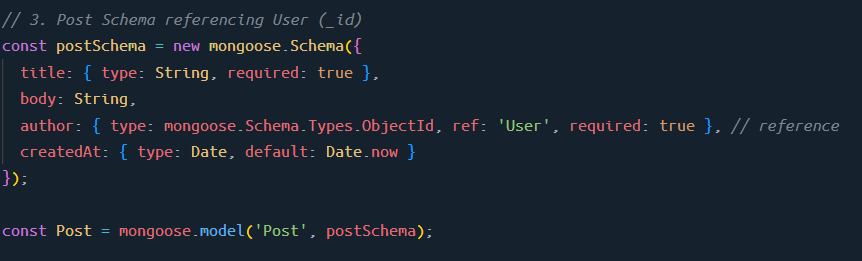
**What is a Schema?**

* A **Schema** in Mongoose defines the structure of documents within a MongoDB collection.
* It specifies fields, data types, validations, default values, and more.

**What is a Model?**

* A **Model** is a constructor compiled from the schema.
* It provides an interface to interact with the database — create, read, update, and delete documents.

****

****

**Q16. What are middleware in Mongoose?**

**Middleware** in Mongoose are functions that run **before** or **after** certain lifecycle events on documents or models.

They allow you to add logic to run automatically during operations like saving, updating, validating, or removing documents.

**Types of Middleware in Mongoose**

1. **Pre middleware** – runs before an operation.
2. **Post middleware** – runs after an operation.

**Common Middleware Use Cases**

* Validate or modify data before saving.
* Hash passwords before storing.
* Log or audit changes.
* Cleanup related data after removal.

****

**Q17. How do you handle relationships in MongoDB with Mongoose?**

Handling relationships in MongoDB with Mongoose is done mainly in two ways:

**Embedding documents** and **Referencing documents** (also called population).

**1. Embedding (Denormalization)**

* You store related data **inside** the parent document as a nested object or array.
* Good for closely related data that is always accessed together.

Example: A blog post with embedded comments.

****

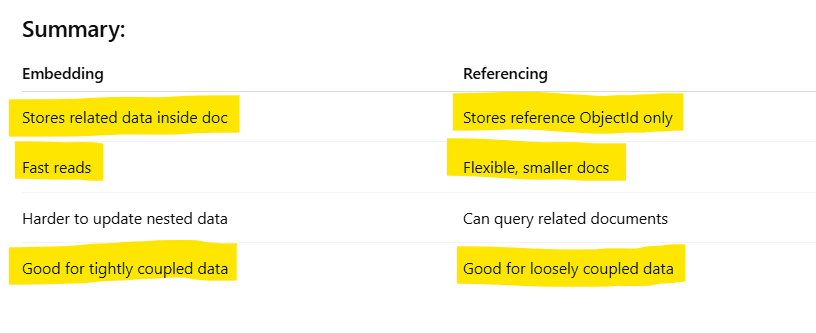
**Pros:** Faster reads since all data is in one document.  
**Cons:** Can cause document size to grow large and harder to update nested data independently.

**2. Referencing (Normalization with Population)**

* You store only the **ObjectId** of the related document.
* Mongoose **provides .populate()** method to fetch related documents.
* Useful when related data is large or shared across documents.

Example: Users and Posts relationship where a post references its author:





**Q22.** **How do you handle large datasets in MongoDB?**

1. **Schema Design for Large Data**

* **Embed vs. Reference**:
  + Embed related documents if they are frequently accessed together.
  + Use references for large or less-frequently accessed data.
* **Avoid deep nesting**: MongoDB documents can only be 16MB — keep nesting reasonable.
* **Use flat and lean schemas**: Store only what’s needed

**2. Indexing**

* **Create indexes** on fields used in queries, sorting, or filtering.
* Use **compound indexes** if querying multiple fields.
* Use **covered queries** to return results directly from the index.

****

**3.** **Sharding (Horizontal Scaling)**

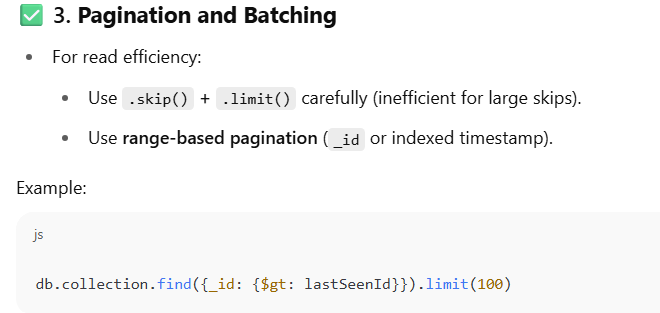
* Split large collections across multiple shards (servers).
* Choose a good **shard key** that avoids:
  + Hot spots (too many inserts in one shard).
  + Poor cardinality (too few unique values).
* MongoDB handles routing automatically once sharded.

**4. Aggregation Framework Optimization**

* Use $match **early in pipeline** to filter before processing.
* Use $project to limit fields.
* Avoid loading entire documents if not needed.
* Run explain() on aggregation pipelines to understand performance.

**5. Monitoring and Profiling**

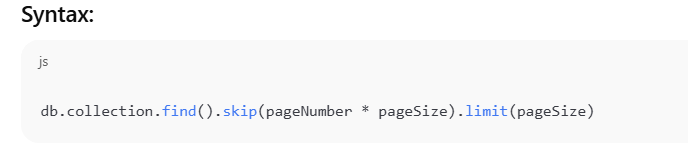
* Use MongoDB tools:
  + **Atlas Performance Advisor**
  + mongotop, mongostat
  + Query profiler: db.setProfilingLevel()
  + db.collection.explain("executionStats")



Q23. **How do you perform pagination efficiently in MongoDB?**

**(e.g., skip/limit vs. using \_id based pagination)**

**1. skip() / limit() Pagination (Offset-based)**

****

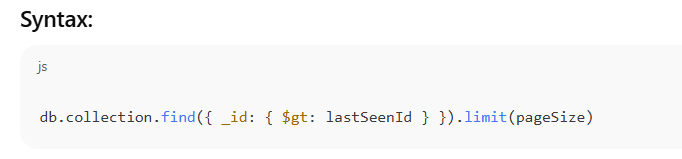
**Pros:**

* Simple to implement.
* Good for small datasets or admin dashboards.

**Cons:**

* **Performance degrades** with high skip values — MongoDB must scan and discard skipped documents.
* Not suitable for infinite scroll or APIs with deep pagination.

**2.** **\_id-based Pagination (Cursor-based / Range-based)**

****

**Pros:**

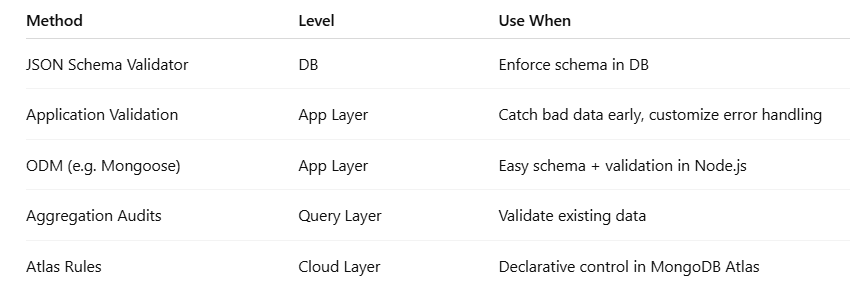
* **Very efficient** — uses indexed field (e.g., \_id, createdAt, etc.).
* Ideal for infinite scroll or real-time feeds.
* Consistent under changing data (no skipping over new inserts).

**Cons:**

* Cannot jump to arbitrary pages (no page number).
* Requires maintaining the **last seen document's key**.

****

**Q24. How do you ensure data validation in MongoDB?**

****

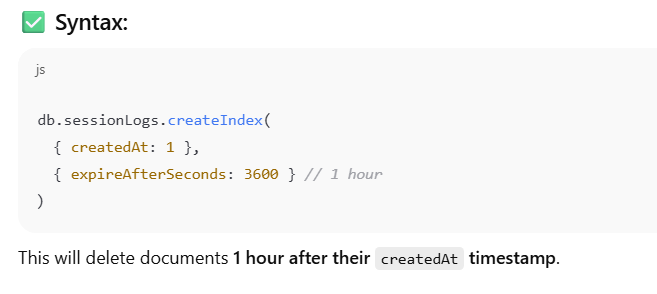
**Q25. What’s the use of MongoDB’s TTL index? Give a practical use case.**

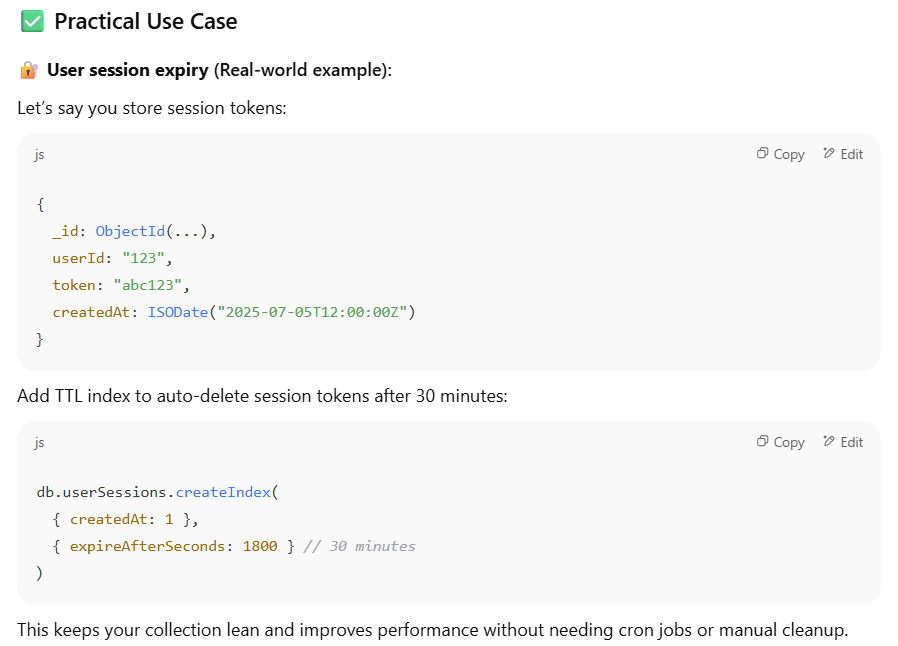
A **TTL (Time-To-Live) index** in MongoDB automatically **removes documents** from a collection after a certain period — based on a timestamp field.

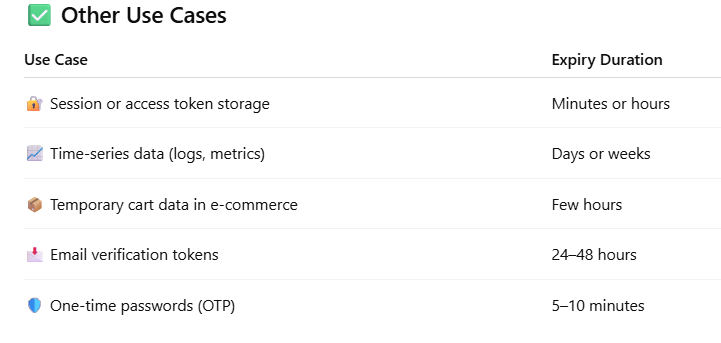
It’s useful for **automatic data expiration** without manual cleanup scripts.

**How TTL Index Works**

* You create a TTL index on a field that stores a **date/time** value.
* MongoDB automatically deletes documents **after the date + expiration window**.
* Cleanup runs every 60 seconds (not immediate).





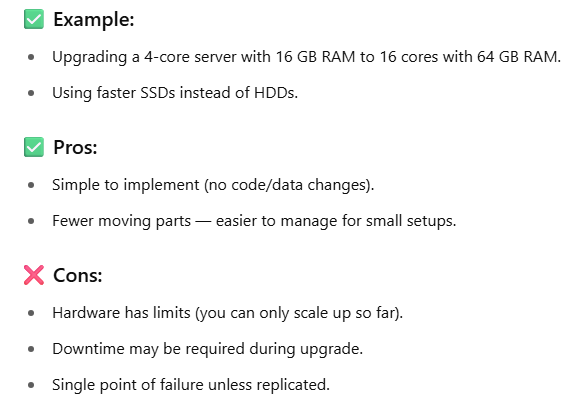


**Q26. Explain Vertical and Horizenta Scalling in MongoDB.**

**1. Vertical Scaling (Scaling Up)**

**📌 Definition:**

Vertical scaling means **adding more resources (CPU, RAM, Disk)** to a **single MongoDB server** to handle increased load.

****

**2. Horizontal Scaling (Scaling Out)**

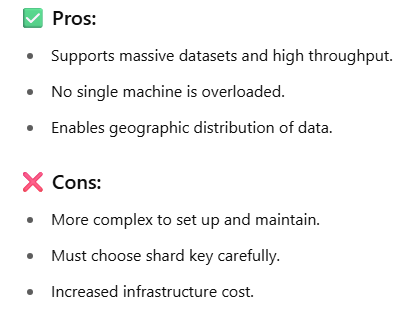
**📌 Definition:**

Horizontal scaling means **adding more servers (nodes)** to distribute data and load across multiple machines.

In MongoDB, this is done via **sharding**.

✅ How it works:

* A large collection is split (sharded) across multiple **shards** (nodes).
* Each shard holds a portion of the data based on a **shard key**.
* A **mongos router** routes queries to the correct shard(s).

****