|  |  |  |
| --- | --- | --- |
| ****1. Existing Document**** **Assume this is the document already in the comments collection:**   |  | | --- | | {  "\_id": 1,  "name": "GFG NOIDA",  "lang": "Python",  "member\_since": 10  } | |  The filter { name: 'GFG NOIDA' } matches this document.   The $set operator updates the specified fields.  {  "\_id": 1,  "name": "GFG",  "lang": "JavaScript",  "member\_since": 51  } | |
| ****No Matching Document**** **Assume the comments collection is empty** or no document matches { name: 'GFG NOIDA' }.   * The upsert: true option ensures a new document is **inserted**.  |  | | --- | | The new document created will be:  {  "name": "GFG",  "lang": "JavaScript",  "member\_since": 51  } | | Key Points to Remember:  1. **updateOne** updates only the first matching document. 2. **$set** modifies or creates fields. 3. **upsert: true** creates a new document if no match is found. | |
| **Creating an Index******Basic Syntax:**** To create an index, you use the createIndex() method:   |  | | --- | | db.collection.createIndex({ fieldName: 1 });   1 specifies ascending order (used for sorting).   -1 specifies descending order. | | ****Single-Field Index**** Suppose you have a collection called users and you want to create an index on the name field in ascending order:  **db.users.createIndex({ name: 1 });** | | ****Compound Index**** A **compound index** is used for multiple fields. If you want to create an index on both name and age:  **db.users.createIndex({ name: 1, age: -1 });**   name is sorted in ascending order.   age is sorted in descending order. | | ****Unique Index**** To ensure that the email field in your collection has **unique** values:  **db.users.createIndex({ email: 1 }, { unique: true });**  This prevents duplicate values for the email field. | | ****TTL Index (Time-To-Live)**** A TTL index is used to automatically delete documents after a certain time. For example: ****db.logs.createIndex({ createdAt: 1 }, { expireAfterSeconds: 3600 });****  * Documents with a createdAt field will be deleted **after 1 hour (3600 seconds)**.   **Note:** The createdAt field must be a date. |  **Viewing Indexes** To see the indexes created on a collection:  db.collection.getIndexes();  example:  **db.users.getIndexes();** |
| **Deleting an Index******Basic Syntax:**** To delete an index, you use the **dropIndex()** method:  db.collection.dropIndex("indexName"); |
| Steps to Delete an Index  **Find the Index Name:** Use the getIndexes() method to list all indexes in the collection.  db.users.getIndexes();   |  | | --- | | [  {  "v": 2,  "key": { "\_id": 1 },  "name": "\_id\_"  },  {  "v": 2,  "key": { "name": 1 },  "name": "name\_1"  }  ]  In this case, the index name for the name field is name\_1.  **db.users.dropIndex("name\_1");** | |
| ****Deleting All Indexes**** If you want to drop **all non-default indexes** (excluding the \_id index), use:  db.collection.dropIndexes();  db.users.dropIndexes(); |
| **Summary******Create Index****  * Single Field: db.collection.createIndex({ fieldName: 1 }); * Compound Index: db.collection.createIndex({ field1: 1, field2: -1 }); * Unique Index: db.collection.createIndex({ fieldName: 1 }, { unique: true }); * TTL Index: db.collection.createIndex({ fieldName: 1 }, { expireAfterSeconds: X });  ****Delete Index****  * Drop a Specific Index: db.collection.dropIndex("indexName"); * Drop All Indexes: db.collection.dropIndexes(); |
| **Checking Execution Stats for a Query** Suppose you have a users collection and you want to search for documents where name is "John":   |  | | --- | | **db.users.find({ name: "John" }).explain("executionStats");** | | **{** **"queryPlanner": {** **"namespace": "test.users",** **"winningPlan": {** **"stage": "COLLSCAN"** **}** **},** **"executionStats": {** **"nReturned": 1,** **"executionTimeMillis": 2,** **"totalDocsExamined": 5** **}****}** | |