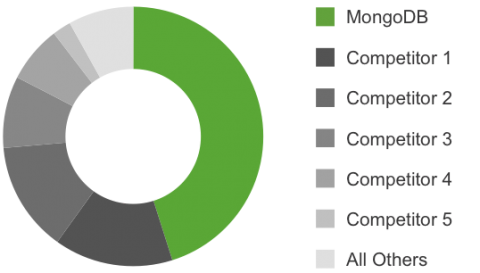
How to perform Atomic Operations through MongoDB?

The NoSQL database system has been able to gain momentum in the past few years due to its flexibility and other benefits. Mongo is the leader when it comes to NoSQL.



**Image Source:** [Mongo](https://www.mongodb.com/leading-nosql-database)

If you want to jump into the NoSQL database, you should strongly consider [**MongoDB certification**](https://www.simplilearn.com/big-data-and-analytics/mongodb-certification-training). It will give you a strong foundation and skills to use MongoDB to solve real-world problems. There is a considerable job demand for MongoDB professionals.

There are plenty of amazing features of MongoDB, and one of them is an atomic operation. However, Mongo does not support atomic operations in multi-document tasks. In the upcoming sections of this article, we will go deep into atomic operations, its use, and how you can apply it to a real-world issue without any difficulty.

## An introduction to atomic operations in MongoDB

The atomic operations in database terminology is a chained and sophisticated database operations series. In this series, either all aspects of the series will be affected, or nothing will be altered. In atomic operations, there is no such thing as a partial database change.

In atomic operations, there is only a room for complete database updates and changes. In case of any partial updates, the whole database will roll back.

We use atomic operations in a case where the partial update will create more harm in comparison to rolling back. There are some instances in the real world where we need to maintain our database in this manner. In the latter part of this article, we will discuss more in depth about it.

We can explain atomic operations in MongoDB clearly with the use of ACID, a.k.a. Atomicity, Consistency, Isolation, and Durability.

* Here is a simple rule of atomicity for every single operation, “either all or none.”
* The consistency property will play a crucial role in atomicity. It will ensure that each transaction will ultimately lead to a valid state of some kind.
* The isolation property of the database will play a part in guaranteeing the systematic process of the concurrent transaction, which means one by one.
* Finally, the durability property will come into play. This property ensures the permanency of the database after each database transaction regardless of errors, crashes, and power loss.

## Some critical points about MongoDB that you need to remember

Before moving forward to checking out how we can apply atomic operations in MongoDB, we will be looking at some of the critical points that you need to keep in your mind in regards to MongoDB:

* MongoDB does not support atomicity for multi-document transactions. However, version 4.0 onwards will support multi-document transactions in several cases.
* It is only possible to use the atomicity feature of MongoDB in a single document (not in case of version 4.0). Suppose, there is a document that consists of 35 fields. In that document, there will either be updates in all 35 fields or none.
* The atomicity feature is only limited to the document level.

## Atomicity in MongoDB

Like said before, we should maintain atomicity in MongoDB by compiling all the related information in a single document, which will update consistently. We can create such type of document via embedded document. The embedded report is for ensuring that every single update that takes place in the document is atomic.

Here is how the document looks like for representing item purchase information.

“

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | Microsoft Windows [Version 10.0.14393]  (c) 2016 Microsoft Corporation. All rights reserved.    C:\Users\Aparajita>cd C:\Program Files\MongoDB\Server\3.2\bin    C:\Program Files\MongoDB\Server\3.2\bin>mongo.exe  MongoDB shell version: 3.2.10  connecting to: test  > use AtomicMongoDB  switched to db AtomicMongoDB  > db.AtominMongoDB.save ({  ... "\_id":101,  ... "item\_name": "Apple iPhone 7",  ... "category": "handset",  ... "warranty\_period": 5,  ... "city": "Toronto",  ... "country": "Canada",  ... "item\_total": 10,  ... "item\_available": 6,  ... "item\_bought\_by": [  ... {  ... "customer": "Mohit",  ... "date": "6-May-2017"  ... },  ... {  ... "customer": "Aparajita",  ... "date": "5-May-2017"  ... },  ... {  ... "customer": "Anita",  ... "date": "4-May-2017"  ... },  ... {  ... "customer": "Abhishek",  ... "date": "3-May-2017"  ... }  ... ]  ... });  WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "\_id" : 101 })  > db.AtominMongoDB.find().pretty();  {  "\_id" : 101,  "item\_name" : "Apple iPhone 7",  "category" : "handset",  "warranty\_period" : 5,  "city" : "Toronto",  "country" : "Canada",  "item\_total" : 10,  "item\_available" : 6,  "item\_bought\_by" : [  {  "customer" : "Mohit",  "date" : "6-May-2017"  },  {  "customer" : "Aparajita",  "date" : "5-May-2017"  },  {  "customer" : "Anita",  "date" : "4-May-2017"  },  {  "customer" : "Abhishek",  "date" : "3-May-2017"  }  ]  }  > |

“

In the above document, we have created a model, embedded document. We have produced a report from the purchase in the item\_bought\_by field. This single document will manage everything about the purchase and the stock. In this document, it will see whether the item that the customer orders are in the stock or not. The customer’s order processes through the item\_available field.

In case of the availability, we will subtract the item\_available field by 1. After we complete that part, we will record the information of the customer, and, i.e., name and the purchase date, in the item\_bought\_by field. We will again look at another document where we will be using the findAndmodify statement to fulfill this purpose.

By using findAndmodify statement, the document will perform search and update activity simultaneously in the report.

“

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | > db.AtominMongoDB.findAndModify({  ... query:{\_id:101,item\_available:{$gt:0}},  ... update:{  ... $inc:{item\_available:-1},  ... $push:{item\_bought\_by:{customer:"CP Jain",date:"6-May-2017"}}  ... }  ... });  {  "\_id" : 101,  "item\_name" : "Apple iPhone 7",  "category" : "handset",  "warranty\_period" : 5,  "city" : "Toronto",  "country" : "Canada",  "item\_total" : 10,  "item\_available" : 6,  "item\_bought\_by" : [  {  "customer" : "Mohit",  "date" : "6-May-2017"  },  {  "customer" : "Aparajita",  "date" : "5-May-2017"  },  {  "customer" : "Anita",  "date" : "4-May-2017"  },  {  "customer" : "Abhishek",  "date" : "3-May-2017"  }  ]  }  > db.AtominMongoDB.find().pretty();  {  "\_id" : 101,  "item\_name" : "Apple iPhone 7",  "category" : "handset",  "warranty\_period" : 5,  "city" : "Toronto",  "country" : "Canada",  "item\_total" : 10,  "item\_available" : 5,  "item\_bought\_by" : [  {  "customer" : "Mohit",  "date" : "6-May-2017"  },  {  "customer" : "Aparajita",  "date" : "5-May-2017"  },  {  "customer" : "Anita",  "date" : "4-May-2017"  },  {  "customer" : "Abhishek",  "date" : "3-May-2017"  },  {  "customer" : "CP Jain",  "date" : "6-May-2017"  }  ]  }  > |

“

In the above document, we searched for the item, setting the ID as 101. If the system finds such a thing, we activate the subtraction function and deduct 1 in the item\_available field. We will also update the field item\_bought\_by in which we insert the name of the customer along with the purchase date.

Finally, we print the full information with the function, find and pretty method. We can see that the item\_available field will come down from 6 to 5 while adding the customer name and the purchase date in the item\_bought\_by field.

## One more example to make you more precise about the use of atomic operations in MongoDB

In the above case, we dealt mainly with the product order and the record keeping of customers. In this example, we will be using the function of the simple book store and make it work out in MongoDB via atomic operations.

Let’s suppose in that book store, and we need to maintain the record of books along with the number of copies available for checkout, including crucial details about checkout.

We should sync the number of copies available, and checkout information must for the program to work. We will be embedding the checkout and available field for ensuring that the two areas will be updated atomically.

“{  
 \_id: 123456789,  
 title: "MongoDB: The Definitive Guide",  
 author: [ "Kristina Chodorow", "Mike Dirolf" ],  
 published\_date: ISODate("2010-09-24"),  
 pages: 216,  
 language: "English",  
 publisher\_id: "oreilly",  
 available: 3,  
 checkout: [ { by: "joe", date: ISODate("2012-10-15") } ]  
}

“

Updating the checkout field with new information is essential. We will be using db.collection.updateOne() method for atomically updating available and checkout field.

“db.books.updateOne (  
 { \_id: 123456789, available: { $gt: 0 } },  
 {  
 $inc: { available: -1 },  
 $push: { checkout: { by: "abc", date: **new** Date() } }  
 }  
)

“

We have set the code in such a way that the operation will transfer a document that stores information, according to the process.

“{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

“

The matchedCount field is responsible for comparing the condition for updates. We can see that 1 document fulfilled the requirements due to which the operation updated 1 report.

There could also be the case where no documents are matched, according to the update condition. In that situation, both the matchedcount and modifiedcount field would be 0. What this means is that you will not be able to purchase the book and continue with a checkout process.

## My Final Say

Finally, we have finished the topic of how you can use atomic operations via MongoDB. It was not that difficult, was it? Although it is not possible to work out with multi-document tasks, but using atomic operations in MongoDB is simple. With that said, MongoDB starting from version 4.0 will be supporting atomic operations in numerous scenarios.

There are plenty of real-world issues where we can use an atomic operation like in purchase record. It will prevent mutual exclusions; hence, it will stop the corruption of data in many ways.

Take a close look at the source codes in the article and follow it systematically. After practicing for a few times, you can naturally apply the atomic operation in the real-world problems where needed.

Do you have any confusions? If yes, feel free to leave a comment below. We will reply to your comment for clearing out your confusions. In case you want to add more insights, you can put forward your opinion in the comment below.