MONGODB - ATOMIC OPERATIONS

 Copyright © tutorialspoint.com

MongoDB does not support **multi-document atomic transactions**. However, it does provide atomic operations on a single document. So if a document has hundred fields the update statement will either update all the fields or none, hence maintaining atomicity at the document-level.

Model Data for Atomic Operations

The recommended approach to maintain atomicity would be to keep all the related information, which is frequently updated together in a single document using **embedded documents**. This would make sure that all the updates for a single document are atomic.

Consider the following products document –

```
{
   "_id":1,
   "product_name": "Samsung S3",
   "category": "mobiles",
   "product_total": 5,
   "product_available": 3,
   "product_bought_by": [
      {
         "customer": "john",
         "date": "7-Jan-2014"
      },
      {
         "customer": "mark",
         "date": "8-Jan-2014"
      }
   ]
}
```

In this document, we have embedded the information of the customer who buys the product in the **product_bought_by** field. Now, whenever a new customer buys the product, we will first check if the product is still available using **product_available** field. If available, we will reduce the value of product_available field as well as insert the new customer's embedded document in the product_bought_by field. We will use **findAndModify** command for this functionality because it searches and updates the document in the same go.

```
>db.products.findAndModify({
   query:{_id:2,product_available:{$gt:0}},
   update:{
      $inc:{product_available:-1},
      $push:{product_bought_by:{customer:"rob",date:"9-Jan-2014"}}
   }
})
```

Our approach of embedded document and using findAndModify query makes sure that the product purchase information is updated only if it the product is available. And the whole of this transaction being in the same query, is atomic.

In contrast to this, consider the scenario where we may have kept the product availability and the information on who has bought the product, separately. In this case, we will first check if the product is available using the first query. Then in the second query we will update the purchase information. However, it is possible that between the executions of these two queries, some other user has purchased the product and it is no more available. Without knowing this, our second query will update the purchase information based on the result of our first query. This will make the database inconsistent because we have sold a product which is not available.