MongoDB ObjectId:

When we store a document in mongodb, mongodb sets the value of the _id property to a long string as below,

_id = “608be0aa98aed41808d91d72”

So, here we’ve 24 characters and every 2 characters represent a byte. So essentially we’ve 12 bytes to uniquely identify identify document in mongodb. So out of this 12 bytes first 4 bytes represent a timestamp. And that is the time this document was created. So, with this 4 bytes you don’t have to create a separate property in your document like createdAT(to store the time of document) because the timestamp is included in the objectId. By the same token if you wanna sort your documents based on their creation time. You can simply sort them by their id property.

// 12 bytes

// First 4 bytes: timestamp

// Next 3 byte: machine identifier (So, two different machine will have two different identifires)

// Next 2 byte: process identifier (So, if you generate two object ids on the same machine but in two different processes these 2 bytes will be different)

// Last 3 bytes: counter (If you're on the same machine in the same process at the same time but generate 2 different documents the couter byte will be different).

So, with this 12 bytes we can uniquely identify a document in mongodb. Having said that there are very very very low chance that we'll generate 2 object ids that are the same for 2 different documents.

Let’s see how this can happen?

In 1 byte we have 8 bits. In each bit we've either 0 or 1. So, how many numbers can we represent in 1 byte or 8 bits i.e.

2^8bits = 256 bits

So, with 1 byte we can store 256 different numbers. Now, the last 3 bytes represent a counter same as the counter in sql server or mysql databases where an id is automatically incremented by 1 in a table. So, auto incrementing number like 1, 2, 3, 4 and so on.

So, how many numbers can be stored in 3 bytes?

2^24bits = 16 Million

So, if at the same time on the same machine in the same process if we generate more than the 16 million documents then the counter will overflow and that's where we may end up with 2 documents with the same objectid. And this is a very unlikely scenario for most applications out there.

So what we can say is, the objectid i.e. "_id" created by mongodb are almost unique but not 100%.

Why we don't have a mechanism in mongodb that guarantees uniqueness?

Eg. in database management systems like sqlserver or mysql in each table we've auto incrementing number that guarantees uniqueness.

So, next time we wanna store a course recored in our database, the id of that course will be the id of the last course + 1. This approach guarantees the uniqueness of this identifiers but it hurts scalability. In mongodb this "_id" that we have above is not generate by the mongodb itself. Instead it is generated by the mongodb driver.

So, we've a mongodb driver that talks to the mongodb and that means we don't have to wait for mongodb to generate the new unique identifiers. And thats why applicatoins built on top of mongodb are highly scalable. We can have several instances of mongodb and we don't have to talk to a central place to get a unique identifier. The driver itself can generate almost unique identifier using these 12 bytes.

// Driver -> MongoDB

So, when we build an app using node and express we use mongoose and mongoose is a abstraction over mongodb driver. So, when we create a new object or new document, mongoose talks to a mongodb driver to generate a new id.

Now, we can also explicitely generate an id if we want to.

Eg.

```
const mongoose = require('mongoose');

const id = new mongoose.Types.ObjectId(); // generates 12 bytes id

console.log(id);


// returns the timestamp extracted from the id of a document. First 4 bytes
console.log(id.getTimestamp());


const isValid = mongoose.Types.ObjectId.isValid('1234'); // to validate mongoose object id

console.log(isValid);
```