MONGODB OVERVIEW: GETTING STARTED WITH MONGODB

Here we provide an overview of the MongoDB database. In subsequent posts we will give more in depth examples of how to use MongoDB.

First, MongoDB is a noSQL big data database. It fits the definition of **big data**, because it scales (i.e., can be made larger) simply by adding more servers to a distributed system. And it is does not require any schema, like an RDBMS database, such as Oracle.

MongoDB data records are stored in JSON (JavaScript Object Notation) format, which is self-describing, meaning the metadata (i.e., schema) is stored together with the data.

(This article is part of our MongoDB Guide. Use the right-hand menu to navigate.)

Command Line Shell

Mongo has an interactive command shell. JavaScript programmers will love this because the syntax is JavaScript. To open the shell you simply type:

```
mongo
```

Concepts

MongoDB records are called **documents**. Each MongoDB database (You can have many.) includes **collections**, which are a set of JSON documents. Each collection and document has an ObjectID created by MongoDB or supplied by the programmer.

To illustrate, suppose we have one database called **products**.

We could have two collections to contain all products grouping them by where they are sold:

Collection	Documents and fields
Products sold in Europe	EAN (European barcode), weight in kilos
Products sold in the USA	UPC (American barcode), weight in pounds

Data storage is cheap and memory and CPU costs more. So, some big data databases, like Cassandra and MongoDB, throw out the idea of a normalized database, which is one of the key principles of an RDBMS database.

For example, with Oracle you would have a product category in a product record. The product category table contains fields common to all of those products. Each product record points to a product category record, so that such common data is not stored more than once:

Product table

Fields: product number, product category

Product Category table

Fields: product category, weight, color

But then you have to do a **join** operation if you want to know the color or weight of a product. But a join is a computationally expensive operation. That takes time. MongoDB would store the data like this:

Product documents

Fields: product number, product category, weight, color

RDBMS programmers say that creates duplication and wastes space. MongoDB programmers would say "yes," but speed is more important than storage.

In other words, MongoDB records might look like this:

Product 1	Category boy's diapers small	Color blue
Product 2	Category boy's diapers large	Color blue
Product 3	Category girl's diapers small	Color pink

Obviously. When you know the category you know the color.

We will illustrate that by creating the products database and adding some products there. Paste these commands into the MongoDB shell.

First create the products database.

```
use products
switched to db products
```

Then these two collections:

```
> db.createCollection("boyDiapers")
{ "ok" : 1 }
> db.createCollection("girlDiapers")
{ "ok" : 1 }
```

>

Then add some data:

```
db.boyDiapers.insert()
db.girlDiapers.insert()
```

Notice two things. First, we use the format db.(collection).insert to add the document. Second, we use the brackets [], which indicates an array, do that we can add more than one document at a time.

Now create some more data so that we can query for data:

```
db.boyDiapers.insert()
db.girlDiapers.insert()
```

Selecting Data

If you use **find** with no arguments it lists all documents. Use **pretty** to display the results in easy-to-read indented JSON format:

```
> db.girlDiapers.find().pretty()
{
"_id" : ObjectId("59d1e9d5ccf50b62c5a7af55"),
"size" : 1,
"color" : "pink",
"brand" : "little angel"
}
{
"_id" : ObjectId("59d1f022ccf50b62c5a7af57"),
"size" : 1,
"color" : "while",
"brand" : "girl large"
}
{
"_id" : ObjectId("59d1f565ccf50b62c5a7af59"),
"size" : 2,
"color" : "while",
"brand" : "girl large"
}
```

Find all girl diapers of size 2 add arguments to the find statement:

```
db.girlDiapers.find({"size":2})
{ "_id" : ObjectId("59d1f565ccf50b62c5a7af59"), "size" : 2, "color" :
"while", "brand" : "girl large" }
```

Now, you could not search both boy's and girl's diapers collections at the same time. MongoDB does not do that. Instead you have to program that in your application that you would code using some driver (See below).

Normalized Documents

We just said that in MongoDB there is no normalization because storage is cheap and computational power expensive. But you can create normalize documents.

For example we can create a sales record for each size 2 girl large document like this with the **diaper** field pointing to the diaper object. That might make more sense in this case as you would not want the diaper collection to grow many times larger each time you make a sale.

```
db.girlDiapers.insert()
```

MongoDB Drivers

Of course, you probably would not use the command line shell for an application. Instead you would write a program to interact with MongoDB using any of the many drivers available. There are drivers for C++, C#, Java, Node.JS, Scala, Python, and more.

For example, to use Python:

```
sudo pip install pymongo
```

Then to query for size 2 diapers across the boy and girl collections:

```
from pymongo import MongoClient
client = MongoClient()
db = client.products
x=db.collection_names()
for i in range(len(x)):
c=x
d = db.get_collection(c)
for e in d.find({"size": 2}):
print(e)
```

Outputs:

```
{'size': 2.0, 'brand': 'boy large white', 'color': 'white', '_id':
ObjectId('59d1f564ccf50b62c5a7af58')}
{'size': 2.0, 'brand': 'girl large', 'color': 'while', '_id':
ObjectId('59d1f565ccf50b62c5a7af59')}
```

In the next post we will get into some more advanced MongoDB topics.