**MongoDB**

# Introduction

Much of the new data (tweets/facebook) is unstructured and semi-structured, so developers also need a database that can efficiently store it.

Unfortunately, the rigidly defined schema-based approach used by relational databases makes it impossible to quickly incorporate new types of data and is a poor fit for unstructured and semi-structured data.

## Advantages

The label NoSQL covers a diverse collection of databases that tend to have at least two elements in common: distributed computing architectures and schemaless design.

They're flexible because, unlike relational databases, NoSQL databases don't require a predefined schema (a.k.a. data model) that demands one way to manage data in columns and rows.

* Massive write performance
* Fast key value access
* Flexible schema
* Easy scaling out/support to cloud
* Sharding - In case of heavy load on database, it can distribute to other nodes easily.
* Supports map reduce for batch processing of huge data.
* Supports server side javascript

## Disadvantage

there are drawbacks to NoSQL, most notably the lack of SQL querying capabilities and ACID (atomic, consistent, isolated and durable) performance.

# Get Started

**download and install mongo db from** <http://www.mongodb.org/downloads?_ga=1.128364879.231085771.1424278489>

**Configure mongo db**

MongoDB requires a [data directory](http://docs.mongodb.org/manual/reference/glossary/#term-dbpath) to store all data. MongoDB’s default data directory path is\data\db. Create this folder using the following commands from a **Command Prompt**:

md \data\d

or you can specify the alternative path using below command -

C:\<InstalledPath>\mongodb\bin\mongod.exe --dbpath d:\test\mongodb\data

# Basic Commands

**Warning : Its case sensitive**

Open command prompt and run below command -

C:\<InstallPath>\mongodb\bin\mongod.exe

This will start our mongo db server. Again open another command prompt and run below command

C:\<installedPath>\mongodb\bin\mongo.exe

It will start the mongo db shell with which we can execute the mongo db commands

fire below command at the mongo shell

* **help**
* **show dbs -** list down all the databases
* **db -** display the current database name.
* **use synechrondb -** will create a new database named as synechrondb

## Insert a collection in database

data is stored in a collection into mongo db in json format. We can insert as many documents into the collection. eg

j = {Name:"Gopal", Department:"APAC", Salary:500}

now insert this document into the database.

db.Employees.insert(j)

The above command will create an 'Employees' collection (if it is not present) and will insert the document in it.

Add few more document/records -

x = {Name:"Vijay", Department:"APAC", Salary:800}

db.Employees.insert(j)

We can create multiple collections and can see the available collections using below command -

**List all the collections present in the database**

db.getCollectionNames()

**List all the documents (or collections) present in a collections**

We can find the document from the collection as below -

db.Employess.find()

**.find() - with predicate**

db.Employees.find({Name:"Gopal"})

it will return the document with this matching key.

MongoDB is case sensitive ie column name 'name' & 'Name' are not same. And the value 'gopal' is not equal to "Gopal"

Let's revisit the id column from the output,later.

**Remove document**

we can remove the specific collection using the predicate

db.<collectionName>.remove({columnName:ValueName})

db.Employees.remove({Name:"Gopal"})

Check whether the document is really removed or not

db.Employees.find()

or can remove all the documents

db.<collectionName>.remove({})

**Null Values**

db.Employees.find( { Salary: **null** } ) // to check not null use exists instead of null

## CRUD Operations

db.Employees.update

(

{

Name:"Gopal"

},

{

$set:

{

Salary:22000

}

}

)

**Indexes**

**Default \_id -** All MongoDB collections have an index on the \_id field that exists by default. If applications

do not specify a value for \_id the driver or the [mongod](http://docs.mongodb.org/manual/reference/program/mongod/" \l "bin.mongod" \o "mongod) will create an \_id field with an [ObjectId](http://docs.mongodb.org/manual/reference/glossary/" \l "term-objectid) value.

The \_id index is unique, and prevents clients from inserting two documents with the same value for the\_id field.

**Single Field Index**

> db.Employees.ensureIndex({"Name":1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

Multi column index

> db.Employees.ensureIndex({"Name":1,"Salary":1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 2,

"numIndexesAfter" : 3,

"ok" : 1

}

**Aggregation**

> db.Employees.aggregate

(

[

{$match:{Department:"APAC"}},

{$group:{\_id:"$Department", total:{$sum:"$Salary"}}}

]

)

{ "\_id" : "APAC", "total" : 28600 }

Grouping on multiple column

> db.Employees.aggregate([{$match:{Department:"APAC"}},{$group:{\_id:{Department:

"$Department",Name:"$Name"}, total:{$sum:"$Salary"}}}])

{ "\_id" : { "Department" : "APAC", "Name" : "Vijay" }, "total" : 6600 }

{ "\_id" : { "Department" : "APAC", "Name" : "Gopal" }, "total" : 22000 }

>

db.orders.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$cust\_id", total: { $sum: "$amount" } } },

{ $sort: { total: -1 } } // -1 is desc order

])

We can use the other aggregation operators like min, max

> db.Employees.find()

{ "\_id" : ObjectId("54eb544fa2ff2093a7c3e54f"), "Name" : "Gopal", "Department" :

"APAC", "Salary" : 22000 }

{ "\_id" : ObjectId("54eb59b2a2ff2093a7c3e550"), "Name" : "Vijay", "Department" :

"APAC", "Salary" : 6600 }

{ "\_id" : ObjectId("54eb59daa2ff2093a7c3e551"), "Name" : "Mahesh", "Department"

: "OTH", "Salary" : 75000 }

> ***db.Employees.aggregate([{$match:{}},{$group:{\_id:{}, MinimumSalary:{$min:"$Sal***

***ary"}}}])***

{ "\_id" : { }, "MinimumSalary" : 6600 }

>

**Distinct**

> db.Employees.distinct("Name")

[ "Gopal", "Vijay" ]

## Users & RoleManagement

**Create a new user**

> use admin

switched to db admin

> db.createUser({user:"tempUser",pwd:"tempUser",roles:[]})

Successfully added user: { "user" : "tempUser", "roles" : [ ] }

**Get the user**

> db.getUser("tempUser")

{

"\_id" : "admin.tempUser",

"user" : "tempUser",

"db" : "admin",

"roles" : [ ]

}

Below document explain the roles management

[http://docs.mongodb.org/manual/reference/privilege-actions/#authr.grantRole](http://docs.mongodb.org/manual/reference/privilege-actions/%23authr.grantRole)

# Transactions

MongoDB does not support multi-document transactions, but provides atomic operations on a single document.

## Joins

MongoDB does not support joins.

MongoDB applications use one of two methods for relating documents:

* [Manual references](http://docs.mongodb.org/manual/reference/database-references/#document-references) where you save the \_id field of one document in another document as a reference. Then your application can run a second query to return the related data. These references are simple and sufficient for most use cases.
* [DBRefs](http://docs.mongodb.org/manual/reference/database-references/#dbref-explanation) are references from one document to another using the value of the first document’s \_idfield, collection name, and, optionally, its database name. By including these names, DBRefs allow documents located in multiple collections to be more easily linked with documents from a single collection.

To resolve DBRefs, your application must perform additional queries to return the referenced documents. Many [drivers](http://docs.mongodb.org/manual/applications/drivers/) have helper methods that form the query for the DBRef automatically. The drivers [[1]](http://docs.mongodb.org/manual/reference/database-references/#official-driver) do not automatically resolve DBRefs into documents.

DBRefs provide a common format and type to represent relationships among documents. The DBRef format also provides common semantics for representing links between documents if your database must interact with multiple frameworks and tools.

comments

{ uid:12345, pid:444, comment="blah" }

{ uid:12345, pid:888, comment="asdf" }

{ uid:99999, pid:444, comment="qwer" }

users

{ uid:12345, name:"john" }

{ uid:99999, name:"mia" }

## Import/Export

We can use mongoexport & mongoimport utilities for this.

**Note :**

1. These utilities are available on server command prompt and not on shell. If ther server is not running then open the command prompt and execute below command -

C:\<InstallPath>\mongodb\bin\mongod.exe

**Export**

C:\Program Files\MongoDB 2.6 Standard\bin>mongoexport --db Test2DB --collection

employeeList --out E:/Study/nodejs/Code/Test2/data/emplist.json

connected to: 127.0.0.1

exported 2 records

**Import**

C:\Program Files\MongoDB 2.6 Standard\bin>mongoimport --db Test2DB --collection

employeeList

connected to: 127.0.0.1

{Name:"Dhirendra", Department:"TIBCO", Salary:3500}

^C

C:\Program Files\MongoDB 2.6 Standard\bin>

now go to shell and find the collection

> db.employeeList.find()

## Other

**MapReduce is not covered in the demo. Please visit below link**

<http://blog.knoldus.com/2013/02/03/joins-now-possible-in-mongodb-2-4/>

**mongoose  - is mongo db driver for windows**

<http://www.w3resource.com/mongodb/installation-windows.php>

**3rd party Admin tools -**

<http://docs.mongodb.org/ecosystem/tools/administration-interfaces/>

**When to use GridFS?**

For documents in a MongoDB collection, you should always use [GridFS](http://docs.mongodb.org/manual/reference/glossary/" \l "term-gridfs) for storing files larger than 16 MB.

**MongoDB as Windows Service**