**MongoDB**

A quick reference guide for MongoDB. This provide Utilities, CRUD syntax, quick reference on concepts replication and sharding concepts, and how to import data from RDBMSs to MongoDB. This quick reference is for Software engineers, and hands on Architects. The reference is built by using MongoDB site and a book ‘MongoDB applied design patterns’. Currently only the SPARK code is on Github @ <https://github.com/cbdesai/MongoMDMViewer> , eventually more samples will be uploaded on Github.

# Install, Configuration and interaction:

1. Download just the community server from <https://www.mongodb.com/download-center?jmp=nav#community>. If on windows it’s a msi so you can click and install. For Lynux, OSx check the instructions.
2. Create directory data/db under your root “c:”
3. Download and install MangoChef: <http://3t.io/mongochef/download/> It’s a msi for windows. So its click and install.
4. You can also try Robomongo and Mongobooster as a client.
5. Mongo has its own shell you can use, you can use intellij and create java program, .net…

*If you have worked with ORM like Hibernet, or Object database like Versant, you will see lot of similarity in concepts. Folloiwing table gives a quick reference for basic operations along with examples:*

1. *Start mongod its under bin directour e.g. c:\program file\mongo\server\3.5\bin*

# Utilities:

|  |  |  |
| --- | --- | --- |
| **Utility** | **Syntax** | **comments** |
| serverStatus | db.serverStatus() |  |
| dbStats | db.stats() |  |
| CollStats | db.collection.stats() e.g. db.users.stats() |  |
| Replicasetstatus | rs.status() |  |
| Processlogging | mongod -v --logpath /var/log/mongodb/server1.log –logappend |  |
| Ganglia |  | mongodb-ganglia |
| Motop |  |  |
| Nagios |  |  |
| Scout |  |  |
| New Relic |  |  |
| Datadog |  |  |

# [quickie](https://www.google.com/search?q=meaning%20of%20quickie&start=0&spell=1) :

Replication: A replica set in MongoDB is a group of mongod processes that maintain the same data set. The data is maintained over multiple node and if required multiple data centers. Data is sncyonized /copied to replica by the mongo processes.

|  |  |
| --- | --- |
| **Master Slave** | **Replication** |
| mongod --master --dbpath /data/masterdb/  mongod --slave --source <masterhostname><:<port>> --dbpath /data/slavedb/  rs.printReplicationInfo()  rs.printSlaveReplicationInfo() | 1. mongod --dbpath /data/masterdb/ 2. mongod --replSet "rs0" 3. mongod --config $HOME/.mongodb/config 4. mongo 5. rs.initiate() 6. rs.conf() 7. rs.add("mongodb1.example.net") 8. rs.add("mongodb2.example.net") 9. rs.status() |

Sharding: Concept similar to MySQL Sharding. Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

|  |  |
| --- | --- |
| **Concept** | **Description** |
| **Shard Keys:** | To distribute the documents in a collection, MongoDB partitions the collection using the shard key. The shard key consists of an immutable field or fields that exist in every document in the target collection. |
| **Hashed Sharding:** | Hashed Sharding involves computing a hash of the shard key field’s value.  Ranged Sharding: Ranged sharding involves dividing data into ranges based on the shard key values. Each chunk is then assigned a range based on the shard key values. |
| **Tag Aware Sharding:** | In sharded clusters, you can tag specific ranges of the shard key and associate those tags with a shard or subset of shards. |
| **A MongoDB shard cluster consists of the following components:** | |
| **Shard:** | Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set. |
| **Mongos:** | The mongos acts as a query router, providing an interface between client applications and the sharded cluster. |
| **Config servers:** | Config servers store metadata and configuration settings for the cluster. As of MongoDB 3.2, config servers can be deployed as a replica set. |

Ref: <https://docs.mongodb.com/manual/tutorial/deploy-sharded-cluster-hashed-sharding/> for shard configuration.

# CRUD & aggregates:

## Simple CRUD

|  |  |  |  |
| --- | --- | --- | --- |
| **Usage** | **Syntax** | **Example** | **Comments** |
| Insert Data | db.collection.insert() | 1. db.users.insert **(**   **{**  name **:** "Joe Smith"**,**  address **:** **{**  street**:** "123 South Street"**,**  city **:** "Oaktown"**,**  state **:** "IL"**,**  zip **:** "12345"  **}**  **}**  **)**   1. db.users.insert({   "name" : "Bill Murry",  "age" : 42,  "status" : "A",  "income" : "100000"  }) | Create some more like this |
| Read/Find | db.collection.find() | 1. db.users.find() 2. db.users.find({name:"Joe Smith"}) 3. db.users.find({name:"Joe Smith"}, {name:1, "address.zip":1}) 4. db.users.find(   { $text: { $search: "Smith", $language: "es" } } | For text search make sure you have index   1. 1 and 0 ascending decending 2. db.getCollection("users").createIndex({"name" : "text"}) |
| Update | db.collection.update() | 1. db.users.update({name:"Joe Smith"},   {$set:{"address.zip":"600015"}})   1. db.users.updateMany({age: {$gt:23}}, {$set:{"income":"100000"}}) |  |
| Delete/Remove | db.collection.remove() | db.users.remove({"address.zip":"600035"}) |  |
|  |  |  |  |
| Aggregates | db.collection.aggregates() | db.users.aggregate([ {$match: {status:"A"}},{ $group: {\_id: "$status", total:{$sum: 1}}}]) | Will give you a aggregated count |

## Crud with Arrays:

|  |  |  |  |
| --- | --- | --- | --- |
| **Usage** | **Syntax** | **Example** | **Comments** |
| Multiple Inserts | db.users.insertMany( [ ] ) | db.users.insertMany(  [  { name: "bob", age: 42, status: "A", },  { name: "ahn", age: 22, status: "A", },  { name: "xi", age: 34, status: "D", }  ]  ) |  |
| Multiple Find | db.users.find (  {  field:{ $in:[]  }  }  ) | db.users.find (  {  name:  { $in:["Joe Bookreader", "Joe Smith"]}  }  ) |  |

## Curser:

|  |  |  |
| --- | --- | --- |
| **Usage** | **Example** | **Comments** |
| Curser | var myCursor = db.users.find({age: { $gt: 10} } );  while (myCursor.hasNext()) {  print(tojson(myCursor.next()));  } |  |
| Curser to Array | var myCursor = db.users.find({age: { $gt: 10} } );  var docArray = myCursor.toArray();  print (docArray.length) |  |

# Data Model example and Patterns:

|  |  |  |
| --- | --- | --- |
| **Model Type** | **Example** | |
| Type1: External: | One-One | |
| db.customer.insert ({\_id: "joe",name: "Joe Bookreader"}) | db.address.insert({patron\_id: "joe",street: "123 Fake Street",city: "Faketon",state: "MA",zip: "12345"}) |
|  | One-One | One-Many |
| Type2: Embedded: | db.customercomplete.insert({\_id: "joe",name: "Joe Bookreader",address: {street: "123 Fake Street",city: "Faketon",state: "MA",zip: "12345"}}) | **db.customermulticomplete.insert(**  **{**  **\_id: "joe",**  **name: "Joe Bookreader",**  **addresses: [**  **{**  **street: "123 Fake Street",**  **city: "Faketon",**  **state: "MA",**  **zip: "12345"**  **},**  **{**  **street: "1 Some Other Street",**  **city: "Boston",**  **state: "MA",**  **zip: "12345"**  **}**  **]**  **})** |
| **type 3: Master -detail with fixed/known array like order details** | db.publisher.insert({name: "O'Reilly Media",founded: 1980,location: "CA",books: [12346789, 234567890]}) | db.books.insert([{\_id: 123456789,title: "MongoDB: The Definitive Guide",author: [ "Kristina Chodorow", "Mike Dirolf" ],published\_date: ISODate("2010-09-24"),pages: 216,language: "English"},{\_id: 234567890,title: "50 Tips and Tricks for MongoDB Developer",author: "Kristina Chodorow",published\_date: ISODate("2011-05-06"),pages: 68,language: "English"}]) |
| **type 4: Master -detail with growing and mutating array like employee in organization** | db.publisher.insert({{\_id: "oreilly",name: "O'Reilly Media",founded: 1980,location: "CA"}); | db.books.insert([{\_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"},{\_id: 234567890,title: "50 Tips and Tricks for MongoDB Developer",author: "Kristina Chodorow",published\_date: ISODate("2011-05-06"),pages: 68,language: "English",publisher\_id: "oreilly"}]); |
| **Type 5a: Making a Tree Structure like Graph Database** | db.categories.insert( { \_id: "MongoDB", parent: "Databases" } )  db.categories.insert( { \_id: "dbm", parent: "Databases" } )  db.categories.insert( { \_id: "Databases", parent: "Programming" } )  db.categories.insert( { \_id: "Languages", parent: "Programming" } )  db.categories.insert( { \_id: "Programming", parent: "Books" } )  db.categories.insert( { \_id: "Books", parent: null } ) | db.categories.createIndex( { parent: 1 } );  db.categories.findOne( { \_id: "MongoDB" } ).parent  db.categories.find( { parent: "Databases" } )  p.s.: There is nothing in the database it’s the data that is establishing hierarchy. In practical situation all these tree structure can mutate and cause developer errors. |
| **Type 5B: Making a Tree Structure like Graph Database** | db.categories.insert( { \_id: "MongoDB", ancestors: [ "Books", "Programming", "Databases" ], parent: "Databases" } )  db.categories.insert( { \_id: "dbm", ancestors: [ "Books", "Programming", "Databases" ], parent: "Databases" } )  db.categories.insert( { \_id: "Databases", ancestors: [ "Books", "Programming" ], parent: "Programming" } )  db.categories.insert( { \_id: "Languages", ancestors: [ "Books", "Programming" ], parent: "Programming" } )  db.categories.insert( { \_id: "Programming", ancestors: [ "Books" ], parent: "Books" } )  db.categories.insert( { \_id: "Books", ancestors: [ ], parent: null } ) | db.categories.createIndex( { ancestors: 1 } )  db.categories.findOne( { \_id: "MongoDB" } ).ancestors  db.categories.find( { ancestors: "Programming" } ) |
| **Type 6: Master details with event recording example ariline and cruse ship seats or library books** | db.bookinlibrary.insert(  {  \_id: 123456789,  title: "mongodb: the definitive guide",  author: [ "kristina chodorow", "mike dirolf" ],  published\_date: new ISODate("2010-09-24"),  pages: 216,  language: "english",  publisher\_id: "oreilly",  available: 3,  checkout: [ { by: "joe", date: new ISODate("2012-10-15") } ]  }); | db.books.update (  { \_id: 123456789, available: { $gt: 0 } },  {  $inc: { available: -1 },  $push: { checkout: { by: "abc", date: new Date() } }  }  ) |
| **Type 7: buckets and categories** | db.bookinshelves.insert(  { title : "Moby-Dick" ,  author : "Herman Melville" ,  published : 1851 ,  ISBN : 0451526996 ,  topics : [ "whaling" , "allegory" , "revenge" , "American" ,  "novel", "nautical" , "voyage" , "Cape Cod" ]  }); | db.volumes.createIndex( { topics: 1 } )  db.volumes.findOne( { topics : "voyage" }, { title: 1 } ) |

## 

## 

# Indexes:

|  |  |  |
| --- | --- | --- |
| **Index Name** | **Collection** | **Index Syntax** |
| Single Field | {  "\_id": ObjectId("570c04a4ad233577f97dc459"),  "score": 1034,  "location": { state: "NY", city: "New York" }  } | db.records.createIndex( { score: 1 } )  db.records.find( { score: 2 } )  db.records.find( { score: { $gt: 10 } } ) |
| Embedded Field |  | db.records.createIndex( { "location.state": 1 } )  db.records.find( { "location.state": "CA" } )  db.records.find( { "location.city": "Albany", "location.state": "NY" } ) |
| Embedded Document |  | db.records.createIndex( { location: 1 } )  db.records.find( { location: { city: "New York", state: "NY" } } ) |
| Compound Index | {  "\_id": ObjectId("9560c04a4ad233577f97dc459"),  "item": "Banana",  "category": ["food", "produce", "grocery"],  "location": "4th Street Store",  "stock": 4,  "type": "cases"  } | db.products.createIndex( { "item": 1, "stock": 1 } )  db.products.find( { item: "Banana" } )  db.products.find( { item: "Banana", stock: { gt: 5 } } ) |
| Multi-key index | { \_id: 5, type: "food", item: "aaa", ratings: [ 5, 8, 9 ] }  { \_id: 6, type: "food", item: "bbb", ratings: [ 5, 9 ] }  { \_id: 7, type: "food", item: "ccc", ratings: [ 9, 5, 8 ] }  { \_id: 8, type: "food", item: "ddd", ratings: [ 9, 5 ] }  { \_id: 9, type: "food", item: "eee", ratings: [ 5, 9, 5 ] } | db.inventory.createIndex( { ratings: 1 } )  db.inventory.find( { ratings: [ 5, 9 ] } ) |
| Index Arrays with Embedded Documents | {  \_id: 1,  item: "abc",  stock: [  { size: "S", color: "red", quantity: 25 },  { size: "S", color: "blue", quantity: 10 },  { size: "M", color: "blue", quantity: 50 }  ]  }  {  \_id: 2,  item: "def",  stock: [  { size: "S", color: "blue", quantity: 20 },  { size: "M", color: "blue", quantity: 5 },  { size: "M", color: "black", quantity: 10 },  { size: "L", color: "red", quantity: 2 }  ]  } | db.inventory.createIndex( { "stock.size": 1, "stock.quantity": 1 } )  db.inventory.find( { "stock.size": "M" } )  db.inventory.find( { "stock.size": "S", "stock.quantity": { $gt: 20 } } )  db.inventory.find( ).sort( { "stock.size": 1, "stock.quantity": 1 } )  db.inventory.find( { "stock.size": "M" } ).sort( { "stock.quantity": 1 } ) |
| Compound Bounds of Non-array Field and Field from an Array | {  \_id: 1,  item: "ABC",  ratings: [ { score: 2, by: "mn" }, { score: 9, by: "anon" } ]  }  {  \_id: 2,  item: "XYZ",  ratings: [ { score: 5, by: "anon" }, { score: 7, by: "wv" } ]  } | db.survey2.createIndex( { "item": 1, "ratings.score": 1, "ratings.by": 1 } )  db.survey2.find( { item: "XYZ", "ratings.score": { $lte: 5 }, "ratings.by": "anon" } |
| Compound $elemMatch | {  \_id: 1,  item: "ABC",  ratings: [  { score: { q1: 2, q2: 5 }, certainty: { q1: 2, q2: 3 } },  { score: { q1: 8, q2: 4 }, certainty: { q1: 10, q2: 10 } }  ]  }  {  \_id: 2,  item: "XYZ",  ratings: [  { score: { q1: 7, q2: 8 }, certainty: { q1: 5, q2: 5 } },  { score: { q1: 9, q2: 5 }, certainty: { q1: 7, q2: 7 } }  ]  } | db.survey4.createIndex( {  "ratings.score.q1": 1, "ratings.score.q2": 1,  "ratings.certainty.q1": 1, "ratings.certainty.q2": 1  } )  db.survey4.find(  {  "ratings.score": { $elemMatch: { q1: 5, q2: 5 } },  "ratings.certainty": { $elemMatch: { q1: 7, q2: 7 } },  }  ) |
| Text Index | db.messages.insert({"subject":"Joe owns a dog", "content":"Dogs are man's best friend", "likes": 60, "year":2015, "language":"english"})  db.messages.insert({"subject":"Dogs eat cats and dog eats pigeons too", "content":"Cats are not evil", "likes": 30, "year":2015, "language":"english"})  db.messages.insert({"subject":"Cats eat rats", "content":"Rats do not cook food", "likes": 55, "year":2014, "language":"english"}) | db.messages.createIndex({"subject":"text","content":"text"})  db.messages.find({$text: {$search: "cat"}}, {score: {$meta: "textScore"}}).sort({score:{$meta:"textScore"}}) |

|  |  |  |
| --- | --- | --- |
| **Index Properties** | **Collection** | **Index Syntax** |
| TTL | db.log\_events.insert( {  "createdAt": new Date(),  "logEvent": 2,  "logMessage": "Success!"  } ) | db.log\_events.createIndex( { "createdAt": 1 }, { expireAfterSeconds: 3600 } )  p.s. The documents themselves start to expire not the Index. |
| Unique |  | db.members.createIndex( { "user\_id": 1 }, { unique: true } )  Same as RDBMS will not let you insert record for the unique id. |
| Partial |  | db.restaurants.createIndex(  { cuisine: 1, name: 1 },  { partialFilterExpression: { rating: { $gt: 5 } } }  )  Only partial set of documents are indexed. e.g no null , gt 5 |
| sparse |  | db.scores.createIndex( { score: 1 } , { sparse: true } )  Index document only if the field is present. Field can be null or non null |

# Sample data Models

## A product Data Model: Record Album:

db.recordalbum.insert(

**{**

"\_id" **:** ObjectId**(**"577c7870dacd5ae163f162e5"**),**

"sku" **:** "00456"**,**

"type" **:** "Audi Album"**,**

"title" **:** "A Love Supreme"**,**

"description" **:** "by John Clotrane"**,**

"asin" **:** "800000A118M"**,**

"shipping" **:** **{**

"weight" **:** 6.0**,**

"dimension" **:** **{**

"width" **:** 10.0**,**

"height" **:** 10.0**,**

"depth" **:** 1.0

**}**

**},**

"pricing" **:** **{**

"list" **:** 1200.0**,**

"retail" **:** 1100.0**,**

"savings" **:** 100.0**,**

"pct\_savings" **:** 8.0

**},**

"details" **:** **{**

"title" **:** " A Love Supreme [Original Recording Reissued]"**,**

"artist" **:** "John Coltrane"**,**

"genre" **:** **[**

"Jazz"**,**

"General"

**],**

"tracks" **:** **[**

"A Love Supreme, Part I: Acknowledgment"**,**

"A Love Supreme, Part II: Resolution"**,**

"A Love Supreme, Part III: Pursuance"**,**

"A Love Supreme, Part iV: Plasm"

**]**

**}**

**});**

## A Social network Data Model:

**users:**

**{**

"\_id" **:** "T4Y2345AC"**,** "name" **:** "Rick"**,**

"profile" **:** **{** "age" **:** NumberInt**(**23**),** "location" **:** "Chicago"**,** "intrest" **:** "Table Tenis" **},**

"followers" **:** **{**

"T4Y234567AC" **:** **{** "name" **:** "Jared"**,** "circle" **:** **[** "Python"**,** "Author" **]** **},**

"T4Y234697AJ" **:** **{** "name" **:** "Bernie"**,** "circle" **:** **[** "Python" **]** **},**

"T4Y234856AB" **:** **{** "name" **:** "Jared"**,** "circle" **:** **[** "Python"**,** "Speaker" **]** **}** **},**

"circles" **:** **{**

"10gen" **:** **{**

"T4Y05678AD" **:** **{** "name" **:** "Jared"**,** "Age" **:** NumberInt**(**24**)** **},**

"T4Y30245AE" **:** **{** "name" **:** "Max"**,** "Age" **:** NumberInt**(**34**)** **},**

"T4Y98564AF" **:** **{** "name" **:** "Bernie"**,** "Age" **:** NumberInt**(**14**)** **},**

"T4Y3689AH" **:** **{** "name" **:** "Paul"**,** "Age" **:** NumberInt**(**20**)}**

**}**

**},**

"blocked" **:** **{**

"GH1056578e" **:** **{** "name" **:** "John"**,** "circle" **:** **[** "Fiction"**,** "Author" **]** **},**

"JN1058880n" **:** **{** "name" **:** "James"**,** "circle" **:** **[** "NonFiction"**,** "Author" **]** **},**

"LM1056990d" **:** **{** "name" **:** "Rex"**,** "circle" **:** **[** "Children's books"**,** "Author" **]** **}**

**}**

**}**

**posts**

**{**

"\_id" **:** ObjectId**(**"577d8c0c7588fe159cd2ed7e"**),**

"by" **:** **{** "id" **:** "T11111BC"**,** "name" **:** "Rick" **},**

"circles" **:** **[** "10gen"**,** "11gen" **],**

"type" **:** "Status"**,**

"ts" **:** ISODate**(**"2016-07-06"**),**

"details" **:** **{** "test" **:** "Loving MondgoDB"**,** "icon" **:** "Thmbsup" **},**

"comments" **:** **[**

**{**"by"**:{** "id"**:**"T4Y05678AD"**,**"name"**:**"Jared"**},**"ts"**:**ISODate**(**"2015-01-10"**),**"text"**:**"Right on"**},**

**{**"by"**:{**"id"**:**"T4Y30245AE"**,**"name"**:**"Max"**},**"ts"**:**ISODate**(**"2015-02-11"**),**"text"**:**"Liekit"**},**

**{**"by"**:{**"id"**:**"T4Y98564AF"**,**"name"**:**"Bernie"**},**"ts"**:**ISODate**(**"2015-03-15"**),** "text"**:**"You know" **},**

**{**"by"**:{**"id"**:**"T4Y3689AH"**,** "name" **:** "Paul" **},** "ts"**:**ISODate**(**"2015-04-01"**),** "text" **:** "Agree" **}**

**]**

**}**

**walls**

**{**

"\_id"**:**ObjectId**(**"577d973b7588fe159cd2ed84"**),**

"user\_id"**:**"T11111BC"**,**"month"**:**"201412"**,**

"posts"**:[**

**{**

"id"**:**ObjectId**(**"577d8c0c7588fe159cd2ed7e"**),**"ts"**:**ISODate**(**"2016-07-03"**),**

"by"**:{**"id"**:**"T4Y30245AE"**,**"name"**:**"Max"**},**

"circles"**:[**"\*public\*"**],**

"type"**:**"status"**,**

"detail"**:{**"text"**:**"Loving MongoDB"**},**

"comments\_shown"**:**NumberInt**(**3**),**

"comments"**:[**

**{**"by"**:{**"id"**:**"T4Y30245AE"**,**"name"**:**"Max"**},**"ts"**:**ISODate**(**"2015-02-"**),**"text"**:**"Likeit"**},**

**{**"by"**:{**"id"**:**"T4Y98564AF"**,**"name"**:**"Bernie"**},**"ts"**:**ISODate**(**"2015-03-15"**),**"text"**:**"Youknow"**},**

**{**"by"**:{**"id"**:**"T4Y3689AH"**,**"name"**:**"Paul"**},**"ts"**:**ISODate**(**"2015-04-01"**),**"text"**:**"Agree"**}**

**]**

**},**

**{**

"id"**:**ObjectId**(**"577d8c0c7588fe159cd2ed8e"**),**"ts"**:**ISODate**(**"2016-07-06"**),**

"by"**:{**"id"**:**"T4Y30245AE"**,**"name"**:**"Max"**},**

"circles"**:[**"\*public\*"**],**

"type"**:**"status"**,**

"detail"**: {**"text"**:**" Loving Python"**},**

"comments\_shown"**:**NumberInt**(**3**),**

"comments"**:[**

**{**"by"**:{**"id"**:**"T4Y30245AE"**,**"name"**:**"Max"**},**"ts"**:**ISODate**(**"2015-02-01"**),** "text"**:**"Vacations?"**},**

**{**"by"**:{**"id"**:**"T4Y98564AF"**,**"name"**:**"Bernie"**},**"ts"**:**ISODate**(**"2015-03015"**),**"text"**:**"See you"**},**

**{**"by"**:{**"id"**:**"T4Y3689AH"**,**"name"**:**"Paul"**},**"ts"**:**ISODate**(**"2015-04-01"**),**"text"**:**"Fasitivty"**}**

**]**

**}**

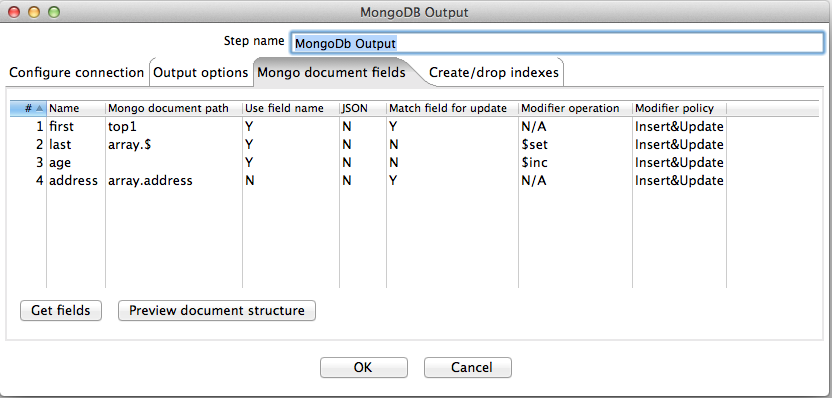
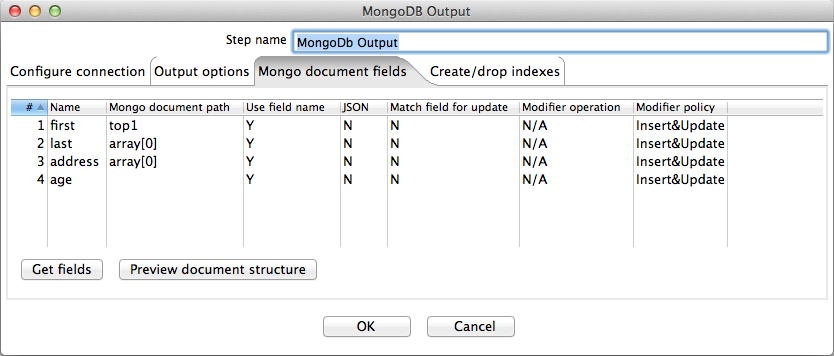
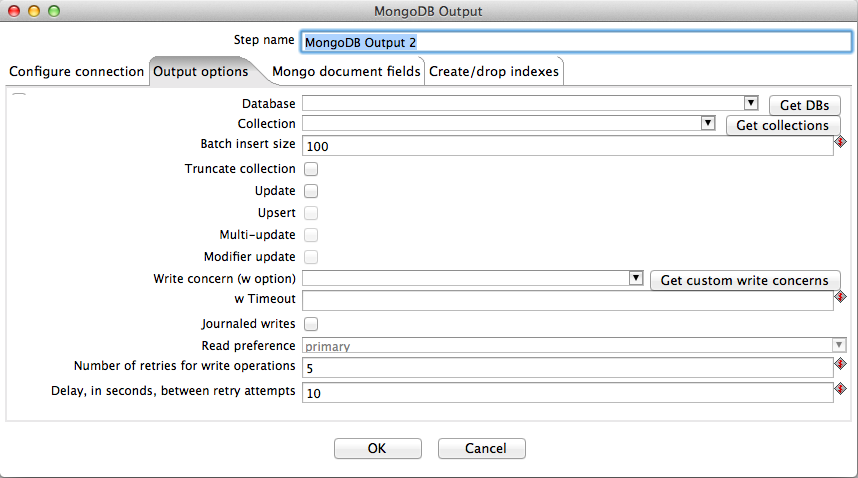
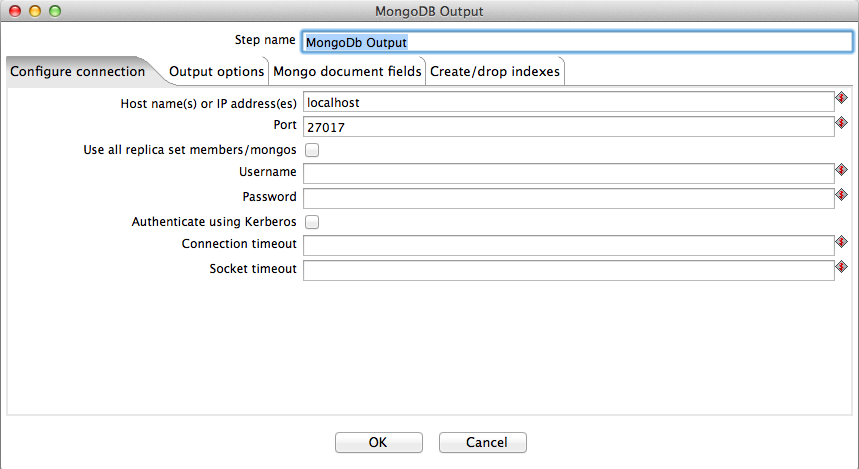
**]**

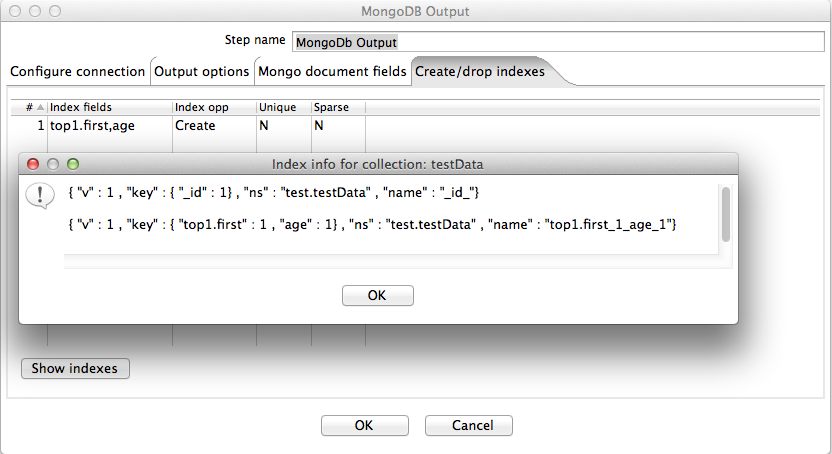
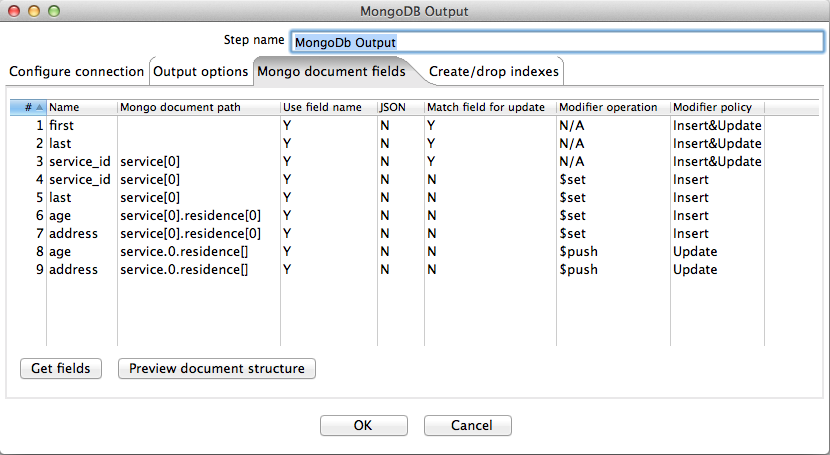
**}**

## RDBMS to MongoDB

[The data structures are going to be very different between the two and hence you will need utility and techniques to convert the models.]

## Using ETL Tool Pentaho:





## Using SPARK as ETL:

*A simple Spark program can run on multiple nodes,can read from database and convert it to JSON. Which an be impoted to MongoDB using mongoimport.*

public class JsonToJdbc implements Serializable {

private static final String MYSQL\_USERNAME = "chetan";

private static final String MYSQL\_PWD = "";

private static final String MYSQL\_CONNECTION\_URL =

"jdbc:mysql://localhost:3306/test?user=" + MYSQL\_USERNAME + "&password=" + MYSQL\_PWD;

private static final JavaSparkContext sc =

new JavaSparkContext(new SparkConf().setAppName("JsonToJdbc").setMaster("local"));

private static final SQLContext sqlContext = new SQLContext(sc);

public static void main(String[] args) {

DataFrame custDf = sqlContext.jsonFile("/home/chetan/demodata/transinfo.json");

custDf.registerTempTable("mytable");

DataFrame recordlist = sqlContext.sql("select custid, name, lastname, age, country,amount from mytable");

recordlist.insertIntoJDBC(MYSQL\_CONNECTION\_URL, "customer1", false);

//if reuried read the table and convert it to Jason

Map<String, String> options = new HashMap<String, String>();

options.put("url", MYSQL\_CONNECTION\_URL);

options.put("dbtable", "customer1");

DataFrame jdbcDF = sqlContext.jdbc(options.get("url"),options.get("dbtable"));

RDD <String> JSONStr = jdbcDF.toJSON();

JSONStr.saveAsTextFile("/home/chetan/demodata/JSONToJDBCOut.JSON");

System.out.println("printing rdd ".concat(String.valueOf(JSONStr.count())));

}

}