MONGODB VS CASSANDRA: NOSQL DATABASES COMPARED



Here we make a side by side comparison of MongoDB versus Cassandra. We provide examples, with commands and code, and not just a narrative explanation.

In sum, Cassandra is the modern version of the relational database, albeit where data is grouped by column instead of row, for fast retrieval. MongoDB stores records as documents in JSON format. It has a JavaScript shell and a rich set of functions which makes it easy to work with. Both systems are designed to scale enormously.

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

Data Structure

Apache Cassandra is a column-oriented database. MongoDB stores records in JSON format. The MongoDB shells also support JavaScript so that you can build up queries and data conversion and manipulation in steps, saving each operation in a JavaScript variable.

A JSON data record is self-describing, because the field name and the data value is stored in the same place, i.e., inside the document. While a schema is not required with MongoDB, as JSON by definition does not need one, you can make one:

```
var schema = new mongoose.Schema({
cachedContents : {
largest : String,
non_null : Number,
null : Number,
```

```
top : ,
smallest : String,
format : {
displayStyle : String,
align : String
}
}
});
```

In this Introduction to Cassandra, we explain that Cassandra tables use a schema, like a traditional relational database. But it does away with the notion of database **normalization**. Oracle administrators would say that the Cassandra schema is **flat**. The reason for this is storage across a network of commodity servers is cheap, so there is no reason to make joins to retrieve data, which slows down data retrieval. Instead, redundancy is OK. Also Cassandra tables do not require every field to be populated. So there is no overhead of storing empty data.

Cassandra also provides JSON support:

```
use json;

CREATE type json.sale ( id int, item text, amount int );

CREATE TABLE json.customers ( id int  PRIMARY KEY, name text, balance int,
sales list> );

INSERT INTO json.customers (id, name, balance, sales)
VALUES (123, 'Greenville Hardware', 700,
 ) ;
select * from customers;

 id  | balance | name                | sales
-----+---------+---------------------+-------------------------------------
----------------------------------------
 123 |     700 | Greenville Hardware |
```

Adding Data

Cassandra:

```
CREATE TABLE Library.book (
ISBN text,
copy int,
title text,
PRIMARY KEY (ISBN, copy)
 );

INSERT INTO  Library.book (ISBN, copy, title) VALUES('1234',1, 'Bible');
```

MongoDB:

```
db.collection.insertOne( { isbn: 100 } )
{
"acknowledged" : true,
"insertedId" : ObjectId("5c4493aa750820eae9756a15")
}
```

Create Index

Cassandra:

```
CREATE TABLE  Library.patron (
ssn int PRIMARY KEY,
checkedOut set
);

INSERT INTO Library.patron (ssn, checkedOut) values (123,{'1234','5678'});

create index on Library.patron (checkedOut);
```

MongoDB:

```
db.address.createIndex( { "location": "2dsphere"} )
```

Clustering

We explain how to Set Up a Cassandra Cluster and MongoDB Cluster Installation.

MongoDB (mongod daemon) uses a Mongo Master, Mongo Shard, and Mongo Config server to replicate data. The Query server, aka router, (mongos daemon) determines to which server to send queries and data operations, like adding a record.

Cassandra has no configuration server to control the operation of other servers. There is no master-slave relationship. Instead there is a **ring** of servers, each serving equal functions, albeit storing different parts (i.e., shards) of the data.

Sharding

Sharding means distributing data across a cluster. This is done by applying an algorithm across part of all of a document or index. MongoDB and Cassandra both provide a fine level of control over this.

Cassandra does this with Cassandra Partition key, Composite key, and Clustering Columns and Using Tokens to Distribute Cassandra Data. In brief, each table requires a unique primary key. The first field listed is the **partition key**, since its hashed value is used to determine the node to store the data. If those fields are wrapped in parentheses then the **partition key** is composite. Otherwise the first field is the partition key. Any fields listed after the primary key are called **clustering columns**. These store data in ascending or descending order within the partition for the fast retrieval of similar values. All the fields together are the primary key.

And here MongoDB sharding is explained. Basically, you assign different parts of the data to different servers using an index. For example, records with the index customers could be on one set

of servers and vendors on the other. But if you want a completely random distribution then you use a hashed value for the index. You can also assign data to servers using a range of values.

```
sh.enableSharding("tobacco")
{ "ok" : 1 }
```

Data Consistency through Replication

Replication means making more than one copy of data to prevent against data loss due to hardware or other failure.

MongoDB uses a concept call **ReplicaSets**, configured with rs.initiate:

```
rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration
for the set",


ConfigReplSet:SECONDARY> rs.status()
{
        "set" : "ConfigReplSet",
```

We explain Configuring Apache Cassandra Data Consistency. You create replication like this:

```
CREATE KEYSPACE Library
      WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' :
3 };
```

Support for GEO Operations

MongoDB has superior support for storing and querying geographical data. One obvious reason for that is the universal format for that is GEOJson. Take a look at Track Tweets by Geographic Location and Query MongoDB by GeoLocation.

Here is a MongoDB query to find records near a certain longitude and latitude:

```
db.address.find ({
  location: {
     $near: {
       $geometry: {
          type: "Point" ,
          coordinates:
       },
       $maxDistance: 4,
       $minDistance: 0
     }
   }
})
```

And since the mongo shell supports Javascript you can save queries in variables:

```
var connecticut = db.address.find ({location:
     {$geoWithin:
       {$geometry: {
       type: "Polygon",
       coordinates:
 ,
   ,
   ,
   ,
   ]]
}}}});
```

Stress Testing

We talk about memory management and performance tuning in Stress Testing and Performance Tuning Apache Cassandra and MongoDB Memory Usage