Topics

1 NoSQL Overview

(Not only SQL became shortened to NoSQL)

2 MongoDB

An introduction for beginners covering MongoDB for the next 3 weeks

NoSQL

Relational databases

- have been used for a few decades and store data in a **structured format**

Data is subdivided into groups called tables

Each unit of data is called a COLUMN and it has a defined type, size and constraints

The combined columns is called a ROW

Scaling horizontally is a problem

NoSQL databases

- are not primarily built on tables and as a result do not use SQL for data manipulation

NoSQL is not a database but a term to refer to databases that attempt to solve problems of scalability and availability against atomicity and consistency

ACID transactions from RDMS

|  |  |
| --- | --- |
| Atomicity | Everything in a transaction succeeds or it is rolled back |
| Consistency | A transaction cannot leave the database in an inconsistent state |
| Isolation | one transaction cannot interfere with another |
| Durability | a completed transaction persists even after applications restart |

Although these are excellent qualities they are incompatible with availability and performance on applications of web scale.

Example :

if a company like Amazon were to use a system like this imagine how slow it would be if I proceed to buy a book and a transaction is being done it will lock apart of the database , specifically locking the inventory , and every other person in the world will have to wait until I complete my transaction. Obviously this doesn't work well

Leads to BASE

|  |  |
| --- | --- |
| Basic availability | each request is guaranteed a response successful or failed execution |
| soft state | the state of the system may change over time at times without any input. It will eventually become consistent |
| eventual consistency | the database may be momentarily inconsistent but will be consistent eventually |

Why NoSQL

Schemaless data representation

Without a schema means that you don't have to think too far ahead to define a structure and you can continue to evolve overtime

Development time

It is reported that there would be less development time because one doesn't have to deal with complex SQL queries. If you remember ask JL you needed to join tables to get data that was stored across multiple tables in order to create your final views for the user.

Speed

Even with small amounts of data that you have if you can deliver it in milliseconds rather than hundreds of a millisecond especially when using mobile devices you have a hype probability of convincing users of your system that it's a good idea

Plan ahead for scalability

It means it can scale out easily with no limit by adding more servers for distributing data

List of NoSQL databases

* There are different NoSQL databases:
* Document
  1. MongoDB
  2. CouchDB
* Key-value
  1. Redis
  2. Membase
* XML
  1. BaseX
* Column
  1. BigTable
  2. Hadoop/Hbase
* Graph

Neo4J

You can go deeper, but we are limited on time to cover everything

MongoDB - intro

MongoDB

MongoDB is a document-oriented NoSQL database used for high volume data storage.

It differs from a relational database.

It provides High performance, high availability , and easy scalability.

MongoDB works on the concept of collections and documents

It scales up easier compared to a relational database.

MongoDB is a powerful, flexible, and scalable general-purpose database.

It provides the following features:

Indexing

Aggregations

File Storage

Special collection types

What is MongoDB – more

It is a document-oriented NoSQL database used for high volume data storage.

Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system.

A single MongoDB server typically has multiple databases.

Collections

Collections contain sets or groups of documents and function which is the equivalent of relational database tables.

Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Documents

Documents consist of key-value pairs which are the basic unit of data in MongoDB.

Documents have dynamic schema.

Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Repeat -- simplified

A database contains collections

Collections contain documents

Each document can be different

Varying by size

Varying by content

Scalability

Often discussed with databases

Meaning: From smaller to very large complex systems

Why important: Data grows at a fast pace 🡺 need to be scalable

How to scale

1 Large machines can be used to scale up

- Expensive

- There may be a limit on physical machines

2 Partitioning

- storage space achieved by adding servers and computers to your clusters

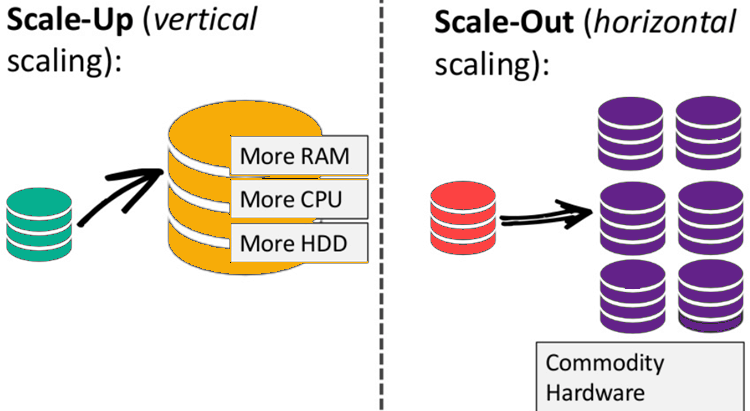
- Cheaper

- Added difficulty managing 1000s of machines

Types of scaling

horizontally vs vertically

Visual Sample



**Vertical scaling**

Vertical scaling can essentially resize your server with no change to your code. It is the ability to increase the capacity of existing hardware or software by adding resources. Vertical scaling is limited by the fact that you can only get as big as the size of the server.

**Example: Apartment building**

apartment building that has many rooms and floors where people move in and out all the time. In this apartment building, 200 spaces are available but not all are taken at one time. So, in a sense, the apartment scales vertically as more people come and there are rooms to accommodate them.

Note: If the 200-space capacity is not exceeded, life is good.

**Restaurant - capacity**

**Horizontal scalability**

Means increasing capacity by connecting multiple hardware or software entities so that they work as a single logical unit. When servers are clustered , the original server is being scaled out horizontally.

Example: **Highway 4 lanes** handle 2000 cars/hour

Add more offices and Condos and you need 8000/hour.

Can handle it but have bottlenecks

Adding more lanes gets expensive and takes time

How MongoDB can scale

Scales out by splitting documents across multiple servers

Back to Basics repeat

DOCUMENT

- the basic unit of data

- equivalent to a row in a relational database

COLLECTION

- similar idea to a table, but not a fixed schema

One MongoDB instance can host multiple databases

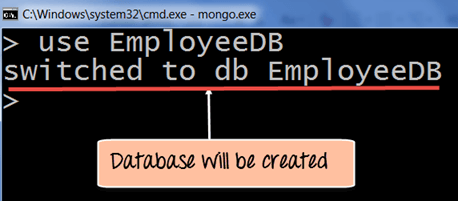
CREATE DATABASE

CREATE

MongoDB creates a database, if it does not already exist, when you insert the first document into your database

The "use" command to create a database

**use EmployeeDB**



MongoDB will now switch to the database you just created

Reserved Database Names

admin

Name of the root database

Users that are added to admin have access to all databases

Admin users only can run certain commands

local

This Database stores any local connections on a single server

config

a config server stores the clusters metadata great

**Create another sample database**

> use mydb 🡸 THIS SWITCHES AND/OR CREATES

The output would look like the previous example

switched to db mydb

To check what database, you are using

> db

mydb

**SEE databases**

show dbs

>show dbs

local 0.78125GB

test 0.23012GB Notice that MYDB is not there

**To display the database, you need to insert at least one document into it.**

db.movie.insert({"name":"tutorials point"}) 🡸AN INSERT COMMAND

show dbs

local 0.78125GB

mydb 0.23012GB 🡸 NOW IT WILL SHOW

test 0.23012GB

**ASIDE:** In MongoDB default database is test.

If you did not create any database, then collections will be stored in test database.

**DROP DATABASE**

The dropDatabase() Method

MongoDB **db.dropDatabase()** command is used to drop an existing database.

> use mydb

switched to db mydb

>db.dropDatabase()

>{ "dropped" : "mydb", "ok" : 1 }

**Proof**

>show dbs

local 0.78125GB

test 0.23012GB

>

**CREATE COLLECTION**

Basic syntax of **createCollection()** method

>use test

switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }

**SEE COLLECTIONS**

>show collections

mycollection 🡸 here it is

system.indexes

Collection Name rules

A collection is identified by its name

Some rules for the name

- cannot be an empty string ""

- cannot contain the null Terminator character \0

- cannot start with the reserved prefix such as system

- cannot include the reserved character $

- cannot exceed max size 64 bytes

- Cannot have spaces

And is

- Case sensitive

**Create collection using an insert**

One of the easiest way to create a collection is to insert a record (which is nothing but a document consisting of Field names and Values) into a collection.

If the collection does not exist a new one will be created.

Example: Using insert

db.myEmployeeDB.insert(

{

"Employeeid" : 1,

"EmployeeName" : "Martin"

}

)

show collections

mycollections

myEmployeesDB 🡸

system.indexes

DROP COLLECTIONS

**db.dropDatabase()** removes the current database and all data inside the database.

>db.mycollection.drop()

true

>

Subcollections

you can use subcollections to organize a collection

Subcollections are separated by the . Character

Example:

The Collection known as blog has two subcollections:

blog.posts

blog.authors

Documents

Key

Every document has a unique key or \_ID

- Used to identify a document in a collection

Example:

{ "greeting" : "Hello, World" }

Key: "greeting"

Value: "Hello, world"

A document can contain more than one key/value pair

{"greeting" : "Hello, world!", "foo" : 3}

Notice the 2 values are different.

Integer

String

Aside:

\_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows −

\_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)

Duplicate Keys

Not allowed because there are duplicate keys

Example:

{"greeting" : "Hello, world!", "greeting" : "Hello, MongoDB!“}

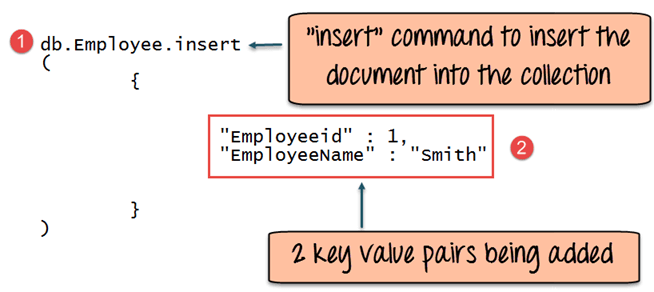
**Adding documents using insert() command**

MongoDB provides the **insert () command** to insert documents into a collection. You sae it above.

The following example shows how this can be done.

**Step 1)** Write the "insert" command

**Step 2)** Within the "insert" command, add the required Field Name and Field Value for the document which needs to be created.

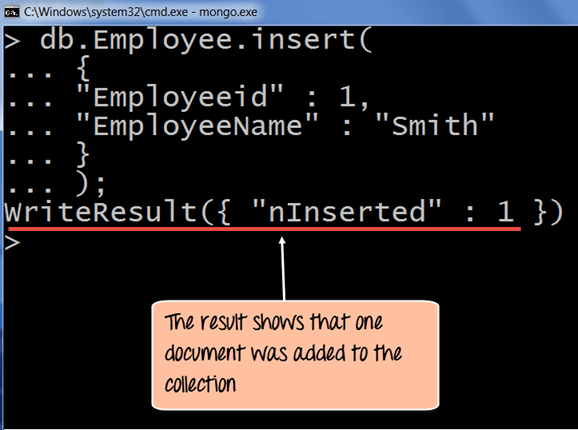
[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio5.png)

**Code Explanation:**

1. The first part of the command is the "**insert statement"** which is the statement used to insert a document into the collection.
2. The second part of the statement is to add the Field name and the Field value, in other words, what is the document in the collection going to contain.

If the command is executed successfully, the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio6.png)

The output shows that the operation performed was an insert operation and that one record was inserted into the collection.

Insert a new document

into a new collection

create the collection empDetails

> db.createCollection("empDetails")

{ "ok" : 1 }

InsertOne() method

db.empDetails.insertOne(

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9848022338"

})

**OUTPUT:**

{

"acknowledged" : true,

"insertedId" : ObjectId("5dd62b4070fb13eec3963bea")

}

Inserting multiple documents

insertMany() method

You need to pass an array of documents … meaning multiple documents

db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

**OUTPUT:**

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("5dd631f270fb13eec3963bed"),

ObjectId("5dd631f270fb13eec3963bee"),

ObjectId("5dd631f270fb13eec3963bef")

]

}

Another set of samples for INSERT

BUT … we are going to insert the following into *empDetails*

insertOne

This will insert one document

db.empDetails.insertOne ( {"title" : "My Blog Post",

"content" : "Here is my blog post"

})

insertMany can be different looking data

db.empDetails.insertMany(

[

{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9000012345"

},

{

First\_Name: "Rachel",

Last\_Name: "Christopher",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Rachel\_Christopher.123@gmail.com",

phone: "9000054321"

},

{

First\_Name: "Fathima",

Last\_Name: "Sheik",

Date\_Of\_Birth: "1990-02-16",

e\_mail: "Fathima\_Sheik.123@gmail.com",

phone: "9000054321"

}

]

)

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("5dd631f270fb13eec3963bed"),

ObjectId("5dd631f270fb13eec3963bee"),

ObjectId("5dd631f270fb13eec3963bef")

]

}

db.empDetails.drop()

Remove a document

The *remove* function deletes documents.

Reinsert the many data above

Do the find db.empDetails.find()

{ "\_id" : ObjectId("5fb29639d1db3d91a34ac04e"), "First\_Name" : "Radhika", "Last\_Name" : "Sharma", "Date\_Of\_Birth" : "1995-09-26", "e\_mail" : "radhika\_sharma.123@gmail.com", "phone" : "9000012345" }

{ "\_id" : ObjectId("5fb29639d1db3d91a34ac04f"), "First\_Name" : "Rachel", "Last\_Name" : "Christopher", "Date\_Of\_Birth" : "1990-02-16", "e\_mail" : "Rachel\_Christopher.123@gmail.com", "phone" : "9000054321" }

{ "\_id" : ObjectId("5fb29639d1db3d91a34ac050"), "First\_Name" : "Fathima", "Last\_Name" : "Sheik", "Date\_Of\_Birth" : "1990-02-16", "e\_mail" : "Fathima\_Sheik.123@gmail.com", "phone" : "9000054321" }

Following example will remove all the documents whose First\_name is …

db.empDetails.remove ({"First\_Name" : "Radhika"})

How to check it

db.empDetails.find()

{ "\_id" : ObjectId("5fb29639d1db3d91a34ac04f"), "First\_Name" : "Rachel", "Last\_Name" : "Christopher", "Date\_Of\_Birth" : "1990-02-16", "e\_mail" : "Rachel\_Christopher.123@gmail.com", "phone" : "9000054321" }

{ "\_id" : ObjectId("5fb29639d1db3d91a34ac050"), "First\_Name" : "Fathima", "Last\_Name" : "Sheik", "Date\_Of\_Birth" : "1990-02-16", "e\_mail" : "Fathima\_Sheik.123@gmail.com", "phone" : "9000054321" }

PRETTY()

Just an extra to make it nicer to read – don't believe this is on the course but showing it anyway

db.empDetails.find().pretty()

{

"\_id" : ObjectId("5fb29639d1db3d91a34ac04f"),

"First\_Name" : "Rachel",

"Last\_Name" : "Christopher",

A bit better layout.

"Date\_Of\_Birth" : "1990-02-16",

"e\_mail" : "Rachel\_Christopher.123@gmail.com",

"phone" : "9000054321"

}

{

"\_id" : ObjectId("5fb29639d1db3d91a34ac050"),

"First\_Name" : "Fathima",

"Last\_Name" : "Sheik",

"Date\_Of\_Birth" : "1990-02-16",

"e\_mail" : "Fathima\_Sheik.123@gmail.com",

"phone" : "9000054321"

}

Remove All Documents

If you do not specify deletion criteria, then MongoDB will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**

db.empDetails.remove({})

WriteResult({ "nRemoved" : 2 })

**Proof**

db.empDetails.find()