

# Relational Databases

RDBMS databases have features like

- Persistence
- Integration
- SQL
- Transaction - Concurrency (Multiple Threads)
- Reporting

## ACID (Atomicity, Consistency, Isolation, Durability)

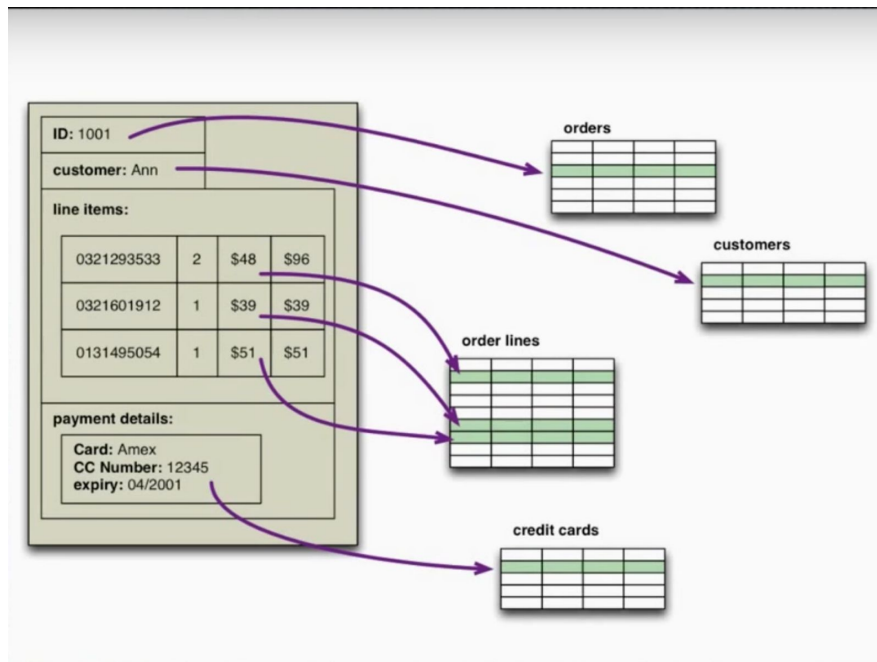
- **Atomicity** - all or nothing
- **Consistency** - The [consistency](#) property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including [constraints](#), [cascades](#), [triggers](#), and any combination thereof
- **Isolation** - The [isolation](#) property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed sequentially, i.e., one after the other. Providing isolation is the main goal of [concurrency control](#).
- **Durability** - The [durability](#) property ensures that once a transaction has been committed, it will remain so, even in the event of power loss, [crashes](#), or errors.

## Issues with Relational Databases

### Impedance Mismatch - Data is scattered

A single application object's information is scattered in multiple tables. Because of which application has to assemble it each time when retrieving it and de-assemble it when persisting it.

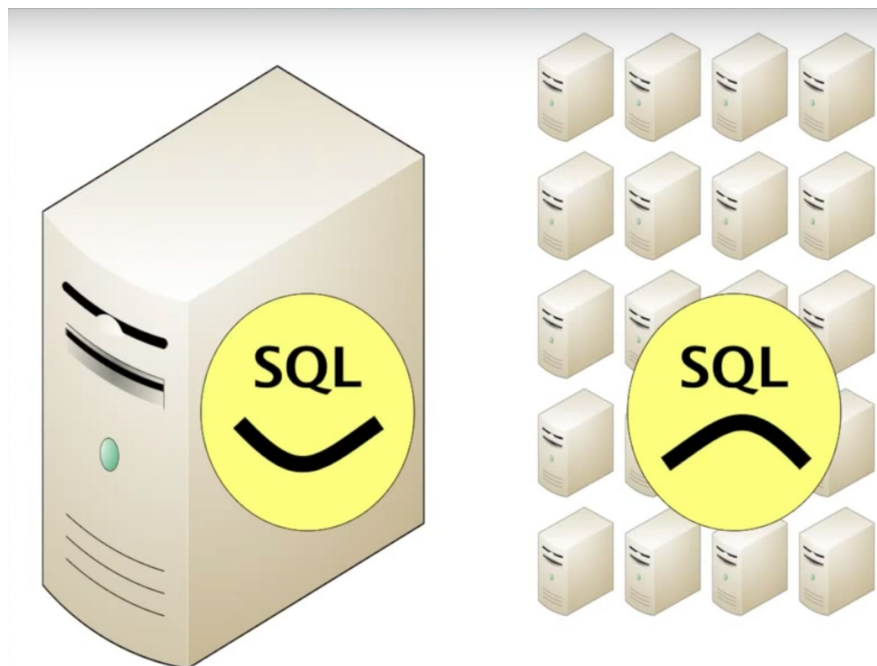
This the reason why we use ORMs



Database community tried to solve it by creating **Object Databases**. But they didn't become popular.

## Scaling

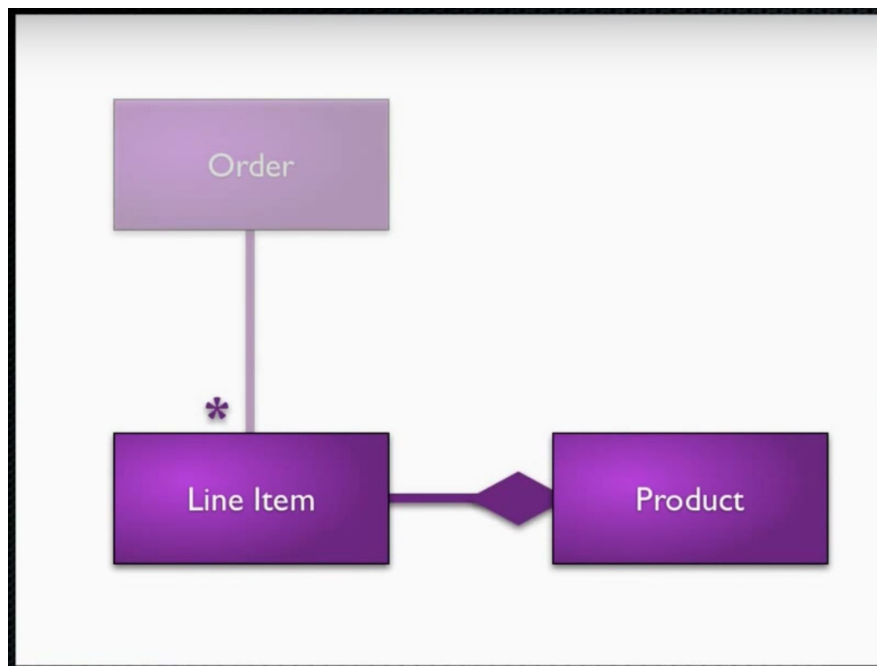
Relational databases are very difficult to run in distributed environment. For huge amount of data they need to be scale up not horizontally. SQL and transaction don't work well distributed environment.



To solve scaling issue

- Google created Bigtable
- Amazon created Dynamo

Product	revenue	prior revenue
321293533	3083	7043
321601912	5032	4782
131495054	2198	3187
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...



In nosql this is possible using map reduce but it's hard

# NoSQL Databases

[https://www.youtube.com/watch?v=qI\\_g07C\\_Q5I](https://www.youtube.com/watch?v=qI_g07C_Q5I)

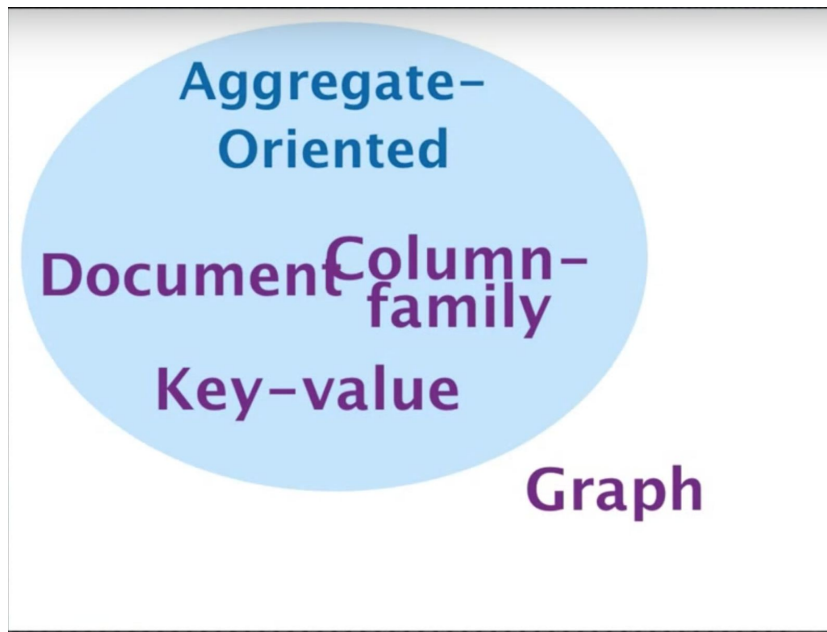
To solve Relational Databases issues, Johan Oskarsson arranged a meetup and created NoSQL twitter hashtag. From there NoSQL name became popular.

BASE (Basically Available, Soft State, Eventual consistency) analysis of NoSQL

## **Possible** Common Characteristics of NoSQL

- Non Relational
- Open Source
- Cluster Friendly
- 21st Century Web
- Schema-less

## Types of NoSQL database



### Aggregate Oriented

An aggregate (Document, Object) lives on a single node (machine in a cluster) because of clustering is easy

## Key-Value

E.g. Project Voldemort, riak, redis,

## Document

E.g. MongoDB, Ravendb, CouchDB relax

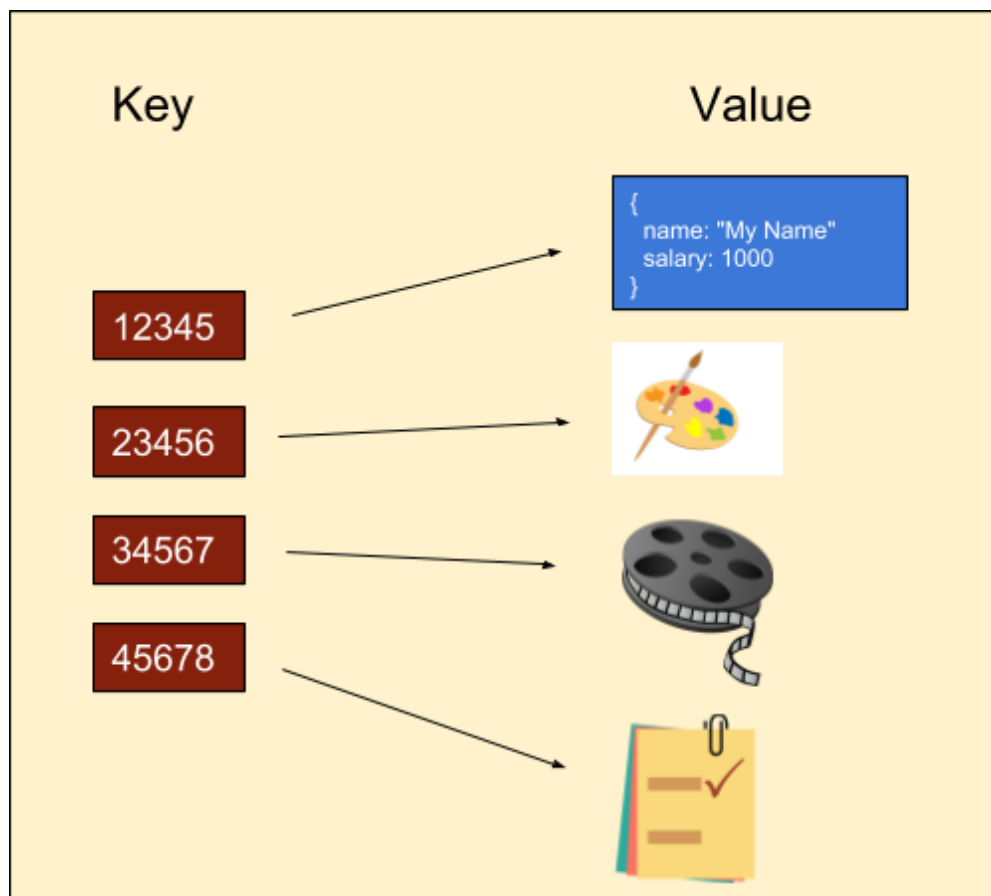
## Column Family

E.g. Cassandra, Apache HBASE

## Graph

E.g. Neo4j

## Key Value - NoSQL database



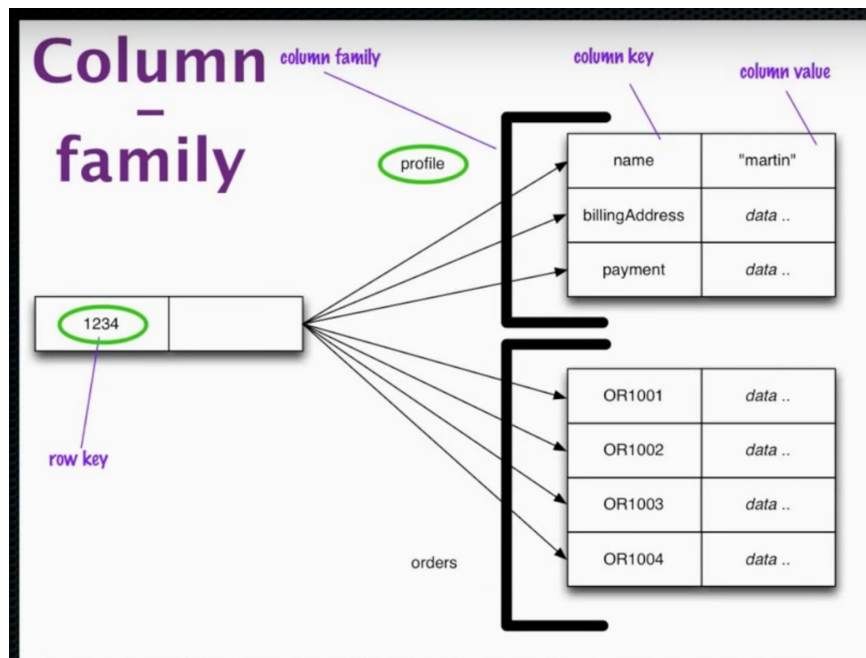
## Document - NoSQL database

### Document

```
{ "id": 1001,  
  "customer_id": 7231,  
  "line-items": [  
    { "product_id": 4555, "quantity": 8 },  
    { "product_id": 7655, "quantity": 4 }, { "product_id": 8755,  
      "quantity": 2 }  
  ],  
  "id": 1002,  
  "customer_id": 9831,  
  "line-items": [  
    { "product_id": 4555, "quantity": 3 },  
    { "product_id": 2155, "quantity": 4 } ],  
  "discount-code": "Y" }
```

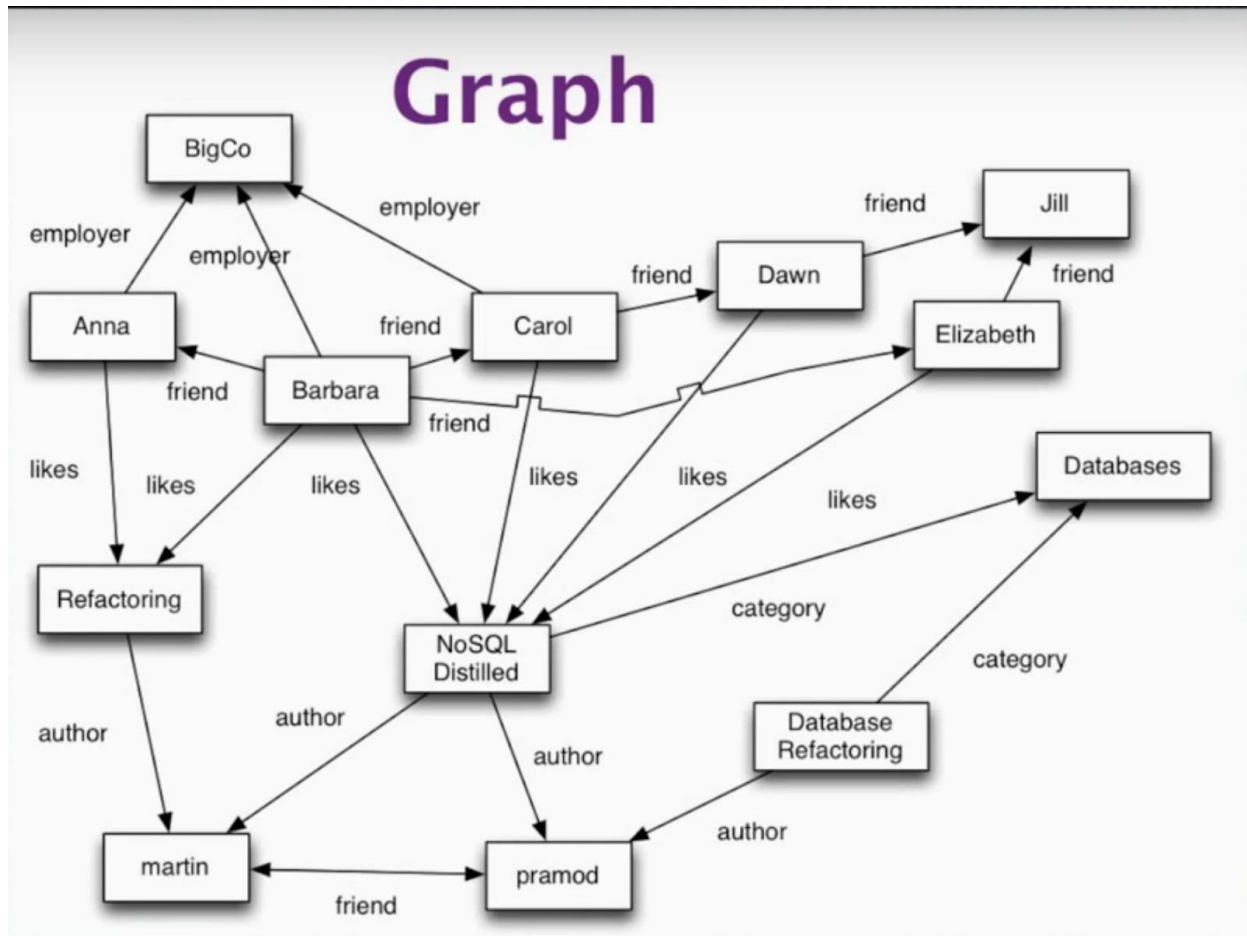
no  
schema

## Column Family - NoSQL database



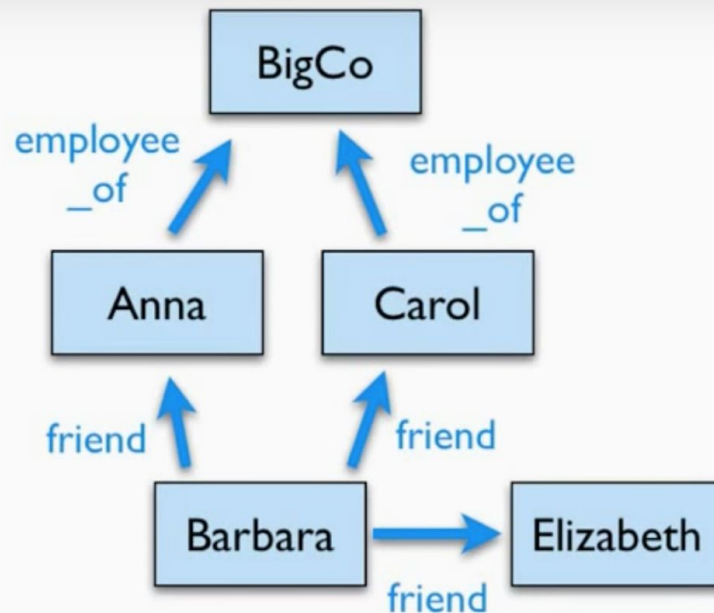
# Graph - NoSQL database

Based on relationship between nodes of data. Relational databases don't perform well when joining a lot of tables.



Query language for Graph databases

# Graph



```
START barbara = node:nodeIndex(name = "Barbara")  
MATCH (barbara)-[:FRIEND]->(friend_node)  
RETURN friend_node.name,friend_node.location
```

## MongoDB

<https://docs.mongodb.com/manual/>

## Installation

### Download Community version

#### Mac & Linux

<https://www.mongodb.com/download-center#community>

Unzip the downloaded file. Rename the folder to "mongodb"

Move mongodb folder to any location where you keep your development installations

E.g. I moved it to



/Users/sheraz/dev/mongodb

Tip: create environment variable for mongodb and use it for all other configuration

```
export MONGO_HOME=/Users/sheraz/dev/mongodb
```

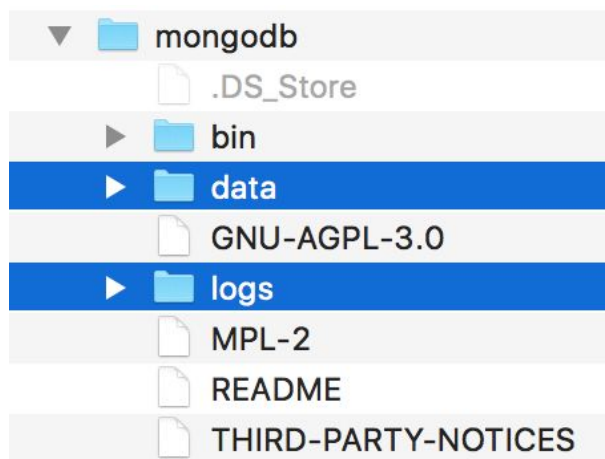
### Add bin folder to PATH

```
export PATH=$MONGO_HOME/bin:$PATH
```

## Setup Database

### Data and logs folder

Make **data** and **logs** folder in **mongodb** folder



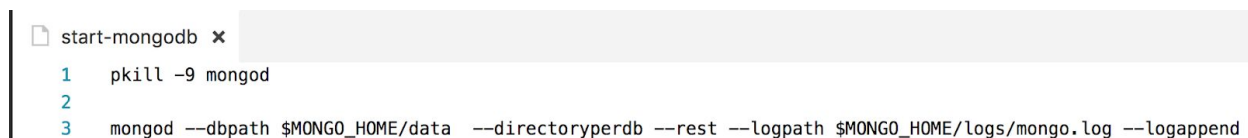
### Startup script

Create a mongodb startup script e.g. start-mongodb for easy start/restarting mongodb.

Add commands below in the script

```
pkill -9 mongod
```

```
mongod --dbpath $MONGO_HOME/data --directoryperdb --rest --logpath  
$MONGO_HOME/logs/mongo.log --logappend
```



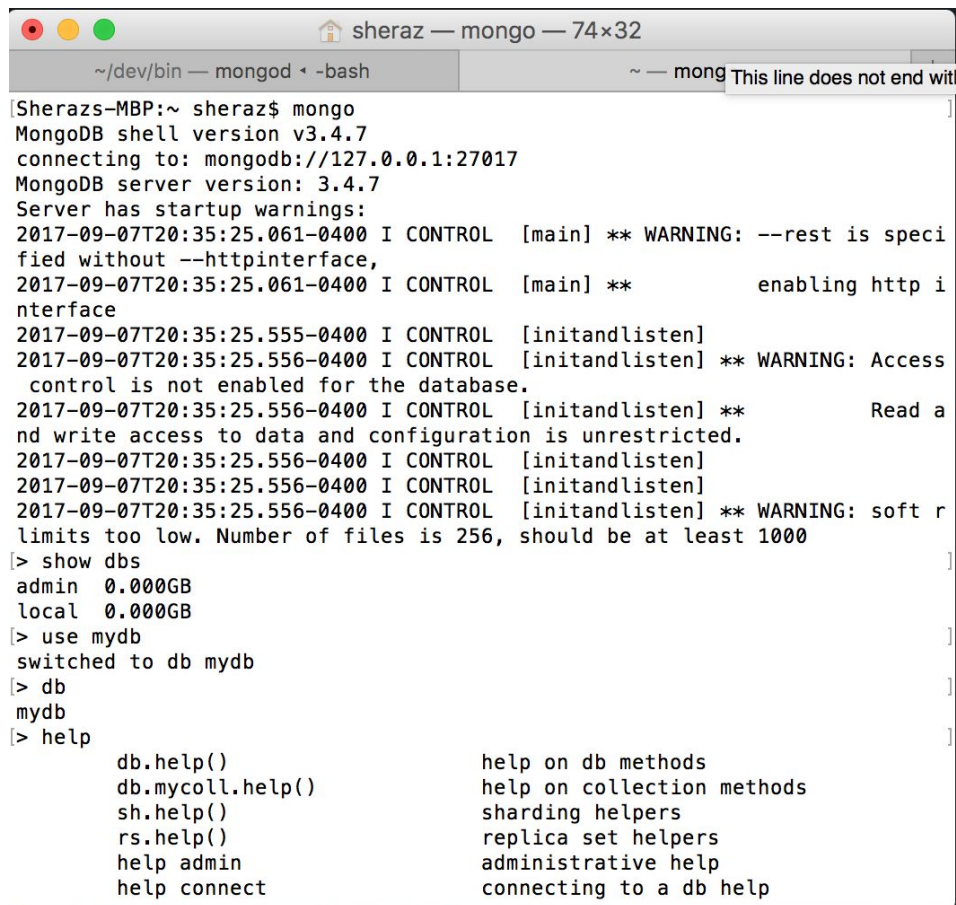
## Start mongoDB

\$ start-mongod

## Using mongod shell

If mongod/bin is in path and mongod process is running then we can use mongod shell

\$ mongo

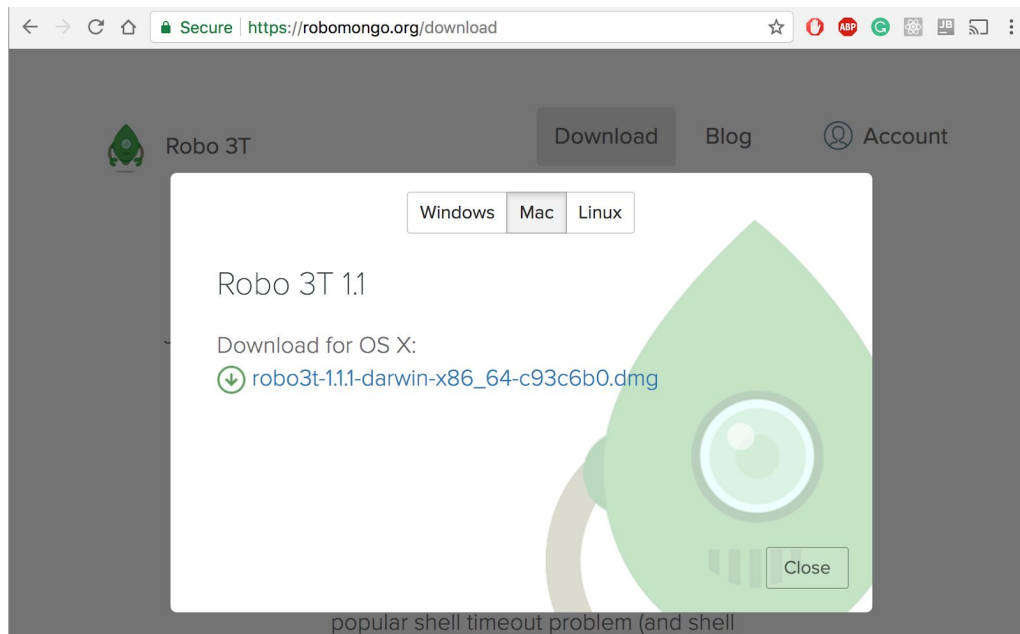


```
Sherazs-MBP:~ sheraz$ mongo
MongoDB shell version v3.4.7
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.7
Server has startup warnings:
2017-09-07T20:35:25.061-0400 I CONTROL [main] ** WARNING: --rest is specified without --httpinterface,
2017-09-07T20:35:25.061-0400 I CONTROL [main] ** enabling http interface
2017-09-07T20:35:25.555-0400 I CONTROL [initandlisten]
2017-09-07T20:35:25.556-0400 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-09-07T20:35:25.556-0400 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2017-09-07T20:35:25.556-0400 I CONTROL [initandlisten]
2017-09-07T20:35:25.556-0400 I CONTROL [initandlisten]
2017-09-07T20:35:25.556-0400 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
[> show dbs
admin 0.000GB
local 0.000GB
[> use mydb
switched to db mydb
[> db
mydb
[> help
db.help() help on db methods
db.mycoll.help() help on collection methods
sh.help() sharding helpers
rs.help() replica set helpers
help admin administrative help
help connect connecting to a db help
```

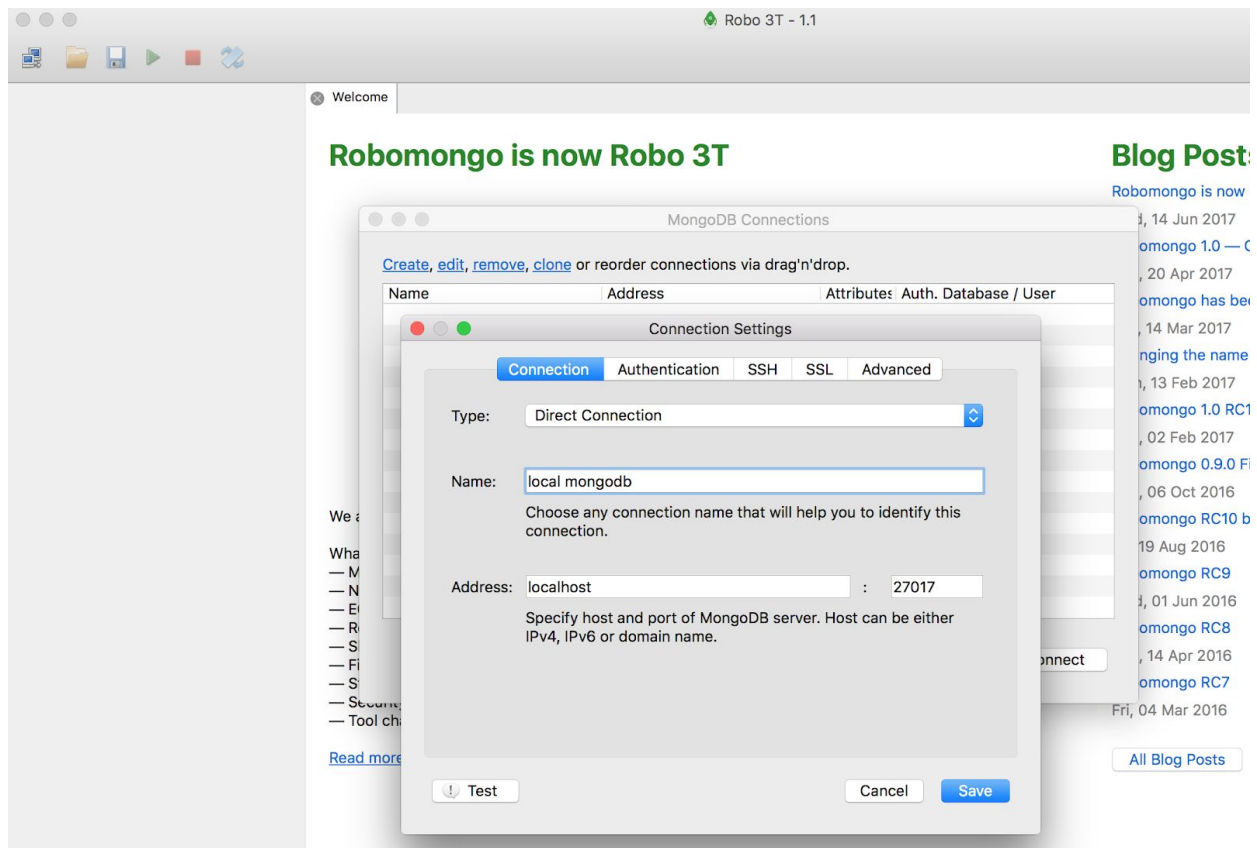
## MongoDB IDE/Explorer - Robomongo

### Download and install robomongo

<https://robomongo.org/download>



Create new localhost connection, save it and connect it.



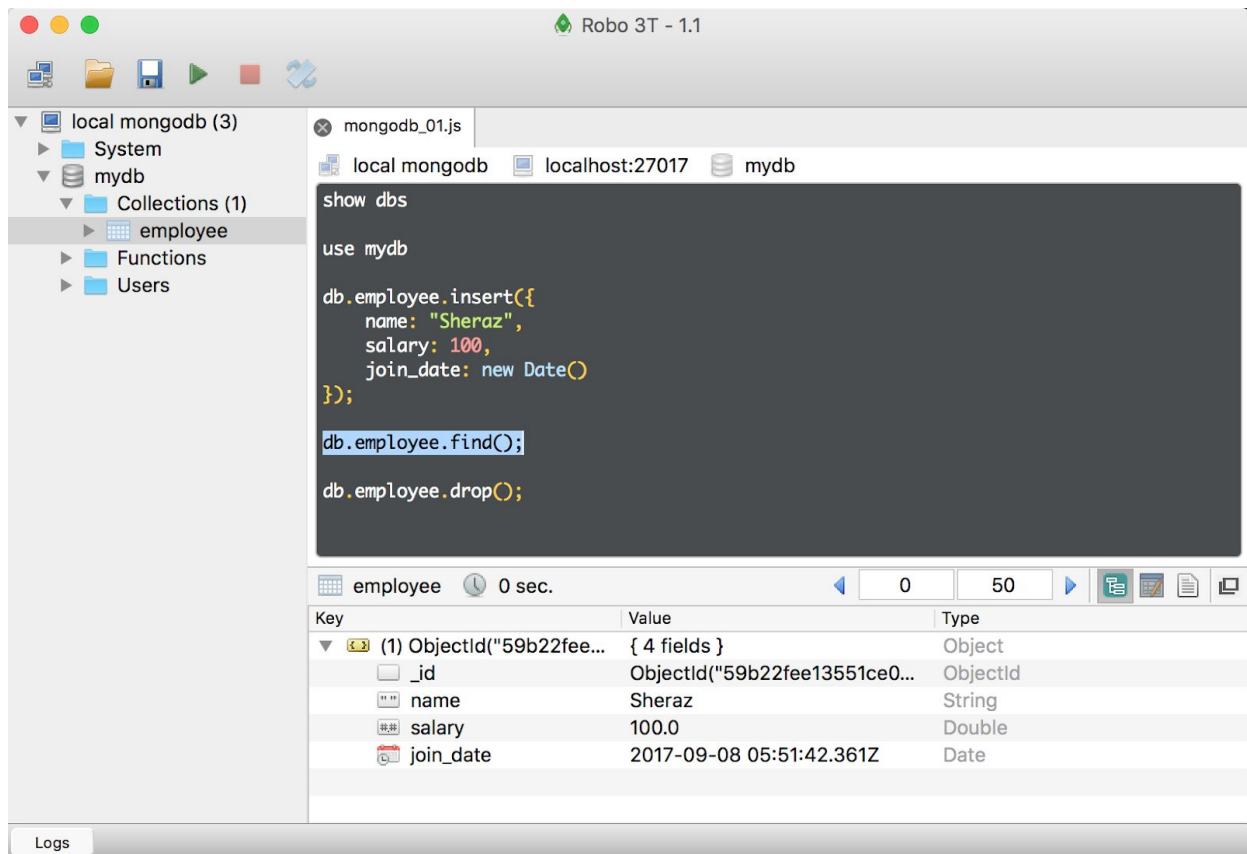
## Using robomongo

### Create or Use existing script

Right click connection name -> click "Open Shell"

If you have existing script then

Click open button -> browse and open your script



### Running script or Command

To run entire script do not select anything in the script and hit command + enter

To run single command select a command and hit command + enter

DB admin commands do not end with semicolon but javascript commands can.

# RDBMS vs NoSQL - MongoDB Terminologies

<https://docs.mongodb.com/manual/reference/sql-comparison/#terminology-and-concepts>

<https://docs.mongodb.com/manual/reference/glossary/>

SQL Terms/Concepts	MongoDB Terms/Concepts
database	<a href="#">database</a>
table	<a href="#">collection</a>
row	document or BSON document
column	<a href="#">field</a>
index	<a href="#">index</a>
table joins	<a href="#">\$lookup</a> , <a href="#">embedded documents</a>
primary key Specify any unique column or column combination as primary key.	<a href="#">primary key</a> <a href="#">In MongoDB, the primary key is automatically set to the <code>_id</code> field.</a>
aggregation (e.g. group by)	aggregation pipeline

## Mongo Shell - Javascript

<https://docs.mongodb.com/manual/tutorial/write-scripts-for-the-mongo-shell/#differences-between-interactive-and-scripted-mongo>

By default mongo shell uses Javascript to interact with mongodb.

Here are some of the shell helper commands and its JavaScript equivalents

Shell Helpers	JavaScript Equivalents
show dbs, show databases	db.adminCommand('listDatabases')
use <db>	db = db.getSiblingDB('<db>')
show collections	db.getCollectionNames()

show users	db.getUsers()
show roles	db.getRoles({showBuiltinRoles: true})
show log <logname>	db.adminCommand({ 'getLog' : '<logname>' })
show logs	db.adminCommand({ 'getLog' : '*' })
it	<pre> cursor = db.collection.find() if ( cursor.hasNext() ){   cursor.next(); } </pre>

> show dbs  
> show databases  
Show all the databases

> show roles  
Show all mongodb built-in roles

> show users  
Show all users

> use <db>  
Switch database

## Printing results

Mongo shell do not have console.log().

To print result we can:

- Just write variable name  
> myVar
- Use the print() function  
> print(myVar);

This is same behavior as browser and node console.

## Help

### Shell help commands

> help

```
> help admin
> help connect
> help keys
> help misc
> help mr
```

## JavaScript help commands

```
help("");
help("connect");
help("mr");
help("keys");
help("admin");
help("misc");
```

**NOTE:** Go over all `help()`;

## Database

To use or to switch to an existing or non existing database we can use **use** command

```
> use mydb
```

After this command a global variable **db** is created.

In all subsequent commands when we have to use current selected database then we use it through **db** variable.

```
> print(db);
> db.stats();
> db.help();
> db.listCommands();
```

**NOTE:** Go over all the commands mentioned in `db.help()`; and `db.listCommands()`;

## Collection

Collections are just like RDBMS table.

### Create Collection

```
db.createCollection("person") ;
```

## List all Collections

```
show collections
```

## Drop Collection

```
db.person.drop();
```

## Basic CRUD

<https://docs.mongodb.com/manual/crud/>

First object in find(), update(), and remove() is **Filter** Object. You will find more details on Filters in following sections.

```
db.person.insert({
  name: "Sheraz",
  salary: 100
});
```

```
db.person.find({});
```

```
db.person.update({}, {
  name: "Sheraz",
  salary: 200
});
```

```
db.person.remove({});
```

## Data Types and Values

All JSON/BSON data types are valid in mongodb

ObjectID, null, String, Number base 64 bit floating points, boolean, Date, Arrays, Regular Expression (given between //), Complex JSON objects

## Retrieve selected fields

<https://docs.mongodb.com/manual/tutorial/project-fields-from-query-results/>

find() will always return all the fields in a document. To retrieve selected field we have to set field name to 1.

```
db.person.find({}, {name: 1});
```

Even if we retrieve selected fields, still find() will return \_id. To turn off \_id we use 0.



```
db.person.find({}, {name: 1, _id: 0});
```

## Update selected fields

<https://docs.mongodb.com/manual/reference/operator/update-field/>

update() expects the entire object (with all the fields) even if we want to update some of the fields. To overcome it we can use **\$set** Update Operator. **\$set** operator update given field's value.

<https://docs.mongodb.com/manual/reference/operator/update/set/>

```
db.person.update({}, {$set: {name: "Chaudhry"}});
```

First argument object is filter/criteria and second object is what to update;

Below command update all employee, sets salary=1500 where salary < 1500

```
db.employee.update({salary: {$lt: 1500}}, {$set: {salary: 1500}}, {multi: true});
```

Lookup other update operator

<https://docs.mongodb.com/manual/reference/operator/update/>

## Filter

<https://docs.mongodb.com/manual/tutorial/query-documents/>

First object in find(), update(), and remove() is **Filter** Object. Just like in SQL we write WHERE clause in SELECT, UPDATE and DELETE.

## Exact match (Just like = in SQL)

```
// Exact match
db.person.find({name: "Sheraz", salary: 100});
```

## Comparison

<https://docs.mongodb.com/manual/reference/operator/query-comparison/>

```
// Comparison < ($lt), > ($gt), <= ($lte), >= ($gte),
db.person.find({salary: {$lt: 200}});
```

## In

```
// In ($in), Not In ($nin)
db.person.find({name: {$in: ["Sheraz", "Chaudhry"]}});
```

## Logical

<https://docs.mongodb.com/manual/reference/operator/query-logical/>

```
// And ($and), Or ($or)
db.person.find({$or: [
  {name: "Sheraz"},
  {salary: {$gt: 50}}
]});
```

## Loading external Script in Console

Create a script in your computer

E.g.

~/dev/mogo

oad("/path/my-script-name.js");

Run the function inside the script MAY BE

## Files

~/dbshell - History of mongo shell commands

~/mongorc.js - startup script

/etc/mongorc.js - Global startup script

## Commands

\$set, \$mul, \$pop -1 1, \$push, \$slice, \$pull, \$gt, \$lt, \$gte, \$lte

## Aggregation

<https://docs.mongodb.com/manual/aggregation/>

<https://docs.mongodb.com/manual/reference/operator/aggregation-group/>

There are 3 types of Aggregation.

- Aggregation Pipeline,
- Map-Reduce,
- Single Purpose Aggregation Operations

## Aggregation Pipeline

<https://docs.mongodb.com/manual/core/aggregation-pipeline/>

## Map-Reduce

<https://docs.mongodb.com/manual/core/map-reduce/>

## Single Purpose Aggregation Operations

<https://docs.mongodb.com/manual/reference/aggregation/>

## SQL and Mongodb Aggregation Comparison

<https://docs.mongodb.com/manual/reference/sql-aggregation-comparison/>

## Privilege Action

<https://docs.mongodb.com/manual/reference/privilege-actions>

## Roles

### Built-in Roles

<https://docs.mongodb.com/manual/core/security-built-in-roles/>

### User defined Roles

<https://docs.mongodb.com/manual/core/security-user-defined-roles/>

## User

<https://docs.mongodb.com/manual/core/security-users/>

<https://docs.mongodb.com/manual/tutorial/create-users/>

### Create user

```
db.createUser (  
  {
```

```
    user: "myuser",  
    pwd: "password123",  
    roles: [  
        { role: "readWrite", db: "mydb" }  
    ]  
}  
);
```

## Drop User

```
db.dropUser("myuser");
```

## Enable Authentication

<https://docs.mongodb.com/manual/tutorial/enable-authentication/>



