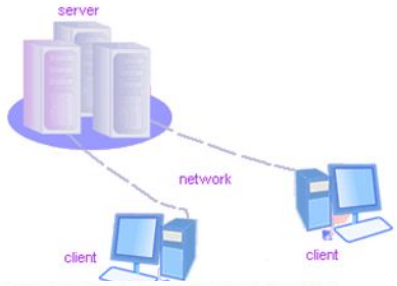
NoSQL

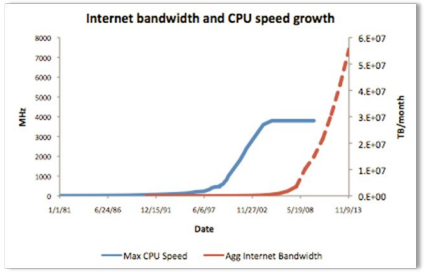
Carrera Programador full-stack

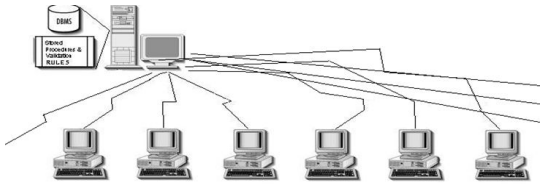
Introduccion

Contexto

- 1970's Aparecen las bases de datos relacionales
  - El almacenamiento es costoso
  - Los datos se normalizan
  - El almacenamiento es abstraído de la aplicación
- 1980's Aparecen versiones comerciales de las RDBMS
  - Modelo cliente/servidor
  - SQL emerge como estándar
- 1990's Las cosas empiezan a cambiar
  - Cliente/servidor => arquitectura 3-niveles
  - Aparecen el internet y la web
- 2000's Web 2.0
  - Aparece "Social Media"
  - Aceptación de E-Commerce
  - Continúan bajando precios de HW
  - Incremento masivo de datos coleccionados

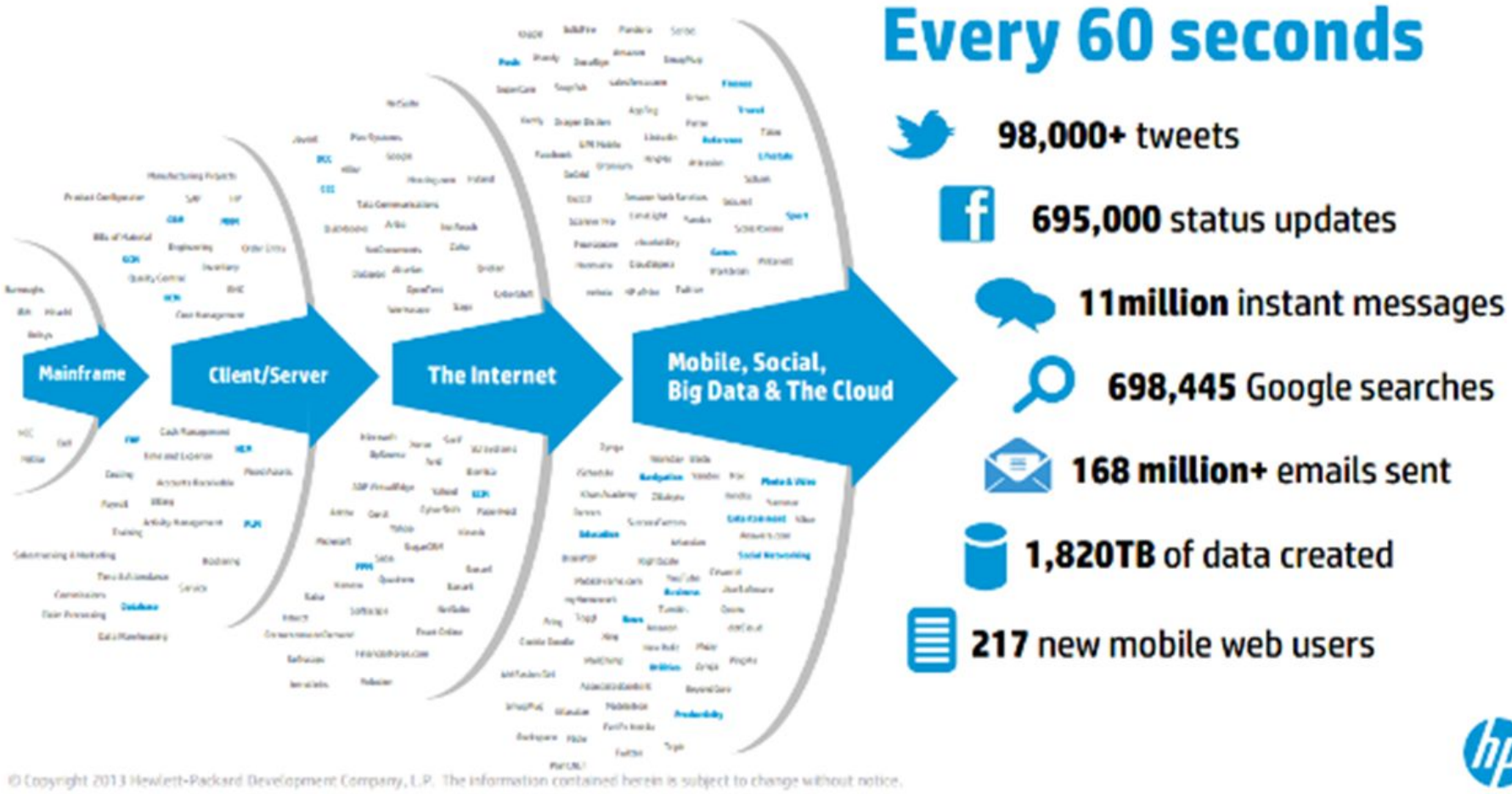






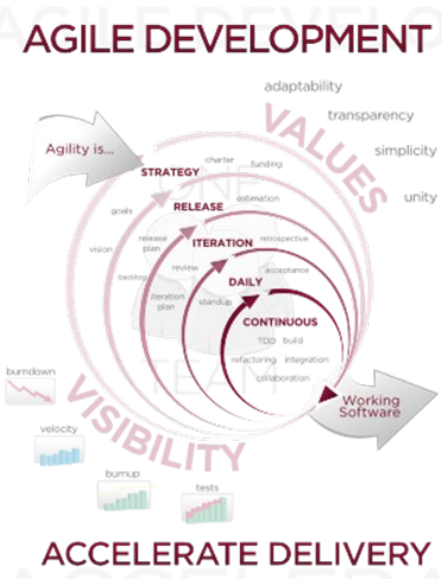
Contexto

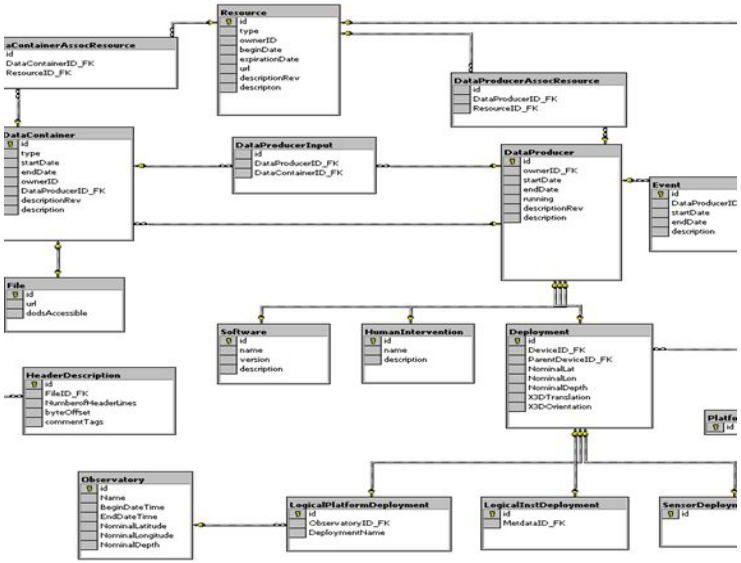
PRINCIPALES CAMBIOS QUE SE PRODUJERON EN LA TECNOLOGÍA Y EN LOS ÚLTIMOS 25 AÑOS



Desarrollo de Software

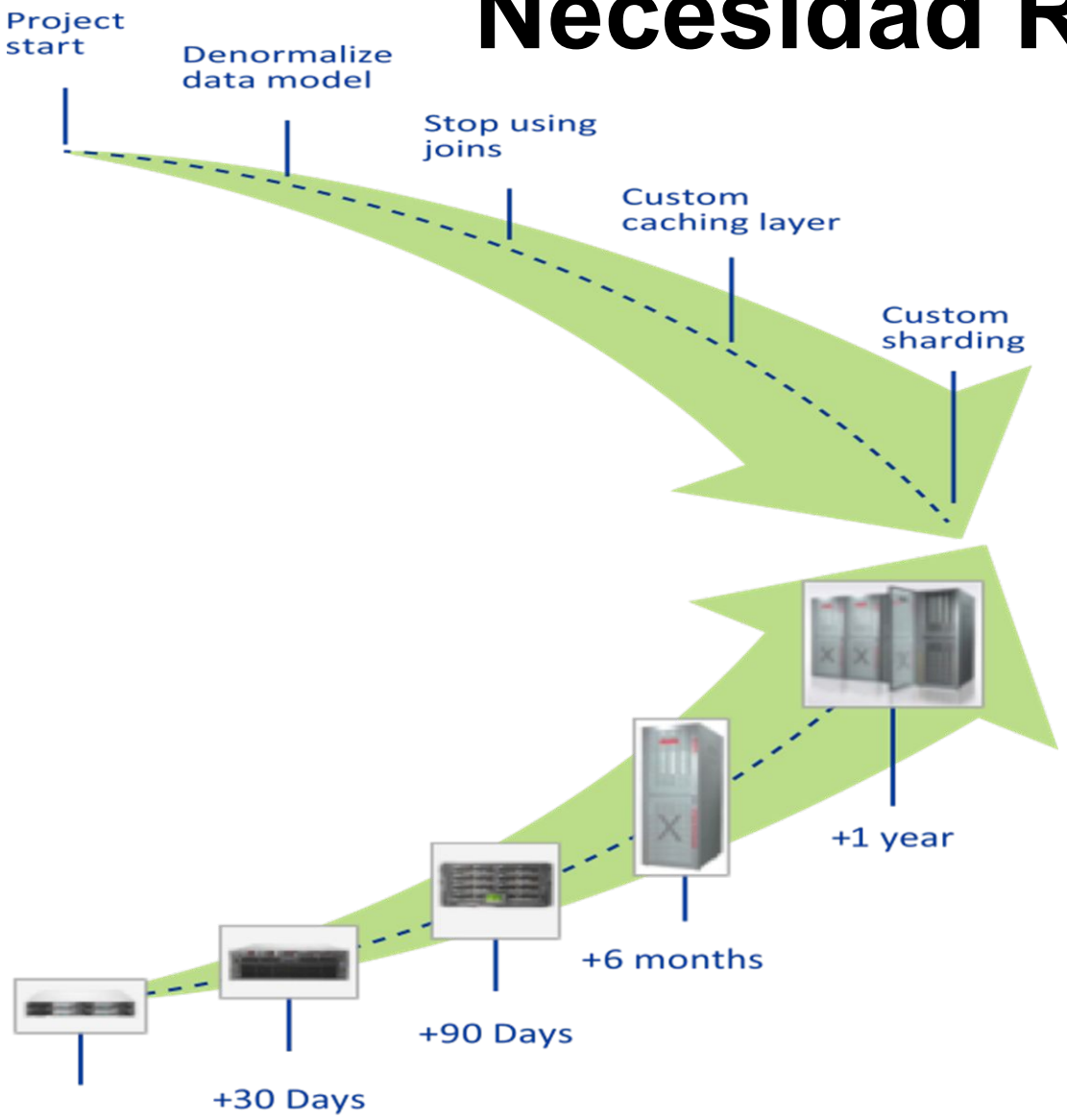
- **Metodología de desarrollo ágil**
  - Ciclos de desarrollo cortos
  - Constante evolución de requerimientos
  - Flexibilidad de diseño





- **Esquema relacional**
  - Difícil de evolucionar
    - Migraciones lentas y difíciles
    - En sincronía con la aplicación
  - Pocos desarrolladores interactúan directamente con la base de datos

Soluciones intermedias y Necesidad Real



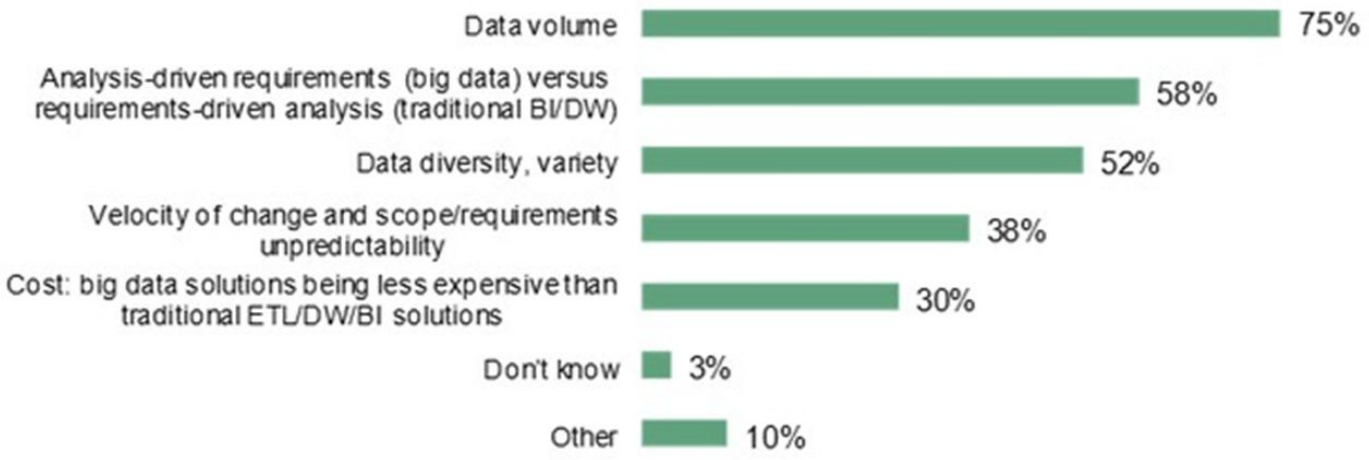
NECESIDAD REAL

- Escalabilidad horizontal
- Más resultados en tiempo real
- Desarrollo más veloz
- Modelo de datos flexible
- Bajo costo inicial
- Bajo costo de operación

Big Data

- Big Data es el sector de IT que hace referencia a **grandes conjuntos de datos** que por la velocidad a la que se generan, la capacidad para tratarlos y los **múltiples formatos y fuentes,** es necesario procesarlos con mecanismos distintos a los tradicionales.
- "**Volumen masivo de datos**, tanto **estructurados como no-estructurados**, los cuales son **demasiado grandes y difíciles de procesar** con las bases de datos y el software tradicionales." (ONU, 2012)
- En ambientes tradicionales de BI y DW primero se generan los requerimientos y luego las aplicaciones. Dicho de otra forma, los requerimientos direccionan las aplicaciones. En Big Data es al revés, ya que se utiliza la exploración de datos libre para generar hipótesis para encontrar un patrón

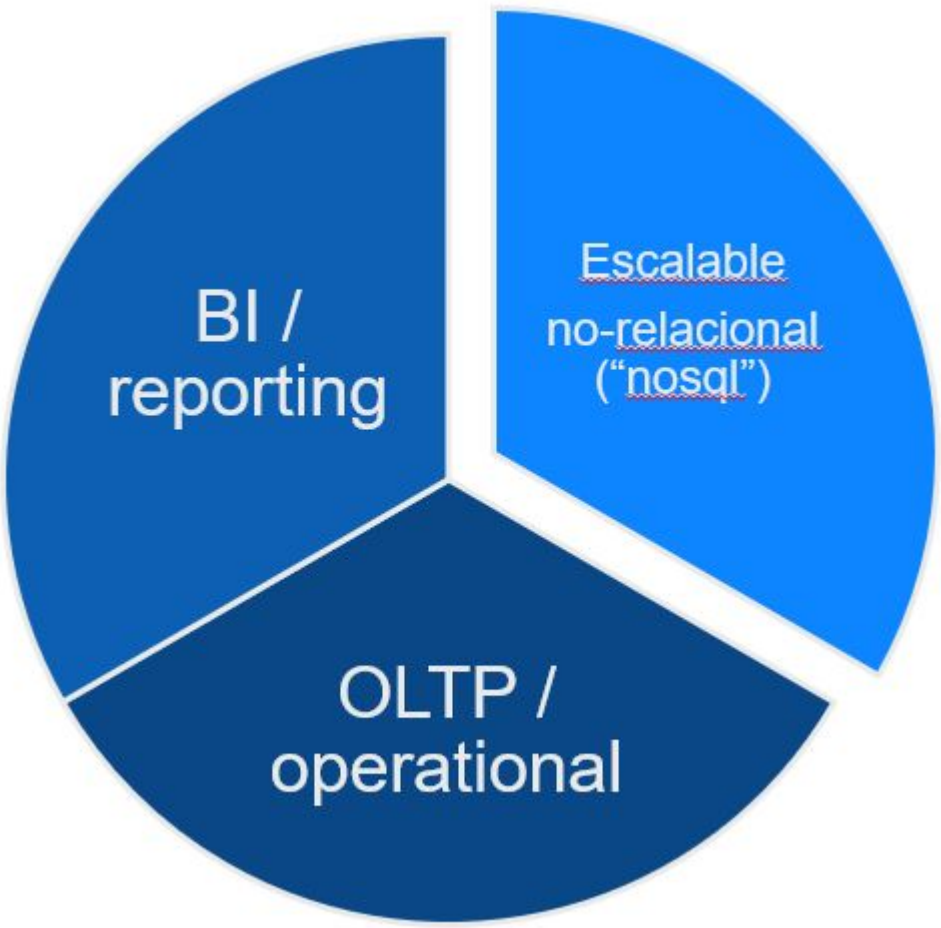
¿Cuales de las 4 Vs tienen mayor influencia?



- Volumen
- Velocidad
- Variedad
- "Veracidad"

- El costo es un factor en muchos casos. Las tecnologías utilizadas en Big Data son más económicas que las tradicionales

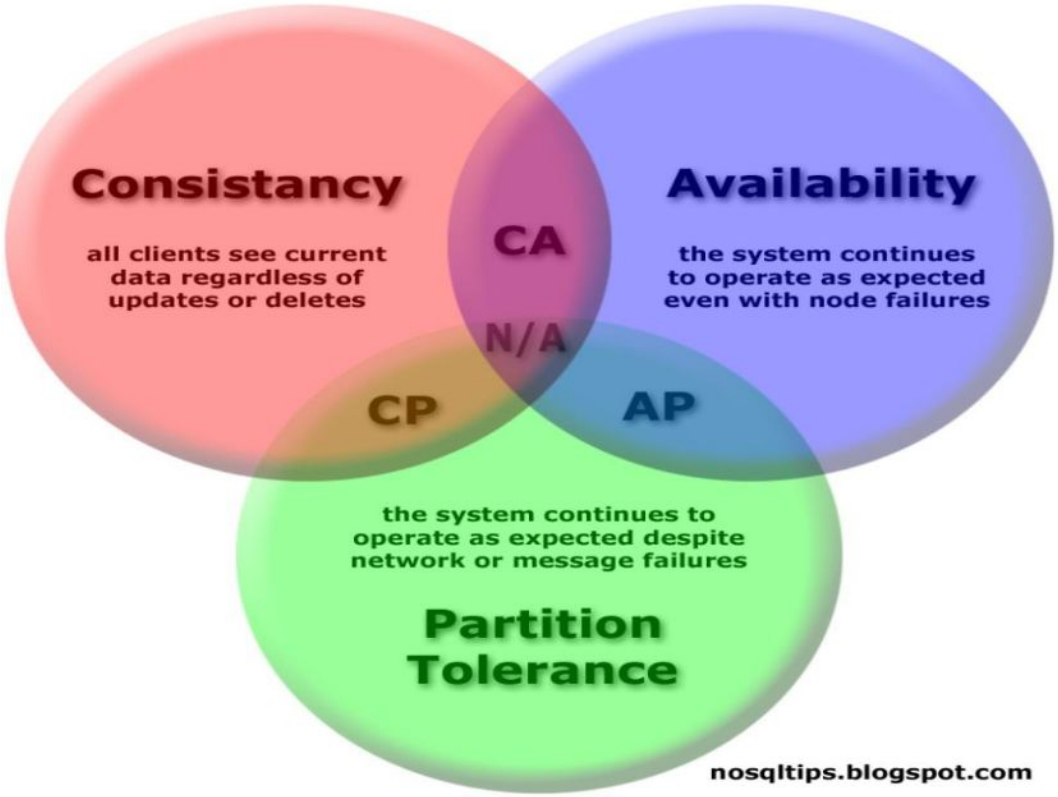
NoSQL al rescate



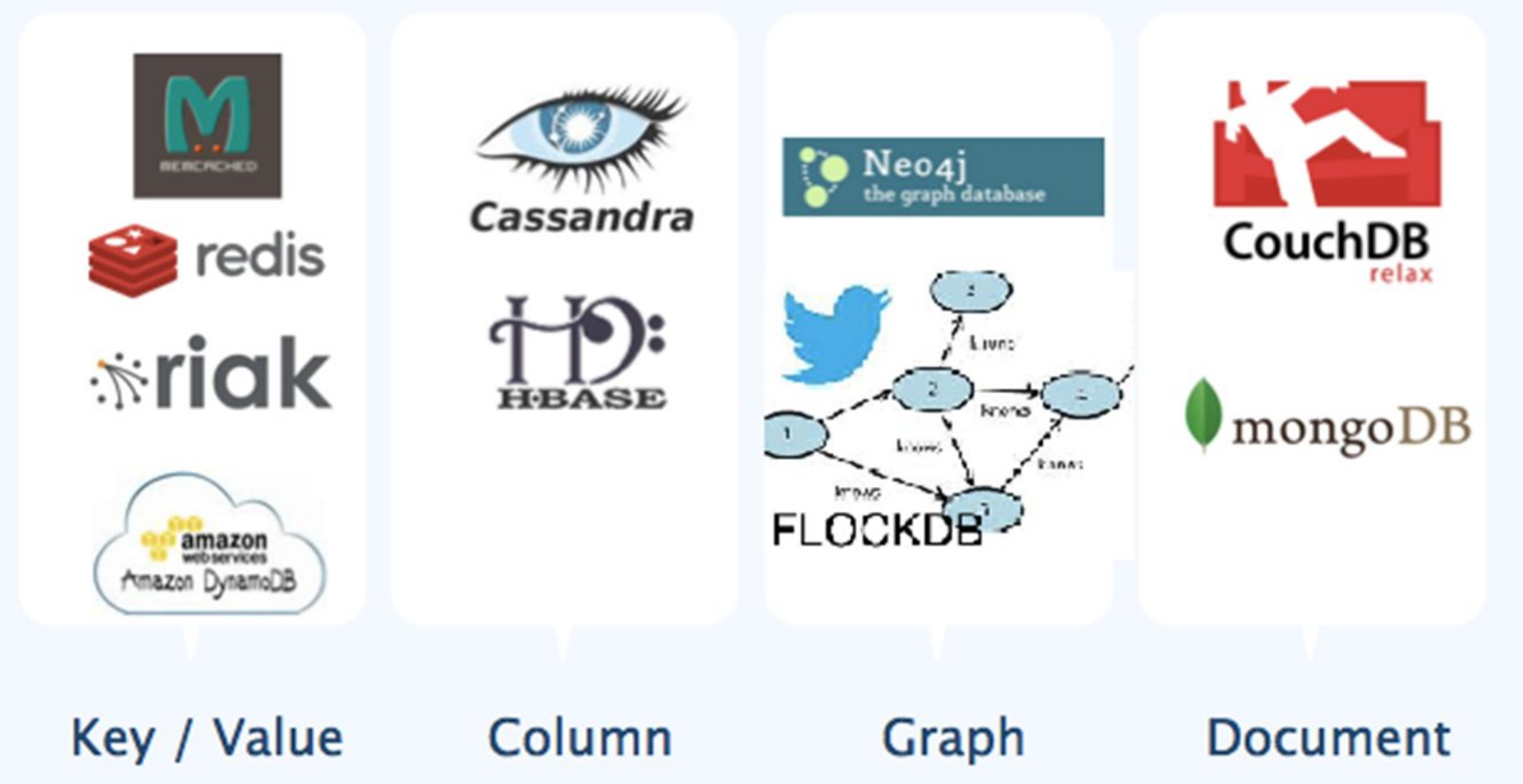
- velocidad y escalabilidad
  - consultas ad hoc limitadas
  - no son muy transaccionales
  - no usan SQL/no hay estándares
- se acoplan bien al modelo OO
- ágiles

NoSQL

- Sistema de gestión de bases de datos que difieren del modelo clásico de bases de datos relacionales: no usan SQL como lenguaje de consulta, los datos almacenados no requieren estructuras fijas como tablas, no garantizan consistencia plena y escalan horizontalmente.

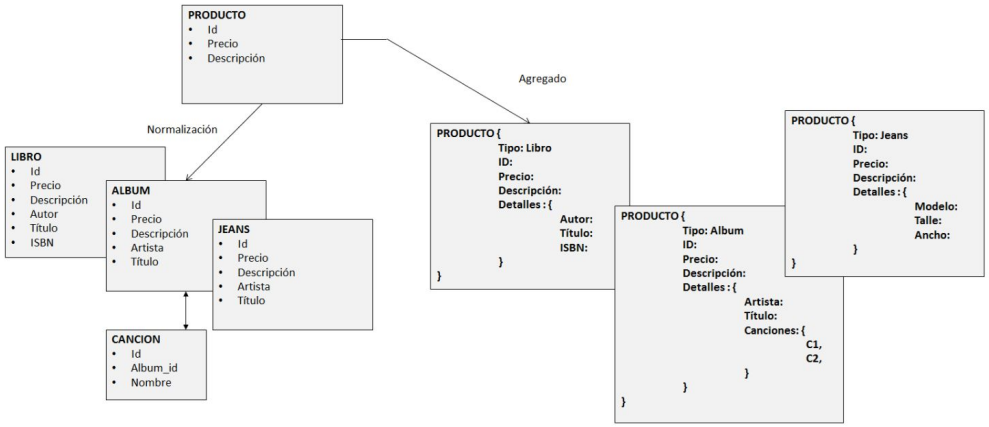


NoSQL - Tipos


memcached
redis
riak
amazon web services
Amazon DynamoDB
Cassandra
HBASE
Neo4j
the graph database
FLOCKDB
CouchDB
relax
mongoDB
Key / Value
Column
Graph
Document

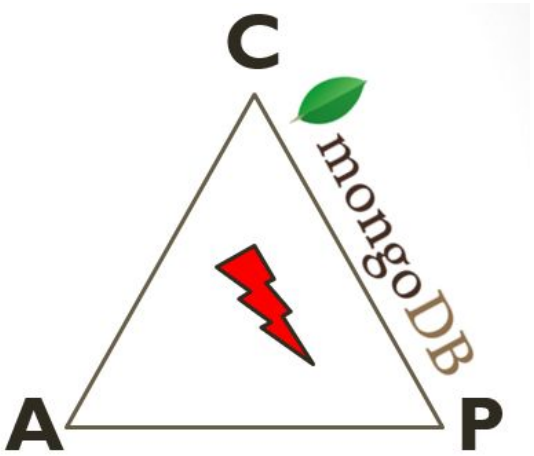
NoSQL - Principios básicos

- **De-Normalización**
  - Copiar los mismos datos en múltiples documentos para simplificar u optimizar el procesamiento de las consultas.
- **Agregados**
  - Un agregado es un registro complejo que permite listas y otras estructuras anidadas incluidas.
  - Minimizan de las relaciones uno a muchos a través de entidades anidadas y consecuentemente la reducción de operaciones de join



NoSQL - Teorema CAP

- **Consistency**
  - Todas las réplicas contienen la misma versión del dato
- **Availability**
  - El sistema permanece operativo a pesar qué algunos nodos fallen
- **Partition tolerance**
  - Múltiples puntos de entrada
  - El sistema permanece operativo ante divisiones del sistema



Teorema CAP: Es imposible satisfacer los 3 nodos al mismo tiempo

NoSQL - ACID vs BASE

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

- **B**asically **A**vailable (CP)
- **S**oft-state
- **E**ventually consistent (AP)

Relacional vs NoSQL

Relacional	**NoSQL**
Tablas (Filas y Columnas)	Key-Value Documentos XML Grafos Columnas
ACID (atomicity, consistency, isolation, durability)	BASE (basically available, soft state, eventual consistency)
Confirmación en 2 fases (two-phase commit)	Transacciones atómicas al nivel de documentos
Uniones (JOINS)	No hay uniones (JOINS)

NoSQL - MongoDB

- Su nombre surge de la palabra en inglés
- “hu**mongo**us” (que significa enorme).
- MongoDB guarda estructuras de datos en documentos tipo **JSON** (JavaScript Object Notation) con un esquema dinámico.
- Internamente MongoDB almacena los datos en formato **BSON** (Binary JavaScript Object Notation).
- BSON está diseñado para tener un almacenamiento y velocidad más eficiente.

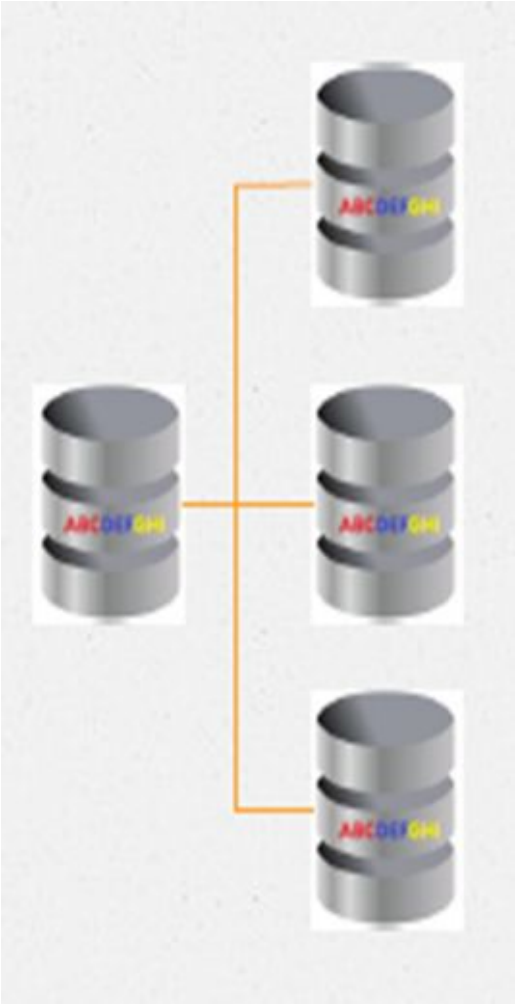
MongoDB - Instalación

1. Instalar MongoDB (Download MongoDB Community Server | MongoDB)
   a. Community Edition (https://fastdl.mongodb.org/windows/mongodb-windows-x86_64-7.0.0-signed.msi)
2. Instalar NoSQL Booster (GUI para MongoDB)
   a. Downloads - NoSQLBooster for MongoDB
3. Actualizar variable Path
   a. Buscar el directorio donde se instaló MongoDb (por ejemplo C:\Program Files\MongoDB\Server\3.2\bin)
   b. Añadir MongoDb a la variable Path (Inicio > Equipo > Propiedades del Sistema > Opciones avanzadas)
4. Crear una carpeta (con los permisos adecuados) para guardar la base de datos
   a. C:/Data/Db es la carpeta default para MongoDb
   b. Si quisiéramos tener otro path debemos ejecutar desde consola: mongod --dbpath ruta/nueva-a/la-carpeta-db
5. Abrir conexión desde Consola de Windows (Símbolo del Sistema) ejecutando el comando mongodb
6. Mantener la consola de conexión abierta y abrir una nueva consola para operar sobre la base

MongoDB - Cualidades

- **High Performance**
  - Alta performance en lectura y escritura.
  - Índices para lograr performance al consultar
- **Rich Query Language**
  - Lenguaje para Data Aggregation, Text Search y Geospatial Queries.
- **High Availability**
  - Usando replicación (replica set) se logra: **automatic failover** y **data redundancy**.
- **Horizontal Scalability**
  - Es una funcionalidad core. El **Sharding** distribuye la data across clusters.

MongoDB - Replicación

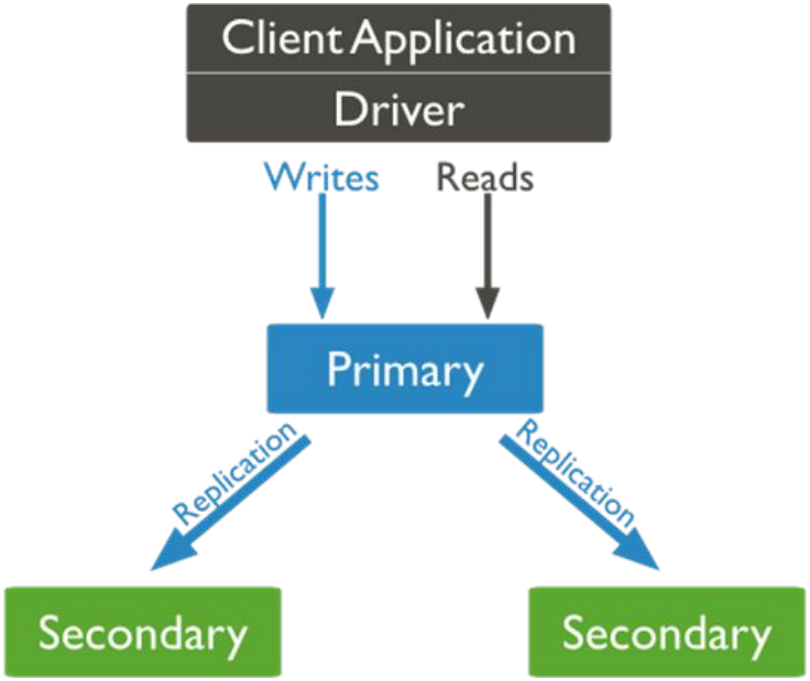


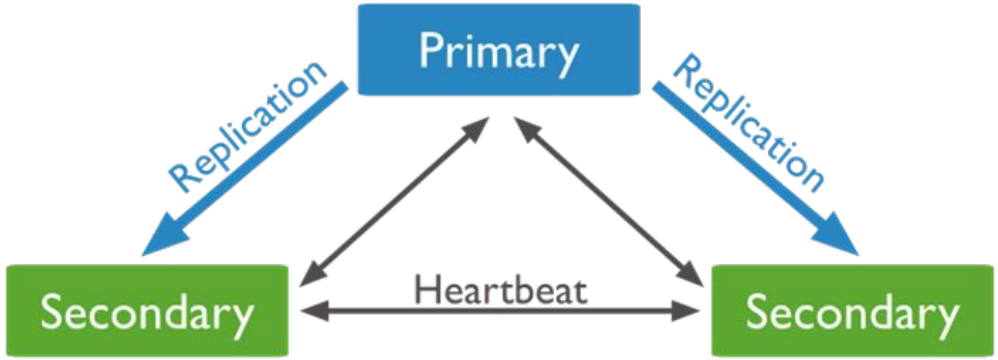
- Grupo de procesos mongod (nodos) que mantienen el mismo data set. Otorgando redundancia de datos y alta disponibilidad

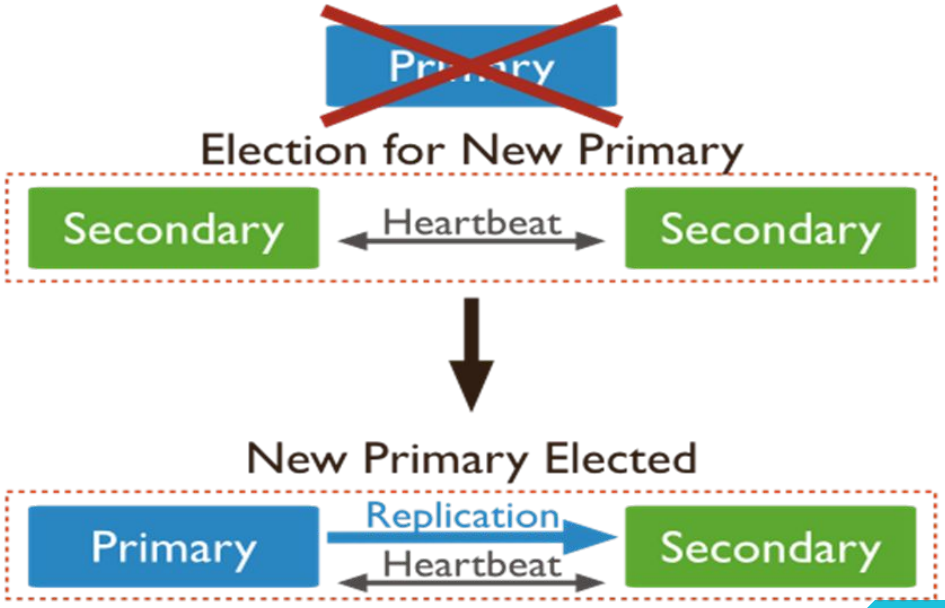
Facilidades

- Redundancia y disponibilidad
- Protección ante caídas
- Failure recovery
- Disaster Recovery

MongoDB - Replicación





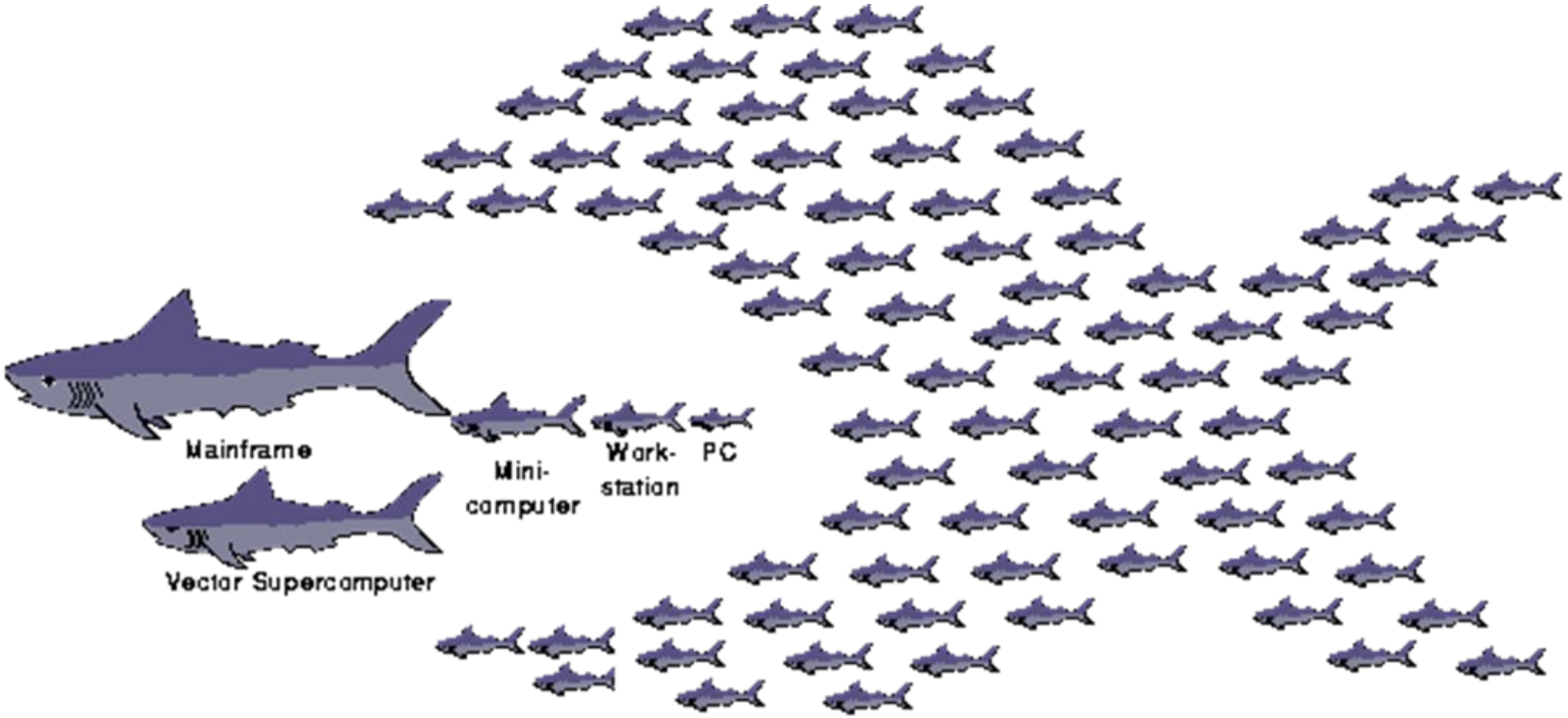


MongoDB - Replicación

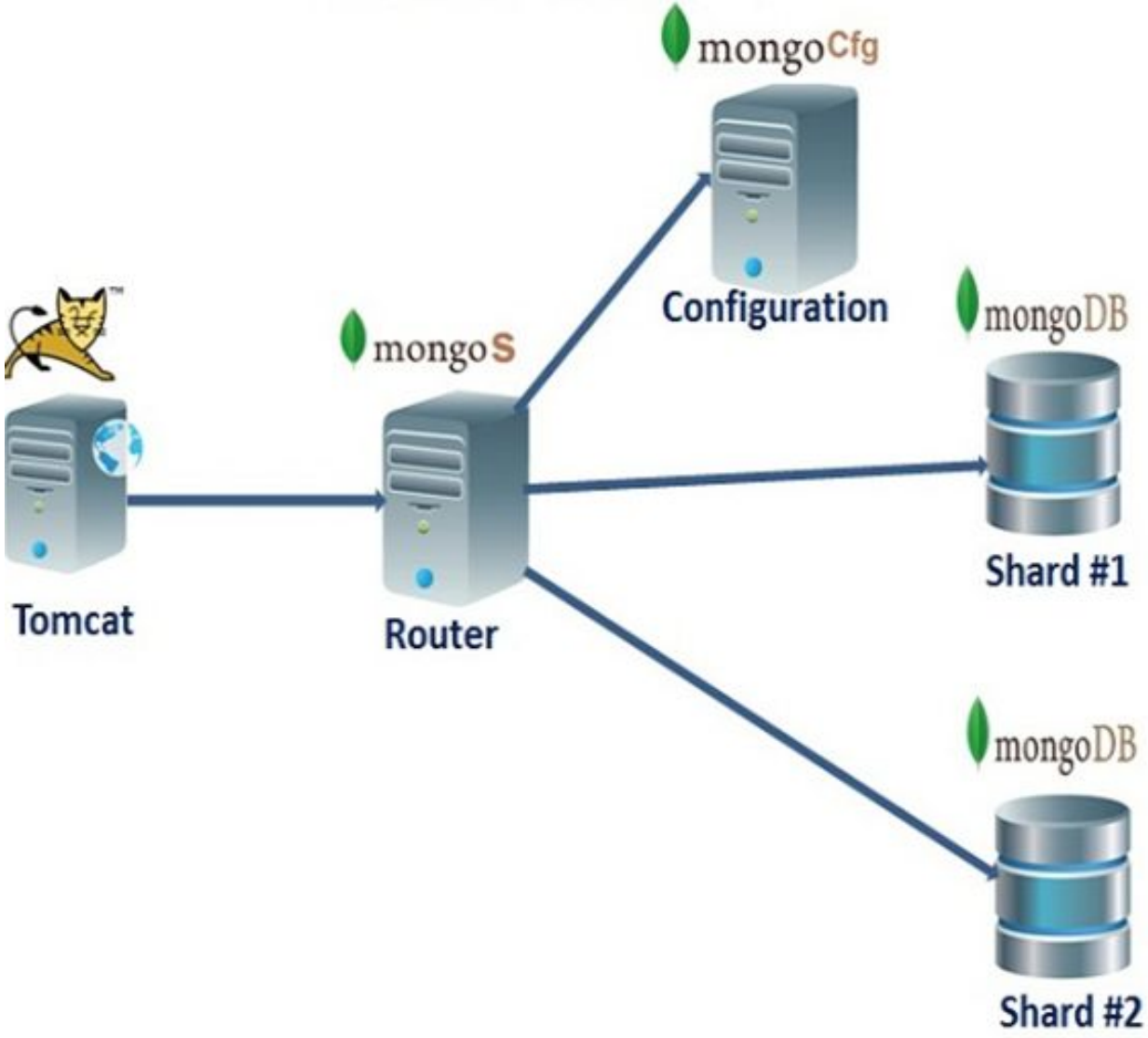
- Un nodo es designado como maestro (Primary).
- Los otros nodos son esclavos, o secundarios (Secondary).
- Mongo utiliza solo replicación por Master/Slave

MongoDB - Sharding

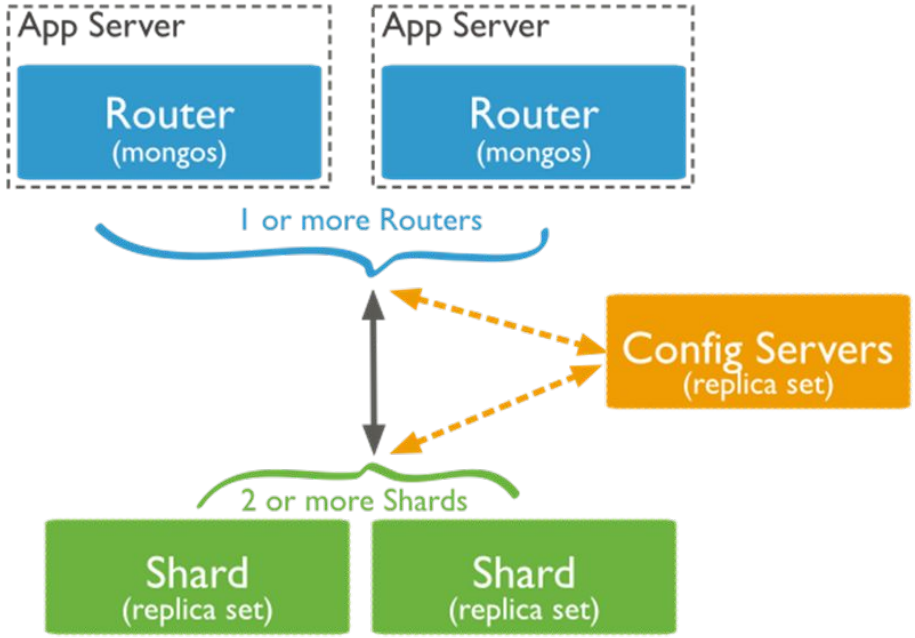
- Uno de los primeros requerimientos que justificaron la aparición del movimiento NoSQL fue la necesidad de poder trabajar en forma eficiente en ambientes basados en clusters, **ESCALAMIENTO HORIZONTAL**

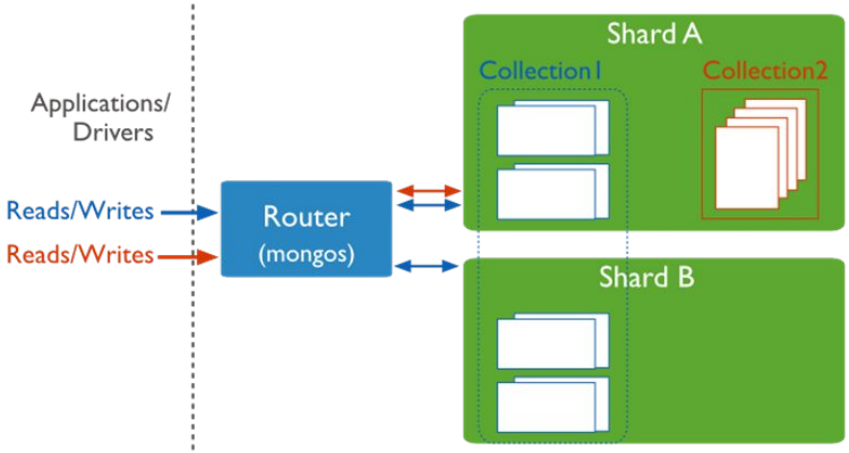


MongoDB - Sharding

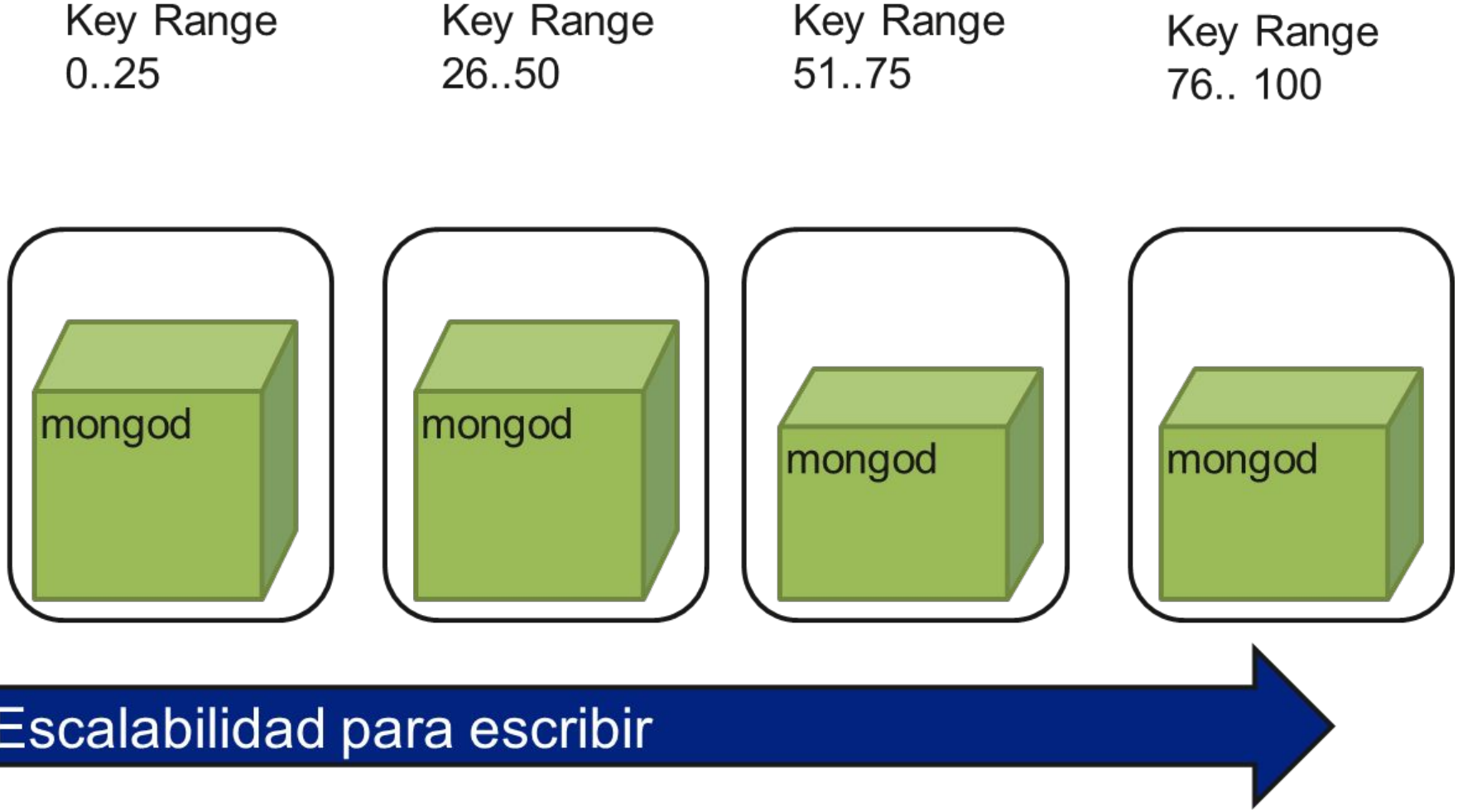


MongoDB - Sharding



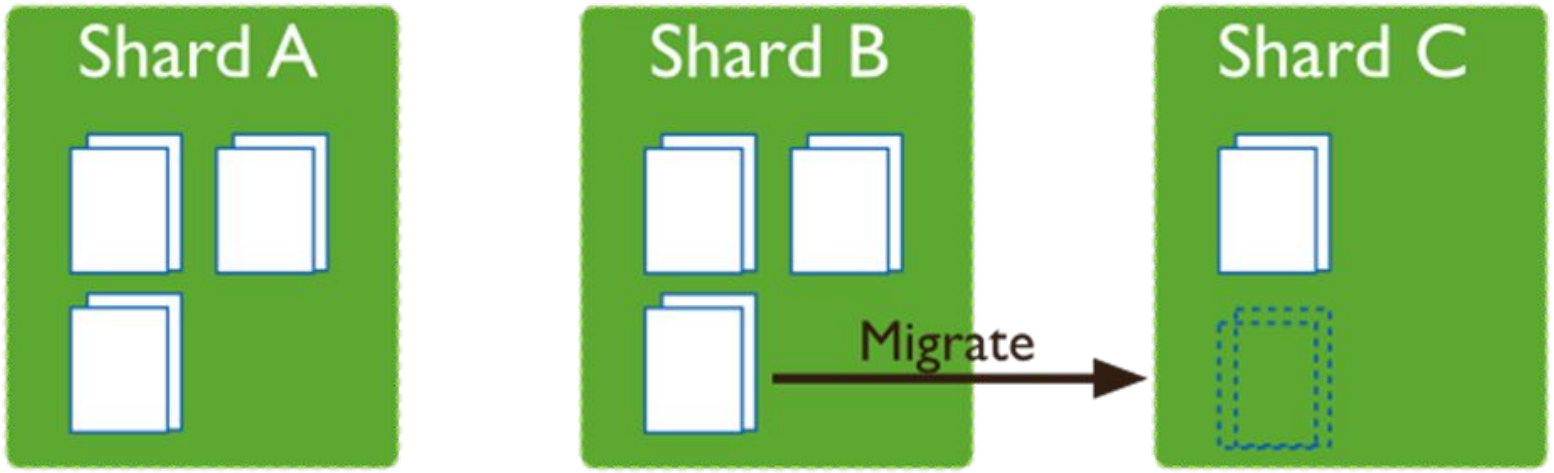


MongoDB - Sharding

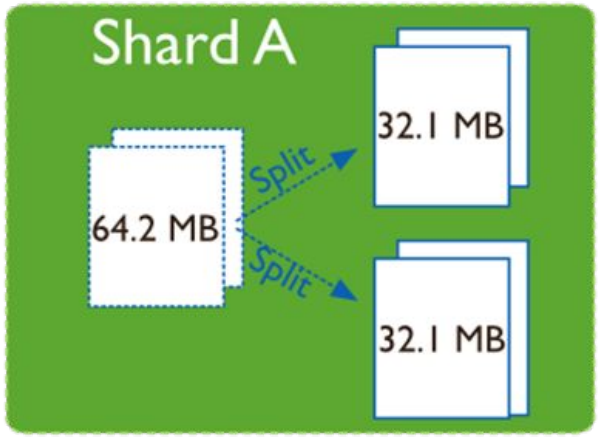


MongoDB - Sharding

MIGRACION



SPLIT



MongoDB - Estructura

RDBMS<		MongoDB
Database	⇨	Database
Table, View	⇨	Collection
Row	⇨	Document (JSON, BSON)
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨	Reference
Partition	⇨	Shard

```
{
  name: "sue",                          ← field: value
  age: 26,                              ← field: value
  status: "A",                          ← field: value
  groups: [ "news", "sports" ]          ← field: value
}
```

MySQL

```
START TRANSACTION;
INSERT INTO contacts VALUES
    (NULL, 'joeblow');
INSERT INTO contact_emails VALUES
   ( NULL, "joe@blow.com",
      LAST_INSERT_ID() ),
   ( NULL, "joseph@blow.com",
      LAST_INSERT_ID() );
COMMIT;
```

MongoDB

```
db.contacts.save( {
    userName: "joeblow",
    emailAddresses: [
      "joe@blow.com",
      "joseph@blow.com" ] } );
```

MongoDB - Base de datos & Collections

MongoDB - Estructura de documentos

```
var mydoc = {
               _id: ObjectId("5099803df3f4948bd2f98391"),
               name: { first: "Alan", last: "Turing" },
               birth: new Date('Jun 23, 1912'),
               death: new Date('Jun 07, 1954'),
               contribs: [ "Turing machine", "Turing test", "Turingery" ],
               views : NumberLong(1250000)
             }
```

- MongoDB almacena los datos internamente como documentos BSON.
- BSON es una representación binaria de los documentos JSON
- El valor de un campo puede ser cualquiera de los tipos de datos BSON, incluyendo otros documentos, matrices y matrices de documentos.

MongoDB - Modelado de relaciones

Relaciones uno a uno con documentos embebidos

```
Colección Personas
{ _id: "u0001",
nombre: "Juan Martín Hernandez" }
```

```
Colección Direcciones
{ persona_id: "u0001",
calle: "Malabia 2277",
ciudad: "CABA",
provincia: "CABA",
codPostal: "1425" }
```

```
Colección Personas
{ _id: "u0001",
  nombre: "Juan Martín Hernandez",

  direccion:{calle: "Malabia 2277",
             ciudad: "CABA",
             provincia: "CABA",
             codPostal: "1425" }
}
```

MongoDB - Modelado de relaciones

Relaciones uno a muchos con documentos embebidos

Colección Personas
```
{ _id: "u0001",
nombre: "Juan Martín Hernandez" }
```

Colección Direcciones
```
{ persona_id: "u0001",
calle: "Malabia 2277",
ciudad: "CABA",
provincia: "CABA",
codPostal: "1425" }
```

```
{persona_id: "u0001",
calle: "Av. Santa Fe 3455",
ciudad: "Mar del Plata",
provincia: "Buenos Aires",
codPostal: "7600" }
```

Colección Personas
```
{ _id: "u0001",
  nombre: "Juan Martín Hernandez",

  direcciones:[{calle: "Malabia 2277",
                ciudad: "CABA",
                provincia: "CABA",
                codPostal: "1425" },

               {calle: "Av. Santa Fe 3455",
                ciudad: "Mar del Plata",
                provincia: "Buenos Aires",
                codPostal: "7600" }
              ]
}
```

MongoDB - Modelado, cuando usar?

- **Logging de Eventos**
  - las bases de datos basadas en documentos puede loguear cualquier clase de eventos y almacenarlos con sus diferentes estructuras.
  - Pueden funcionar como un repositorio central de logueo de eventos.
- **CMS, blogging**
  - su falta de estructura predefinida hace que funcionen bien para este tipo de aplicaciones.
- **Web-analytics / Real-Time analytics**
  - Almacenar cantidad de vistas a una página o visitantes únicos.
- **Commerce**
  - A menudo requieren tener esquemas flexibles para los productos y órdenes

MongoDB - CRUD

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

MongoDB - Ejemplo de CRUD

```
> db.user.insert({
        first: "John",
        last : "Doe",
        age: 39
})
```

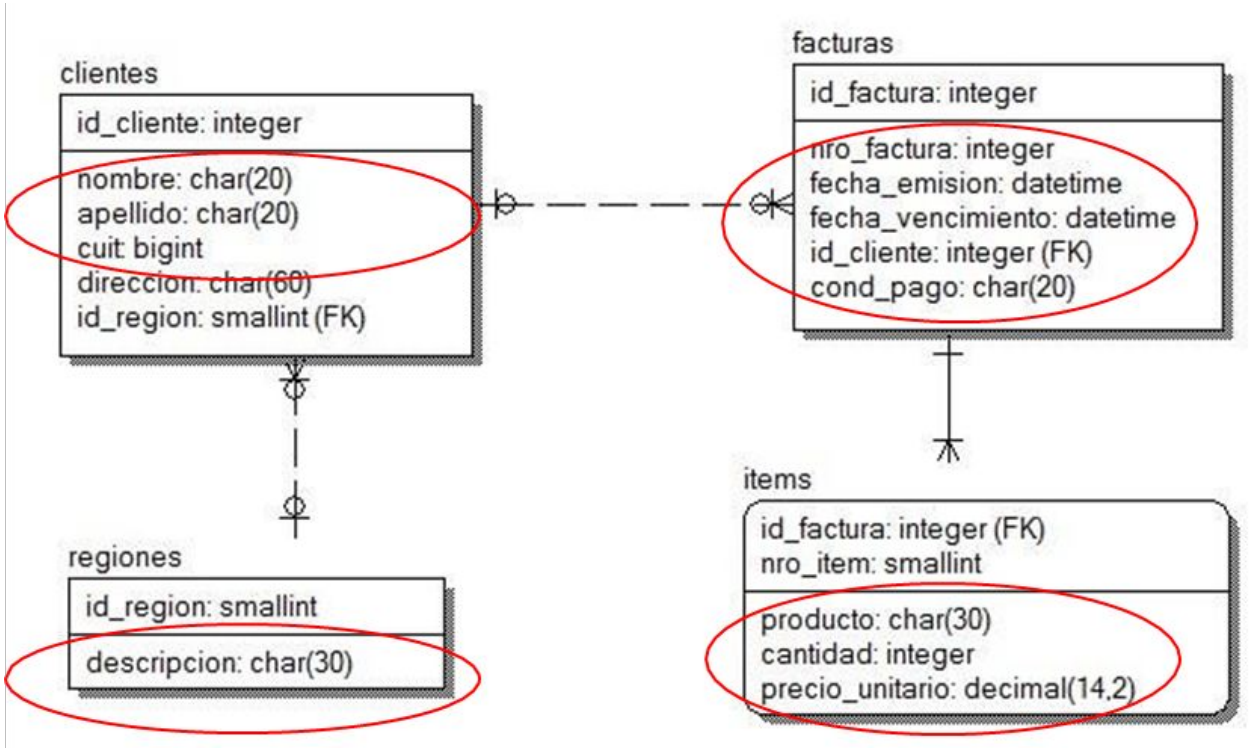
```
> db.user.find ()
{
        "_id" : ObjectId("51…"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39
}
```

```
> db.user.update(
        {"_id" : ObjectId("51…")},
        {
            $set: {
                age: 40,
                salary: 7000}
        }
)
```

```
> db.user.remove({
        "first": /^J/
})
```

MongoDB - Caso Práctico

- **Armemos un modelo que contenga la información de las facturas y todos sus ítems, detallando el nombre, apellido, cuit y región del cliente al que se le emitió la factura, para poder realizar consultas desde un portal de factura de la forma más performante posible**



MongoDB - Operaciones

- **Inicio de servidor mongo por consola**
  - c:> mongod
- **Inicio de cliente mongo por consola**
  - c:> mongosh
- **Ver Bases de datos**
  - Show dbs
- **Ver colecciones**
  - show collections

MongoDB - Operaciones sobre una colección

- **Ver todos los comandos:**
  - db.facturas.help()
- **Ver todos los documentos:**
  - db.facturas.find()

```
> db.facturas.find()
{ "_id" : ObjectId("534b52b443ad02d44397b71b"), "nroFactura" : 1448, "fechaEmisi
on" : ISODate("2014-02-20T00:00:00Z"), "fechaVencimiento" : ISODate("2014-03-22T
00:00:00Z"), "condPago" : "30 Ds FF", "cliente" : { "nombre" : "Martin", "apelli
do" : "Zavasi", "cuit" : 2038373771, "region" : "CABA" }, "item" : [  {  "produc
to" : "CORREA 10mm",  "cantidad" : 2,  "precio" : 134 } ] }
{ "_id" : ObjectId("534b52bc43ad02d44397b71c"), "nroFactura" : 1449, "fechaEmisi
on" : ISODate("2014-02-20T00:00:00Z"), "fechaVencimiento" : ISODate("2014-02-20T
00:00:00Z"), "condPago" : "CONTADO", "cliente" : { "nombre" : "Martín", "apellid
o" : "Zavasi", "cuit" : 2038373771, "region" : "CABA" }, "item" : [     {
"producto" : "TUERCA 2mm",      "cantidad" : 6,         "precio" : 60 },
{       "producto" : "CORREA 10mm",     "cantidad" : 12,        "precio" : 134 }
 ] }
```

- **Cantidad de documentos en la colección**
  - db.facturas.count()
- **Espacio ocupado pro los documentos de la colección**
  - db.facturas.find()

MongoDB - Operaciones sobre una colección

- **Buscar documento para cliente con determinado apellido**
  - db.facturas.find({"cliente.apellido":"Malinez"})
- **Consultar los primeros dos documentos**
  - db.facturas.find().limit(2)
- **Consultar documentos salteando los primeros dos documentos**
  - db.facturas.find().skip(2)
- **Visualización mejorada**
  - db.facturas.find().pretty()
- **Consultar dos documentos, salteando los dos primeros documentos de una colección, mostrándolos en un modo mejorado.**
  - db.facturas.find().limit(2).skip(2).pretty()
- **Consultar documentos sólo mostrando algunos datos**
  - db.facturas.find({"cliente.apellido":"Malinez"}, {"cliente.cuit":1, "cliente.region":1})
- **Buscar documento para cliente con dos criterios**
  - Para Zavasi teníamos dos documentos: db.facturas.find({"cliente.apellido":"Zavasi"}).pretty()
  - Agregamos un criterio: db.facturas.find({"cliente.apellido":"Zavasi", "nroFactura":1001.0}).pretty()
- **Ordenamiento en forma ascendente**
  - db.facturas.find().sort({nroFactura:1})
- **Ordenamiento en forma descendente**
  - db.facturas.find().sort({nroFactura:-1})