MÁSTER: "Ciencia de Datos e Ingeniería de Computadores"

Universidad de Granada

ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Big data I: Cloud Computing y Almacenamiento Masivo de Datos

Agenda

1. **Introducción a las bases de datos NoSQL.**
   - Concepto. Principales características.
   - BBDD distribuidas. Teorema CAP.
   - RDBMS vs NoSQL.
   - Arquitectura de las bases de datos NoSQL.
     ACID vs BASE.
   - Para qué es útil NoSQL. ¿Qué tipo de BD utilizo?
   - Tipos de bases de datos NoSQL.
   - NoSQL en Cloud Computing. Uso para Big Data.

Agenda

2. MongoDB: una BD NoSQL orientada a Documentos.

- Introducción. Características principales: Consultas Ad hoc, Indexación, Replicación, Balanceo de carga, Agregación, etc.
- Áreas de aplicación.
- Documentación e instalación sobre diferentes sistemas operativos.
- Utilidades de mongoDB: mongo, mongostat, mongotop, etc.
- Herramientas gráficas para administración y uso de MongoDB.
- La "shell" mongo.
- Correspondencia entre conceptos RDBMS con los conceptos de MongoDB.
- Operaciones CRUD. Índices.
- Agregación (pipeline) y MapReduce.
- Transacciones.
- Replicación y "sharding".
- MongoDB y Hadoop.

NoSQL. Introducción

- **NoSQL** – "not only SQL" – es una categoría general de sistemas de gestión de bases de datos que difiere de los RDBMS en diferentes modos:
  - No tienen schemas, no permiten JOINs, no intentan garantizar ACID y escalan horizontalmente.
  - Tanto las bases de datos NoSQL como las relacionales son tipos de **Almacenamiento Estructurado**.
- El término fue acuñado en 1998 por Carlo Strozzi y resucitado en 2009 por Eric Evans
  - Evans sugiere mejor referirse a esta familia de BBDD de nueva generación como **"Big Data"** mientras que Strozzi considera ahora que **NoREL** es un mejor nombre

NoSQL. Introducción

Las BBDD NoSQL se han diseñado para potenciar aspectos como:

- **Distribución:**

  La distribución de los datos proporciona beneficios adicionales.

- **Escalabilidad:**

  Mediante la distribución, la escalabilidad horizontal es prácticamente ilimitada.

- **Disponibilidad:**

  La distribución y la replicación mantienen los datos disponibles.

- **Flexibilidad:**

  Nuevos tipos de datos se pueden incorporar posteriormente a la creación de una BD.

NoSQL. Introducción

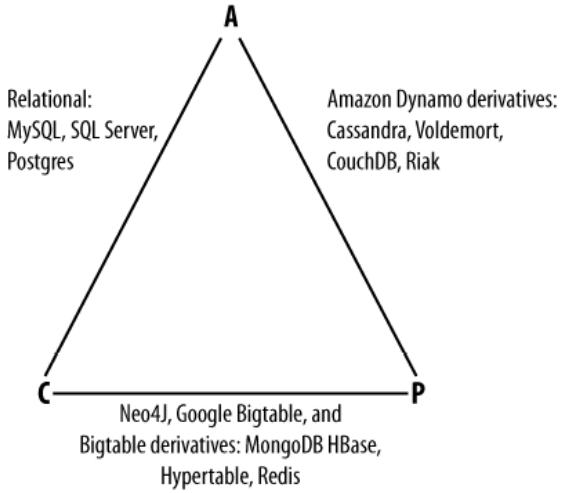
- La principal diferencia radica en cómo guardan los datos (por ejemplo, *almacenamiento de un recibo*):
  - En un RDBMS tendríamos que partir la información en diferentes tablas y luego usar el DML de la parte servidora para transformar estos datos en objetos de la vida real.
  - En NoSQL, simplemente guardas el recibo:
    - NoSQL es **libre de schemas**, tú no diseñas tus tablas y su estructura por adelantado
- **¡¡¡NoSQL no es la panacea!!!**
  - Si tus datos son relacionales, quedarte con tu RDBMS sería generalmente la opción correcta …
    - Aunque algunos NoSQL como MongoDB siempre puede ser una alternativa en determinados casos.

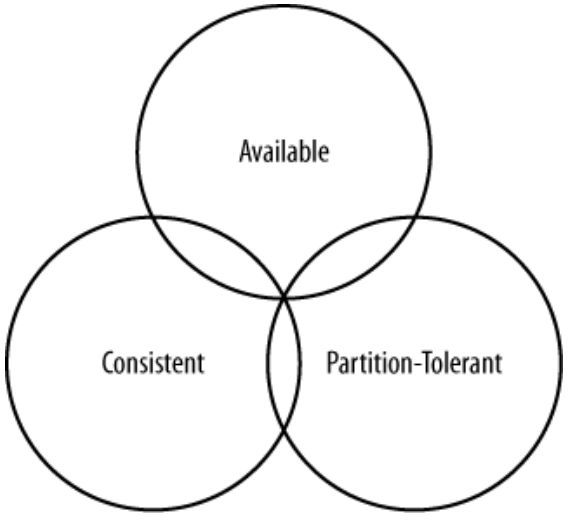
NoSQL. Conceptos BBDD distribuidas

- **Almacenes basados en columnas y filas**. RDBMS almacenan las filas de modo continuo en disco, mientras que algunas NoSQL guardan así las columnas.
- **Consistencia eventual**: si no se realizan nuevas actualizaciones a un elemento de datos, todos los accesos del elemento devolverán el último valor actualizado.
- **Sharding**: es una partición horizontal en una BD donde las filas de una tabla se mantienen de modo separado.
- **Replicación maestro-maestro** es un método de replicación de BD que permite almacenar datos en un grupo de nodos y su actualización por cualquier miembro del grupo.
- **Replicación maestro-esclavo** donde un sólo elemento se designa como "maestro" de un datastore y es el único nodo que permite modificar datos.
- **Particionado** es la división lógica de una BD lógica y sus partes constituyentes en un conjunto de partes independientes.
- **Modelo de consistencia**: garantías de consistencia sobre los datos que ofrece el sistema a las aplicaciones (write & read consistency: one, all, quorum, etc.).

RDBMS vs NoSQL. El teorema CAP

- Teorema de Brewer: "es imposible para un sistema computacional distribuido ofrecer simultáneamente las siguientes tres garantías":
  - ***C**onsistency* : todos los nodos ven los mismos datos al mismo tiempo.
  - ***A**vailability (Disponibilidad):* garantiza que cada petición recibe una respuesta acerca de si tuvo éxito o no. Cada cliente siempre puede leer y escribir.
  - ***P**artition tolerance:* el sistema tiene que seguir funcionando aunque existan fallos o caídas parciales en alguna de las partes en que se divida el sistema.
- Equivalente a:
  - **"You can have it good, you can have it fast, you can have it cheap: pick two."**





RDBMS vs NoSQL. ACID vs. BASE

- En el mundo relacional estamos familiarizados con las transacciones ACID (Atomiciad, Consistencia, Isolation (aislamiento) y Durabilidad), que garantizan la consistencia y estabilidad de las operaciones pero requieren una gestión de bloqueo sofisticada:
  - **Atomicidad:** es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto, no puede quedar a medias ante un fallo del sistema. Por ejemplo, en el caso de una transacción bancaria o se ejecuta tanto el depósito como la deducción o ninguna acción es realizada.
  - **Consistencia:** *Integridad*. La propiedad de consistencia sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido (de acuerdo con las restricciones y reglas establecidas en la BD).
  - **Aislamiento:** La ejecución concurrente de dos transacciones da como resultado un estado del sistema que sería el mismo que si se hubieran ejecutado secuencialmente.
  - **Durabilidad:** *Persistencia*. Es la propiedad que asegura que una vez establecida una transacción, ésta persistirá y no se perderán los cambios aunque falle el sistema.

RDBMS vs NoSQL. ACID vs. BASE

- Las BBDD NoSQL son repositorios de almacenamiento más optimistas , siguen el **modelo BASE**:
  - **BA**sic availability: disponibilidad de los datos según el teorema CAP. Siempre se obtiene una respuesta del sistema a una petición de datos aunque esta sea un fallo o que sean inconsistentes o estén en fase de cambio.
  - **S**oft-state: el estado del sistema cambia constantemente a lo largo del tiempo, incluso aunque no hayan entradas de datos en ese periodo, debido a la consistencia eventual.
  - **E**ventual consistency: eventualmente, el sistema se volverá consistente a partir de que deje de recibir datos. Los datos se propagarán pero el sistema seguirá recibiendo datos sin evaluar la consistencia de los datos para cada transacción antes de avanzar a la siguiente.
- **BASE es una alternativa flexible a ACID** para aquellos almacenes de datos que no requieren un adherencia estricta al modelo relacional.

NoSQL. Caracteristicas

- A menudo ofrecen sólo **garantías de consistencia débiles**, como por ejemplo *eventual consistency*, o transacciones restringidas a elementos de datos simples.
- Emplean una **arquitectura distribuida**, donde los datos se guardan de modo redundante en distintos servidores, a menudo usando tablas hash distribuidas.
- Suelen ofrecer **estructuras de datos sencillas** como arrays asociativos o almacenes de pares clave-valor.

NoSQL. Características

- Fáciles de usar en clústers de balanceo de carga convencionales → facilitan escalabilidad horizontal.
- Guardan datos persistentes (no sólo cachés).
- No tienen esquemas fijos y permite la migración del esquema sin tener que ser reiniciadas o paradas.
- Suelen tener un sistema de consultas propio en vez de usar un lenguaje de consultas estándar.
- Tienen propiedades ACID en un nodo del clúster y son "eventualmente consistentes" en el clúster.

¿Por qué se necesita NoSQL?

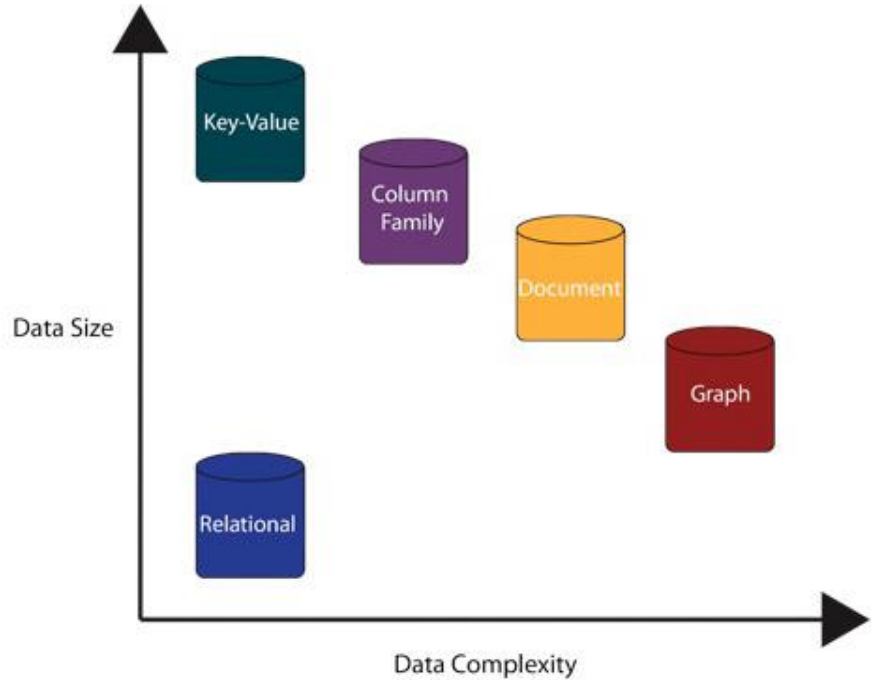
- Las aplicaciones web modernas, la computación ubicua y el Big Data presentan desafíos muy distintos a las que presentan los sistemas empresariales tradicionales (e.j. sistemas bancarios):
  - Datos a escala web.
  - Procesamiento masivo de datos.
  - Alta frecuencia de lecturas y escrituras.
  - Cambios frecuentes del esquema de datos.
  - Las aplicaciones sociales (no bancarias) no necesitan el mismo nivel de ACID.

¿Por qué se necesita NoSQL?

- Desafíos en un sistema de información que son difíciles de resolver usando tecnología de bases de datos relacionales:
  - La BD no escala con el tráfico a un coste aceptable.
  - El tamaño del esquema de datos crece desproporcionadamente.
  - El sistema de información genera muchos datos temporales que no corresponden al almacén de datos principal (carritos de compra, personalización de portales).
  - La BD ha sido desnormalizada por razones de rendimiento o por conveniencia para utilizar los datos en una aplicación.
  - La BD contiene grandes cantidades de texto o imágenes en columnas como BLOBs.
  - Se ejecutan consultas sobre los datos que implican relaciones jerárquicas complejas; recomendaciones o consultas de inteligencia de negocio.
  - Se usan transacciones locales que no necesitan ser muy durables.
    - Los sitios AJAX tienen muchos casos de uso de este estilo.

NoSQL. Tipos de BBDD

- **Orientadas a columnas (Col):** Pensadas para realizar consultas y agregaciones sobre grandes cantidades de datos. Funcionan de forma parecida a las bases de datos relacionales, pero almacenando columnas de datos en lugar de registros.
- **Orientadas a documentos (Doc):** Gestionan datos semi estructurados. Es decir documentos. Estos datos son almacenados en algún formato estándar como puede ser XML, JSON o BSON.
- **De clave-valor (Key):** Guardan tuplas que contienen una clave y su valor. Cuándo se quiere recuperar un dato, simplemente se busca por su clave y se recupera el valor.
- **Basadas en grafos (Graf):** Basadas en la teoría de grafos utilizan nodos y aristas para representar los datos almacenados. Son muy útiles para guardar información en modelos con muchas relaciones, como redes y conexiones sociales.



BBDD orientadas a Columnas

- Buenas en:
  - Modelado de datos natural.
  - Amigables al programador.
  - Desarrollo rápido.
  - Orientadas a la web: CRUD.
- Ejemplos destacables:
  - Cassandra: incluida en esta categoría, aunque en realidad sigue un modelo híbrido entre orientada a columnas y clave-valor. Es utilizada por Facebook y Twitter (aunque dejaron de usarla para almacenar tweets).
  - HBase. Escrita en Java y mantenida por el Projecto Hadoop de Apache, se utiliza para procesar grandes cantidades de datos. La utilizan Facebook, Twitter o Yahoo.

BBDD orientadas a Documentos

- Buenas en:
  - Modelado de datos natural.
  - Amigables al programador.
  - Desarrollo rápido.
  - Orientadas a la web: CRUD
- En esta categoría encontramos:
  - MongoDB: probablemente la base de datos NoSQL más famosa del momento. Algunas compañías que actualmente utilizan MongoDB son Foursquare o eBay.
  - CouchDB: es la base de datos orientada a documentos de Apache. Una de sus interesantes características es que los datos son accesibles a través de una API Rest. Este sistema es utilizado por compañías como Credit Suisse y la BBC.

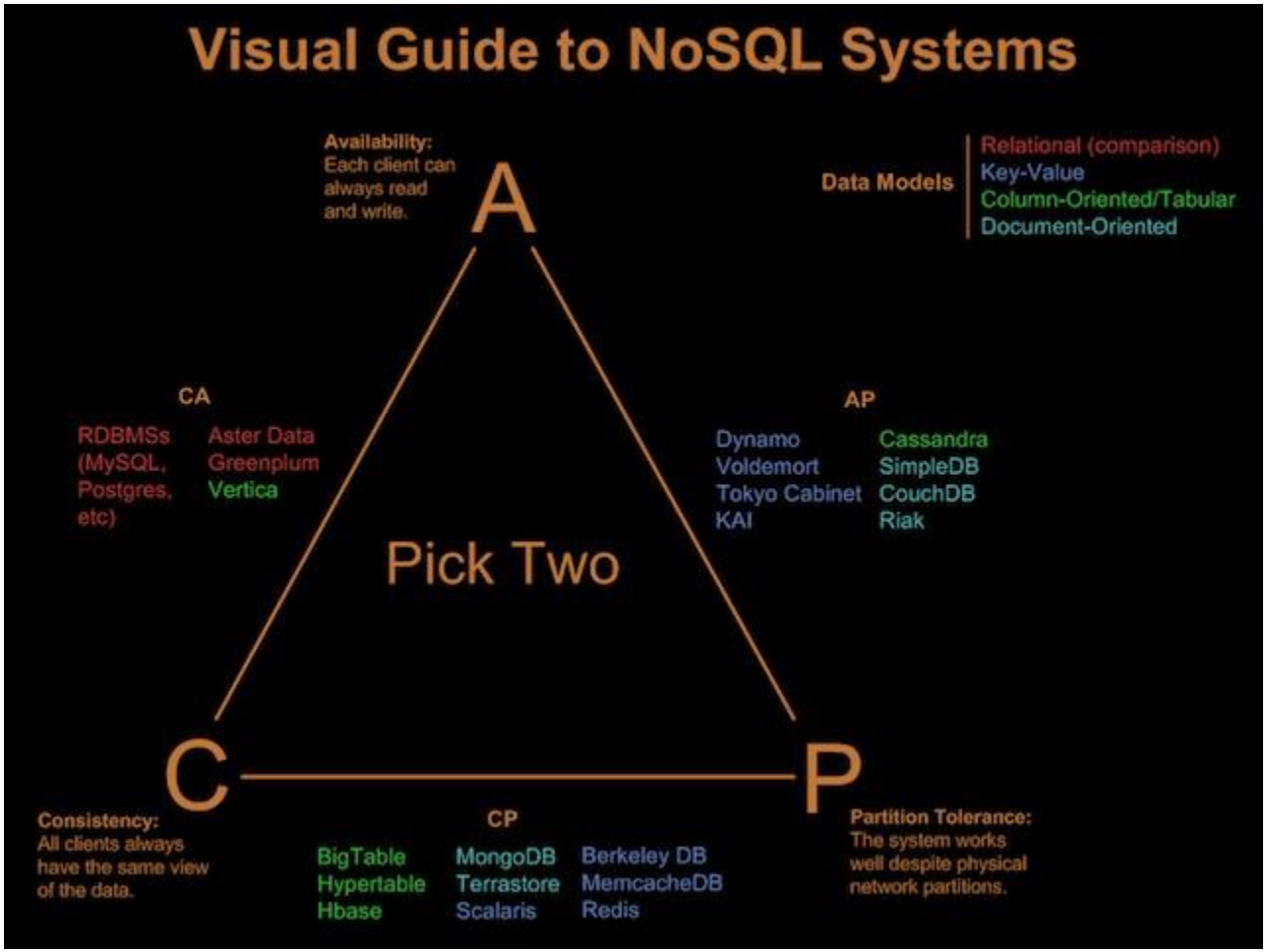
BBDD orientadas a Clave-Valor

- Su precursor fue Google BigTable.
- Buenas en:
  - Gestión de tamaño.
  - Cargas de escrituras masivas orientas al stream.
  - Alta disponibilidad.
  - MapReduce
- Ejemplos: Redis, Memcached, Riak, Cassandra (híbrido Key-value/column oriented), Amazon DinamoDB (Key-value/document oriented)

BBDD orientadas a Grafos

- Buenas en:
  - Modelar directamente un dominio en forma de grafo, una manera común de representar y entender datasets.
  - Ofrecer excelente rendimiento cuando los datos están interconectados y no tabulares.
  - Realizar operaciones transaccionales que exploten las relaciones entre entidades.
- Ejemplos: Neo4J, OrientDB, HyperGraphDB.

Teorema de CAP. Clasificación



NoSQL y Big Data

Dos tipos de procesamiento en Big Data:

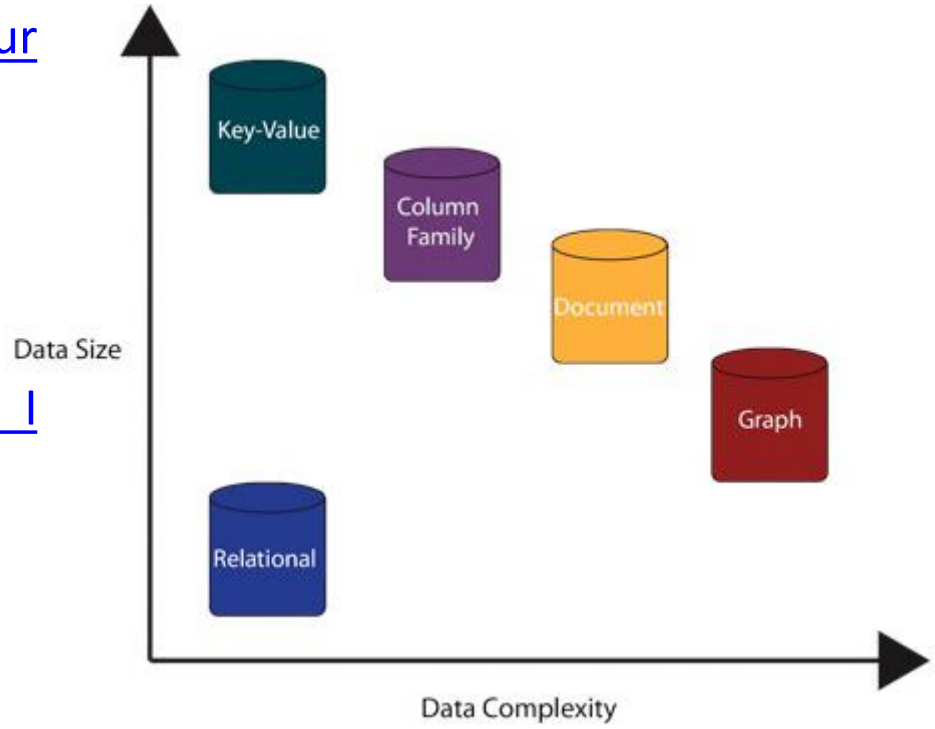
- **Big Data Operacional.** Las BBDD NoSQL pueden cubrir un amplio espectro de aplicaciones y están optimizadas para determinadas aplicaciones específicas. Las tecnologías NoSQL son más rápidas y escalan más rapidamente y a un menor coste que los RDBMS. Las soluciones NoSQL se integran perfectamente en los sistemas de Cloud Computing, haciéndolas mas fácil de implementar, mantener y mas económicas. Además, proporcionan algo de inteligencia en tiempo real sobre los datos almacenados.
- **Big Data Analítico.** A medida que se incrementa el volumen de datos operacionales almacenado, se plantean una serie de tareas analíticas retrospectivas que provean de valor añadido a los negocios. Estas analíticas precisan del uso de sistemas de BBDD que soporten procesamiento masivamente paralelo (MPP) y de MapReduce. Algunas NoSQL proporcionan soporte nativo para MapReduce con lo que se puede el realizar análisis sobre los datos operacionales almacenados "in situ".

NoSQL. Cloud Computing

- BigTable es un sistema de gestión de base de datos distribuido creado por Google, de alta eficiencia y propietario.
  - Google Cloud Datastore (App Engine NoSQL Data Storage) motor de BD NoSQL factorizado de Google App Engine.
    - BD de columnas que soporta transacciones ACID, tiene alta disponibilidad a través de centros de replicación y ofrece consultas SQL-like.
- Amazon DynamoDB es un servicio de bases de datos NoSQL rápido y totalmente gestionado que permite almacenar y recuperar de manera fácil y económica cualquier cantidad de datos y atender cualquier nivel de tráfico. Ofrece a los clientes una opción de consistencia eventual o fuerte, dependiendo de los precios.
- Comparativa:
  - http://www.theregister.co.uk/2013/05/16/google_datastore/

¿Qué tipo de BD elijo?

- Algunas orientaciones y comparaciones pueden encontrarse en:
  - 35+ Use Cases For Choosing Your Next NoSQL Database
  - Five Reasons to Use NoSQL
  - Which freaking database should I use?
  - Comparativa de BBDD NoSQL



DB-Engines Ranking

- Complete ranking
- Relational DBMS
- Key-value stores
- Document stores
- Graph DBMS
- Time Series DBMS
- RDF stores
- Object oriented DBMS
- Search engines
- Multivalue DBMS
- Wide column stores
- Native XML DBMS
- Content stores
- Event Stores
- Navigational DBMS

Special reports

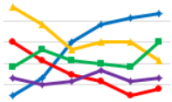
- Ranking by database model
- Open source vs. commercial

Featured Products

DB-Engines Ranking

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the method of calculating the scores.



trend chart

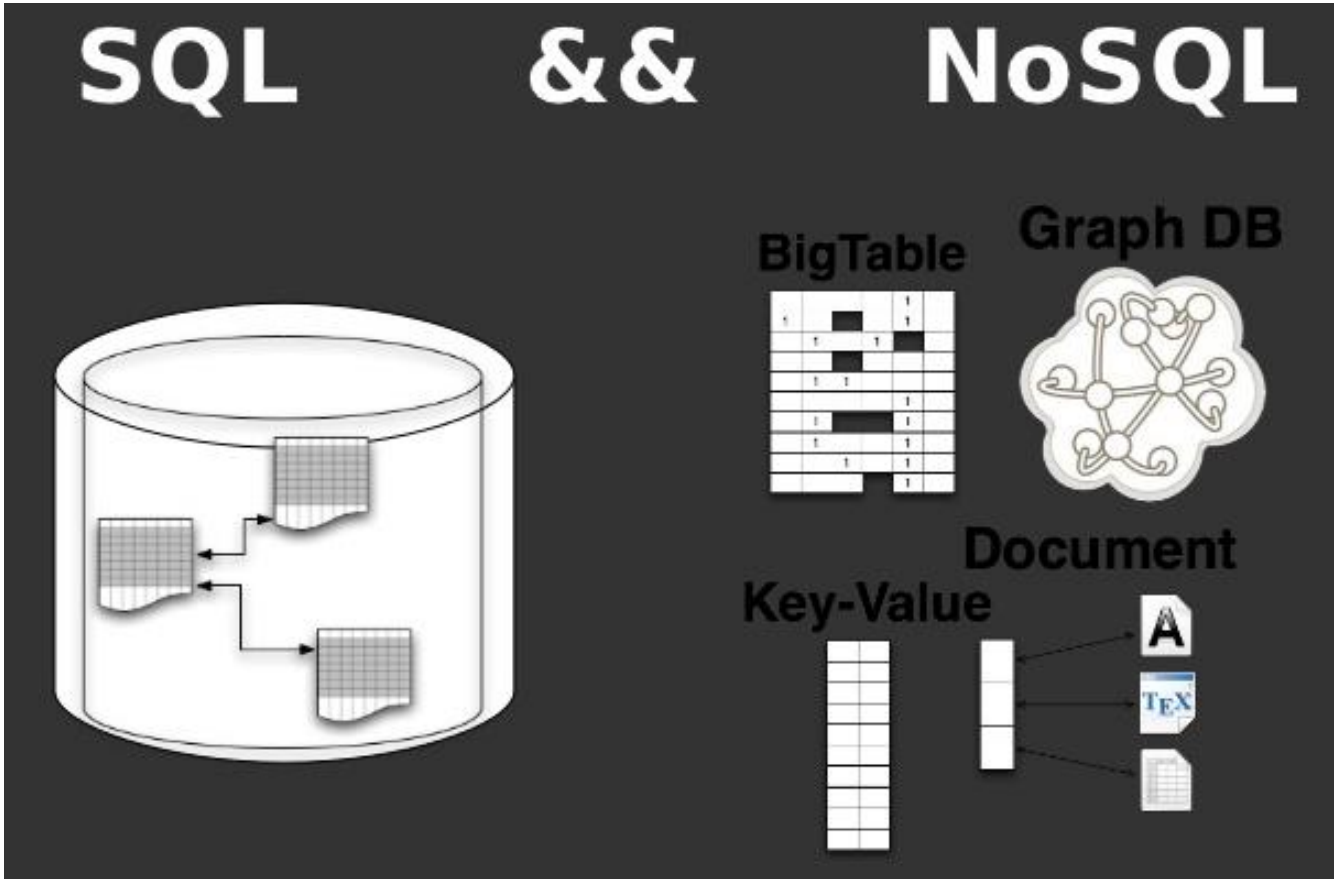
322 systems in ranking, March 2017

Rank Mar 2017	Rank Feb 2017	Rank Mar 2016	DBMS	Database Model	Score Mar 2017	Score Feb 2017	Score Mar 2016
1.	1.	1.	Oracle	Relational DBMS	1399.50	-4.33	-72.51
2.	2.	2.	MySQL	Relational DBMS	1376.07	-4.23	+28.36
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1207.49	+4.04	+71.00
4.	4.	↑ 5.	PostgreSQL	Relational DBMS	357.64	+3.96	+58.01
5.	5.	↓ 4.	MongoDB	Document store	326.93	-8.57	+21.60
6.	6.	6.	DB2	Relational DBMS	184.91	-2.99	-3.02
7.	↑ 8.	7.	Microsoft Access	Relational DBMS	132.94	-0.45	-2.09
8.	↓ 7.	8.	Cassandra	Wide column store	129.19	-5.19	-1.14
9.	9.	↑ 10.	SQLite	Relational DBMS	116.19	+0.88	+10.42
10.	10.	↓ 9.	Redis	Key-value store	113.01	-1.03	+6.79
11.	11.	11.	Elasticsearch	Search engine	106.23	-2.08	+26.06
12.	12.	↑ 13.	Teradata	Relational DBMS	73.53	-2.06	-0.53
13.	13.	↓ 12.	SAP Adaptive Server	Relational DBMS	70.13	-1.61	-6.52
14.	14.	14.	Solr	Search engine	63.99	-3.70	-5.38
15.	15.	15.	HBase	Wide column store	58.98	-0.26	+6.57
16.	↑ 17.	↑ 17.	FileMaker	Relational DBMS	54.57	-0.62	+6.64
17.	↓ 16.	↑ 18.	Splunk	Search engine	54.09	-1.94	+10.36
18.	18.	↑ 19.	SAP HANA	Relational DBMS	50.06	-2.39	+10.07
19.	↑ 20.	↑ 22.	MariaDB	Relational DBMS	46.88	+1.53	+17.00
20.	↓ 19.	↓ 16.	Hive	Relational DBMS	44.62	-3.33	-5.89
21.	21.	↓ 20.	Neo4j	Graph DBMS	34.32	-1.95	+1.96
22.	22.	↑ 23.	Amazon DynamoDB	Document store	31.13	-1.06	+8.90
23.	23.	↑ 24.	Couchbase	Document store	30.04	-1.14	+4.24
24.	24.	↓ 23.	Memcached	Key-value store	30.04	-0.49	+0.80
25.	25.	↓ 21.	Informix	Relational DBMS	26.89	-0.35	-4.97
26.	26.	↓ 25.	CouchDB	Document store	22.94	-0.49	-0.45

Persistencia Políglota

- **Toda empresa va a acabar teniendo una variedad de tecnologías de almacenamiento para diferentes tipos de datos**
  - El código de nuestros sistemas va a tener acceso a diferentes repositorios de datos.
  - Muchos datos seguirán guardándose en almacenes relacionales pero debemos empezar a **preguntarnos cómo queremos utilizar los datos y sólo entonces decidir qué tecnología es mejor:**
    - Usar NoSQL para una característica particular de una aplicación:
      - Rápido procesado batch.
      - Logging distribuido.
      - Para grandes tablas.
    - Usar una RDBMS para reporting y necesidad de ACID.
  - Se complica la lógica de la aplicación y el despliegue.
  - Responsabilidades administrativas adicionales.
- Hay que **¡elegir la herramienta adecuada para cada trabajo!**

Persistencia Políglota



MongoDB

- **MongoDB** (de la palabra en ingles “hu**mongo**us” que significa enorme) es un sistema de base de datos NoSQL ”open source” orientado a documentos escrito en C++.
- Existen versiones para instalar en diversas plataformas: Windows, Linux, OS X.
- También está disponible como servicio empresarial en la nube (MMS) a través de:

  https://mms.mongodb.com, y que se puede integrar con Amazon Web Services (AWS)

MongoDB. Características Principales

- **BD** basada en **documentos**:
  - Los documentos (objetos) se acoplan perfectamente en los tipos de datos de los lenguajes de programación.
  - Los documentos incrustados y las colecciones reducen la necesidad de reuniones.
  - Dispone de un esquema dinámico que facilita el polimorfismo.
- **Alto rendimiento**:
  - El carácter compuesto de los documentos, que pueden incluir otros documentos y colecciones relacionadas, hace más rápidas las escrituras y las lecturas.
  - Los índices pueden incluir claves definidas sobre los documentos incrustados y sobre las colecciones.
  - Opción de escritura de flujos de datos sin protocolo de reconocimiento.
- **Alta disponibilidad**:
  - Servidores replicados con restablecimiento automático maestro.
- **Fácil escalabilidad**:
  - El "sharding" automático distribuye una colección de datos entre diferentes máquinas.
  - Las lecturas eventualmente-consistentes se pueden distribuir a través de servidores replicados.

MongoDB. Características Principales

- **Consultas Ad hoc**
  - MongoDB soporta la búsqueda por campos, consultas de rangos y expresiones regulares.
  - Las consultas pueden devolver un campo específico del documento pero también pueden ser una función JavaScript definida por el usuario.
- **Indexación**
  - Cualquier campo en un documento de MongoDB puede ser indexado, al igual que es posible hacer índices secundarios.
    - El concepto de índices en MongoDB es similar a los encontrados en base de datos relacionales.
- **Replicación**
  - MongoDB soporta el tipo de **replicación maestro-esclavo.**
    - El maestro puede ejecutar comandos de lectura y escritura.
    - El esclavo puede copiar los datos del maestro y sólo se puede usar para lectura o para copia de seguridad, pero no se pueden realizar escrituras. El esclavo puede elegir un nuevo maestro en caso del que se caiga el servicio con el maestro actual.

MongoDB. Características Principales

- **Balanceo de carga**
  - MongoDB se puede escalar de forma horizontal usando el concepto de "shard".
  - El desarrollador elije una llave shard, la cual determina cómo serán distribuidos los datos de una colección. Los datos son divididos en rangos (basado en la llave shard) y distribuidos a través de múltiples shard.
  - MongoDB tiene la capacidad de ejecutarse en múltiples servidores, balanceando la carga y/o duplicando los datos para poder mantener el sistema funcionando en caso que exista un fallo de hardware.
- **Almacenamiento de archivos**
  - MongoDB puede ser utilizado con un sistema de archivos (GridFS), con balanceo de carga y la replicación de datos utilizando múltiples servidores para el almacenamiento de archivos.
  - Esta función (que es llamada GridFS) está incluida en los drivers de MongoDB y disponible para los lenguajes de programación que soporta MongoDB.
- **Agregación**
  - La función MapReduce y el operador `aggregate()` pueden ser utilizados para el procesamiento por lotes de datos y operaciones de agregación.
  - Estos mecanismos permiten que los usuarios puedan obtener el tipo de resultado que se obtiene cuando se utiliza el comando SQL "group-by".
- **Ejecución de JavaScript del lado del servidor**
  - MongoDB tiene la capacidad de realizar consultas utilizando JavaScript, haciendo que estas sean enviadas directamente a la base de datos para ser ejecutadas: `db.system.js.save`
    - http://docs.mongodb.org/manual/tutorial/store-javascript-function-on-server/

MongoDB. Áreas de aplicación

- Almacenamiento y registro de eventos.
- Para sistemas de manejo de documentos y contenido.
- Comercio Electrónico.
- Juegos.
- Problemas sobre grandes volúmenes de datos.
- Aplicaciones móviles.
- Almacén de datos operacional de una página Web.
- Manejo de contenido.
- Almacenamiento de comentarios:
  - Votaciones.
  - Registro de usuarios.
  - Perfiles de usuarios.
  - Sesiones de datos.
- Proyectos que utilizan metodologías de desarrollo iterativo o ágiles.
- Manejo de estadísticas en tiempo real.

MongoDB. Conceptos Clave

1. MongoDB tiene el concepto de **"base de datos"** con el que estamos familiarizados (schema en el mundo relacional).
   - Dentro de un servidor MongoDB podemos tener 0 o más BBDD, cada una actuando como un contenedor de todo lo demás.
2. Una base de datos puede tener una o más **"colecciones"**, equivalente en el mundo relacional a una "tabla".
3. Las colecciones están hechas de 0 o más **"documentos"**, donde un documento puede considerarse equivalente a una fila de una tabla de un RDBMS.
4. Un documento está compuesto de uno o varios "**campos**" que son equivalentes a las columnas de una fila. La estructura está compuesta por "key-value pairs" parecido a las matrices asociativas en un lenguaje de programación.
5. Los "**índices**" en MongoDB funcionan como los de los RDBMS.
6. Los "**cursores**" son utilizados para acceder progresivamente a los datos recuperados con una consulta
   - Pueden usarse para contar o moverse hacia delante entre los datos.

MongoDB. Elementos de los documentos

- Además de los datos de un documento, MongoDB siempre introduce un campo adicional `_id`
  - Todo documento tiene que tener un campo `_id` único
  - Este campo `_id` puede contener un valor de cualquier tipo BSON, excepto un array
    - Más info en http://docs.mongodb.org/manual/core/document/
- Podemos generar el identificador nosotros:
  - `x = "55674321R"`
  - `y = ObjectId("507f191e810c19729de860ea")`
- o dejarle a MongoDB que lo haga, eso lo hace si no asignamos un valor a _id.
  - El tipo de ese campo es `ObjectId`:

    http://docs.mongodb.org/manual/reference/object-id/
  - Es un tipo de datos de **12 bytes, donde 4 bytes representan un timestamp, 3 un identificador de máquina, 2 el identificador del proceso y 3 restantes un contador**
    - Tiene el atributo `str` y los métodos `getTimeStamp()` y `toString()`
  - Es preferible que MongoDB lo genere por nosotros
- El campo `_id` es indexado lo que explica que se guardan sus detalles en la colección del sistema `system.indexes`

MongoDB. Ejemplo de documento

```
{
    "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
    "Last Name": "PELLERIN",
    "First Name": "Franck",
    "Age": 29,
    "Address": {
        "Street": "1 chemin des Loges",
        "City": "VERSAILLES"
    }
}
```

Documentación e instalación

- La documentación completa de MongoDB puede encontrarse en:
  - http://docs.mongodb.org/manual/
- Instrucciones para instalar MongoDB en Windows:
  - Descargar la última versión para Windows.
  - Seleccionar Windows 64-bit 2008 R2+ para Windows 7 o superior. Si se va a instalar en Windows 7 debe descargarse instalar este parche.
  - Ejecutar el archivo de instalación .msi, e indicarle el directorio donde queréis instalarlo. (`c:\mongodb` en nuestros ejemplos).
  - Crear directorio de datos: `mkdir c:\mongodb\data\db`
  - Se puede configurar como servicio en Windows.
- Iniciar el servidor: `mongod.exe --dbpath c:\mongodb\data\db`
  - Iniciar con archivo configuración:
  - `mongod --config c:\mongodb\mongodb.conf`
- Conectarse mediante el cliente `mongo` abrir un cmd y ejecutar: `c:\mongodb\bin\mongo.exe`
- Comenzar a usar MongoDB.
- Instrucciones para instalar MongoDB en Linux

Utilidades de MongoDB

- Utilidades disponibles para el manejo y la administración del sistema de base de datos:
  - `mongo`: es un Shell interactivo que permite a los desarrolladores ver, insertar, eliminar y actualizar datos en su base de datos. Entre otras funciones, también permite la replicación de información, configurar los Shards, apagar los servidores y ejecutar JavaScript.
  - `mongostat`: es una herramienta de línea de comandos que muestra una lista de estadísticas de una instancia de MongoDB en ejecución.
  - `mongotop`: es una herramienta de línea de comandos que provee un método para mostrar la cantidad de tiempo empleado por lectura o escritura de datos en una instancia.
  - `mongosniff`: es una herramienta de línea de comandos que provee un sniffing en la base de datos haciendo un sniffing en el tráfico de la red que va desde y hacia MongoDB.
  - `mongoimport/mongoexport`: es una herramienta de línea de comandos que facilita la importación/exportación de contenido desde JSON, CSV o TSV.
  - `mongodump/mongorestore`: es una herramienta de línea de comandos para la creación de una exportación binaria del contenido de la base de datos.

MongoDB. Herramientas gráficas

- Recopilación de herramientas gráficas para la administración y uso de MongoDB.
- MongoDB tiene una interfaz administrativa accesible yendo a: http://localhost:28017/. Siempre que se haya iniciado con la opción: `mongod --rest` o se haya añadido al fichero de configuración la línea `rest=true`.
- Robo3T (antes robomongo, open source). Incluye una shell completamente compatible con la shell mongo.
- Studio 3T, es un IDE más completo que tiene una versión gratuita (la Core) para uso no comercial o educacional. Características en las diferentes versiones: https://studio3t.com/compare

MongoDB. Mongo Shell

- El shell Mongo se comporta como un buen shell de UNIX:
  - Ofrece autocompletado (usando el tabulador)
  - Te permite moverte por la historia de comandos con el cursor para arriba o abajo o moverte al primer o último comando con las combinaciones de teclas (CTRL-a y CTRL-e).
  - Ofrece un objeto implícito llamado `db` que representa a la base de datos
    - Usa por defecto la BD `test` salvo que establezcamos otra mediante el comando `use <database-name>`
  - Las colecciones se crean automáticamente cuando insertamos el primer documento en ellas
  - Es un shell en JavaScript que no distingue entre enteros y números en coma flotante, todo número se representa en JavaScript como un número de coma flotante de 64 bits.
- Conectarse a nuestro servidor mongod:
  - <database> -> db_mdat_<ID> ; <user> -> mdat_<ID>
  - Desde cliente mongo: mongo hadoop.ugr.es:27017/<database> -u <user> -p <clave>
    [--authenticationDatabase admin] (no vá, pues sólo se puede acceder a MongoDB en hadoop desde local)
  - Desde hadoop.ugr.es: mongo localhost:27017/<database> -u <user> -p <clave>
    [--authenticationDatabase admin]
  - Desde cliente Robo3T (RoboMongo): Crear una conexión con los siguientes valores:

En Address: localhost, puerto: 27017; poner: en Database y en Default Database <database>, en User Name <user>, poner vuestra clave de mongoDB en Password: y seleccionar Auth Mechanism a SCHRAM-SHA-1.

Para acceder a través de un túnel SSH, debéis rellenar la pestaña SSH de la siguiente manera: activar el "check box" Use SSH Tunnel, poner SSH Address: hadoop.ugr.es, SSH User Name: <user>, seleccionar Password como SSH Auth Method y en User Password ponéis vuestra clave de usuario en hadoop.ugr.es (no la de mongodb). Pichad Test a ver si va bien la conexión, si es así, guardarla con el nombre que consideréis apropiado.

MongoDB. Mongo Shell

- Comandos útiles:
  - `help` – muestra ayuda.
  - `db.help()` – muestra ayuda de los métodos de la BD .
  - `db.<collection>.help()` – detalla qué métodos se pueden aplicar a una colección.
  - `show dbs` – imprime una lista de las bases de datos del servidor.
  - `use <database-name>` – cambia la base de datos a `<db>`, haciendo que `db` apunte la BD seleccionada.
  - `show collections` – imprime todas las colecciones de la base de datos actual.
  - `show users` – imprime los usuarios de la BD.
- Referencia comandos mongo shell.

MongoDB. Correspondencia SQL vs MongoDB

MYSQL EXECUTABLE	ORACLE EXECUTABLE	MONGODB EXECUTABLE
mysqld	oracle	mongod
mysql	sqlplus	mongo

SQL TERM	MONGODB TERM
database	database
table	collection
index	index
row	document
column	field
joining	embedding & linking

MongoDB. Correspondencia SQL vs MongoDB

SQL	MONGODB
`CREATE TABLE users (name VARCHAR(128), age NUMBER)`	`db.createCollection("users")`
`INSERT INTO users VALUES ('Bob', 32)`	`db.users.insert({name: "Bob", age: 32})`
`SELECT * FROM users`	`db.users.find()`
`SELECT name, age FROM users`	`db.users.find({}, {name: 1, age: 1, _id:0})`
`SELECT name, age FROM users WHERE age = 33`	`db.users.find({age: 33}, {name: 1, age: 1, _id:0})`
`SELECT * FROM users WHERE age > 33`	`db.users.find({age: {$gt: 33}})`
`SELECT * FROM users WHERE age <= 33`	`db.users.find({age: {$lte: 33}})`

MongoDB. Correspondencia SQL vs MongoDB

SQL	MONGODB
SELECT * FROM users WHERE age > 33 AND age < 40	db.users.find({age: {$gt: 33, $lt: 40}})
SELECT * FROM users WHERE age = 32 AND name = 'Bob'	db.users.find({age: 32, name: "Bob"})
SELECT * FROM users WHERE age = 33 OR name = 'Bob'	db.users.find({$or:[{age:33}, {name: "Bob"}]})
SELECT * FROM users WHERE age = 33 ORDER BY name ASC	db.users.find({age: 33}).sort({name: 1})
SELECT * FROM users ORDER BY name DESC	db.users.find().sort({name: -1})
SELECT * FROM users WHERE name LIKE '%Joe%'	db.users.find({name: /Joe/})
SELECT * FROM users WHERE name LIKE 'Joe%'	db.users.find({name: /^Joe/})
SELECT * FROM users LIMIT 10 SKIP 20	db.users.find().skip(20).limit(10)
SELECT * FROM users LIMIT 1	db.users.findOne()

MongoDB. Correspondencia SQL vs MongoDB

SQL	MONGODB
SELECT DISTINCT name FROM users	db.users.distinct("name")
SELECT COUNT(*) FROM users	db.users.count()
SELECT COUNT(*) FROM users WHERE AGE > 30	db.users.find({age: {$gt: 30}}).count()
SELECT COUNT(AGE) FROM users	db.users.find({age: {$exists: true}}). count()
UPDATE users SET age = 33 WHERE name = 'Bob'	db.users.update({name: "Bob"}, {$set: {age: 33}}, {multi: true})
UPDATE users SET age = age + 2 WHERE name = 'Bob'	db.users.update({name: "Bob"}, {$inc: {age: 2}}, {multi: true})
DELETE FROM users WHERE name = 'Bob'	db.users.remove({name: "Bob"})
CREATE INDEX ON users (name ASC)	db.users.ensureIndex({name: 1})
CREATE INDEX ON users (name ASC, age DESC)	db.users.ensureIndex({name: 1, age: -1})
EXPLAIN SELECT * FROM users WHERE age = 32	db.users.find({age: 32}).explain()

MongoDB. Operaciones CRUD. Insert()

- Operaciones CRUD: Create (insertar) , Read (consultar), Update (actualizar) and Delete (eliminar) “records”.
- CRUD vs IFUR:

Create	=>	**I**nsert
Read	=>	**F**ind
Update	=>	**U**pdate
Delete	=>	**R**emove

- BD ejemplo de Restaurantes. Importar:
  - Entrar en hadoop.ugr.es
  - mongoimport -u <user> -p <clave> --db <vuestraBD> --collection restaurants --type json --drop --file /var/tmp/restaurantes1.json
- Para INSERTAR documentos en una colección se usa el método `insert()`:

MongoDB. Operaciones CRUD. Insert()

```
db.restaurants.insert(
   {
      "address" : {
         "street" : "2 Avenue",
         "zipcode" : "10075",
         "building" : "1480",
         "coord" : [ -73.9557413, 40.7720266 ]
      },
      "borough" : "Manhattan",
      "cuisine" : "Italian",
      "grades" : [
         {
            "date" : ISODate("2014-10-01T00:00:00Z"),
            "grade" : "A",
            "score" : 11
         },
         {
            "date" : ISODate("2014-01-16T00:00:00Z"),
            "grade" : "B",
            "score" : 17
         }
      ],
      "name" : "Vella",
      "restaurant_id" : "41704620"
   }
```

MongoDB. Operaciones CRUD. Insert()

- Inserta en la colección `restaurants`, de la base de datos (`db`) un documento con los valores que se indican.
- El método devuelve un objeto WriteResult con el estado de la operación:
  - `WriteResult({ "nInserted" : 1 })`
- Si la colección `restaurants` no existiese previamente, se crearía automáticamente cuando insertamos un documento en ella.
- `db` hace referencia a la base de datos sobre la que estamos actuando (si no se cambia mediante `use`, por defecto se usa la base de datos `test`).
- Si se inserta sin el campo `_id`, mongod inserta un valor para ese campo que es único para cada documento de la colección

MongoDB. Operaciones CRUD. Find()

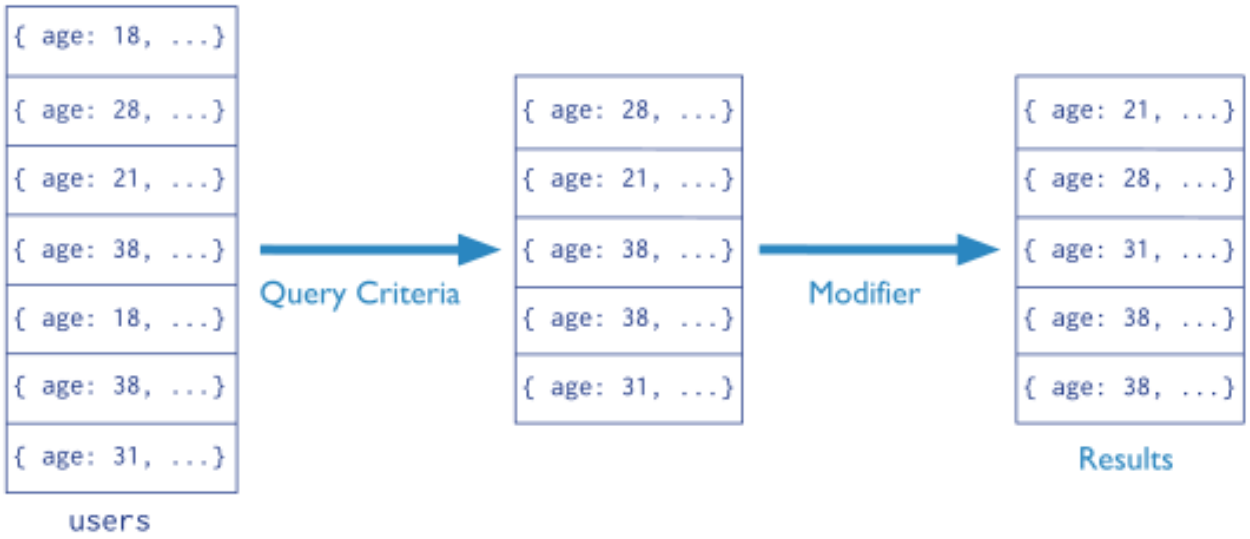
- CONSULTA: mediante el método `find()`:

```
db.users.find(                          <- collection
    { age: { $gt: 18 } },               <- query criteria
    { name: 1, address: 1 }             <- projection
).limit(5)                              <- cursor modifier
```

- Ejemplo de consulta y resultado (`.sort({age: -1}` ordenaría descendentemente):

Collection | Query Criteria | Modifier

```
db.users.find( { age: { $gt: 18 } } ).sort( {age: 1 } )
```



MongoDB. Operaciones CRUD. Find()

- Operadores para los criterios de consulta:

Operador	Descripción	Ejemplo de consulta
$eq	Igual que	db.ships.find({class:{$eq:'P'}})
$gt	Mayor que	db.ships.find({class:{$gt:'P'}})
$gte	Mayor o igual que	db.ships.find({class:{$gte:'P'}})
$lt	Menor que	db.ships.find({class:{$lt:'P'}})
$lte	Menor o igual que	db.ships.find({class:{$lte:'P'}})
$ne	Distinto a	db.ships.find({class:{$ne:'P'}})
$in	Igual a alguno de los elementos de un array	db.ships.find({class:{$in:['P', 'Q']}})
$exists	Si un atributo existe o no	db.ships.find({type:{$exists:true}})
$regex	Expresiones regulares tipo Perl	db.ships.find({name : {$regex:'^USS\\sE'}})
$type	Busca campos de un determinado tipo en un campo de un documento	db.ships.find({name : {$type:2}})

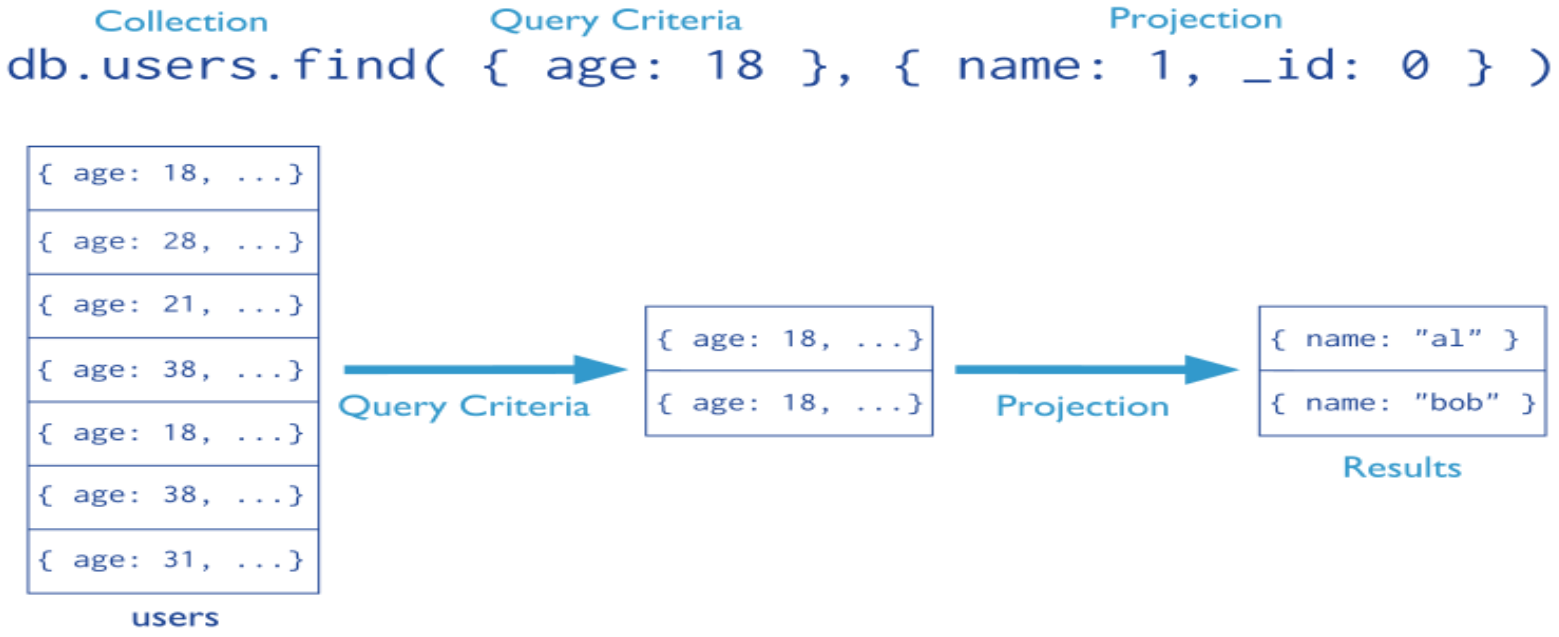
MongoDB. Operaciones CRUD. Find()

- El operador `$exists` se utiliza para encontrar qué documentos contienen o no el campo indicado
  - `db.restaurants.find({address: {$exists: true}})`
- Si queremos utilizar el operador booleano **OR** tenemos que hacer uso del operador `$or` y asociarle un array de tuplas clave/valor sobre los que realizar el OR:
  - `db.restaurants.find(`
    `{ $or:[{ "cuisine": "Italian" }, { "address.zipcode": "10462" }]})`
- Dado que los arrays en MongoDB son objetos de primera categoría se puede comprobar la inclusión de un elemento dentro de un array al igual que si compararamos con un único valor:
  - `db.restaurants.find( {grades :`
    `{"date" : ISODate("2014-10-01T00:00:00Z"), "grade" : "A", "score" : 11}})`

  devolverá los restaurantes que incluyan el grado descrito por (`{date:,grade,:score:}`)
- Un valor de tipo `ObjectId` asociado al campo `_id` puede seleccionarse como:
  - `db.restaurants.find({_id: ObjectId("570217367587ecbe68ef080a")})`

MongoDB. Operaciones CRUD. Find()

- Proyección: se determina mediante el segundo argumento de find(). El valor 1 indica que se muestre el campo y el valor 0 que se excluya el campo de los resultados mostrados.



- Ejemplo: `db.restaurants.find( { "cuisine": "Italian" , "address.zipcode": "10075" }, {name:1,address:1})`

MongoDB. Operaciones CRUD. Find()

- Cursores: el resultado de una consulta (`find()`) se devuelve mediante un objeto cursor.
- Éste puede ser asignado a una variable. Si no se asigna a una variable, mongo shell muestra los 20 primeros documentos. Se puede iterar usando este procedimiento:

```
var myCursor = db.restaurants.find({ "cuisine": "Italian"})
while (myCursor.hasNext()) {
        printjson(myCursor.next());
}
```

- Otro modo de iterarlo: `myCursor.forEach(printjson);`
- Paginación. Se soporta con los métodos de cursor `skip` y `limit`
  - `db.restaurants.find( { "cuisine": "Italian"}).sort({name:1}).limit(2).skip(1)`
- Conteo. También se pueden contabilizar resultados:
  - `db.restaurants.count({ "grades.score": { $lt: 5 } } )`
  - `db.restaurants.find({ "grades.score": { $lt: 5 } } ).count()`

MongoDB. Operaciones CRUD. Update()

- UPDATE: por defecto actualiza sólo un documento, si se pone `multi: true` se actualizan todos los documentos que cumplan el criterio.

```
db.users.update(                          <-- collection
    { age: { $gt: 18 } },                 <-- update criteria
    { $set: { status: "A" } },            <-- update action
    { multi: true }                       <-- update option
)
```

-
```
db.restaurants.update(
{ "restaurant_id" : "41156888" },
{ $set: { "address.street": "East 31st Street" } })
```

- El operador `$set` hace que se mantengan los valores para el resto de los campos en los documentos existentes y sólo se cambie el del campo al que se aplica el operador.

MongoDB. Operaciones CRUD. Update()

- El operador `$unset` elimina un campo del documento de salida: `db.books.update( { _id: 1 }, { $unset: { tags: 1 } } )`
- El operador `$inc` se usa para incrementar el campo por una cantidad positiva o negativa:

```
db.unicorns.update({name: 'Pilot'}, {$inc: {vampires: -2}})
```

- Podemos añadir un nuevo elemento en un "array" usamos el modificador `$push`, p.e. añadir una nueva valoración para un restaurante:

```
db.restaurants.update({ "restaurant_id" : "41156888" },{$push: {grades:
  {"date" : ISODate("2016-01-02T00:00:00.000Z"), "grade" : "A", "score" : 14}}})
```

- Si queremos que se inserte un nuevo documento cuando intentamos actualizar uno no existente, añadimos un tercer parámetro `{upsert: true}`

```
db.hits.update({page: 'unicorns'}, {$inc: {hits: 1}},
{upsert: true});
```

- Si queremos que `update()` actualice todos los documentos que cumplen una expresión, el 4º parámetro tiene que ponerse a `true`

```
db.restaurants.update(
{ "address.zipcode": "10016", cuisine: "Other" },{$set: { cuisine: "Category To Be Determined" },
  $currentDate: { "lastModified": true }}, { multi: true})
```

MongoDB. Operaciones CRUD:Remove()

- REMOVE: por defecto elimina todos los documentos que cumplan el criterio, si se pone la opción: `justOne: true`, se elimina sólo uno.

- La modificación de un solo documento siempre es atómica, incluso si la operación de escritura modifica varios documentos incrustados dentro de ese documento.
- Ninguna otra operación es atómica. Se puede intentar aislar una operación de escritura que afecta múltiples documentos utilizando el operador `$isolated`.

```
db.foo.update(
  { status : "A" , $isolated : 1 },
  { $inc : { count : 1 } },
  { multi: true }
)
```

MongoDB. Operaciones CRUD

- MongoDB vs SQL

```
db.users.insert (          <-- collection
   {
     name: "sue",          <-- field: value
     age: 26,              <-- field: value
     status: "A"           <-- field: value
   }                       } document
)
```

```
INSERT INTO users                          <-- table
            ( name, age, status )          <-- columns
VALUES      ( "sue", 26, "A" )             <-- values/row
```

```
db.users.update(                           <-- collection
   { age: { $gt: 18 } },                   <-- update criteria
   { $set: { status: "A" } },              <-- update action
   { multi: true }                         <-- update option
)
```

```
UPDATE users          <-- table
SET    status = 'A'   <-- update action
WHERE  age > 18       <-- update criteria
```

```
db.users.remove(           <-- collection
   { status: "D" }         <-- remove criteria
)
```

```
DELETE FROM users     <-- table
WHERE  status = 'D'   <-- delete criteria
```

MongoDB. Índices

- Los índices mejoran el rendimiento de las consultas y operaciones de ordenación en MongoDB
  - Operan de modo similar a los de los RDBMS
    - MongoDB los guarda como un B-Tree que permite la recuperación de listas de claves ordenadas
- Se crean con la sentencia `ensureIndex()`, identificando el sentido de ordenación por campo ascendente (`1`) o descendente (`-1`):

```
db.restaurants.ensureIndex({name: 1});
```

- Para conocer los índices de la colección, escribir:

```
db.restaurants.getIndexes();
```

- Y se eliminan con `dropIndex()`:

```
db.restaurants.dropIndex({name: 1});
```

- Para asegurarnos que el índice es único, usamos el atributo `unique`:

```
db.restaurants.ensureIndex({name: 1}, {unique: true});
```

- Los índices también pueden ser compuestos:

```
db.restaurants.ensureIndex({cusine: 1, name: -1});
```

MongoDB. Índices

- Para saber si se está usando un índice o no, usamos `explain()`:

```
db.restaurants.find({"cuisine":"Italian"}).sort({name:1}).explain()
```

Si la salida incluye algo como esto:

```
"winningPlan" : {
        ...
        "inputStage" : {
            "stage" : "IXSCAN",
            "keyPattern" : {
                "name" : 1.0
            },
            "indexName" : "name_1",
        ...
```

es que se está usando un índice para resolver la consulta (en este caso, el índice que habíamos creado sobre el campo name anteriormente), si no incluye una `"stage": "IDXSCAN"` indica que la ejecución de la consulta no está usando un índice, bien porque no haya ninguno definido, bien porque la condición de consulta no haga óptimo su uso.

MongoDB. Índices

- Si usamos `explain("executionStats")`:

```
db.restaurants.find({"cuisine":"Italian"}).sort({name:1}).explain("executionStats")
```

MongoDB ejecuta el optimizador de consultas para seleccionar el mejor plan de ejecución, ejecuta ese plan de ejecución y devuelve las estadísticas de ejecución de ese plan, por ejemplo:

```
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 1069,
        "executionTimeMillis" : 24,
        "totalKeysExamined" : 25359,
        "totalDocsExamined" : 25359,
        "executionStages" : {
            "stage" : "FETCH",
            "filter" : {
                "cuisine" : {
                    "$eq" : "Italian"
                }
            },
            "nReturned" : 1069,
            "executionTimeMillisEstimate" : 20,
            "works" : 25360,
            "advanced" : 1069,
            "needTime" : 24290,
            "needYield" : 0,
            "saveState" : 198,
            "restoreState" : 198,
            "isEOF" : 1,
            "invalidates" : 0,
            "docsExamined" : 25359,
            "alreadyHasObj" : 0,
            "inputStage" : {
                "stage" : "IXSCAN",
                "nReturned" : 25359,
                "executionTimeMillisEstimate" : 20,
…
```

MongoDB. Índices

- La colección `db.system.indexes` contiene detalles de todos los índices de una BD de MongoDB:

```
db.system.indexes.find()
```

- Es posible crear índices sobre atributos que se guardan como "arrays" de valores (Multikey Index), así todos los valores del "array" son indizados
  - En un índice compuesto, sólo se permite incluir un campo con un "Multikey Index"
- Resumen de operaciones de indexación:

Index	Comando
Creación de un índice	`db.universe.ensureIndex({galaxy : 1})`
Eliminación de un índice	`db.universe.dropIndex({galaxy : 1})`
Creación de un índice compuesto	`db.universe.ensureIndex({galaxy : 1, quadrant : 1, planet : 0})`
Eliminar un índice compuesto	`db.universe.dropIndex({galaxy : 1, quadrant : 1, planet : 0})`
Creación de un índice compuesto en "background"	`db.universe.ensureIndex({galaxy : 1, quadrant : 1, planet : 0},{unique : true, background : true})`

MongoDB. Agregación

- Las operaciones de agregación procesan registros de datos y devuelven resultados.
  - Es una técnica para simplificar código de aplicación y limitar el uso de recursos, ya que usa operaciones nativas en C++
- MongoDB 2.2 introdujo un **framework** de agregación modelado **en torno al concepto de pipelines de procesamiento de datos.**
  - A través de varios pasos los documentos son transformados en resultados agregados.
  - Filtros, transformaciones, agrupamientos, ordenación y cómputos son algunas de las operaciones.
- Además MongoDB también permite el uso de MapReduce para hacer agregación.
- Finalmente, MongoDB ofrece un conjunto de operadores sencillos de agregación como `count()`, `distinct()` o `group()`
  - http://docs.mongodb.org/manual/reference/operator/aggregation/interface/

Nombre	Descripción
`db.collection.aggregate()`	Proporciona acceso a la pipeline de agregación.
`db.collection.group()`	Agrupa documentos en una colección mediante una clave especificada y realiza un agregación simple. (obsoleto, se usa ahora incluido en "pipeline" de aggregate)
`db.collection.mapReduce()`	Realiza agregación map-reduce para conjuntos de datos extensos.

MongoDB. Pipeline de Agregación

- MongoDB incorpora un mecanismo para poder agregar datos de documentos en diferentes pasos.
  - Cada paso toma como entrada un conjunto de documentos y como salida produce un conjunto de documentos.
  - Hay operaciones que en un paso dado mantendrán el mismo número de documentos de entrada pero hay otras que los pueden reducir (filtrar).
- La siguiente tabla muestra equivalencias entre cláusulas de agregación, incluyendo operaciones de filtrado, agrupación, proyección, ordenación y cálculo, de SQL al framework de agregación de MongoDB:

SQL	MongoDB
WHERE	$match
GROUP BY	$group
HAVING	$match
SELECT	$project
ORDER BY	$sort
LIMIT	$limit
SUM	$sum
COUNT()	db.records.count()

MongoDB. Pipeline de Agregación

- Operaciones:
  - `$project` – cambia el conjunto de documentos modificando claves y valores. Es de tipo 1 a 1.
  - `$match` – es una operación de filtrado que reduce el conjunto de documentos generando un nuevo conjunto de los mismos que cumple alguna condición, ej. `{ $match: { <query> } }`
  - `$group` – hace el agrupamiento en base a claves o campos indexados, reduciendo el número de documentos.
  - `$sort` – ordenar en ascendente o descendente, dado que es computacionalmente costoso debería ser uno de los últimos pasos de la agregación .
  - `$skip` – permite saltar entre el conjunto de documentos de entrada, por ejemplo avanzar hasta el décimo documento de entrada. Se suele usar junto con `$limit`
  - `$limit` – limita el número de documentos a procesar.
  - `$unwind` – desagrega los elementos de un array en un conjunto de documentos. Esta operación incrementa el número de documentos para el siguiente paso.
- Ejemplos y documentación de estos operadores en:
  - http://docs.mongodb.org/manual/reference/operator/aggregation-pipeline/

MongoDB. Agrupación

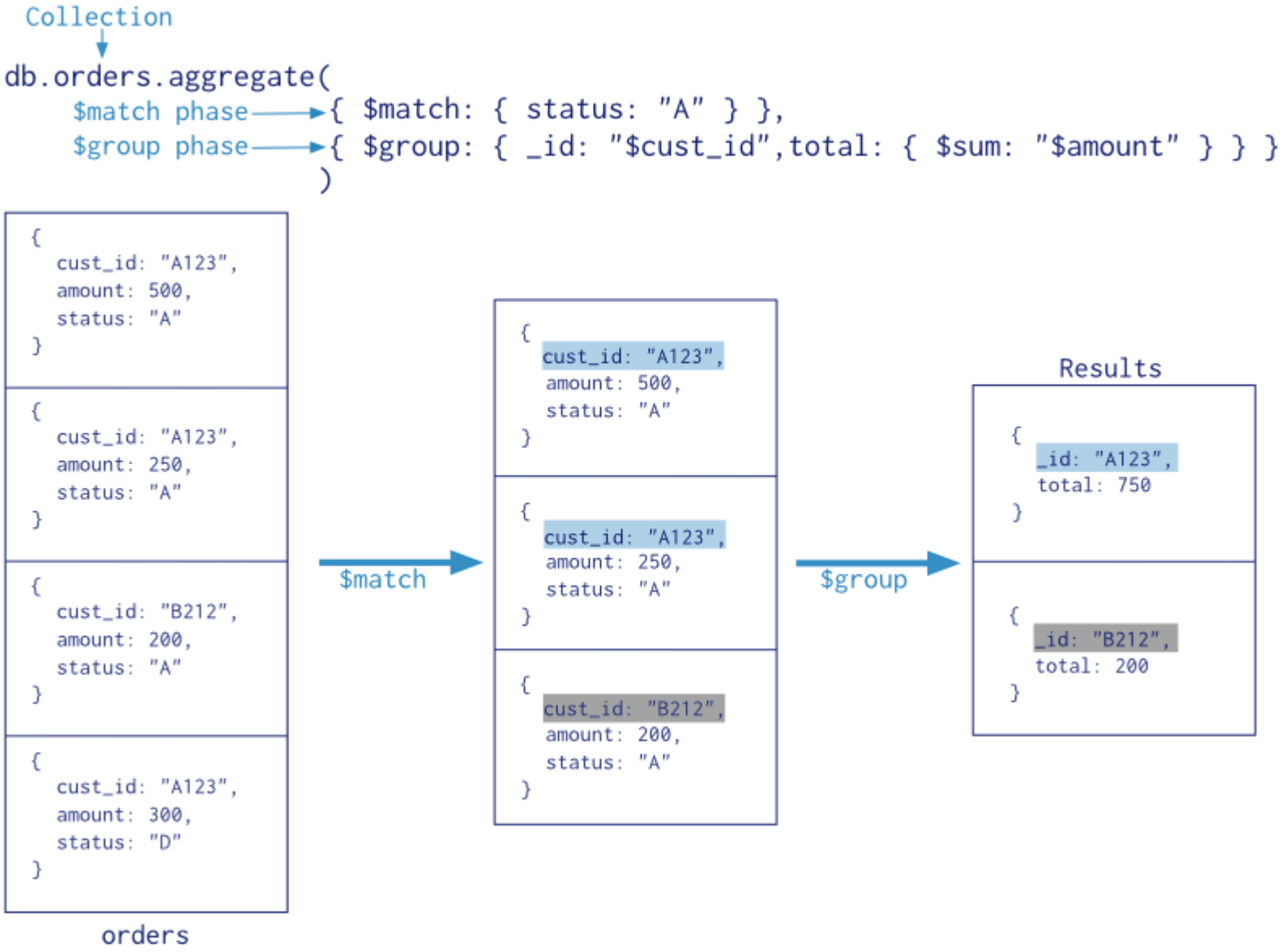
- A menudo agrupamos documentos para agregar valores en subconjuntos de documentos.
- Ejemplo, dado un conjunto de artículos:

```
db.article.aggregate([
   { $group :  {
     _id : "$author",
     docsPerAuthor : { $sum : 1 }, /*almacena la cantidad de documentos por cada autor*/
     viewsPerAuthor : { $sum : "$pageViews" } /*almacena la cantidad de vistas por autor*/
   }}
]);
```

MongoDB. Acumuladores sobre Grupos

Descripción	Ejemplo
$sum Suma sobre el valor definido de todos los documentos de la colección.	`db.ships.aggregate([{$group : {_id : "$armador", cant_embarcados : {$sum : "$tripulacion"}}}])`
$avg Calcula la media sobre los valores seleccionados de todos los documentos.	`db.ships.aggregate([{$group : {_id : "$armador", cant_embarcados : {$avg : "$tripulacion"}}}])`
$min Obtiene el mínimo de los valores seleccionados de todos los documentos.	`db.ships.aggregate([{$group : {_id : "$armador", cant_embarcados : {$min : "$tripulacion"}}}])`
$max Obtiene el máximo de los valores seleccionados de todos los documentos.	`db.ships.aggregate([{$group : {_id : "$armador", cant_embarcados : {$max : "$tripulacion"}}}])`
$push Devuelve un array de todos los valores que resultan de la aplicación de una expresión a cada documento en un grupo de documentos que comparten el mismo grupo por clave.	`db.ships.aggregate([{$group : {_id : "$armador", clases : {$push: "$clase"}}}])`
$addToSet Lo mismo que $push pero sin valores duplicados	`db.ships.aggregate([{$group : {_id : "$armador", clases : {$addToSet : "$clase"}}}])`
$first Obtiene el primer documento de un conjunto, sólo tiene sentido despues de un $sort.	`db.ships.aggregate([{$group : {_id : "$armador", primer_clase : {$first : "$clase"}}}])`
$last Obtiene el ultimo documento de un conjunto, sólo tiene sentido despues de un $sort.	`db.ships.aggregate([{$group : {_id : "$armador", ult_clase : {$last : "$clase"}}}])`

MongoDB. Pipeline de Agregación

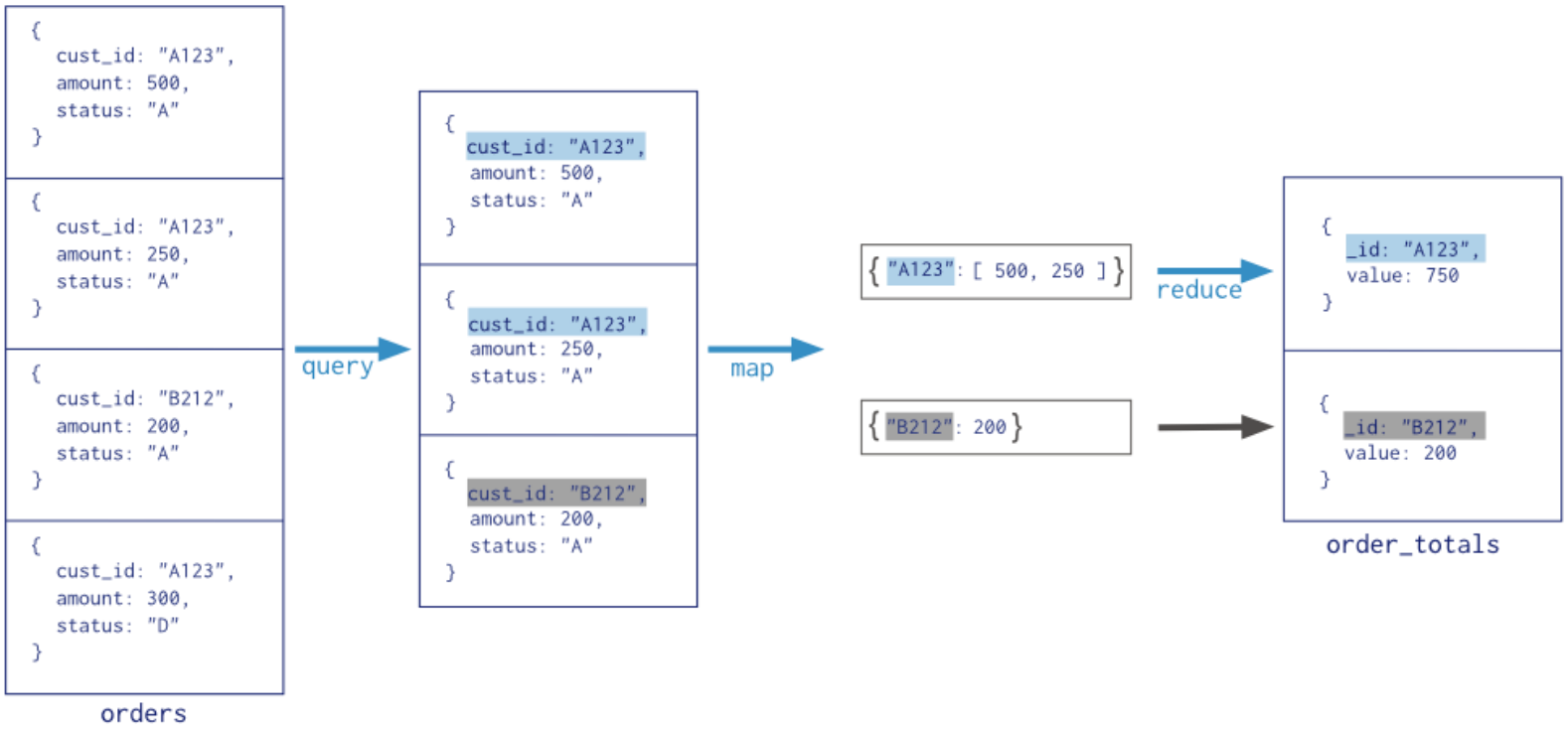


MongoDB. MapReduce

- MapReduce es un enfoque para procesar datos que tiene dos beneficios:
  - Puede ser paralelizado permitiendo que grandes volúmenes de datos sean procesados a través de varios cores/CPUs y máquinas,
  - El procesamiento se puede describir mediante el uso de JavaScript.
- Se materializa a través de dos pasos:
  1. Mapear los datos transformando los documentos de entrada en pares clave/valor.
  2. Reducir las entradas conformadas por pares clave y array de valores asociados a esa clave para producir el resultado final.

MongoDB. MapReduce

```
Collection
    ↓
db.orders.mapReduce(
        map      ──►  function() { emit( this.cust_id, this.amount ); },
        reduce   ──►  function(key, values) { return Array.sum( values ) },
                      {
        query    ──►    query: { status: "A" },
        output   ──►    out: "order_totals"
                      }
                  )
```



MongoDB. Ejemplo1 de MapReduce

- En este ejemplo vamos a contar el número de hits por día en un portal web.
  - Donde cada hit está representado con un log como:

```
resource    date
index       Jan 20 2010 4:30
index       Jan 20 2010 5:30
...         ...
```

  - Generando tras el procesamiento la siguiente salida:

```
resource year month day count
index    2010 1     20  2
...
```

- Para la función `map` emitiremos pares compuestos por una clave compuesta (`resource`, `year`, `month`, `day`) y un valor `1`, generando datos como:

```
{resource: 'index', year: 2010, month: 0, day: 20} => [{count: 1},
{count: 1}]
```

- La función `reduce` recoge cada dato intermedio y genera un resultado final:

```
{resource: 'index', year: 2010, month: 0, day: 20} => {count: 3}
```

- Documentación en:
  - http://docs.mongodb.org/manual/reference/method/db.collection.mapReduce/

MongoDB. Ejemplo1 de MapReduce

- Creamos la colección de entrada con un conjunto de documentos:

```
db.hits.insert({resource: 'index', date: new Date(2010, 0, 20, 4, 30)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 20, 5, 30)});
...
```

- La función map:

```
var map = function() {
    var key = {resource: this.resource, year: this.date.getFullYear(), month: this.date.getMonth(), day: this.date.getDate()};
    emit(key, {count: 1});
};
```

- La función reduce:

```
var reduce = function(key, values) {
    var sum = 0;
    values.forEach(function(value) {
        sum += value['count']; //sum se incrementa con el valor del campo
                               //count
    });
    return {count: sum};
};
```

- Ejecutamos mapReduce() (out: indica la colección que almacenará el resultado, {inline:1} indica que la colección generada no se almacenará, se mostrará en línea):

```
db.hits.mapReduce(map, reduce, {out: {inline:1}})
```

MongoDB. Ejemplo2 de MapReduce

- Considere la colección orders (pedidos) que contiene los documentos de acuerdo con siguiente prototipo:

```
{ _id: ObjectId("50a8240b927d5d8b5891743c"),
cust_id: "abc123",
ord_date: new Date("Oct 04, 2012"),
status: 'A',
price: 25,
items: [ { sku: "mmm", qty: 5, price: 2.5 },
         { sku: "nnn", qty: 5, price: 2.5 } ]
}
```

- Vamos a calcular el total del pedido para cada cliente.

```
var mapFunction1 = function() {
                      emit(this.cust_id, this.price);
                   };
var reduceFunction1 = function(keyCustId, valuesPrices) {
                         return Array.sum(valuesPrices);
                      };
```

MongoDB. Ejemplo2 de MapReduce

```
db.orders.mapReduce(
                        mapFunction1,
                        reduceFunction1,
                        { out: "map_reduce_ejemplo" }
                       )
```

- Resultado:

```
db.orders.find()

db.map_reduce_ejemplo.find()
```

MongoDB. Ejemplo3 de MapReduce

- Vamos a usar mapReduce en la colección de orders (pedidos) tomando los documentos que tienen un valor de ord_date mayor que 01/01/2012. Agruparemos por el campo `item.sku` , y calcularemos la cantidad de pedidos y la cantidad total pedida para cada sku (artículo). La operación concluye mediante el cálculo de la cantidad media por pedido para cada valor de sku.
- Map:

```
var mapFunction2 = function() {
            for (var idx = 0; idx < this.items.length; idx++) {
                var key = this.items[idx].sku;
                var value = {
                            count: 1,
                            qty: this.items[idx].qty
                            };
                emit(key, value);
            }
};
```

MongoDB. Ejemplo3 de MapReduce

- Se define la función de reduce correspondiente con dos argumentos `keySKU` y `countObjVals`:
  - `countObjVals` es un array cuyos elementos son los objetos agrupados por keySKU pasados por la función map a la función reduce.
  - La función reduce el array countObjVals a un sólo objeto reducedValue que contiene el recuento y los campos qty.
  - En reducedVal, el campo `count` contiene la suma de los campos `count` de cada elemento del array, y el campo `qty` contiene la suma de los campos `qty` de cada elemento del array.
- Reduce:

```
var reduceFunction2 = function(keySKU, countObjVals) {
                        reducedVal = { count: 0, qty: 0 };

                        for (var idx = 0; idx < countObjVals.length; idx++) {
                            reducedVal.count += countObjVals[idx].count;
                            reducedVal.qty += countObjVals[idx].qty;
                        }
                        return reducedVal;
                    };
```

MongoDB. Ejemplo3 de MapReduce

- Se define la función de finalize que modifica el objeto `reducedVal` para añadir un campo calculado denominado avg y devuelve el objeto modificado.

- Finalize:

```
var finalizeFunction2 = function (key, reducedVal) {

                    reducedVal.avg = reducedVal.qty/reducedVal.count;

                    return reducedVal;
                };
```

MongoDB. Ejemplo3 de MapReduce

- Se realiza la operación MapReduce sobre la colección orders usando las funciones mapFunction2, reduceFunction2 y finalizeFunction2.
- MapReduce:

```
db.orders.mapReduce(  mapFunction2,
                      reduceFunction2,
                     {
                       out: { merge: "map_reduce_ejemplo" },
                       query: { ord_date:
                                   { $gt: new Date('01/01/2012') }
                              },
                       finalize: finalizeFunction2
                     }
                    )
```

- Esta operación utiliza el campo de `query` para seleccionar sólo aquellos documentos con `ord_date` superior a `new Date (01/01/2012)`. El resultado se guarda en la colección `map_reduce_ejemplo`. Si ya existe la colección `map_reduce_example`, la operación combinará los contenidos existentes con los resultados de esta operación MapReduce.

MongoDB. Ejemplo3 de MapReduce

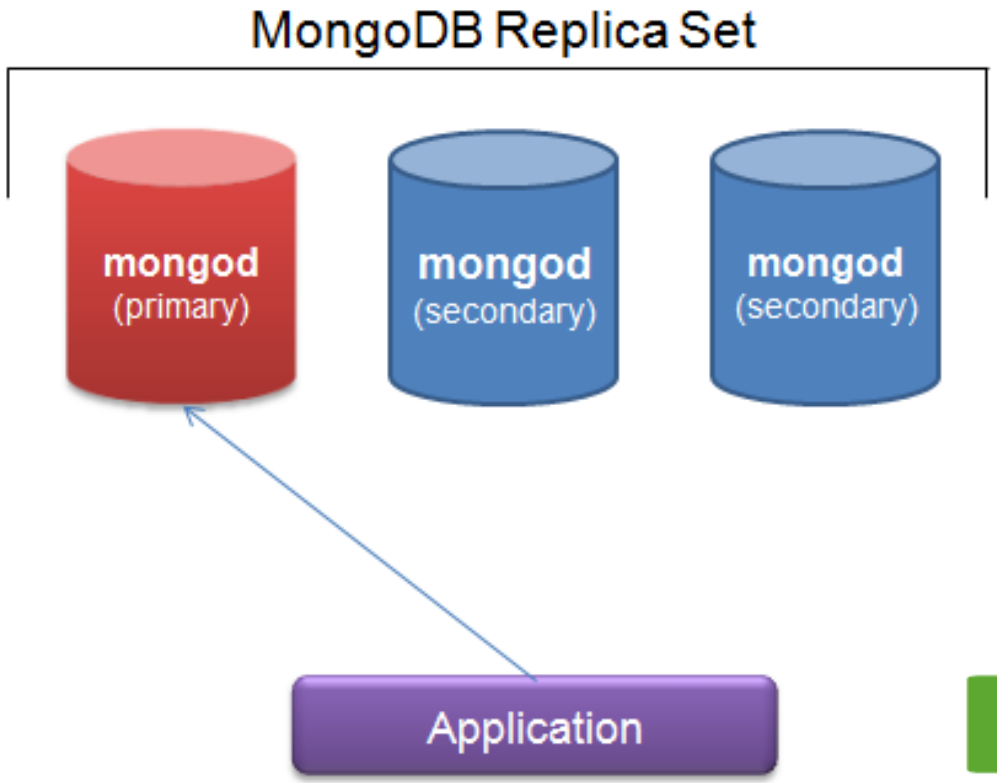
- También se puede ejecutar el comando MapReduce mediante db.runCommand():

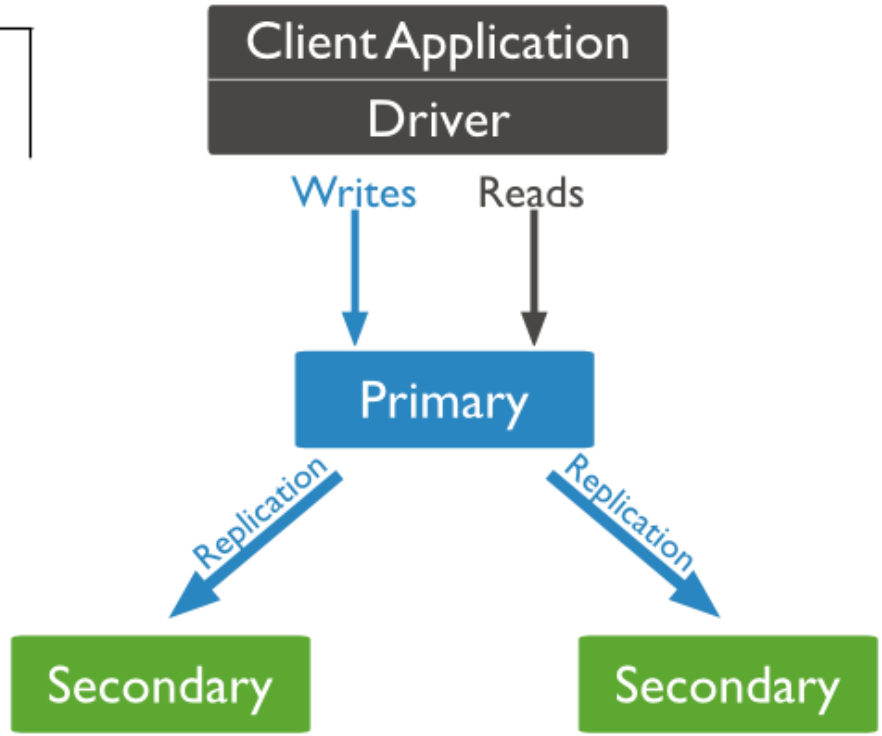
```
db.runCommand({
  mapReduce: "orders",
  map:function() {
              for (var idx = 0; idx < this.items.length; idx++) {
                    var key = this.items[idx].sku;
                    var value = {
                                count: 1,
                                qty: this.items[idx].qty
                                };
                    emit(key, value);
                } },
   reduce: function(keySKU, countObjVals) {
                           reducedVal = { count: 0, qty: 0 };
                           for (var idx = 0; idx < countObjVals.length; idx++) {
                               reducedVal.count += countObjVals[idx].count;
                               reducedVal.qty += countObjVals[idx].qty;
                           }
                           return reducedVal;};
   finalize:function (key, reducedVal) {
                        reducedVal.avg = reducedVal.qty/reducedVal.count;
                        return reducedVal;
                },
   out: {merge: "map_reduce_ejemplo" },
   query: {ord_date:{$gt: new Date('01/01/2012')}
})
```

MongoDB. Transacciones

- MongoDB no soporta transacciones distribuidas de colecciones.
  - Soporta transacciones atómicas en documentos lo que significa que uno siempre verá, o todos los cambios sobre un documento, o ninguno.
  - Si el documento incrusta colecciones en un tamaño no superior a 16 MB entonces la atomicidad de la transacción se garantizará.
- Algunas alternativas para garantizar la atomicidad de colecciones:
  - Estructura la aplicación de modo que quepa en un sólo documento haciendo uso de **atomicidad a nivel de documento de MongoDB**: `$inc`, `$set` o algunas más complejas como `findAndModify`
    - http://docs.mongodb.org/manual/reference/command/findAndModify/
  - Tolerar un poco de inconsistencia.
  - Hacer uso de un **two-phase commit** programáticamente para transacciones con varios documentos:
    - Es análogo a crear referencias manuales entre colecciones en "joins"
      - http://docs.mongodb.org/manual/tutorial/perform-two-phase-commits/

MongoDB. Replicación





MongoDB. Replica Set

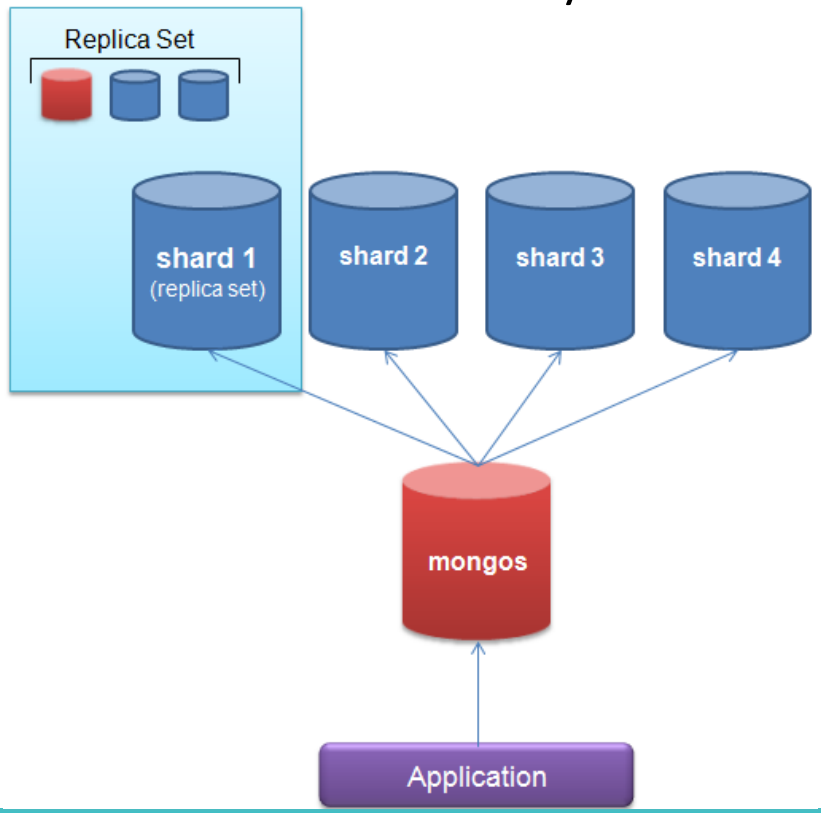
- La replicación se basa en el modelo Maestro/Esclavo, las escrituras son enviadas a un único servidor, el **maestro**, que sincroniza su estado a uno o varios esclavos. Todos juntos configuran un ReplicaSet.
- Cuando se lee de un nodo esclavo no hay garantía que el dato leído esté actualizado.
- Si se lee y escribe del nodo primario sí hay garantía de consistencia.
- En el periodo entre la caída de un nodo primario y un nuevo nodo primario es elegido, la consistencia está comprometida.
  - Todo nodo tiene un nivel de prioridad usado en el proceso de elección.
  - Si el nivel de prioridad es puesto a 0, ese nodo no puede ser nodo primario.
- El modo de escritura en un ReplicaSet está configurado por:
  - El valor del parámetro `w` indica el número de nodos que han de reconocer el write. Si tenemos tres nodos, dos puede ser un buen valor para `w`, o la alternativa `w='majority'.`
  - El valor del parámetro `j`. Con el valor `1` los datos se escribirán en el journal del nodo maestro.

MongoDB. Replica Set

- Documentación en:
  - http://docs.mongodb.org/manual/tutorial/deploy-replica-set-for-testing/
- Un listado de métodos para iniciar la replicación disponible en:
  - http://docs.mongodb.org/manual/reference/method/js-replication/

MongoDB. Sharding

- MongoDB soporta auto-sharding, una técnica de escalabilidad horizontal que particiona los datos a través de varios servidores.
  - Una implementación *elemental* podría poner los datos con nombre que empieza entre A-M en el servidor 1 y el resto en el servidor 2.

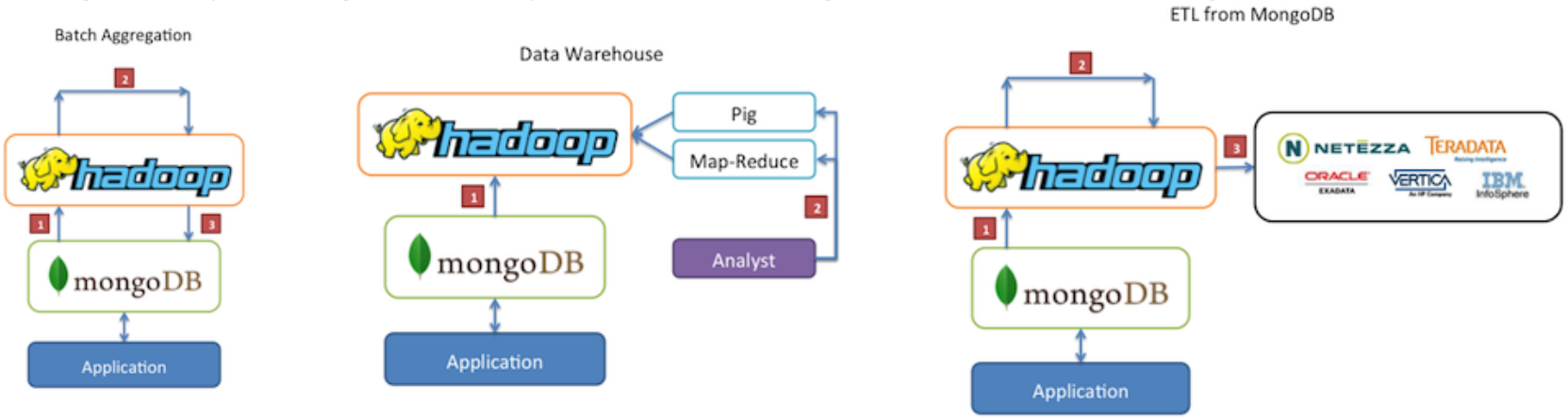


MongoDB. Sharding

- El sharding consiste en dividir una colección en varios nodos y luego acceder a sus contenidos a través de un nodo especial que actúa como router (`mongos`).
- Es una decisión que se toma cuando se despliega la base de datos.
- Para usar sharding es necesario que cada colección declare un ***shard-key.***
  - Clave definida para uno o varios campos de un documento.
  - Debe ayudar a dividir la colección en fragmentos (chunks).
  - Los chunks son luego distribuidos entre nodos, lo más equitativamente posible.
- La instancia `mongos` usa la clave shard-key para determinar el chunk y así el nodo utilizado.
- Las condiciones que un shard-key ha de cumplir son:
  - Cada documento tiene un shard-key.
  - El valor del shard-key no puede modificarse.
  - Debe ser parte de un índice y debe ser el primer campo de un índice compuesto.
  - No puede haber un índice único a no ser que esté conformado por el shard-key .
  - Si no se usa el shard-key en una operación de lectura esta petición llegará a todos los shards.
  - La clave de shard debe ofrecer suficiente cardinalidad para poder utilizar todos los shards.
- Documentación: “Convert a Replica Set to a Replicated Sharded Cluster”.
  - http://docs.mongodb.org/manual/tutorial/convert-replica-set-to-replicated-shard-cluster/

MongoDB y Hadoop

- MongoDB y Hadoop son una combinación poderosa y pueden ser utilizados para realizar tareas de análisis y procesado de datos almacenados en MongoDB.
  - MongoDB Connector for Hadoop
    - http://docs.mongodb.org/ecosystem/tools/hadoop/
- Casos de uso:
  - Agregación batch, data warehouse y datos ETL
- MongoDB y Hadoop:
  - http://docs.mongodb.org/ecosystem/tutorial/getting-started-with-hadoop/
- BigData y MongoDB: http://www.mongodb.com/learn/big-data



Referencias

- MongoDB sitio oficial:
  - http://www.mongodb.com/
- Uso de mongo Shell:
  - https://docs.mongodb.com/manual/mongo/
- Mongo Shell Quick Reference:
  - http://docs.mongodb.com/manual/reference/mongo-shell/
- "Little MongoDB Book" por Karl Seguin:
  - http://openmymind.net/2011/3/28/The-Little-MongoDB-Book/
- MongoDB: Primer contacto por Thomas Jaspers:
  - https://blog.codecentric.de/en/2012/11/mongodb-first-contact/
- MongoDB: la BBDD NoSQL más popular del mercado:
  - http://www.slideshare.net/dipina/mongodb-la-bbdd-nosql-ms-popular-del-mercado
- Hoja Resumen sobre MongoDB:
  - https://blog.codecentric.de/files/2012/12/MongoDB-CheatSheet-v1_0.pdf
  - https://dhodgin.files.wordpress.com/2016/11/mongo-shell-cheat-sheet-v3-4.pdf

Referencias