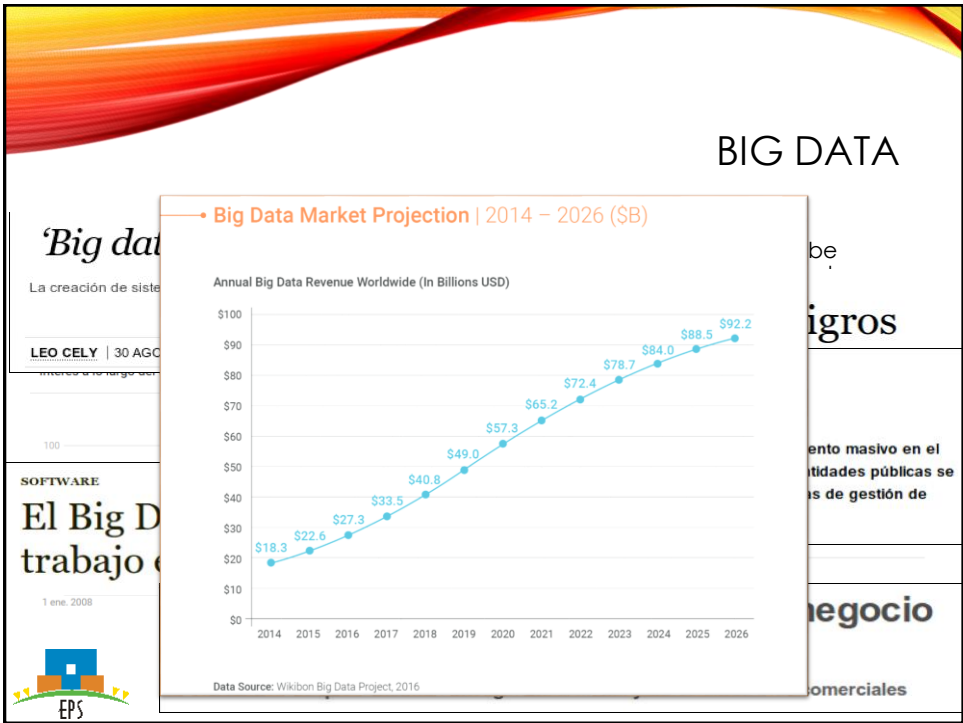


BASES DE DATOS DISTRIBUIDAS

Sistemas informáticos I

3.6 BIG DATA Y BASES DE DATOS NoSQL

Bases de datos distribuidas




¿QUÉ ES BIG DATA?

- Obtener información útil a partir de **volúmenes de datos** tan **grandes** que hasta hace pocos años no era posible adquisición y procesamiento
 - "Big Data" es similar a "Small Data", solo que con mayores volúmenes de datos
 - La diferencia de tamaño requiere de soluciones diferentes
 - Técnicas
 - Herramientas
 - Arquitecturas
- El objetivo es resolver problemas nuevos y mejorar la solución de problemas viejos:
 - Sistemas de recomendación (*Amazon*)
 - Escuchar las redes sociales (*sentiment analysis*)
 - Predecir eventos y actuar en consecuencia:
 - Mantenimiento predictivo de maquinaria
 - Identificación de clientes en riesgo de abandono
 - Predicción de epidemias (dónde, cuándo, quién)
 - Predicción de crímenes (dónde, cuándo, quién)
 - Predicción de consumo energético
 - Etc., etc., etc., etc...

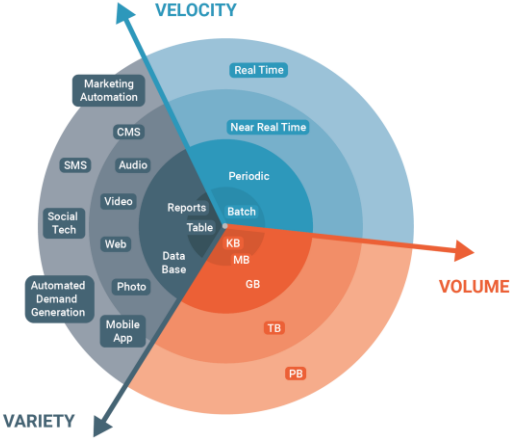
EPS


¿Y POR QUÉ AHORA?

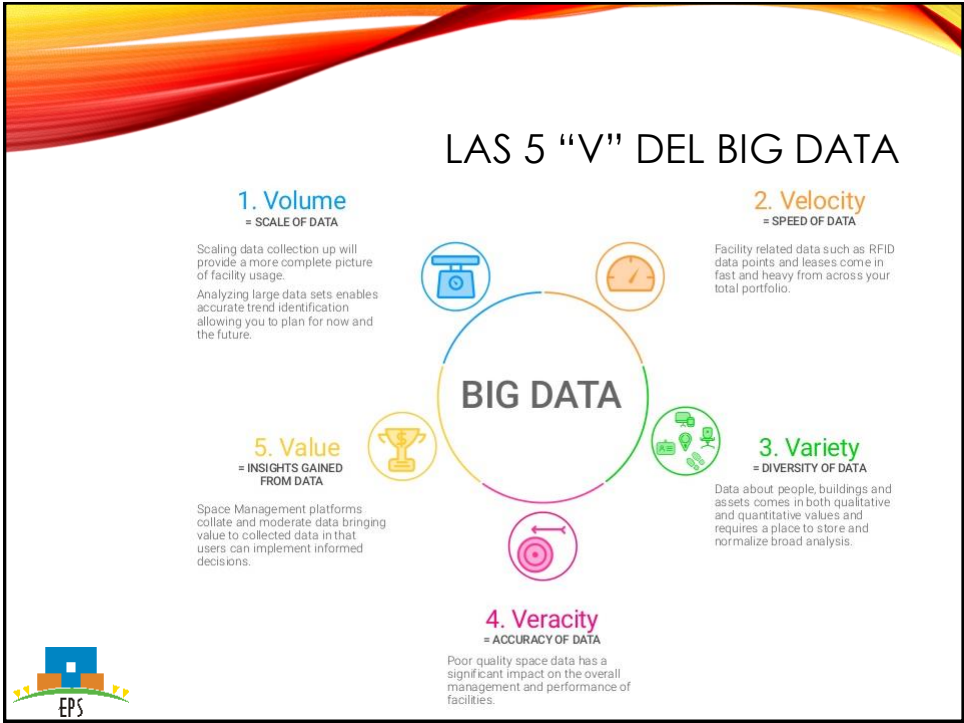
- Fundamentalmente se han dado dos condiciones:
 - Incremento exponencial en la cantidad de datos generados y disponibles
 - Cada día (y cada vez más) nuestra actividad genera una gran cantidad de datos:
 - Con el objetivo de mantener un registro: notas de estudiantes, evolución de pacientes...
 - Información que suministramos: fichaje en el trabajo, entrada en nuestro correo o cualquier actividad en la nube, pagos con tarjeta...
 - Sin que nos demos cuenta: contaje de vehículos en una vía, cámaras de vigilancia, registro de visita en sitios web, registro de lo que compramos...
 - Revolución tecnológica/social:
 - Web 2.0
 - Smartphones
 - Internet de las cosas
- Aparición de tecnologías de bajo coste que permiten su almacenamiento y procesamiento
 - Alta confiabilidad y velocidad de transmisión de datos
 - Bajo coste de sensores, cámaras, etc.



LAS 3 “V” DEL BIG DATA







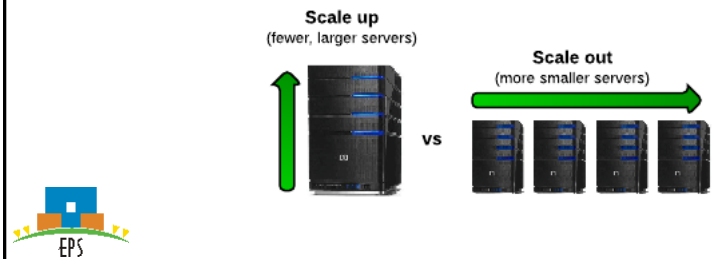
PROBLEMÁTICA DEL BIG DATA

- Volumetría de datos:
 - En 2011 se calculaba que el volumen de datos en el mundo era un poco más de 10^{21} bytes (1 ZB)
 - Se estima que actualmente está alrededor de los 3-4 ZB
 - Se cree que para 2040 alcanzará los 40 ZB
- Los modelos tradicionales de procesamiento y almacenamiento (vigentes desde ~1970) no son suficientes para algunas casuísticas
- Surgen dos necesidades, interrelacionadas:
 - Procesamiento distribuido
 - Almacenamiento distribuido
- Entre otras cosas, implica que los datos pueden estar en sitios físicos distintos y posiblemente replicados

EPS

ESCALABILIDAD

- Para entender las soluciones propuestas, vamos a concentrarnos en la V de volumen (y su amiga inseparable la V de velocidad)
- Trabajar con mayores volúmenes de datos implica aumentar la capacidad de almacenamiento y procesamiento de los sistemas
- Dos tipos posibles de crecimiento (escalabilidad):
 - Vertical (scale up)
 - Horizontal (scale out)



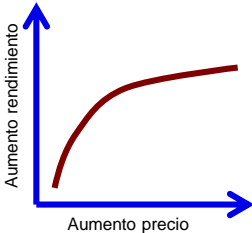
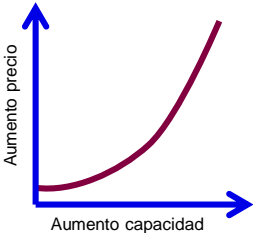

ESCALABILIDAD VERTICAL

- Incrementar la potencia de la máquina en la que se ejecuta el software
 - Ya sea nuevos y mejores componentes o cambiar el ordenador completo



ESCALABILIDAD VERTICAL

- Ventajas
 - Es más simple si el software está preparado
- Desventajas
 - Por más preparado que esté el software, más temprano que tarde encontraremos limitaciones.
 - Windows Server 2008 R2 Datacenter = máx 2 TB de RAM
 - Es muy caro



ESCALABILIDAD HORIZONTAL




- Distribuir la carga de trabajo entre varios ordenadores conectados entre sí
 - Normalmente estos ordenadores son de gama baja/media (más baratos)






ESCALABILIDAD HORIZONTAL

- Ventajas:
 - Los límites son mucho más altos (potencialmente miles de ordenadores conectados)
 - Cuando más capacidad, más barato respecto de escalabilidad vertical
 - Normalmente escalabilidad lineal: si duplico el número de ordenadores, duplico el rendimiento (predictibilidad)
- Desventajas:
 - Requiere de software específicamente diseñado e implementado para ejecutarse en varios ordenadores a la vez (procesamiento distribuido)



TEOREMA CAP

- La escalabilidad horizontal implica cierta probabilidad de que falle uno de los nodos conectados o la comunicación entre ellos
- En estas condiciones hay tres propiedades deseables:
 - Consistencia: para resumir, que todos los nodos contengan valores consistentes entre sí en todo momento
 - Disponibilidad (**A**vailability): garantía de que cada petición a un nodo reciba una confirmación de si ha sido o no resuelta satisfactoriamente
 - Tolerancia al **P**articionado: que pueda fallar un nodo o conexión y el sistema siga funcionando
- El teorema CAP establece la imposibilidad de que un sistema ofrezca las tres propiedades simultáneamente, solo pudiendo cubrirse simultáneamente dos de ellas:
 - Sistemas CA: SGBDR
 - Sistemas CP: mayoría BBDD NoSQL (Ejem: MongoDB)
 - Sistemas AP: Ejem: Apache Cassandra



NUEVA REALIDAD, NUEVA PROBLEMÁTICA

- Problema de caída de servidores cuando tengo tantos
- Las BBDD se pueden distribuir, pero a costa de sus propiedades ACID (atomicidad, consistencia, aislamiento y durabilidad)



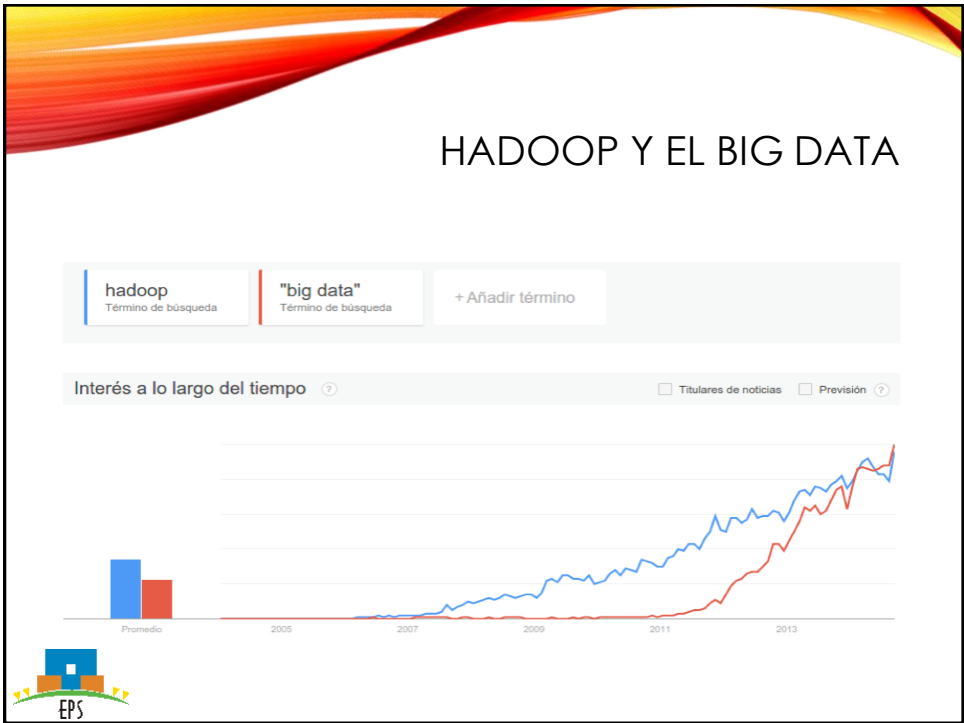
- El procesamiento distribuido en un contexto big data requiere el uso de modelos computacionales no estándar:
 - Más allá del modelo de servidor único, e incluso de modelos de computación paralela tradicionales (p.ej., *message passing interface*, MPI)
 - El objetivo es simplificar tareas de programación complejas
- La primera tecnología que tuvo éxito (y facilitó la expansión del big data) fue *Apache Hadoop*
 - Implementación (código abierto) del modelo de programación *MapReduce*, el más popular hasta el momento
 - Basado en el sistema de ficheros HDFS




ORIGEN DE LA TECNOLOGÍA HADOOP

- Curiosamente (o no), *HDFS* y *MapReduce* se basan en publicaciones de trabajos de investigación de científicos de Google:
 - The Google File System (2003)
 - MapReduce: Simplified Data Processing on Large Clusters (2004)
- Ambas ideas fueron implementadas para un motor de búsqueda e indexación (*Apache Nutch*) por Doug Cutting
 - Más adelante lo contrató Yahoo! y dio mucho impulso a la iniciativa

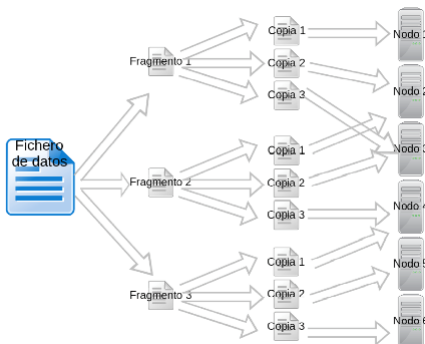




- ## CARACTERÍSTICAS DE HADOOP
- Cubre necesidades de almacenamiento y procesamiento masivo de datos
 - Las tareas se ejecutan en una red (*cluster hadoop*) de ordenadores conectados entre sí (nodos) que se reparten la tarea
 - La suma total de capacidad de proceso y almacenamiento es igual a la suma de capacidad de proceso y almacenamiento de cada uno de sus nodos
 - Esto dota al sistema de escalabilidad horizontal y capacidad de crecer según las necesidades
- 

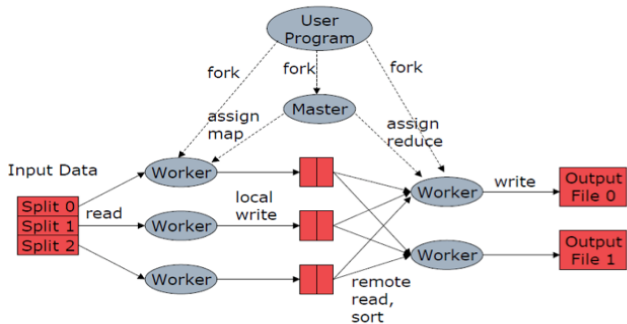
HDFS – HADOOP DISTRIBUTED FILE SYSTEM

- Permite aprovechar y trabajar con la capacidad total de almacenamiento de todos los ordenadores a la vez, mostrándonosla como si fuera uno solo
- Es un sistema de almacenamiento tolerante a fallos (mediante replicación)



MAPREDUCE

- Un problema objetivo debe ser paralelizable → divisible en sub tareas que puedan ser ejecutadas por separado
 - Primero se divide el problema en problemas menores (etapa Map)
 - Luego los problemas más pequeños son resueltos paralelamente
 - Finalmente, el conjunto de soluciones a los problemas menores es sintetizado en una solución al problema original (etapa Reduce)



MAPREDUCE

- ¿Qué tiene de “mágico”?
 - Hace fácil lo difícil (procesamiento paralelo)
- Ejemplo: clasificar y contar libros de una biblioteca
 - Los *mappers* recorren las salas, clasifican los libros y los cuentan: generan la cuenta de libros de cada categoría que han encontrado
 - Los *reducers* cogen las listas anteriores y cada uno suma los libros de una categoría. Al final tengo total de libros por cada categoría (y total de libros si sumo eso)
- Puedo reducir a la mitad el tiempo de los *mappers* si duplico su cantidad (cada uno tiene que hacer la mitad)



EL ECOSISTEMA HADOOP

- No es sólo *Hadoop*, sino una serie de sistemas de apoyo y complemento
 - Sqoop: transferencia de ficheros entre HDFS y bases de datos
 - Flume: transferencia de datos generados de forma continua (p.ej., logs) a ficheros HDFS
 - Oozie: permite definir y ejecutar flujos de trabajo sobre Hadoop
 - Hive: Motor SQL sobre ficheros HDFS. Traduce de forma transparente al usuario consultas SQL a programas map-reduce. Esta traducción ralentiza el proceso
 - Cludera Impala: funcionalidad equivalente a Hive pero puede llegar a ser 100 veces más rápido: algoritmos distribuidos propios (no map-reduce) que trabajan en memoria principal
 - Pig: permite utilizar lenguaje más sencillo que MapReduce en Java (Pig Latin) para procesar datos
 - Hue: interfaz gráfica para componentes principales de Hadoop
 - Y muchos más
- Configurar y mantener actualizados todos los componentes Hadoop puede ser una tarea compleja
- Como alternativa, se ofrecen instalaciones de pago, con todo listo para ser instalado y usado: CDH de Cloudera, HDP de HortonWorks, MapR...



LIMITACIONES DEL MAPREDUCE

- Complejidad:
 - aunque reduce la dificultad de la programación paralela, su implementación es a bajo nivel y no trivial.
- Rigidez:
 - las soluciones siempre deben expresar en dos etapas con semántica muy estricta.
 - A veces obliga a soluciones forzadas y poco naturales.
 - En el extremo, cierto tipo de problemas no pueden ser solucionados con este método.
- Antigüedad



APACHE SPARK

- Alternativa principal a *Hadoop*
- Versión 1.0 en junio de 2014
- Modelo de programación más sencillo y más potente (incluye pero no se limita a *MapReduce*)
- Conjunto de herramientas mejor integradas entre sí
- Rendimientos entre 10 y 100 veces mejor que el *MapReduce* de Hadoop
 - Preferencia por procesamiento en memoria.



BASES DE DATOS NoSQL

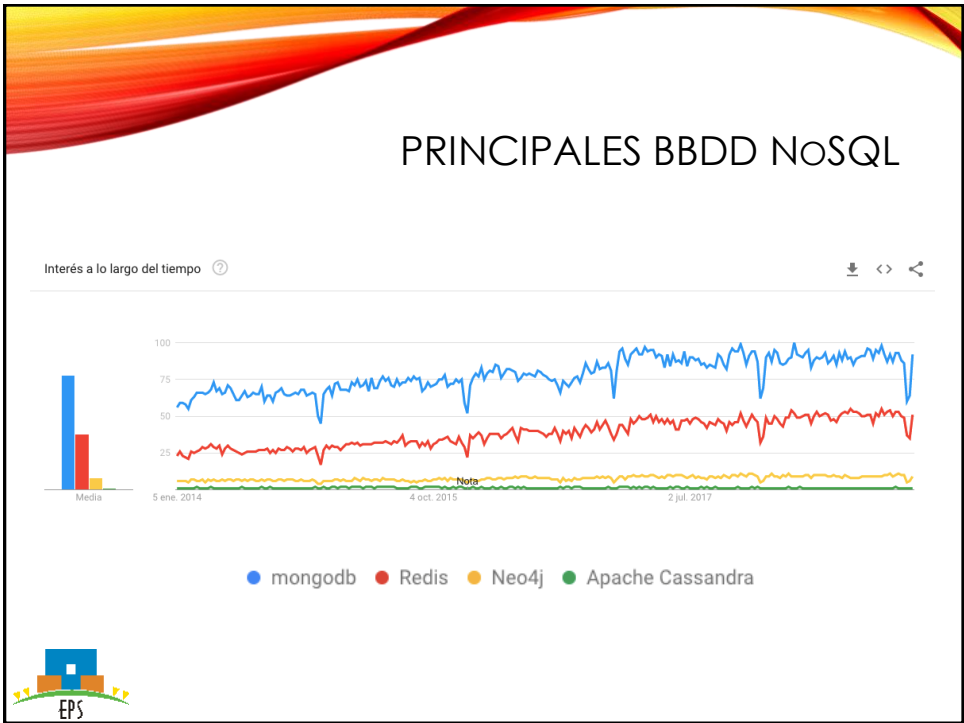
- Bases de datos sin esquemas
- Mayormente utilizan interfaces distintas al SQL
- En general dan soporte al almacenamiento de grandes cantidades de datos mediante escalabilidad horizontal
- Operan sobre infraestructuras distribuidas, como *Hadoop*
- Son flexibles



¿POR QUÉ NoSQL?

- Las BBDD relacionales imponen esquemas y estructuras rígidas que no siempre son adecuadas para un tratamiento rápido y flexible de la información.
- Cuando hablamos de BBDD NoSQL estamos hablando en realidad de BBDD no relacionales, que escapan del modelo habitual de filas y columnas para almacenar la información modelada en forma de entidades y relaciones.
- Tipos de BBDD NoSQL:
 - Basadas en pares clave-valor:
 - Utilizan combinaciones nombre:valor, normalmente en memoria y de acceso rápido. Ejemplo: [Redis](#).
 - Basadas en grafos:
 - Utilizan un grafo para establecer conexiones entre datos, así como mecanismos de consulta más eficientes. Ejemplos: [Neo4j](#) y [JanusGraph](#).
 - Basadas en columnas:
 - Utilizan filas y columnas, pero con nombres y formatos variables entre filas. Pueden verse como BBDD basadas en clave-valor bidimensionales. Ejemplos: [Cassandra](#) y [Hbase](#).
 - Basadas en documentos:
 - Utilizan documentos en vez de tablas, y se caracterizan por su gran flexibilidad y capacidad de almacenamiento. Ejemplos: [CouchBase](#) y [MongoDB](#).






BASES DE DATOS DOCUMENTALES


- Se basan en el modelo clave-valor, pero permitiendo el uso de meta-datos para aportar mayor expresividad.
- La unidad organizativa de la información es el documento, que goza de alta flexibilidad. Cada uno consta de un ID único.
- Los datos se agrupan en colecciones y documentos, que serían como las tablas y las filas, respectivamente, en las BBDD Relacionales.
- Suelen basarse en el formato JSON preferentemente, si bien pueden usar también XML.
- La principal ventaja es la flexibilidad, ya que no tienen estructuras predefinidas. Podemos tener documentos diferentes entre sí. Esto permite cambios ágiles, sin tener que modificar estructuras internas predefinidas, así como consultas más naturales y reducción de la verbosidad en la mayoría de los casos.
- Sin embargo, la utilización de esquemas flexibles también las hace propensas a errores de introducción de datos, por lo que es necesario implementar métodos de saneado y limpieza de datos.

EPS



- MongoDB (del inglés **humongous**, “enorme”) es un sistema de BBDD orientado a documentos. Es gratis y de código abierto. Muy utilizada en la industria.
- Permite trabajar con documentos de manera distribuida.
- Los documentos pueden tener estructuras diferentes (schemaless).
 - Este aspecto, ventajoso, puede volverse un inconveniente (que en las BBDD Relacionales está mejor controlado por su rigidez). A veces son necesarias reglas explícitas.
 - No existe validación ni integridad referencial. Deben ser implementadas.
- Otros aspectos a destacar:
 - Características ACID comprometidas
 - Más bien se habla de CAP (Consistencia, Disponibilidad y Tolerancia al particionado).
 - Se garantiza la atomicidad sólo a nivel de documento.
 - Permite el escalado horizontal de manera sencilla.
 - Sistema de sharding para distribuir información en diferentes clusters.
 - Buen rendimiento en lectura.
 - Peor cuando el número de operaciones de escritura es muy alto.
 - Funciona bien para grandes cantidades de datos almacenados.
 - No es adecuado para transacciones complejas.







- Definiciones previas:
 - **Documento**: unidad básica de almacenamiento. La información se guarda en formato BSON (Binary JSON). Se permiten documentos embebidos en otros.
 - **Colección**: grupos de documentos.
 - **Base de Datos**: contenedores físicos para almacenar colecciones.
 - **Cluster**: almacena varias bases de datos. Al tratarse de una BBDD distribuida, permite una gran escalabilidad horizontal.

```
{
  name: "al",
  age: 18,
  status: "D",
  groups: [ "politics", "news" ]
}
```

Collection






Documentos en MongoDB

```
{
  "address": {
    "building": "1007",
    "coord": [ -73.856077, 40.848447 ],
    "street": "Morris Park Ave",
    "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
    { "date": { "$date": "1393804800000" }, "grade": "A", "score": 2 },
    { "date": { "$date": "1378857600000" }, "grade": "A", "score": 6 },
    { "date": { "$date": "1358985600000" }, "grade": "A", "score": 10 },
    { "date": { "$date": "1322006400000" }, "grade": "A", "score": 9 },
    { "date": { "$date": "1299715200000" }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```

Documentos Embebidos





- Comandos básicos CRUD del cliente MongoDB (<https://docs.mongodb.com>):
 - Mostrar bases de datos:
 - show dbs
 - Creación y utilización de bases de datos:
 - use nombre_base_de_datos
 - Borrado de bases de datos en uso:
 - db.dropdatabase
 - Mostrar colecciones:
 - show collections
 - Crear colección en una base de datos en uso:
 - db.createCollection(nombre, opciones)
 - Se crea automáticamente cuando se inserta un documento.
 - Insertar un documento en una colección:
 - db.nombre_coleccion.insert(nombre_documento)
 - db.nombre_coleccion.insertOne(documento)
 - db.nombre_coleccion.insertMany(documentos)
 - Eliminar colección:
 - db.nombre_coleccion.drop()
 - Búsqueda:
 - db.nombre_coleccion.find()
 - Búsqueda con salida formateada:
 - db.nombre_coleccion.find().pretty()





- Comandos básicos CRUD del cliente MongoDB (<https://docs.mongodb.com>):
 - Actualizar datos de documentos:
 - `db.nombre_coleccion.update(criterio_seleccion, datos)`
 - Actualizar documento completo:
 - `db.nombre_coleccion.save({_id:ObjectId(), datos})`
 - Eliminar documentos de una colección:
 - `db.nombre_coleccion.remove(criterio_seleccion)`
 - Ordenar documentos en una colección:
 - `db.nombre_coleccion.find().sort({KEY:i})`
 - Donde `i` puede ser 1 (ascendente) o -1 (descendente)
 - Se pueden indicar múltiples campos a indexar
 - Creación de índices:
 - `db.nombre_coleccion.ensureIndex({KEY:i})`
 - Donde `i` puede ser 1 (ascendente) o -1 (descendente)
 - Agregación en la búsqueda:
 - `db.nombre_colección.aggregate(operacion_de_agregacion)`





Tipos BSON


- String
- Integer
- Boolean
- Double
- Min/ Max
- Arrays
- Timestamp
- Object
- Null
- Symbol
- Date
- Object ID
- Binary data
- Code
- Regular expression







- Ejemplos básicos:
 - > use mi_base_de_datos
 - > db.mi_coleccion.insertOne({ x: "Jose Antonio" })
 - > db.mi_coleccion.find()
{ "_id" : ObjectId("5da9f5c2b173ecf9651a2658"), "x" : "Jose Antonio" }
 - > db.mi_coleccion.insert({y:"Roberto"})
 - > db.mi_coleccion.insert({z:"Alvaro"})
 - > db.mi_coleccion.save({ _id:ObjectId("5da9f5c2b173ecf9651a2658"), m:"Jose" })
 - > db.mi_coleccion.find()
{ "_id" : ObjectId("5da9f5c2b173ecf9651a2658"), "m" : "Jose" }
{ "_id" : ObjectId("5da9fc0eb173ecf9651a2659"), "y" : "Roberto" }
{ "_id" : ObjectId("5da9fc39b173ecf9651a265a"), "z" : "Alvaro" }
 - > db.mi_coleccion.remove({y:"Roberto"})
 - > db.mi_coleccion.update({z:"Alvaro"}, {\$set:{z:"Alvaro y Roberto"}})
 - > db.mi_coleccion.find()
{ "_id" : ObjectId("5da9f5c2b173ecf9651a2658"), "m" : "Jose" }
{ "_id" : ObjectId("5da9fc39b173ecf9651a265a"), "z" : "Alvaro y Roberto" }





- Ejemplos con documentos embebidos:
 - > use mi_base_de_datos
 - > db.createCollection("inventario")
 - > db.inventario.insertMany([
 { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
 { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
 { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
 { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
 { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
 - > db.inventario.find({status:"D"})
{ "_id" : ObjectId("5da9e5f1b173ecf9651a2650"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
{ "_id" : ObjectId("5da9e5f1b173ecf9651a2651"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, "w" : 30, "uom" : "cm" }, "status" : "D" }
 - > db.inventario.find({\$in: [{"A","D"}]})
{ "_id" : ObjectId("5da9e5f1b173ecf9651a264e"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("5da9e5f1b173ecf9651a264f"), "item" : "notebook", "qty" : 50, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "A" }
{ "_id" : ObjectId("5da9e5f1b173ecf9651a2650"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
{ "_id" : ObjectId("5da9e5f1b173ecf9651a2651"), "item" : "planner", "qty" : 75, "size" : { "h" : 22.85, "w" : 30, "uom" : "cm" }, "status" : "D" }
{ "_id" : ObjectId("5da9e5f1b173ecf9651a2652"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w" : 15.25, "uom" : "cm" }, "status" : "A" }





MongoDB Compass

mi_base_de_datos.inventario

DOCUMENTS 5TOTAL SIZE 560BTOTAL SIZE 4.0KBAVG. SIZE 112BAVG. SIZE 4.0KBINDEXES 1

DocumentsAggregationsExplain PlanIndexes

FILTERstatus: \$in: ["A","D"]

OPTIONS

FIND

RESET

...

INSERT DOCUMENT


VIEW


LIST

TABLE

Displaying documents 1 - 5 of 5

inventario					
	_id ObjectId	item String	qty Double	size Object	status String
1	5da9e5f1b173ecf9651a264e	"journal"	25	{ } 3 fields	"A"
2	5da9e5f1b173ecf9651a264f	"notebook"	50	{ } 3 fields	"A"
3	5da9e5f1b173ecf9651a2650	"paper"	100	{ } 3 fields	"D"
4	5da9e5f1b173ecf9651a2651	"planner"	75	{ } 3 fields	"D"
5	5da9e5f1b173ecf9651a2652	"postcard"	45	{ } 3 fields	"A"






```
> db.inventario.find( { status: "A", $or: [ { qty: { $lt: 30 } } ], { item: /p/ } } )
{ "_id" : ObjectId("5da9e5f1b173ecf9651a264e"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("5da9e5f1b173ecf9651a2652"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w" : 15.25, "uom" : "cm" }, "status" : "A" }


> db.inventario.find( { size: { h: 14, w: 21, uom: "cm" } } )
{ "_id" : ObjectId("5da9e5f1b173ecf9651a264e"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }

> db.inventario.find( { "size.h": { $lt: 15 } } )
{ "_id" : ObjectId("5da9e5f1b173ecf9651a264e"), "item" : "journal", "qty" : 25, "size" : { "h" : 14, "w" : 21, "uom" : "cm" }, "status" : "A" }
{ "_id" : ObjectId("5da9e5f1b173ecf9651a264f"), "item" : "notebook", "qty" : 50, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "A" }
{ "_id" : ObjectId("5da9e5f1b173ecf9651a2650"), "item" : "paper", "qty" : 100, "size" : { "h" : 8.5, "w" : 11, "uom" : "in" }, "status" : "D" }
{ "_id" : ObjectId("5da9e5f1b173ecf9651a2652"), "item" : "postcard", "qty" : 45, "size" : { "h" : 10, "w" : 15.25, "uom" : "cm" }, "status" : "A" }
```


Ejemplo de proyección de campos:


```
db.inventario.find( { status: "A" }, { item: 1, status: 1 } )
{ "_id" : ObjectId("5da9e5f1b173ecf9651a264e"), "item" : "journal", "status" : "A" }
{ "_id" : ObjectId("5da9e5f1b173ecf9651a264f"), "item" : "notebook", "status" : "A" }
{ "_id" : ObjectId("5da9e5f1b173ecf9651a2652"), "item" : "postcard", "status" : "A" }
```







- Ejemplos con Arrays:
 - ```
> db.inventario2.insertMany([
 {item: "journal", qty: 25, tags: ["blank", "red"], dim_cm: [14, 21] },
 {item: "notebook", qty: 50, tags: ["red", "blank"], dim_cm: [14, 21] },
 {item: "paper", qty: 100, tags: ["red", "blank", "plain"], dim_cm: [14, 21] },
 {item: "planner", qty: 75, tags: ["blank", "red"], dim_cm: [22.85, 30] },
 {item: "postcard", qty: 45, tags: ["blue"], dim_cm: [10, 15.25] }
]);
```
  - ```
> db.inventario2.find( { tags: { $all: ["red", "blank"] } } )
{ "_id" : ObjectId("5da9eb28b173ecf9651a2653"), "item" : "journal", "qty" : 25,
  "tags" : [ "blank", "red" ], "dim_cm" : [ 14, 21 ] }
{ "_id" : ObjectId("5da9eb28b173ecf9651a2654"), "item" : "notebook", "qty" : 50,
  "tags" : [ "red", "blank" ], "dim_cm" : [ 14, 21 ] }
{ "_id" : ObjectId("5da9eb28b173ecf9651a2655"), "item" : "paper", "qty" : 100,
  "tags" : [ "red", "blank", "plain" ], "dim_cm" : [ 14, 21 ] }
{ "_id" : ObjectId("5da9eb28b173ecf9651a2656"), "item" : "planner", "qty" : 75,
  "tags" : [ "blank", "red" ], "dim_cm" : [ 22.85, 30 ] }
```
 - ```
> db.inventario2.find({ dim_cm: { $elemMatch: { $gt: 22, $lt: 30 } } })
{ "_id" : ObjectId("5da9eb28b173ecf9651a2656"), "item" : "planner", "qty" : 75,
 "tags" : ["blank", "red"], "dim_cm" : [22.85, 30] }
```





- Ejemplos de tratamiento de texto y expresiones regulares:
  - ```
> db.tiendas.insert( [
  { _id: 1, name: "Java Hut", description: "Coffee and cakes" },
  { _id: 2, name: "Burger Buns", description: "Gourmet hamburgers" },
  { _id: 3, name: "Coffee Shop", description: "Just coffee" },
  { _id: 4, name: "Clothes Clothes Clothes", description: "Discount clothing" },
  { _id: 5, name: "Java Shopping", description: "Indonesian goods" }
] )
```
 - ```
> db.tiendas.createIndex({ name: "text", description: "text" })
```
  - ```
> db.tiendas.find( { $text: { $search: "java coffee shop" } } )
{ "_id" : 3, "name" : "Coffee Shop", "description" : "Just coffee" }
{ "_id" : 1, "name" : "Java Hut", "description" : "Coffee and cakes" }
{ "_id" : 5, "name" : "Java Shopping", "description" : "Indonesian goods" }
```
 - ```
> db.tiendas.find({ $text: { $search: "java shop -coffee" } })
{ "_id" : 5, "name" : "Java Shopping", "description" : "Indonesian goods" }
```
  - ```
> db.tiendas.find({name:{$regex:"Shop", $options: "Si"}})
{ "_id" : 3, "name" : "Coffee Shop", "description" : "Just coffee" }
{ "_id" : 5, "name" : "Java Shopping", "description" : "Indonesian goods" }
```







- Relaciones en MongoDB
 - Pueden ser 1:1, 1:N, N:1 o N:N
 - Embebidas o referenciadas.
- Relaciones referenciadas:
 - id_libro = ObjectId()
 - db.libro.insert({ _id:id_libro, titulo: "El Quijote", autor: "Cervantes"})
 - db.libro.insert({cod_libro: id_libro, fecha_prestamo:"10/02/2019"})
 - db.libro.insert({cod_libro: id_libro, fecha_prestamo:"20/05/2019"})
 - db.libro.insert({cod_libro: id_libro, fecha_prestamo:"10/10/2019"})
 - db.libro.aggregate([{
\$lookup:
{
from: "libro",
localField: "_id",
foreignField: "cod_libro",
as: "Prestamos"
}
}, { \$match : { titulo : "El Quijote" } }]).pretty()

Salida

```
{
  "_id" : ObjectId("5daacbf03d65c90e415f96e"),
  "titulo" : "El Quijote",
  "autor" : "Cervantes",
  "Prestamos" : [
    {
      "_id" : ObjectId("5daac913d65c90e415f970"),
      "cod_libro" : ObjectId("5daacbf03d65c90e415f96e"),
      "fecha_prestamo" : "10/02/2019"
    },
    {
      "_id" : ObjectId("5daac913d65c90e415f971"),
      "cod_libro" : ObjectId("5daacbf03d65c90e415f96e"),
      "fecha_prestamo" : "20/05/2019"
    },
    {
      "_id" : ObjectId("5daac913d65c90e415f972"),
      "cod_libro" : ObjectId("5daacbf03d65c90e415f96e"),
      "fecha_prestamo" : "10/10/2019"
    }
  ]
}
```






- Utilización de MongoDB desde Python
 - Necesario instalar la librería "pymongo"
 - Ejemplo:

```
from pymongo import MongoClient

def main():
    mongoClient = MongoClient("localhost", 27017);
    db = mongoClient.mi_base_de_datos;
    collection = db.inventario;
    cursor = collection.find({"status":"D"});
    for elemento in cursor:
        print ("%s" %(elemento['item']));

    mongoClient.close();

if __name__ == '__main__':
    main()
```



- Planes de Ejecución
 - `db.inventario3.find({ quantity: { $gte: 100, $lte: 200 } }).explain("executionStats")`

COLLSCAN

nReturned: 3

Execution Time: 0 ms

Documents Examined: 10

DETAILS

```
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    ...
    "winningPlan" : {
      "stage" : "COLLSCAN",
      ...
    },
    ...
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 3,
    "executionTimeMillis" : 0,
    "totalKeysExamined" : 0,
    "totalDocsExamined" : 10,
    "executionStages" : {
      "stage" : "COLLSCAN",
      ...
    },
    ...
  },
  ...
}
```

Se escanea la colección completa de documentos **(muy costoso)**

La consulta devuelve 3 documentos

Inexistencia de índice

Total de documentos involucrados en la consulta, donde todos se escanean **(muy costoso)**

- Planes de Ejecución
 - Creación de un índice (B-tree) sobre "quantity"
 - `db.inventario3.createIndex({ quantity: 1 })`
 - `db.inventario3.find({ quantity: { $gte: 100, $lte: 200 } }).explain("executionStats")`

FETCH

nReturned: 3

Execution Time: 0 ms

DETAILS

IXSCAN

nReturned: 3

Execution Time: 0 ms

Index Name: quantity_1

Multi Key Index: no

DETAILS

```
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    ...
    "winningPlan" : {
      "stage" : "FETCH",
      "inputStage" : {
        "stage" : "IXSCAN",
        "keyPattern" : {
          "quantity" : 1
        },
        ...
      },
      ...
    },
    "rejectedPlans" : [ ]
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 3,
    "executionTimeMillis" : 0,
    "totalKeysExamined" : 3,
    "totalDocsExamined" : 3,
    "executionStages" : {
      ...
    },
    ...
  },
  ...
}
```

Utilización de índice

Ahora sólo se examinan 3 documentos con el índice creado

23

EJERCICIO MONGODB

- Crear una colección de universidades públicas de la ciudad de Madrid que contenga los siguientes documentos:
 - Un documento para cada una de las universidades, que contenga: el nombre, la dirección postal, el teléfono, el e-mail de contacto y la página web.
 - Se definirán también el nombre y número total de estudiantes de los centros de cada una de las universidades.
- Definir consultas que permitan:
 - Mostrar todos los centros de la UAM.
 - Mostrar el total de estudiantes de cada una de las universidades.
 - Mostrar sólo aquellas universidades que tengan facultades de Derecho.



BBDD BASADAS EN GRAFOS

- Si el nuevo paradigma son aplicaciones basadas en *data lakes*, hay oportunidad para BBDD especializadas




- Las nuevas aplicaciones sociales dan origen a muchos datos en forma de red (networks) que son naturalmente representados como grafos


Bases de Datos Basadas en Grafos







- La representación y almacenado de los datos es en forma de grafo
- Se dice que es "whiteboard friendly": lo que dibujas como cajas y líneas en una pizarra lo almacenas directamente en Neo4j.
- Neo4J se centra más en las relaciones entre los datos que en los aspectos comunes entre conjuntos de datos (tales como tablas de filas o colecciones de documentos).
- Define un lenguaje propio (Cypher) para manipulación de los datos, pero existen varios lenguajes capaces de interactuar con Neo4J:
 - Java code, REST, Ruby console, Gremlin y otros.





Representamos instancias



The diagram illustrates data instances in Neo4j. It shows nodes for 'User' (screen_name: @a_ortigosa), 'User' (screen_name: @es_ncl), 'Tweet' (text: RT: La #NationalCyberleagueGC busca ...), and 'Tweet' (text: La #NationalCyberleagueGC busca ...). Relationships include 'author', 'likes', 'retweet', and 'follows'. A red dashed box highlights a specific tweet and its author, which is linked to a screenshot of a tweet from 'National Cyberleague Retos en el Ciberespacio'.






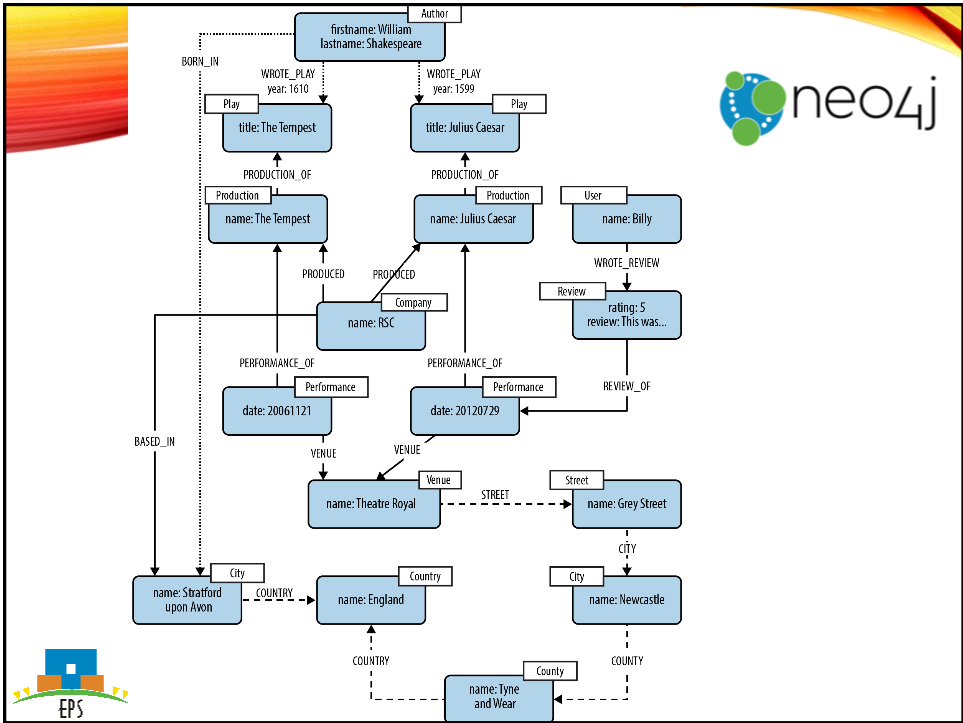
- No se representa esquema, aunque muchas veces para optimización usamos etiquetas e índices → ellos constituyen el esquema para Neo4J.
 - Pero un nodo puede tener múltiples labels.
 - Cypher incluye comandos DDL para manipular el esquema



Etiqueta (label)

Índices





EL LENGUAJE CYPHER

- Cypher busca una representación natural

Actor

nombre:
Tom
Hanks

()


Protagoniza


Película

nombre:
Forest
Gump

()

(:Actor {nombre: 'Tom Hanks'}) - [:Protagoniza] -> (:Película {nombre: 'Forest Gump'})





EJEMPLOS NEO4J (CYPHER)

CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})

CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})

CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})

CREATE (LillyW:Person {name:'Lilly Wachowski', born:1967})

CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})

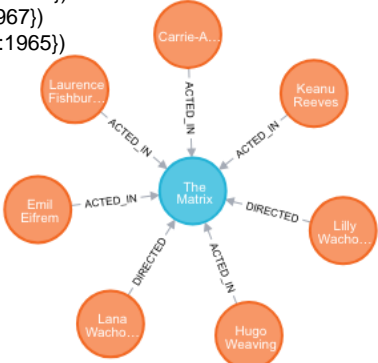
CREATE


(Keanu)-[:ACTED_IN {roles:['Neo']}]>(TheMatrix),


(Carrie)-[:ACTED_IN {roles:['Trinity']}]>(TheMatrix),

(LillyW)-[:DIRECTED]>(TheMatrix),

(LanaW)-[:DIRECTED]>(TheMatrix),







EJEMPLOS DE CYPHER


- Películas protagonizadas por 'Harrison Ford' ('Tom Hanks')


SQL

```
SELECT titulo
FROM actor, pelicula, reparto
WHERE nombre = 'Harrison Ford' AND actor.id=actor_id AND
      pelicula_id=pelicula.id;
```

Cypher

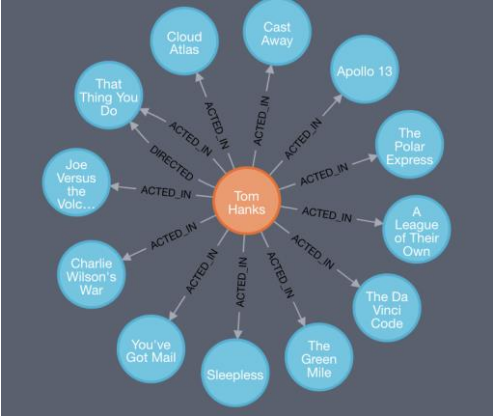
```
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->
      (tomHanksMovies) RETURN tom,tomHanksMovies
```







CYPHER - SALIDA

```
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->
      (tomHanksMovies) RETURN tom,tomHanksMovies
```







EJEMPLOS DE CYPHER


- Actores de la película 'The Matrix'


```
SELECT nombre
FROM actor, reparto, película
WHERE película.titulo='The Matrix' AND
      reparto.película_id=película.id AND
      reparto.actor_id=actor.id;
```

SQL

```
MATCH (:Movie {title:'The Matrix'}) <- [:ACTED_IN] -
      (actores) return actores
```

Cypher








CYPHER - SALIDA

```
MATCH (:Movie {title:'The Matrix'}) <- [:ACTED_IN] -
      (actores) return actores
```

"actores"
{"name": "Emil Eifrem", "born": 1978}
{"name": "Hugo Weaving", "born": 1960}
{"name": "Laurence Fishburne", "born": 1961}
{"name": "Carrie-Anne Moss", "born": 1967}
{"name": "Keanu Reeves", "born": 1964}








EJEMPLOS DE CYPHER

- Lista de actores que no actuaron con 'Tom Hanks' pero que si actuaron con otros actores que actuaron con él
- Aunque en el mundo del cine pueda no tener sentido, en sistemas de recomendación en general...
 - ¿Quién es amigo de tus amigos pero no es tu amigo (todavía)?

```
SELECT * FROM actor WHERE
NOT EXISTS
  (SELECT NULL FROM
   (SELECT pelicula_id FROM reparto WHERE reparto.actor_id = actor.id) peliId
   NATURAL JOIN
   (SELECT pelicula_id FROM reparto WHERE actor_id = (SELECT id FROM actor WHERE nombre = 'Tom Hanks'))
   peliTH)
AND EXISTS
  (SELECT NULL FROM
   (SELECT pelicula_id FROM reparto WHERE reparto.actor_id = actor.id) peliId
   NATURAL JOIN
   (SELECT pelicula_id FROM
    (SELECT id FROM actor WHERE EXISTS
      (SELECT NULL FROM
       (SELECT pelicula_id FROM reparto WHERE reparto.actor_id = actor.id) peliId NATURAL JOIN
       (SELECT pelicula_id FROM reparto WHERE actor_id =
        (SELECT id FROM actor WHERE nombre = 'Tom Hanks')) peliTH)) compisTH,
      reparto WHERE compisTH.id = reparto.actor_id) pelisCompisTH);
```



CYPHER - SALIDA

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]
-(coActors), (coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cocoActors)
WHERE NOT (tom)-[:ACTED_IN]->()<-[:ACTED_IN]-(cocoActors) AND
tom <> cocoActors
RETURN cocoActors.name
```


Table

Text

Code

cocoActors.name
"Zach Grenier"
"Zach Grenier"
"Zach Grenier"
"Cuba Gooding Jr."
"Jack Nicholson"
"Oliver Platt"
"Michael Sheen"
"Frank Langella"
"Aaron Sorkin"

Started streaming 81 records after 2 ms and completed after 31 ms.





Who Uses Neo4j?

Neo4j Powers Everyone: Enterprises & Startups Alike









