

Bases de datos noSQL

Máster en Data Science y Big Data

Miguel Ángel Corella
mcorella@geoblink.com

Marzo 2022

Contenido

1. Introducción
2. Bases de datos noSQL
3. Clave – Valor: Redis
4. Columnar: Cassandra
5. Gra
6. Referencias

Introducción

El problema de RDBMS...

- Las **bases de datos relacionales** han sido (y, en muchos casos, siguen siendo) la piedra angular del almacenamiento de datos en la inmensa mayoría de las empresas.
- Sin embargo, la evolución tecnológica tan grande que se vive desde hace años ha traído consigo **situaciones en las que estos sistemas han demostrado carencias**:
 - Volúmenes de información muy grandes.
 - Volúmenes de usuarios muy grandes.
 - Estructuras de información complejas.
 - Estructuras de información no definidas / muy cambiantes en el tiempo.
- Estas situaciones se empezaron a sufrir, inicialmente, en empresas tecnológicas que vieron crecer de forma exponencial sus necesidades de almacenamiento, infraestructura y respuesta:
 - Google, Amazon, Facebook, Twitter...

...y la solución encontrada

- Ante la situación, estas grandes compañías iniciaron el desarrollo de soluciones propietarias:
 - **Google** desarrolló **BigTable**.
 - **Amazon** desarrolló **DynamoDB**.
 - **Facebook** desarrolló **Cassandra**.
- El éxito de estas soluciones propietarias dio pie a la aparición de nuevos proyectos (privados y *open source*) que conforman el ecosistema que tenemos en la actualidad.



Bases de datos noSQL

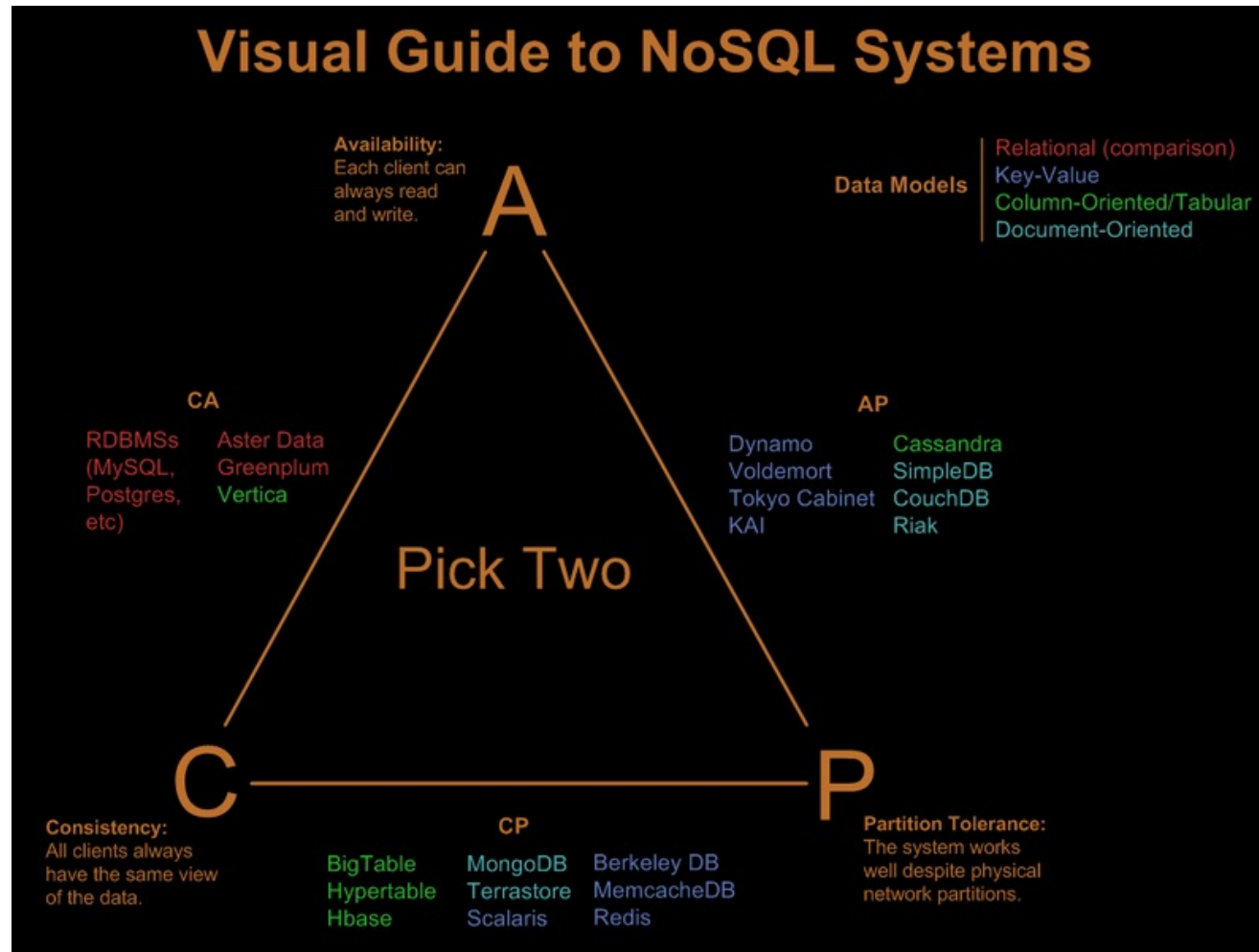
¿Qué es noSQL?

- **NoSQL** hace referencia a las siglas de “**Not Only SQL**” (y no “No SQL” como sugiere el nombre y alguna vez se ha interpretado).
- Este nuevo término se refiere a un **nuevo tipo de gestión de bases de datos** en las que:
 - **No se tiene un esquema** ni fijo ni predefinido de los datos.
 - **No se mantienen relaciones** por lo que no se permiten JOINS (o son muy costosos).
 - **No se utiliza (necesariamente) lenguaje SQL** para la realización de consultas.
 - **No se respeta** (completamente) el principio **ACID**.
 - Se tienen (generalmente) **arquitecturas distribuidas**.
 - Se prima la **flexibilidad, escalabilidad y rendimiento**.
- Es importante entender que, a la luz de estas características, **NoSQL NO ES UN SUSTITUTO DE RDBMS** sino una **alternativa necesaria y útil** en determinadas situaciones.

Teorema CAP (de Brewer)

- Al tratarse de sistemas distribuidos tenemos que tener en cuenta el Teorema CAP (de Brewer).
- Este teorema establece que es imposible para un sistema computacional distribuido ofrecer simultáneamente:
 1. **Consistency:** todos los nodos del sistema distribuido tienen la misma información en el mismo momento.
 2. **Availability:** el sistema siempre está disponible y responde a las solicitudes de forma inmediata.
 3. **Partition tolerance:** el sistema funciona correctamente aunque se pierda parte de la información almacenada en el mismo.

Teorema CAP (de Brewer)



ACID vs. BASE

Atomicity

Una transacción se ejecuta completa o no produce cambios en el sistema.

Consistency

Una transacción lleva el sistema de un estado válido a otro estado válido.

Isolation

Una transacción no compromete nunca los resultados de otra transacción concurrente.

Durability

Una transacción siempre persiste sus resultados incluso ante caídas del sistema.

Basically Available

Una transacción siempre puede ejecutarse (disponibilidad por replicación).

Soft-state

Una transacción puede dar resultados distintos en distintos momentos del tiempo.

Eventual consistency

Una transacción da resultados consistentes si el sistema no cambia durante “un tiempo”.

RDBMS vs. noSQL

Feature	NoSQL Databases	Relational Databases
Performance	High	Low
Reliability	Poor	Good
Availability	Good	Good
Consistency	Poor	Good
Data Storage	Optimized for huge data	Medium sized to large
Scalability	High	High (but more expensive)

Ventajas e inconvenientes de noSQL

Ventajas

- ✓ Permiten manejar volúmenes de información muy grandes.
- ✓ Permiten escalar los sistemas (en volumen y rendimiento) de forma horizontal.
- ✓ Permiten escalar los sistemas con hardware no muy potente y barato.
- ✓ Poseen estructuras de datos “ligeras y flexibles”.
- ✓ La mayoría de ellos son *open source*.

Inconvenientes

- ✗ No hay estándar en términos de infraestructura.
- ✗ No hay estándar en términos de estructura de datos.
- ✗ No hay estándar en términos de lenguajes de consulta.
- ✗ Son sistemas con mucha menos madurez que los RDBMS.
- ✗ Hay casos de uso de RDBMS que no se pueden soportar en NoSQL.

¿Cuándo usar sistemas noSQL?

- Cuando **tu base de datos no escala** a un coste aceptable.
- Cuando **el tamaño de tu esquema de datos ha crecido** desproporcionadamente.
- Cuando **generas muchos datos temporales** que no tiene sentido almacenar “eternamente”.
- Cuando **has desnormalizado tu base de datos para mejorar el rendimiento**.
- Cuando **trabajas con información no estructurada** (imágenes, texto, datos binarios, etc.).
- Cuando **tus consultas no implican muchas relaciones** entre los datos.
- ...

Tipos de bases de datos noSQL



Amazon DynamoDB



redis

Clave – Valor



Google
Big Query



cassandra

Columnares



mongoDB



CouchDB
relax

Documentales



neo4j



ArangoDB

Orientadas a grafos

Clave – Valor



¿Qué es una BD clave – valor?

- Son las bases de datos noSQL más simples en cuanto a arquitectura y funcionamiento.
- Se trata de bases de datos basadas en el uso de tablas hash en las que existe un único valor (simple o complejo) para cada clave única.
- Normalmente, esta tabla hash o tabla de mapeo implementa mecanismos de cache (incluso llegando a tener bases de datos “potencialmente efímeras” mantenidas 100% en memoria).
- Este tipo de bases de datos buscan maximizar el rendimiento de acceso a la información.

Modelo de datos clave – valor

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Características de redis



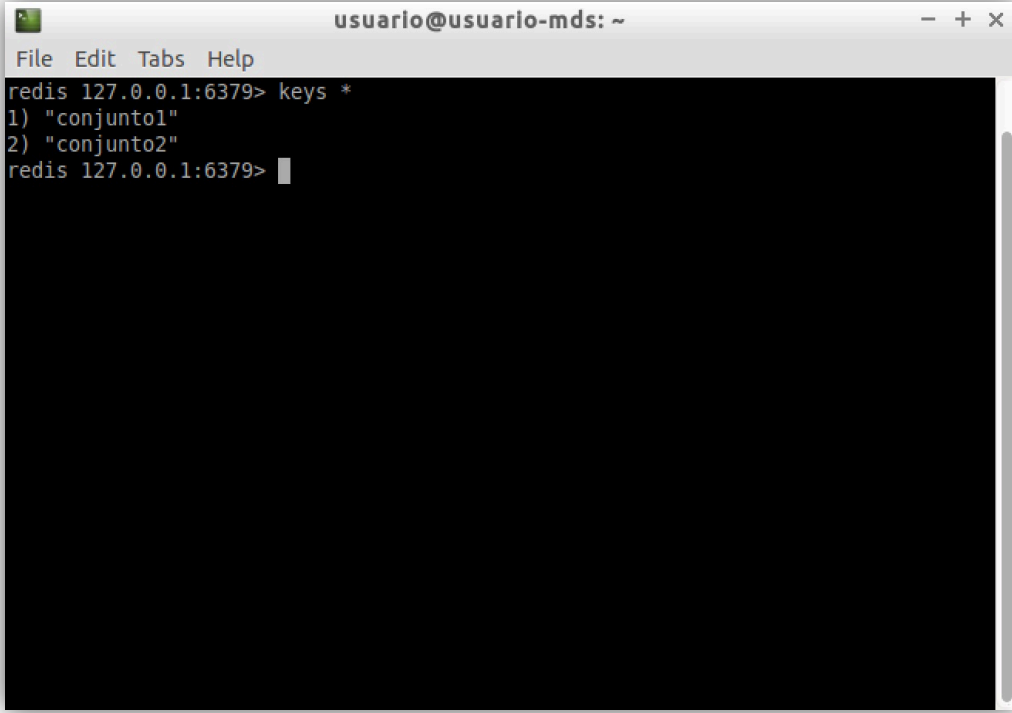
- Mantiene toda la **base de datos en memoria**.
- Únicamente se utiliza el **disco para llevar a cabo operaciones de persistencia** (backup / restore) que deben ser programadas explícitamente por el usuario.
- Al mantener toda la información en memoria, el acceso a la misma es **excepcionalmente rápido**.
- Ofrece un **conjunto muy amplio de tipos de datos** (mayor al que ofrecen otros almacenamientos clave / valor).
- **Permite** llevar a cabo la **réplica de su contenido a múltiples nodos** (esclavos) aunque no de forma automática ni directa.
- Las **operaciones** ejecutadas contra la base de datos son **SIEMPRE atómicas**.

- Desde la propia **consola cliente de redis**:
 - Es la forma “más incómoda” y “menos útil” para trabajar con redis desde el punto de vista de un sistema en producción.
- Desde una **aplicación cliente de redis**:
 - Útiles para llevar a cabo una gestión más cómoda de la base de datos, pero hay que tener en cuenta que las bases de datos en redis tienden a ser “efímeras”.
- Conectándonos desde **diferentes lenguajes de programación**:
 - Es el uso más común y extendido de redis y hay soporte para la inmensa mayoría de lenguajes de programación actuales.

Uso de redis (en Linux)



`redis-cli`

A screenshot of a terminal window titled "usuario@usuario-mds: ~". The window has a menu bar with "File", "Edit", "Tabs", and "Help". The terminal content shows a Redis command prompt "redis 127.0.0.1:6379>" followed by the command "keys *". The output is a list of two keys: "1) 'conjunto1'" and "2) 'conjunto2'". The prompt "redis 127.0.0.1:6379>" is shown again with a cursor at the end.

```
usuario@usuario-mds: ~
File Edit Tabs Help
redis 127.0.0.1:6379> keys *
1) "conjunto1"
2) "conjunto2"
redis 127.0.0.1:6379> 
```

1. **Strings:** es el tipo de dato más básico que nos permite almacenar redis. Cualquier valor asociado a una clave es SIEMPRE una cadena de caracteres (incluso los números).
2. **Hashes:** se trata de colecciones de tipo clave / valor en las que tanto los campos como los valores son estructuras de tipo String. Pueden verse como diccionarios de Python.
3. **Lists:** se trata de colecciones ordenadas (por orden de inserción) de Strings.
4. **Sets:** se trata de colecciones desordenadas de Strings en las que cada elemento sólo puede aparecer una única vez.
5. **Sorted sets:** se trata de colecciones ordenadas (en base a una puntuación interna) de Strings en las que cada elemento sólo puede aparecer una única vez.

Comandos generales



Comando	Descripción
KEYS pattern	Muestra el listado de claves coincidentes con el patrón solicitado
TYPE key	Recupera el tipo asociado a la clave seleccionada
EXISTS key	Chequea si una clave ya existe en la BD
DEL key	Elimina una clave si ésta ya existe en la BD
FLUSHDB	Elimina TODAS las claves de la BD
(P)EXPIRE key seconds	Establece una caducidad para una clave en (mili)segundos
(P)TTL key	Muestra el tiempo restante que le queda a una clave en (mili)segundos
RENAME(NX) key newkey	Renombra (si no existe el nuevo nombre) la clave seleccionada

Juguemos un poco...



01-redis-general.txt

Comandos sobre Strings



Comando	Descripción
[M]SET(NX) key value [key value]	Establece el valor de la[s] clave[s] seleccionadas[s] (sólo si no existe)
[M]GET key [key]	Recupera el valor de la[s] clave[s] seleccionada[s]
STRLEN key	Recupera la longitud del valor de la clave seleccionada
GETRANGE key start end	Devuelve una subcadena de la clave seleccionada (base 0)
APPEND key value	Concatena el valor a la clave seleccionada
INCR key	Incrementa el valor numérico de la clave seleccionada en 1
INCRBY key increment	Incrementa el valor numérico de la clave seleccionada
DECR key	Decrementa el valor numérico de la clave seleccionada en 1

Juguemos un poco...



02-redis-strings.txt

Comandos sobre Hashses



Comando	Descripción
H[M]SET(NX) key field value [field value]	Elimina el campo seleccionado de la clave seleccionada
HGET key field	Chequea si existe el campo en la clave seleccionada
HDEL key field	Establece el valor de un[varios] campos (si no existen)
HEXISTS key field	Recupera el valor del campo seleccionado
HGETALL key	Recupera el valor de todos los campos de la clave
HLEN key	Recupera el número de campos en la clave
HKEYS key	Recupera los nombres de los campos en la clave
HVALS key	Recupera los valores de los campos en la clave

Juguemos un poco...



03-redis-hashes.txt

Comandos sobre Lists



Comando	Descripción
LSET key index value	Añade un elemento a la lista por la izquierda o por la derecha
L RPUSH key value	Establece el valor de la posición indicada de la lista
LLEN key	Recupera el número de elementos de la lista
LRANGE key start stop	Recupera un rango de la lista (base 0)
LINDEX key index	Recupera el elemento en la posición indicada de la lista
LINSERT key BEFORE AFTER pivot value	Inserta un elemento en la lista utilizando otro como pivote
LREM key count value	Elimina un elemento de la lista
L RPOP key	Extrae un elemento de la lista por la izquierda o por la derecha

Juguemos un poco...



04-redis-lists.txt

Comandos sobre Sets



Comando	Descripción
SADD key member [member]	Añade elementos a un conjunto
SCARD key	Recupera el número de elementos del conjunto
SISMEMBER key member	Chequea si un elemento pertenece a un conjunto
SMEMBERS key	Recupera todos los elementos de un conjunto
SREM key member [member]	Elimina elementos de un conjunto
SUNION(STORE key) key1 key2	Une las claves seleccionadas (y almacena)
SDIFF(STORE key) key1 key2	Hace la diferencia de las claves seleccionadas (y almacena)
SINTER(STORE key) key1 key2	Hace la intersección de las claves seleccionadas (y almacena)

Juguemos un poco...



05-redis-sets.txt

Comandos sobre Sorted Sets



Comando	Descripción
ZADD key score member [score member]	Añade elementos al conjunto con puntuación de orden
ZCARD key	Recupera el número de elementos en el conjunto
ZRANGE key start stop [WITHSCORES]	Recupera un rango de elementos del conjunto (base 0)
ZRANGEBYSCORE key start stop [WITHSCORES]	Recupera un rango de puntuaciones del conjunto
ZRANK key member	Recupera la posición de un elemento
ZSCORE key member	Recupera la puntuación de orden de un elemento
ZREM key member [member]	Elimina elementos de un conjunto
ZINCRBY key increment member	Incrementa el valor de la puntuación de un elemento

Juguemos un poco...



06-redis-sorted-sets.txt

Uso de redis desde R y Python



- Comunicación con redis desde R.
 - Paquete **rredis**.
 - Disponible en CRAN.
 - <https://cran.r-project.org/web/packages/rredis/index.html>
- Comunicación con redis desde Python.
 - Paquete / Módulo **redis**.
 - Disponible en PyPI (pip) y Anaconda (conda).
 - <https://pypi.python.org/pypi/redis>

Juguemos un poco...



redis-r.R / redis-python.py

- Mantenimiento de información de sesión.
 - Información de usuario.
 - Carritos de la compra.
- Publicación / Suscripción:
 - Chats.
 - Comunicación LIFO / FIFO entre componentes de un sistema.
- Contadores:
 - Tablas de clasificación.
 - Rankings.

Ejercicio – redis desde Python



- Escribir una función **login** que reciba el nombre de un usuario como parámetro.
- Si el usuario no existe en la BD de redis, la función deberá:
 - Crear una clave de tipo Hash con el nombre **usuario:<nombre>**
 - Añadir a dicha clave un campo **last_login** con el timestamp actual.
 - Añadir a dicha clave un campo **num_logins** con un valor de 1.
- Si el usuario existe en la BD de redis, la función deberá:
 - Actualizar el campo **last_login** del usuario con el timestamp actual.
 - Incrementar el valor del campo **num_logins** en 1.

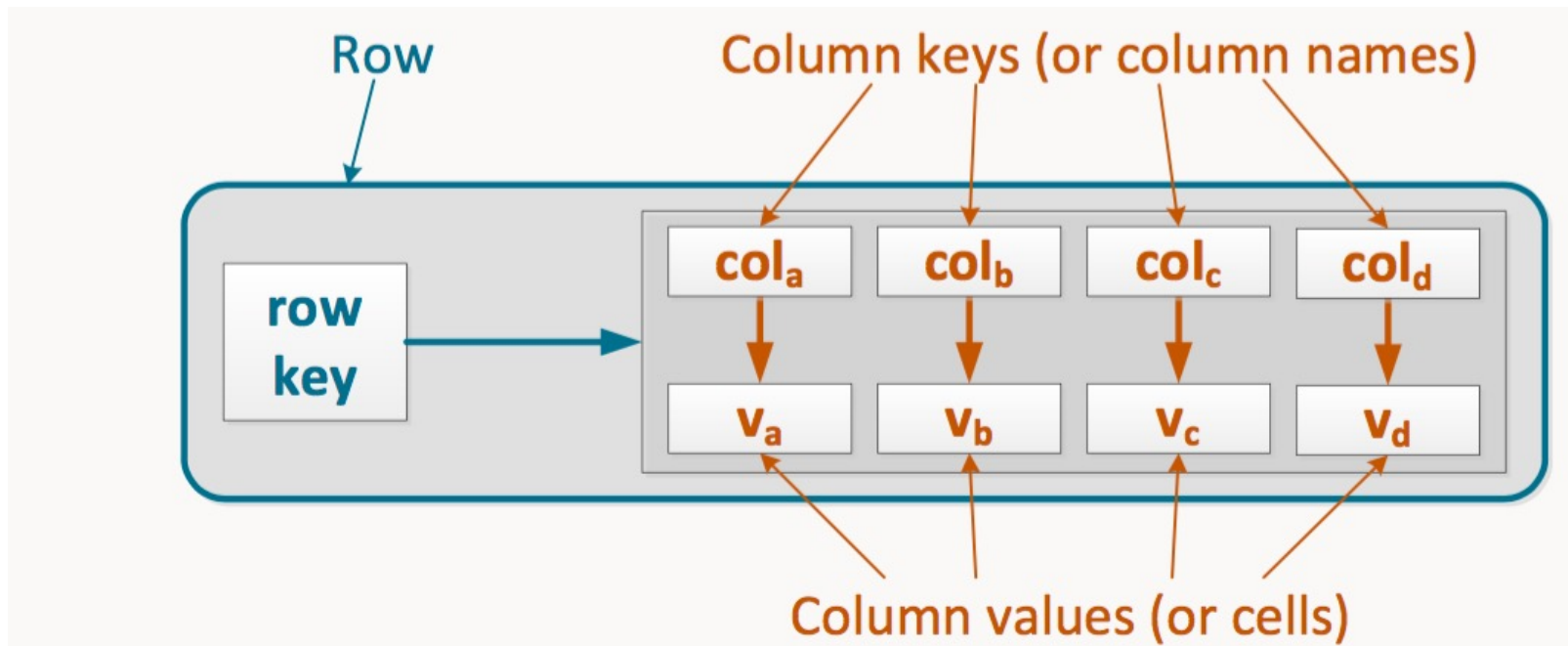
Columnares



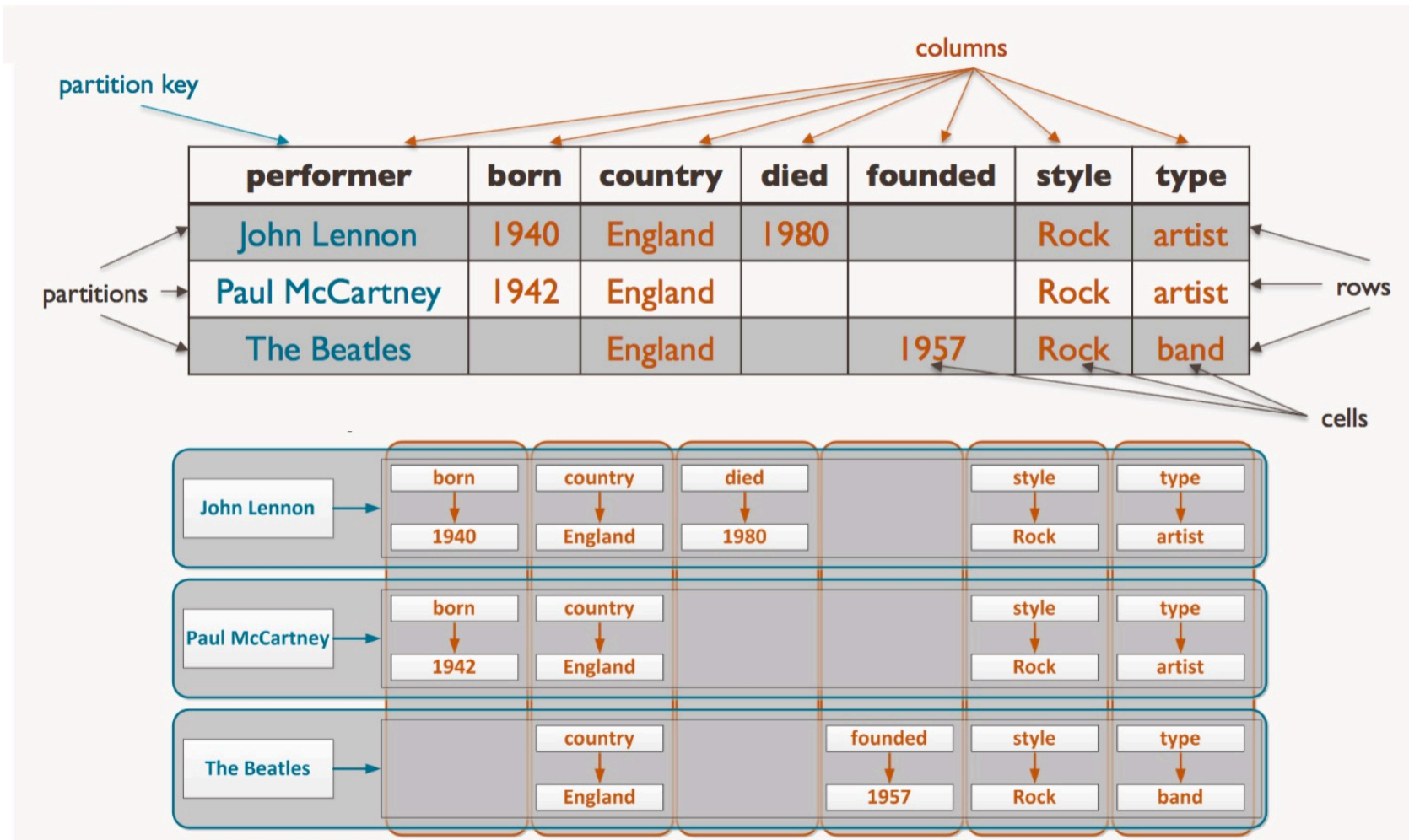
¿Qué es una BD columnar?

- Se trata de bases de datos en las que los datos están organizados por columnas en lugar de por filas como ocurre en los sistemas de bases de datos relacionales.
- La relación con las bases de datos relacionales es directa y las diferencias, desde un punto de vista práctico, son casi nulas.
- Sin embargo, está “reorganización de los datos” permite:
 - Estructura variable: No todas las columnas tienen por qué tener valor para todas las filas.
 - Incremento en rendimiento de lectura: Se accede de forma directa a las columnas seleccionadas en queries en lugar de “descartar” información.
 - Incremento en rendimiento de escritura: Se introducen valores únicamente en las columnas seleccionadas.

Modelo de datos columnar



Modelo de datos columnar



Características de Cassandra



- **Altísima escalabilidad horizontal** que permite la adición de hardware de forma “casi transparente” al sistema.
- **Arquitectura always on** que permite disponer de un sistema sin ningún punto crítico de fallo lo que la hace ideal para entornos críticos donde no se pueden admitir caídas.
- **Estructuras de datos completamente flexibles** que permiten introducir variaciones “al vuelo” sin apenas coste.
- **Robusto sistema de réplica de información** a lo largo del cluster lo que permite al usuario abstraerse de su gestión.
- Funcionamiento **basado en transacciones** y “casi” cumplimiento del **principio ACID**.
- **Optimización de escrituras y lecturas** “masivas”.

Características de Cassandra



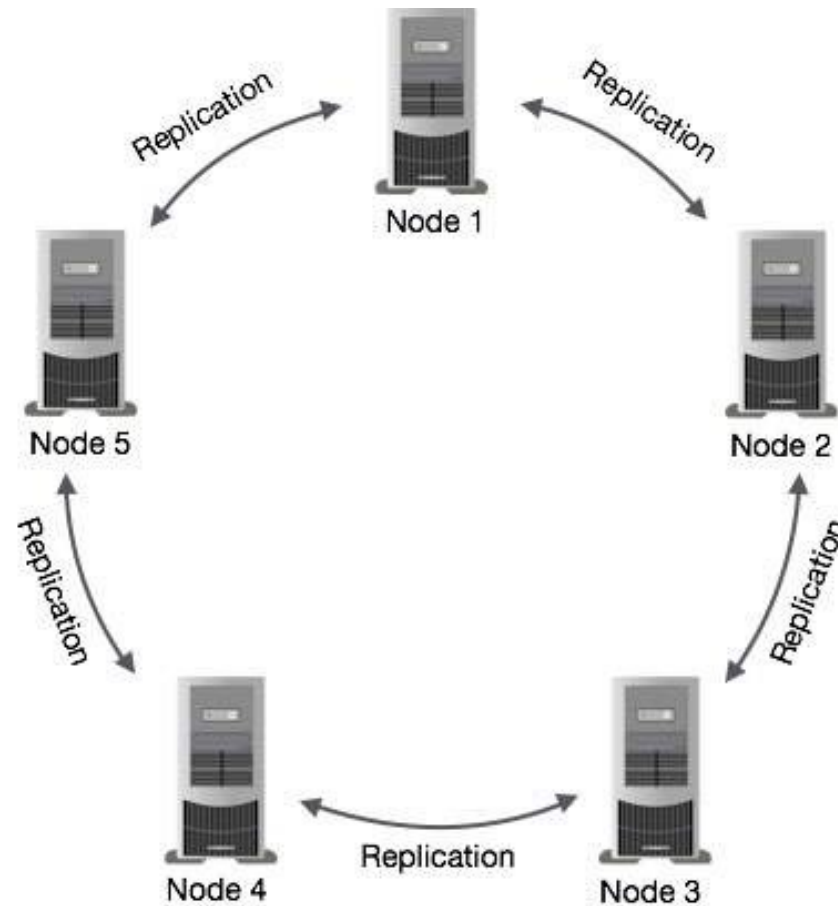
Arquitectura always on

- La arquitectura de Cassandra se basa en tres principios fundamentales:
 1. TODOS los nodos de un cluster son idénticos y cumplen el mismo rol.
 2. TODOS los nodos de un cluster pueden recibir peticiones de lectura y escritura de forma independiente a donde estén los datos.
 3. TODOS los nodos son capaces de solicitar réplicas de la información que mantienen.
- Esto hace que nos encontremos con una arquitectura que **no mantiene ningún tipo de jerarquía maestro / esclavo** entre sus nodos.
- De esta forma **NO EXISTE un único punto de fallo** y el sistema siempre estará disponible (salvo caída masiva de múltiples nodos).

Características de Cassandra



Arquitectura always on



Modelo de datos de Cassandra



- El modelo de datos de Cassandra es muy similar al que nos encontramos en bases de datos relacionales. Sus componentes principales son:
 - **Cluster:** conjunto de nodos que componen una instancia de Cassandra.
 - **Keyspace:** espacio de nombres único bajo el que englobamos un conjunto de tables/column families.
 - **Table/Column family:** conjunto de columnas englobadas bajo un nombre / clave común.
 - **Column:** estructura básica compuesta por un nombre, un valor y un timestamp.

Modelo de datos de Cassandra



- Dada su similitud, podemos encontrar una equivalencia directa entre los componentes de Cassandra y los de un modelo de datos relacional...

Modelo relacional	Modelo en Cassandra
Cluster	Instance
Keyspace	Database
Table/Column family	Table
Column	Column

Modelo de datos de Cassandra



- Aunque el funcionamiento interno de estos componentes hace que tengamos que tener en cuenta algunas diferencias muy relevantes.
- **No hay JOINS:**
 - No se puede llevar a cabo el cruce de información entre dos tablas. Se debe almacenar la información de forma desnormalizada y se recomienda crear tablas específicas para dar solución a consultas específicas.
- **No hay ordenación ni agrupación en tiempo de consulta:**
 - Al primar la velocidad de respuesta y no existir un “nodo maestro” no se puede llevar a cabo la ordenación de datos en el cluster ni la agrupación de los mismos.
- **No trabajamos con SQL para la realización de consultas y creación de estructuras.**
 - Desde versiones muy tempranas de Cassandra disponemos de CQL que es prácticamente equivalente.

Interacción con Cassandra



- Desde la propia **consola cliente de Cassandra**:
 - Es la forma “más incómoda” y “menos útil” para trabajar con Cassandra desde el punto de vista de un sistema en producción.
- Desde una **aplicación cliente de Cassandra**:
 - Útiles para llevar a cabo una gestión más cómoda de la base de datos al igual que haríamos en cualquier base de datos relacional.
- Conectándonos desde **diferentes lenguajes de programación**:
 - Es el uso más común y extendido de Cassandra y hay soporte para la inmensa mayoría de lenguajes de programación actuales.

Uso de Cassandra (en Linux)



cqlsh

```
usuario@usuario-mds: ~  
File Edit Tabs Help  
usuario@usuario-mds:~$ cqlsh  
Connected to Test Cluster at 127.0.0.1:9042.  
[cqlsh 5.0.1 | Cassandra 2.2.12 | CQL spec 3.3.1 | Native protocol v4]  
Use HELP for help.  
cqlsh> |
```

Gestión de keyspaces



Creación de keyspaces

CREATE KEYSPACE <keyspace_name> **WITH** <properties>

- **<keyspace_name>**: nombre del keyspace que queremos crear.
- **<properties>**: conjunto de propiedades que queremos establecer para el keyspace.
 - Es obligatorio definir la propiedad de replicación (*replication*).
 - Indicando el tipo de replicación (*class*)...
 - El más básico es SimpleStrategy.
 - ... y el factor de replicación (*replication_factor*)

```
CREATE KEYSPACE test
WITH replication = {'class' : 'SimpleStrategy',
'replication_factor': 3}
```

Gestión de keyspaces



Listado y selección de keyspaces

DESCRIBE keyspaces

USE <keyspace_name>

- **<keyspace_name>**: nombre del keyspace que queremos seleccionar.

Gestión de keyspaces



Eliminación de keyspaces

DROP KEYSPACE <keyspace_name>

- **<keyspace_name>**: nombre del keyspace que queremos eliminar.

IMPORTANTE: se eliminará el keyspace y todo su contenido. Esta acción no se puede deshacer.

Juguemos un poco...



01-cassandra-keyspaces.txt

Creación de tablas

```
CREATE TABLE <table_name> (  
    <column_name> <column_type>,  
    ...,  
    PRIMARY KEY (column_name)  
);
```

- **<table_name>**: nombre de la tabla que queremos crear.
- **<column_name>**: nombre de la columna que queremos añadir.
- **<column_type>**: tipo de dato asociado a la columna.

IMPORTANTE: la asignación de una clave primaria es OBLIGATORIA.

Gestión de tablas



Tipos de datos

Tipo de dato	Contenido	Descripción
int	integers	Represents 32-bit signed int
bigint	integers	Represents 64-bit signed long
float	integers, floats	Represents 32-bit IEEE-754 floating point
double	integers	Represents 64-bit IEEE-754 floating point
decimal	integers, floats	Represents variable-precision decimal
boolean	booleans	Represents true or false
text	strings	Represents UTF8 encoded string
inet	strings	Represents an IP address, IPv4 or IPv6
blob	blobs	Represents arbitrary bytes
timestamp	integers, strings	Represents a timestamp
text	strings	Represents UTF8 encoded string

Gestión de tablas



Listado y descripción de tablas

DESCRIBE tables

DESCRIBE <table_name>

- **<table_name>**: nombre de la tabla que queremos describir.

Modificación de tablas

ALTER TABLE <table_name>

ADD COLUMN <column_name> <column_type>

ALTER TABLE <table_name>

DROP COLUMN <column_name>

- **<table_name>**: nombre de la tabla que queremos modificar.
- **<column_name>**: nombre de la columna que queremos añadir/eliminar.
- **<column_type>**: tipo de dato de la columna que queremos añadir.

Vaciado / Eliminación de tablas

TRUNCATE TABLE <table_name>

DROP TABLE <table_name>

- **<table_name>**: nombre de la tabla que queremos vaciar / eliminar.

IMPORTANTE: se eliminará la tabla y todo su contenido. Esta acción no se puede deshacer.

Juguemos un poco...



02-cassandra-tables.txt

Creación de índices

```
CREATE INDEX <index_name>  
ON <table_name> (<column_name>)
```

- **<index_name>**: nombre del índice que queremos crear.
- **<table_name>**: nombre de la tabla sobre la que queremos crear el índice.
- **<column_name>**: nombre de la columna sobre la que queremos crear el índice.

IMPORTANTE: Cassandra sólo permite índices sobre columnas individuales.

Gestión de índices



Eliminación de índices

DROP INDEX <index_name>

- **<index_name>**: nombre del índice que queremos eliminar.

Operaciones CRUD



Inserción de datos (Create)

```
INSERT INTO <table_name>  
(column_name_1, ...)  
VALUES  
(column_value_1, ...)
```

- **<table_name>**: nombre de la tabla en la que queremos insertar datos.
- **<column_name>**: nombre de la columna en la que queremos insertar datos.
- **<column_value>**: valor que queremos insertar en la columna.

Operaciones CRUD



Recuperación de datos (Retrieve)

```
SELECT <column_name_1>, ... | *  
FROM <table_name>  
[WHERE <conditions>]  
[ALLOW FILTERING]
```

- **<column_name>**: nombre de la columna que se quiere recuperar.
- **<table_name>**: nombre de la tabla que se quiere recuperar.
- **<conditions>**: múltiples expresiones booleanas (AND | OR) para filtrar las filas a devolver en la consulta.

Operaciones CRUD



Actualización de datos (Update)

```
UPDATE <table_name> SET  
    <column_name_1> = <column_value_1>,  
    ...  
[WHERE <conditions>]
```

- **<table_name>**: nombre de la tabla en la que queremos actualizar datos.
- **<column_name>**: nombre de la columna en la que queremos actualizar datos.
- **<column_value>**: valor que queremos actualizar en la columna.
- **<conditions>**: múltiples expresiones booleanas (AND | OR) para filtrar las filas a devolver en la consulta.

IMPORTANTE: Cassandra sólo permite actualizar filtrando por clave primaria.

Operaciones CRUD



Eliminación de datos (Delete)

```
DELETE FROM <table_name>  
[WHERE <conditions>]
```

- **<table_name>**: nombre de la tabla que se quiere recuperar.
- **<conditions>**: múltiples expresiones booleanas (AND | OR) para filtrar las filas a devolver en la consulta.

IMPORTANTE: se eliminarán las filas seleccionadas. Esta acción no se puede deshacer.

Juguemos un poco...



03-cassandra-crud.txt

Uso de Cassandra desde R y Python



- Comunicación con Cassandra desde R.
 - Paquete **RCassandra** o **RJDBC**.
 - Disponibles ambos en CRAN.
 - Funcionalidad muy limitada.
 - <https://cran.r-project.org/web/packages/RCassandra/index.html>
- Comunicación con Cassandra desde Python.
 - Paquete / Módulo **cassandra-driver**.
 - Disponible en PyPI (pip).
 - <https://pypi.org/project/cassandra-driver/>

Juguemos un poco...



cassandra-python.py

Casos de uso de Cassandra



- Escalado de (algunos) sistemas de base de datos relacionales.
 - Por similitud es la evolución más sencilla.
 - ¡OJO! No son completamente equivalentes.
- Almacenamiento y recuperación de logs.
- Detección de fraude.
- Almacenamiento de información de sensores / Internet of Things.
- Playlists.
- Catálogos de productos.

Ejercicio – Cassandra desde Python



- Crea un nuevo keyspace de nombre **airbnb**.
- Crea una tabla **airbnb_offer** en el keyspace anterior siguiendo la estructura del fichero **airbnb.csv**.
- Escribe un programa que cargue una a una las filas del fichero **airbnb.csv** en la nueva tabla.
- Ejecuta, desde Python, las siguientes consultas (crea los índices/tablas que consideres necesarios):
 - Ofertas con más de 330 días de disponibilidad.
 - Número de ofertas ubicadas en Manhattan.
 - Ofertas del tipo Private room ubicadas en Brooklyn.

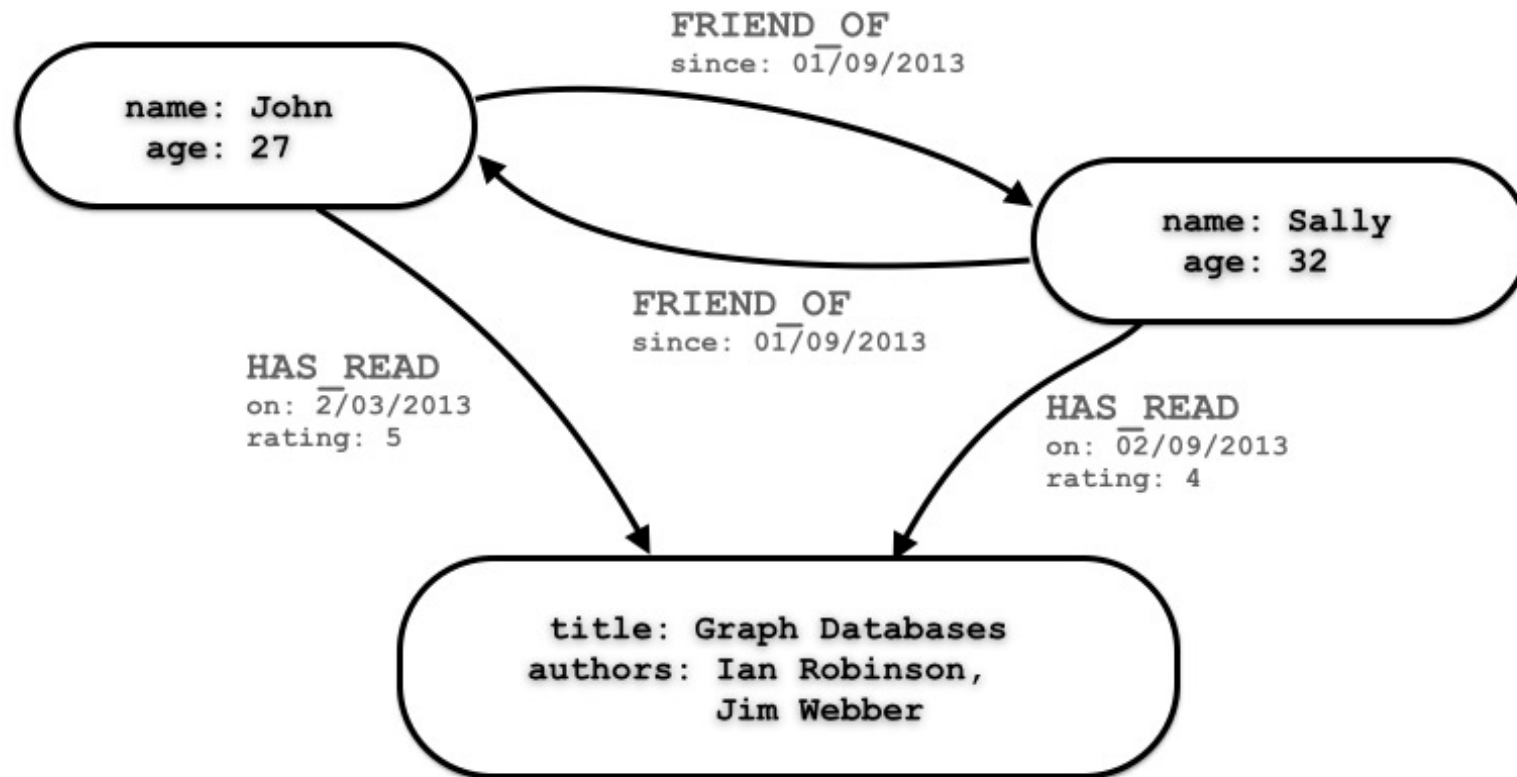
Orientadas a grafos



¿Qué es una BD orientada a grafos?

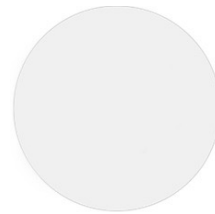
- Es una base de datos en la que la información está estructurada como un conjunto de nodos / objetos y aristas / relaciones.
- Las relaciones establecidas entre los nodos siempre están dirigidas, si bien estas relaciones pueden ser bidireccionales.
- Generalmente este tipo de bases de datos se utilizan para capturar información sobre redes (amigos, seguidores, contactos, etc.).
- En estas situaciones es más importante la explotación de las relaciones entre los objetos que en los valores asociados a los propios objetos.

Modelo de datos orientado a grafos



- Estructura de datos basada en la construcción de una **red / grafo dirigido (o no) de objetos y relaciones (cada uno de ellos con propiedades asociadas)**.
- **Cumple el principio ACID** (mientras no se realice una instalación distribuida).
- Define su **lenguaje de consulta propio** orientado a grafos: **Cypher Query Language (CQL)** → No confundir con Cassandra Query Language (CQL)
- **Entorno de trabajo Web** completamente **integrado** en la instalación del servidor de Neo4j.
- Sólo se puede tener **UNA base de datos por servicio** → Diferentes servicios en diferentes puertos.

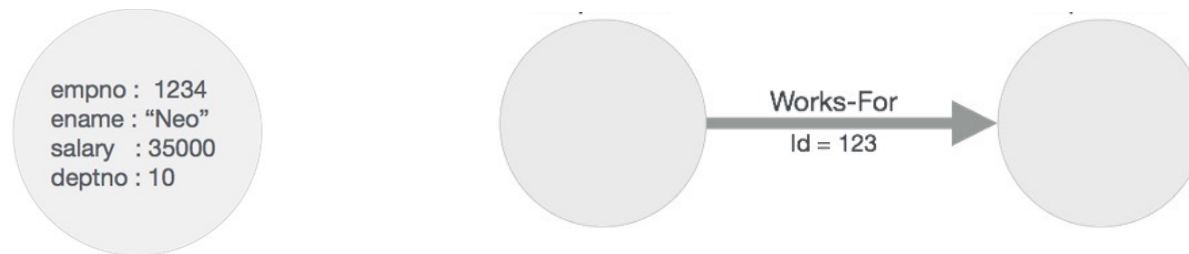
- **Nodo:** unidad fundamental de un grafo. Representa un objeto concreto dentro de nuestra base de datos.



- **Relación:** aristas de nuestro grafo, conectan dos nodos de forma dirigida y representan la presencia de una interacción entre dos objetos / nodos de nuestra base de datos.



- **Propiedad:** se trata de un par clave / valor que puede incorporar información adicional a cualquiera de los elementos de nuestro grafo, tanto nodos, como relaciones.



- **Etiqueta:** se trata de un nombre que permite agrupar nodos y relaciones que poseen características similares (es decir, el "tipo" de nodo / relación). Puede ser múltiple.



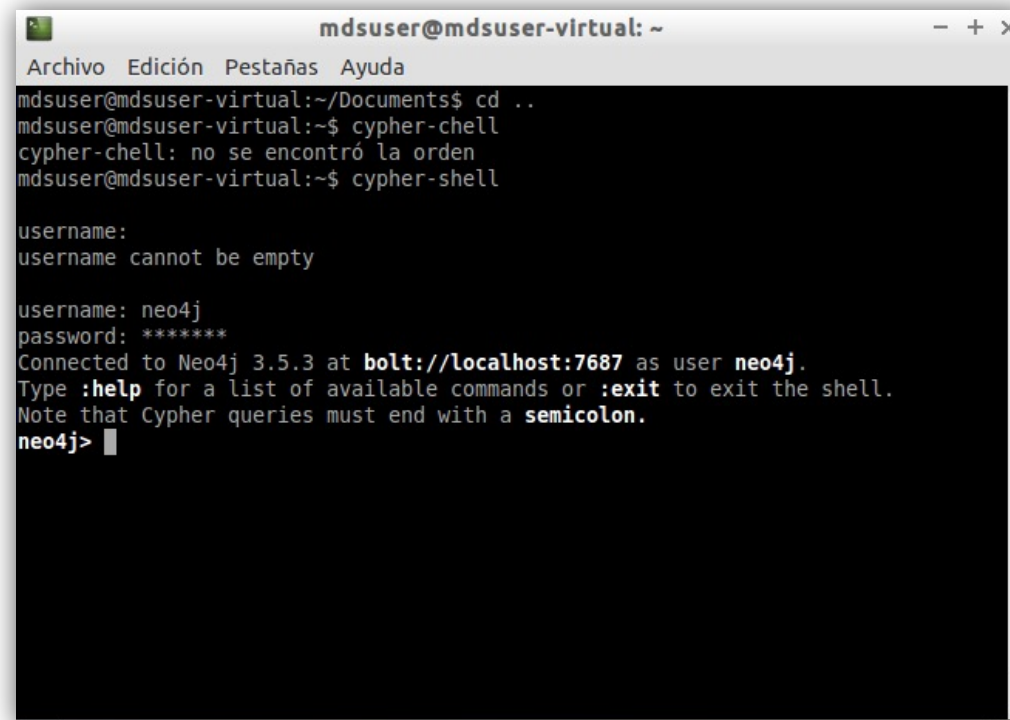
Modelo de datos de neo4j



RDBMS	Neo4j
Database	Database
Table	Graph
Row	Node
Column	Property
Constraints	Relationships
mysqld / oracle	neo4j
mysql / sql plus	cypher-shell
Workbench / Toad	Neo4j Browser

- Desde la propia **consola de cliente de neo4j**:
 - Es la forma “más incómoda” y “menos útil” para trabajar con neo4j desde el punto de vista de un sistema en producción.
- Desde el navegador mediante la **aplicación propia de neo4j**:
 - Útil para llevar a cabo una gestión más cómoda de la base de datos al igual que haríamos con cualquier base de datos relacional.
- Conectándonos desde **diferentes lenguajes de programación**:
 - Es el uso común para llevar a cabo la explotación de bases de datos en neo4j y se disponen de drivers para los principales lenguajes de programación.

cypher-shell

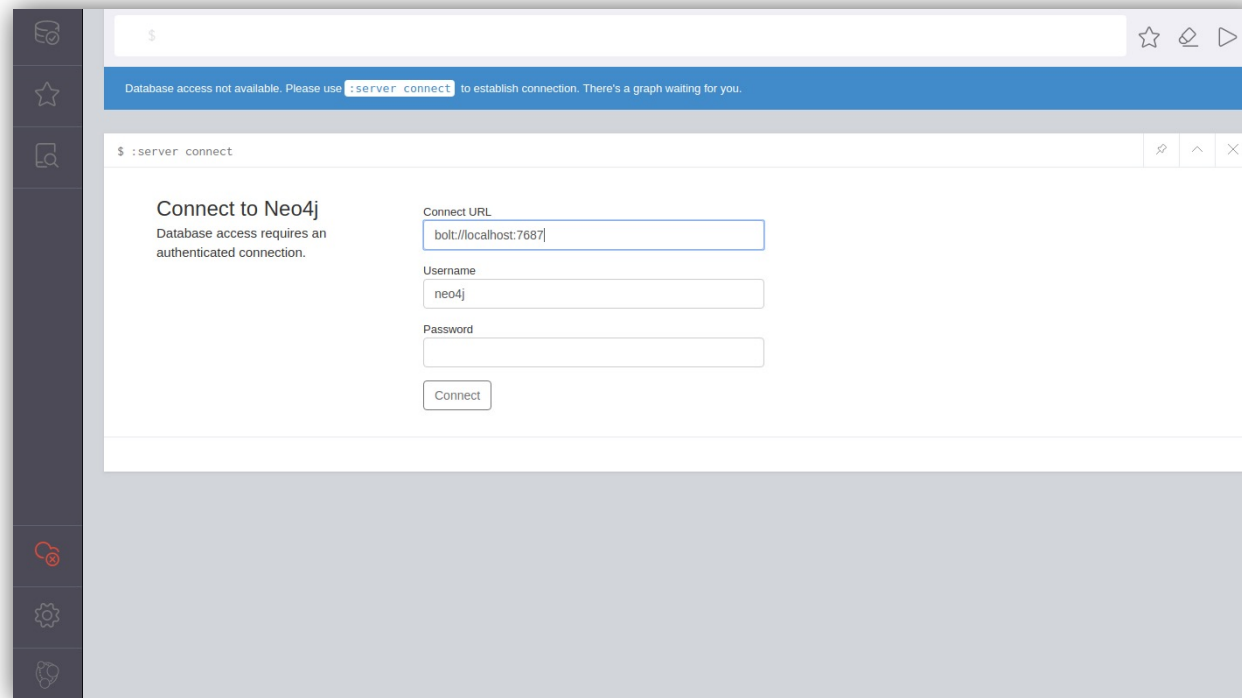


```
mdsuser@mdsuser-virtual: ~  
Archivo Edición Pestañas Ayuda  
mdsuser@mdsuser-virtual:~/Documents$ cd ..  
mdsuser@mdsuser-virtual:~$ cypher-shell  
cypher-shell: no se encontró la orden  
mdsuser@mdsuser-virtual:~$ cypher-shell  
  
username:  
username cannot be empty  
  
username: neo4j  
password: *****  
Connected to Neo4j 3.5.3 at bolt://localhost:7687 as user neo4j.  
Type :help for a list of available commands or :exit to exit the shell.  
Note that Cypher queries must end with a semicolon.  
neo4j> █
```

Uso de neo4j (en Linux)



http://localhost:7474

A screenshot of the Neo4j web interface. The browser address bar shows a dollar sign (\$). A blue banner at the top reads: "Database access not available. Please use :server connect to establish connection. There's a graph waiting for you." Below this, a terminal-like window titled "\$:server connect" contains the text "Connect to Neo4j" and "Database access requires an authenticated connection." To the right of this text are three input fields: "Connect URL" with the value "bolt://localhost:7687", "Username" with the value "neo4j", and "Password" which is empty. A "Connect" button is located below the password field. The left sidebar of the interface contains several icons: a folder, a star, a document, a red circular icon with a white dot, a gear, and a shield.

Uso de neo4j (en Linux)



User: neo4j
Password: neo4j

Connect to Neo4j
Database access requires an authenticated connection.

Connect URL

Username

Password

Connect to Neo4j
Database access requires an authenticated connection.

New password

Repeat new password

Operaciones CRUD



Inserción de nuevos nodos (Create)

```
CREATE (  
    [<node_name>][:<label>...], ...  
    [{<property>: <property_value>, ... }]  
);
```

- **<node_name>**: identificador “temporal” del nodo.
- **<label>**: etiqueta que queremos asociar al nodo.
- **<property>**: nombre de la propiedad que queremos asignar al nodo.
- **<property_value>**: valor de la propiedad que queremos asignar al nodo.

Juguemos un poco...



01-neo4j-create-nodes.txt

Inserción de relaciones entre nuevos nodos (Create)

CREATE

```
(<node_definition_1>
-[:<relationship_type> [{<property>:<property_value>, ...}]]->
(<node_definition_2>);
```

- **<node_definition>**: definición de los nodos a relacionar.
- **<relationship_type>**: etiqueta que queremos asociar a la relación.
- **<property>**: nombre de la propiedad que queremos asignar a la relación.
- **<property_value>**: valor de la propiedad que queremos asignar a la relación.

Inserción de relaciones entre nodos existentes (Create)

MATCH

```
(<node_name_1>:<node_definition>),  
(<node_name_2>:<node_definition>)
```

CREATE

```
(<node_name_1>  
-[:<relationship_type> [{<property>:<property_value>, ...}]]->  
(<node_name_2>);
```

- **<node_name>**: identificador “temporal” del nodo.
- **<node_definition>**: definición de los nodos a relacionar.
- **<relationship_type>**: etiqueta que queremos asociar a la relación.
- **<property>**: nombre de la propiedad que queremos asignar a la relación.
- **<property_value>**: valor de la propiedad que queremos asignar a la relación.

Juguemos un poco...



02-neo4j-create-relationships.txt

Inserción de caminos de nuevos nodos (Create)

CREATE

```
(<node_definition>)  
-[:<relationship_type> [{<property>:<property_value>, ...}]]->  
(<node_definition>)  
-[:<relationship_type> [{<property>:<property_value>, ...}]]->  
(<node_definition>);
```

- **<node_definition>**: definición de los nodos a relacionar.
- **<relationship_type>**: etiqueta que queremos asociar a la relación.
- **<property>**: nombre de la propiedad que queremos asignar a la relación.
- **<property_value>**: valor de la propiedad que queremos asignar a la relación.

Inserción de caminos sobre nodos existentes (Create)

MATCH

```
(<node_name_1>:<node_definition>),  
(<node_name_2>:<node_definition>),  
(<node_name_3>:<node_definition>),
```

CREATE

```
(<node_name_1>  
-[:<relationship_type> [{<property>:<property_value>, ...}]]->  
(<node_name_2>  
-[:<relationship_type> [{<property>:<property_value>, ...}]]->  
(<node_name_3>);
```

- **<node_name>**: identificador “temporal” del nodo.
- **<node_definition>**: definición de los nodos a relacionar.
- **<relationship_type>**: etiqueta que queremos asociar a la relación.
- **<property>**: nombre de la propiedad que queremos asignar a la relación.
- **<property_value>**: valor de la propiedad que queremos asignar a la relación.

Juguemos un poco...



03-neo4j-create-paths.txt

Operaciones CRUD



Recuperación básica de elementos (Retrieve)

MATCH <pattern> **RETURN** <pattern_elements>;

- **<pattern>**: Patrón de elementos del grafo que queremos encontrar.
- **<pattern_elements>**: Elementos concretos del patrón que queremos devolver.

Operaciones CRUD



Patrones de recuperación de nodos (Retrieve)

MATCH (node) **RETURN** node;

Todos los nodos del grafo

MATCH (node:Label) **RETURN** node;

Todos los nodos con una determinada etiqueta

MATCH (node {property: value}) **RETURN** node;

Todos los nodos con un determinado valor de propiedad

MATCH (node {property: value}) **RETURN** node **LIMIT** 10;

Recuperación de nodos limitando el número de resultados

Operaciones CRUD



Patrones de recuperación de nodos (Retrieve)

MATCH (node) **RETURN** labels(node);

Etiquetas de los nodos

MATCH (node) **RETURN** node.property;

Valores de la propiedad de todos los nodos

MATCH (node) **RETURN** node.property as alias;

Valores de la propiedad devueltos con un alias

Juguemos un poco...



04-neo4j-retrieve-nodes.txt

Operaciones CRUD



Patrones de recuperación de relaciones (Retrieve)

MATCH ()-[rel]-() RETURN rel;

Todas las relaciones del grafo

MATCH ()-[rel]->() RETURN rel;

Todas las relaciones con una determinada dirección

MATCH ()-[rel:type] RETURN rel;

Todas las relaciones con un determinado tipo

MATCH ()-[rel {property: value}] RETURN rel;

Todas las relaciones con un determinado valor de propiedad

Patrones de recuperación de relaciones (Retrieve)

MATCH ()-[rel]-() **RETURN** rel **LIMIT** 10;

Recuperación de relaciones limitando el número de resultados

MATCH ()-[rel]-() **RETURN** type(rel);

Tipos de las relaciones

MATCH ()-[rel]-() **RETURN** rel.property;

Valores de la propiedad de todos los nodos

MATCH ()-[rel]-() **RETURN** rel.property as alias;

Valores de la propiedad devueltos con un alias

Juguemos un poco...



05-neo4j-retrieve-relationships.txt

Patrones de recuperación de caminos (Retrieve)

MATCH (node1)--(node2) **RETURN** node1, node2;

Pares de nodos directamente relaciones

MATCH (node1)-->(node2) **RETURN** node1, node2;

Pares de nodos directamente relacionados con dirección

MATCH (node1)-[*min..max]-(node2) **RETURN** node1, node2;

Pares de nodos relacionados con número de saltos entre min y max

Juguemos un poco...



06-neo4j-retrieve-paths.txt

Recuperación condicional de elementos (Retrieve)

```
MATCH <pattern>  
WHERE <conditions>  
RETURN <pattern_elements>;
```

- **<pattern>**: Patrón de elementos del grafo que queremos encontrar.
- **<conditions>**: Conjunto de filtros “booleanos” a aplicar sobre los elementos del grafo.
- **<pattern_elements>**: Elementos concretos del patrón que queremos devolver.

Operaciones CRUD



Patrones de recuperación condicionales (Retrieve)

MATCH (node)

WHERE node.property = value

RETURN node;

Filtrado por propiedad de nodo

MATCH ()-[rel]-()

WHERE rel.property = value

RETURN rel;

Filtrado por propiedad de relación

Operaciones CRUD



Patrones de recuperación condicionales (Retrieve)

MATCH (node)

WHERE <condicion> (AND | OR | XOR | NOT) <condicion>

RETURN node;

Filtrado por múltiples condiciones

MATCH (node)

WHERE exists(node.property)

RETURN node;

Filtrado por existencia de una propiedad

Operaciones CRUD



Patrones de recuperación condicionales (Retrieve)

MATCH (node)
WHERE node.property IN (<values>)
RETURN node;

Filtrado por múltiples valores

MATCH (node)
WHERE node.property (STARTS WITH | ENDS WITH | CONTAINS) 'value'
RETURN node;

Filtrado por texto

Juguemos un poco...



07-neo4j-retrieve-conditions.txt

Actualización de elementos del grafo (Update)

MATCH <pattern> **SET** <pattern_elements>.property = value, ...;

MATCH <pattern> **REMOVE** <pattern_elements>.property, ...;

MATCH <pattern> **SET** <pattern_elements>:label, ...;

MATCH <pattern> **REMOVE** <pattern_elements>:label, ...;

- **<pattern>**: Patrón de elementos del grafo que queremos encontrar.
- **<pattern_elements>**: Elementos concretos del patrón que queremos devolver.

Juguemos un poco...



08-neo4j-update.txt

Eliminación de elementos del grafo (Delete)

MATCH <pattern> **DELETE** <pattern_elements>;

- **<pattern>**: Patrón de elementos del grafo que queremos encontrar.
- **<pattern_elements>**: Elementos concretos del patrón que queremos devolver.

Juguemos un poco...



09-neo4j-delete.txt

- Comunicación con neo4j desde R.
 - Paquete **RNeo4j**.
 - Disponible en CRAN.
 - <https://nicolewhite.github.io/2014/05/30/demo-of-rneo4j-part1.html>
 - <https://nicolewhite.github.io/2014/05/30/demo-of-rneo4j-part2.html>
 - <https://nicolewhite.github.io/2014/06/30/create-shiny-app-neo4j-graphene.html>
- Comunicación con neo4j desde Python.
 - Paquete / Módulo **neo4j-driver**.
 - Disponible en PyPI (pip).
 - <https://github.com/neo4j-examples/movies-python-bolt>

- Detección de fraude.
 - Seguros.
 - Blanqueo de capitales.
- Sistemas de recomendación.
- Redes sociales.
- Bases de conocimiento / Buscadores.
- Gestión y control de permisos.
- Gestión y control de riesgos de IT y operaciones.

Ejercicio – Queries de neo4j

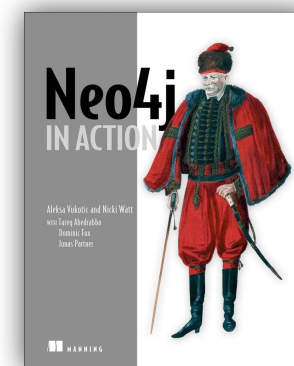
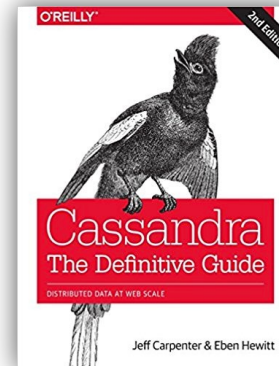
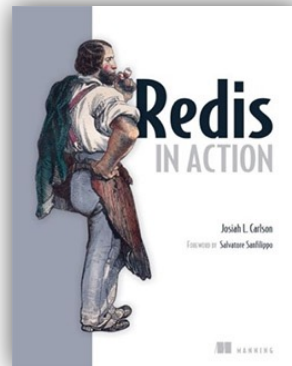
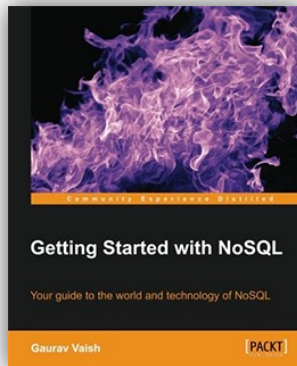


- Títulos de películas en las que aparece Al Pacino.
- Nodos de críticos de cine.
- Actores que han participado en películas de 1996.
- Directores que también son actores de sus películas.
- Películas en las que han participado los actores de Top Gun.

Referencias

Referencias

- Documentación oficial:
 - <https://redis.io/>
 - <http://cassandra.apache.org/>
 - <https://neo4j.com/>
- Tutoriales online:
 - <https://www.tutorialspoint.com/redis/index.htm>
 - <https://www.tutorialspoint.com/cassandra/index.htm>
 - <https://www.tutorialspoint.com/neo4j/index.htm>
- Libros:





Afi Escuela

© 2022 Afi Escuela. Todos los derechos reservados.