APUNTES

Víctor Manuel Peiró Martínez

FIIS 3ºA Ampliación de Bases de Datos

Contenido

Introducción

Tipos de Datos

No estructurados:

- Documentos de Lenguaje Natural
- Audios y Videos
- Necesitan procesos inteligentes para poder acceder a la información (ML o Data Mining)

Semiestructurados:

- Páginas Web
- XML/JSON

Estructurados:

- Bases de Datos
- Información accesible de forma rápida a través de métodos de consulta (SQL)

BBDD Relacionales

Persistencia:

- Problema:
  - Almacenaje en disco, ¿Archivos?
- Solución:
  - BBDD facilita acceso a datos específicos mediante consultas

Concurrencia:

- Problema:
  - Acceso de datos al mismo tiempo puede corromper los datos
- Solución:
  - Propiedades ACID
  - **Atomicidad:** Se ejecuta todo o nada
  - **Consistencia:** Solo se almacena información valida
  - **Aislamiento:** La ejecución de consultas concurrentes tiene como resultado el mismo que una ejecución secuencial
  - **Durabilidad:** Los datos permanecen de forma indefinida

Integración:

- Problema:
  - Acceso a la información desde múltiples aplicaciones
- Solución:
  - Interfaz común para todas las aplicaciones

Modelo Estandarizado:

- Problema:
  - Mismo problema para multitud de situación
- Solución:
  - Normalización del modelo relacional

Desajuste de Impedancia

Es la incompatibilidad entre modelos de datos relaciones y el modelo de la programación orientada a objetos creando desafíos para mapear tablas a objetos y viceversa.

X Relatioal Mapping (XRM)

- Proceso por el cual se crea una relación entre la estructura del modelo relacional y el tipo X de modelo de datos en memoria
- **Problema:**
  - Existen diferencias estructurales que dificultan una relación optima
  - Ejemplo: Jerarquía de clases en POO (ORM)

Object Relational Mapping (ORM)

- **Problema:**
  - Hay estructuras de datos no modelizables como entradas anidadas o listas
- **Solución:**
  - Creación de tablas adicionales

La Caída del Imperio Relacional

Aparecen capas intermedias para la comunicación con BBDD. Aparecen soluciones que actúan como parches para el problema:

- **Problema:** Impedence Mismatch
  - **Solución:** Se crean interfaces que eluden las restricciones de las BBDD relacionales
- **Problema:** Acceso Concurrente
  - **Solución:** Se coordina dicho acceso

Clústeres

Internet ha propiciado un nuevo modelo de negocio donde toda la información es valiosa. Esto ha hecho que las bases de datos hayan incrementado en tamaño

Hay 2 tipos de escalado:

- **Escalado Vertical:** Se usan maquinas más grandes y potentes
  - Sin embargo, hay una limite en rendimiento y un alto coste económico
- **Escalado Horizontal:** Añadimos más maquinas
  - Es más económico, más tolerante a fallos y hay alta redundancia

Sin embargo, al usar clústeres para almacenar datos ocurre un problema. Las BBDD relacionales no están preparadas para trabajar con clústeres. Para esto se ofrecen 3 soluciones:

- Escalado vertical. Esto es caro.
- Usar Oracle RAC y Microsoft Server. Esto también es caro y aparecen más problemas
    - Se virtualiza una maquina en un clúster por lo que si se cae la maquina se puede caer el servicio entero
    - Se usar sharding donde los datos están en diversa maquinas, pero no se aseguran las propiedades ACID.
- Se usan BBDD NoSQL

NoSQL

NoSQL = Not Only SQL. Son bases de datos no relacionales

- Desaparecen restricciones intrínsecas de las BBDD relacionales como:
    - Modelado de datos por tablas
    - Mismo tipo de datos para todos los elementos para una misma entidad
    - Relaciones entre entidades
- No se garantizan las propiedades ACID, pero se usa un modelado de datos agregado
    - Facilita el almacenamiento y acceso en clústeres
    - Solo es necesaria una consulta para acceder a datos agregados
- Son mas simples que las BBDD relacionales
    - Simplificación de Bases de Datos
    - Simplificación de diseño
    - Simplificación de uso y mantenimiento
- Están diseñadas para modelar los datos como son accedidos
- Escalabilidad horizontal
- Desarrollo ágil

Clasificación de BBDD

Orientadas a Documentos

Documento: Conjunto de datos pertenecientes a una entidad

Modelado similar a JSON. Datos mas accesibles al hombre frente a la máquina.

Modelo Agregado: Se permite agrupar/anidar datos en un mismo documento.

Se accede a los documentos por claves siendo estas identificadores o variables indexadas

Ejemplo: MongoDB

Orientadas a Clave-Valor

Solo existe un valor para una clave.

Modelo Agregado: El valor puede tener todo tipo de datos: Variables, listas, sets, tablas hash, arrays asociativos, etc…

Ejemplo: Redis

Orientadas a Columnas

Datos almacenados en tuplas con nombre único, dato y una timestamp para verificar la validez

Similares a BBDD relacionales pero las filas no tienen por qué tener las mismas columnas

- Una columna puede existir en una fila y en otra no
- Las columnas pueden variar a lo largo del tiempo
- Los datos se pueden agregar

Ejemplo: Cassandra

Orientadas a Grafos

Para modelar relaciones y basada en la teoría de grafos

- **Nodos:** Representan entidades
- **Aristas:** Representan relaciones
- **Propiedades:** Representan relaciones

Eficiente para conjuntos de datos asociativos

Ejemplo: Neo4j

Orientadas a Objetos

Información almacenada en forma de objetos, aproximación a POO.

Relaciones mediante punteros

Multimodales

Se combinan varias técnicas mencionadas anteriormente

MongoDB

Base de datos orientada a documentos

Documento: Conjunto de datos pertenecientes a una entidad

Modelado similar a JSON. Datos más accesibles al hombre frente a la máquina.

Modelo Agregado: Se permite agrupar/anidar datos en un mismo documento.

Se accede a los documentos por claves siendo estas identificadores o variables indexadas

Sin esquema, pero se pueden hacer validaciones de esquema con **$jsonSchema**

Documentacion: https://www.mongodb.com

Rendimiento

Se usan índices en variables utilizadas en filtros

Evitar el uso de operaciones join y anidar la información mientras no sea excesiva

Usar indexado inverso cuando se realicen consultas sobre texto

Posibles Problemas de Diseño:

- Muchas colecciones
- Array de muchos elementos
- Mucha información en un solo documento
- Muchos índices

Índices

Tipos de índice:

- Ascendente **(1)** o Descendiente **(-1)**
- De texto **(text)**
- Geoespaciales **(2d o 2dsphere)**

createIndex(): Crea un índice en el campo indicado. Unique para crear un índice único

dropIndex(): Elimina un índice

getIndexes(): Obtener índices de una colección

Ejemplo: db.collection.createIndex({variable: tipo}, {unique: true})

Lectura

find(<criterios>, <proyección>): Devuelve un cursor al listado de documentos que cumplen los criterios de búsqueda

findOne(<criterios>, <proyección>: Devuelve solo un documento que cumpla los criterios

Escritura

insertOne(<documentos>): Inserta un documenta en una colección.

insertMany(<array de documentos>): Inserta varios documentos en una colección

Ambos devuelven los ids insertado

Actualización

Actualiza solo los atributos mencionados

Upsert: Si no existe lo inserta

updateOne(<query>, <documento>, <opciones>): Actualiza un documento

updateMany(<query>, <documento>, <opciones>): Actualiza varios documentos

replaceOne(<query>, <documento>, <opciones>): Reemplaza un documento

Devuelven el resultado de la operación.

findAndModify(query:<document>, update: <document>, remove: <Boolean>): Modifica y elimina un unico elemento, Devuelve el documento sin modificar

Operadores de Modificación

$set: Inicializa o cambia el valor de un campo si existe

$unset: Elimina el campo de un documento

$currentDate: Fecha actual

$inc: Incrementa el valor actual según el valor especificado

$max: Actualiza si el campo es mayor que el especificado

$min: Actualiza si el campo es menor que el especificado

$mul: Multiplica un campo por un valor especificado

$rename: Cambia el nombre de un campo

$pop: Elimina el primero o el ultimo o ultimo elemento de una lista

$push: Añade un elemento a una lista

$pull: Elimina todos los elementos de una listaque cumplan una condicion

Eliminación

deleteOne(<query>): Elimina un documento basándose en la query

deleteMany(<query>): Elimina varios documentos basándose en la query

Filtros

Comparación

$gt: Mayor

$gte: Mayor o igual

$lt: Menor

$lte: Menor o igual

$ne: Distinto

$in: En la lista

$nin: No en la lista

Ejemplo: db.collection.find(edad: {$gte: 18})

Lógicos

$and: Esta en a y b

$or: Esta en a o en b

$nor: No esta en a o en b

$not: No es eso

Ejemplo: db.collection.find({$and: [{edad: 18},{nombre: “Pablo”}]})

Elemento

$exists: Determina si el campo existe

$type: Comprueba si el campo de un determinado tipo

Evaluación

$mod: Realiza el modulo de un campo y compruba si coincide con el que se busca

$text: Busqueda de texto en indices

$where: Busqueda de expresiones JavaScript

Ejemplo: db.students.find({ $text: { $search: "Perico"}})

Expresiones regulares

$regex: Busqueda de expresiones regulares

Expresiones:

- **Coincidencias:**
  - **“texto”:** Coincidencia literal
  - **“^texto”:** Comienza con
  - **“texto$”:** Acaba con
- **Clases de Caracteres**
  - **[caracteres]:** Conjunto de caracteres.
    - **^[abc]: Comienza con a, b o c**
  - **[^caracteres]:** Negación
- **Metacaracteres**
  - **. :** Cualquier carácter
  - **\d:** Digito
  - **\D:** No digito
  - **\w:** Letra, numero o _
  - **\W:** No alfanumerico
  - **\s:** Espacio
  - **\S:** No espacio
- **Cuantificadores**
  - ***:** 0 o mas
  - **+:** 1 o mas
  - **?:** 0 o 1
  - **{n}:** Exactamente n
  - **{n,}:** Al menos n
  - **{n,m}:** Entre n y m
- **|:** Operador or

Geoespaciales

$geometry: Especifica una geometría

$geoIntersects: Documentos que intersectan una geometria

$geoWithin: Documentos que se encuentran completamente dentro de una geometría

$minDistance: Distancia mínima a un punto

$maxDistance: Distancia máxima a un punto

$nearSphere: Distancia geodesica

$near: Distancia cartesiana

Arrays

$elemMatch: Documentos con un array donde al menos uno cumpla una consulta multiple

$all: Todos los elementos de una lista están en el array

$size: Tamaño del array coinciden con uno especificado

$: Devuelve únicamente la primera coincidencia

$slice: Devuelve en función de los índices especificados

Queries Agregadas

Devuelve un listado de los documentos resultado de las operaciones realizadas en cada etapa.

aggregate(): Para hacer consultadas agregadas

$match: Filtra según una condición especificada. Se suele usar al principio para mejorar el rendimiento. Misma sintaxis que find.

$lookup: Para combinar registros de varias colecciones

$group: Agrupa documentos en función de una expresión especificada. Usa acumaladores como:

- **$sum:** suma de campos de los documentos agrupados o numeros
- **$avg:** media de variables de los documentos agrupados
- **$first:** Valor del primer documento agrupado
- **$last:** Valor del último documento agrupado
- **$max:** Valor máximo de los documentos agrupados
- **$min:** Valor mínimo de los documentos agrupados

$unwind: Descompone una lista de uno o varios documentos

$sort: Ordena los documentos

$limit: Limita el número de documentos

$skip: Salta los primeros documentos

$georNear: Ordena los documentos del mas cercano al mas lejano de un punto

$out: Escribe el resultado de una query agregada en una colección

$project: Remodela documentos

Redis

Base de datos de alto rendimiento. Optimizada para usar memoria principal

Usa cache como capa intermedia para sistemas de alta demanda. Usa Least Recently Used Cache

Alamcenamiento en memoria principal y virtual

Diseñado para trabajar en arquitecturas master-slave

Uso de clave-valor para almacenar datos

Tipos de datos:

- Cadena/Cadenas binarias
- Listas
- Conjuntos/ Conjuntos ordenados
- Tablas hash
- Coordenadas

Durabilidad Opcional de datos

Casos de Uso

Bases de Datos:

- Almacenamiento de información de sesiones
- Perfiles de usuarios y preferencias
- Información de cesta de compra

Cache

- Almacenamiento de datos más consultados
- Almacenamiento de páginas web más frecuentadas

Mensajería

- Sistemas de procesamiento distribuido
- Chats

Claves

Se puede usar cualquier tipo de cadena binaria como clave: Cadenas, Imágenes, etc…

Las claves tienen tamaño máximo de 512Mb

Cadenas

Dato básico de Redis

Asignación

SET clave "valor": Permite asignar una clave a un valor. Se permite añadir tiempo de expiración

- **EX:** Especifica un tiempo de expiración en segundos
- **PX:** Especifica un tiempo de expiración en milisegundos
- **NX:** Asigna el valor solo si no existe la clave
- **XX:** Asigna el valor solo si existe la clave

GET clave: Permite obtener el valor de una clave

GETSET clave "valor": Asigna de forma atómica un valor a una clave y devuelve el valor antiguo

MSET c1 "v1" c2 "v2": Asigna de forma atómica una serie de valores a sus claves

MGET c1 c2: Obtiene los valores a varias claves especificas

EXISTS: 1 si existe una clave y 0 si no existe

Expiración

EXPIRE/PEXPIRE clave valor: Asigna un tiempo de expiración

EXPIREAT/PEXPIREAT clave valor: Asigna un momento donde la clave expira

TTL/PTTL clave: Permite mostrar el tiempo hasta que expire la clave. -1 no expira, -2 no existe

PERSIST clave: Elimina el tiempo de expiración

Incrementos y Decrementos

INCR clave: Incrementa el valor en uno

INCRBY clave n: Incrementa el valor en n

INCRBYFLOAT clave n: Incrementa el valor por un valor en coma flotante

DECR clave: Incrementa el valor en uno

DECRBY clave n: Incrementa el valor en uno

Eliminar y Modificar

DEL c1 c2: Elimina una o varias claves

APPEND clave "valor2": Concatena una cadena a otra de una clave

Listas

Son listas enlazadas

El acceso por índice es muy costoso

Inserción, Consulta y Obtención

RPUSH lista "val1" "val2": Insertar por la derecha de la lista

LPUSH lista "val1" "val2": Insertar por la izquierda de la lista

RPOP lista: Obtiene el dato de la derecha de la lista

LPOP lista: Obtiene el dato de por la izquierda de la lista

LRANGE lista inicio fin: Devuelve una sublista comprendida entre el primer índice y el segundo. Índices negativos para empezar desde el final

LTRIM lista inicio final: Recorta una lista

Operadores Bloqueantes

BRPOP: Obtiene un valor de la lista por la derecha, si está vacía, se espera hasta que haya un valor

BLPOP: Obtiene un valor de la lista por la izquierda, si está vacía, se espera hasta que haya un valor

Mas Operadores

LINDEX lista indice: Devuelve el valor de una lista en el índice indicado

LSET lista n "valor": Asigna un valor en una posición n

LINSERT lista: Inserta un valor antes o después de un valor especificado

LLEN: Longitud de la lista

LREM n: Elimina los n primero valores de una lista. Si 0 toda la lista

Tablas Hash

Inserción y Consultas

HSET hash “c1” “v1”: Inserta un par de clave-valor. Devuelve 1 si no existe y lo crea. 0 si ya existe y lo modifica

HSETNX: Inserta un par de clave-valor. Devuelve 1 si no existe y lo crea. 0 si ya existe

HMSET: Inserta in conjunto de pares clave-valor

HGET: Obtiene el valor para una clave especificada

HMGET: Obtiene valores para un conjunto del claves

HGETALL: Devuelve todas las claves y valores

HKEYS: Devuelve todas las claves

HVALS: Devuelve todos los valores

Modificadores

HINCRBY n: Incrementa el valor asociado a una clave en una tabla especificada en n

HINCRBYFLOAT f: Incrementa un valor asociado a una clave con un valor en coma flotante

Otros operadores

HDEL: Elimina una clave especificada

HEXISTS: Devuelve 1 si existe la clave 0 si no.

HLEN: Devuelve el numero de elementos de la tabla hash

HSTRLEN: Devuelve la longitud de una cadena almacenada en una clave

Conjuntos

Inserción y Consulta

SADD set “v1” “v2”: Añade uno o varios elementos. No se insertan elementos repetidos

SPOP: Devuelve y elimina un elemento aleatorio

SRANDMEMBER: Devuelve un miembro aleatorio

SMEMBERS: Devuelve el listado de elementos

SISMEMBER: Devuelve si un elemento pertenece al conjunto

Operaciones de Conjuntos

SDIFF: Devuelve un conjunto con el resultado de la diferencia del primer set y los siguentes

SDIFFSTORE: Almacena un conjunto con el resultado de la diferencia del primer set y los siguientes

SUNION: Devuelve un set con el resultado de la unión de todos los sets especificados

SUNIONSTORE: Almacena un set con el resultado de la unión de todos los sets especificados

SINTER: Devuelve un set con el resultado de la intersección de todos los sets especificados.

SINTERSTORE: Almacena un set con el resultado de la intersección de todos los sets especificados.

Otros Operadores

SREM: Elimina uno o varios valores de un set

SCARD: Devuelve la cardinalidad de un set

SMOVE: Mueve un elemento de un set a otro

Conjuntos Ordenados

Conjuntos cuyos valores tiene una puntuación asociada por la cual se ordenan

Si 2 elementos se tienen la misma puntuación se ordenan por orden lexicográfico

Inserción y Consultas

ZADD set puntuación valor: Inserta uno o varios elementos puntuación/valor. Devuelve el numero de elementos insertados. Si ya existe se actualiza la puntuación

ZRANGE: Devuelve el subconjunto comprendido entre 2 índices. **WITHSCORES** devuelve las puntuaciones tambien

ZREVRANGE: Devuelve el subconjunto comprendido entre 2 índices ordenado de manera inversa

ZRANGEBYSCORE: Devuelve el subconjunto de valores cuya puntuación esta entre 2 valores.

ZREVRANGEBYSCORE: Devuelve el subconjunto de valores cuya puntuación esta entre 2 valores en orden inverso.

ZRANGEBYLEX: Para listas con elementos con la misma puntuación, devuelve un subconjunto ordenado por valor cuyos valores están lexicográficamente comprendidos entre 2 valores.

ZREVRANGEBYLEX: Para listas con elementos con la misma puntuación, devuelve un subconjunto ordenado por valor cuyos valores están lexicográficamente comprendidos entre 2 valores en orden inverso.

Operadores de Conjuntos

ZINTERSTORE: Almacena en un sorted set el resultado de una intersección entre varios conjuntos

ZUNIONSTORE: Almacena en un sorted set el resultado de una union entre varios conjuntos

Eliminación

ZREM: Elimina elementos si existen con valores especificos

ZREMRANGEBYRANK: Elimina elementos entre 2 posiciones

ZREMRANGEBYSCORE: Elimina elemento cuya puntuación esta entre 2 valores

ZREMRANGEBYLEX: Elimina elementos lexicográficamente comprendido entre 2 valores

Otros operadores

ZSCORE: Devuelve la puntuación especifica de un valor

ZRANK: Devuelve la posición de un valor especifico

ZREVRANK: Devuelve la posición de un valor especifico

ZCARD: Devuelve en numero de elementos del set

ZCOUNT: Devuelve el numero de elementos entre 2 posiciones

ZINCRBY n: Incrementa la puntuación de un valor por n

Bits

SETBIT bits val pos: Inserta un bit en la posición especificada

GETBIT: Obtiene el bit en la posición especificada

BITOP: Realiza la operación especificada sobre las claves especificada y las almacena en otra clave

BITCOUNT: Cuenta el numero de bits activos entre las posiciones especificada

BITPOS: Devuelve la posición del primer bit

Iteradores

Uno de los argumentos que toman es el identificador del cursor. Si es 0, se crea un cursor

COUNT para especificar el numero de elementos a iterar

MATCH para restringir el conjunto de elementos sobre el que iterar

SCAN: Itera sobre las claves almacenadas de una base de datos

SSCAN: Itera sobre las claves de un set

HSCAN: Itera sobre las claves de una tabla hash

ZSCAN: Itera sobre las claves de un sorted set

Pipelining

En redis se pueden enviar cadenas de operaciones y esperar mas adelante a obtener la respuesta.

Todos los comandos son serializados y ejecutados secuencialmente de manera atómica

MULTI: Comienzo de la transacción

EXEC: Ejecutar transacción

DISCARD: Descartar la transacción

WATCH: Vigilan variables

SUSCRIBE: Suscribirse a canales especificos

PSUSCRIBE: Suscribirse a canales que cumplan canales especificos

PUBLISH: Publicar un mensaje en un canal

UNSUSCRIBE/PUNSUSCRIBE: Finalizar la suscripción

Cache

Se permite implementar diferentes modalidades de cache

config set: Limite de memoria

Se puede modificar la política cuando se acaba la memoria

noeviction: Error al llegar al limite

allkeys-lru: Least Recently Used puro

volatile-lru: LRU puro sobre elementos que expiran

allkeys-random: Aleatorio sobre todos los elementos

volatile-random: Aleatorio sobre elementos que expiran

volatile-ttl: Elimina elementos que expira antes

Neo4j

Introducción

Nodos: representa entidades de información

- Clasificados y agrupados por etiquetas

Aristas: representan relaciones entre nodos.

- Tienen dirección y tipo y propiedades

Propiedades: tanto los nodos como las aristas tienen campos asociados que proporcionan información adicional.

- Pueden ser números, cadenas, booleanos, …

Creación

Se hace las consultas mediante el lenguaje de Cypher

Nodos

Se crean mediante ()

Ejemplo: (identificador:etiqueta {propiedad:valor , …})

Ejemplo: (identificador:etiqueta WHERE propiedad=valor AND …)

Relaciones

Se crean mediante flechas del siguiente formato - - >

Ejemplo: -[identificador: etiqueta {p1:v1, …}]->

Ejemplo: -[identificador: etiqueta WHERE propiedad=valor AND …]->

Patrones

Combinaciones entre nodos y relaciones

Ejemplo: (nodo)-[relación]->(nodo)

También se puede identificar estructuras especificas en la base de datos

Caminos de longitud definida:

- ***:** Una o mas iteraciones
- ***n:** Exactamente n iteraciones
- ***m..n:** Entre m y n iteraciones
- ***m..:** m o mas iteraciones
- ***..n:** Entre 1 y n iteraciones

Ejemplo: (a)-[*2]->(b)

Repetición de patrón:

- **{m,n}:** Entre m y n iteraciones
- **{1,}:** 1 o mas
- **{0,}:** 0 o mas
- **{n}:** Exactamente n
- **{m,}:** m o mas
- **{,n}:** Entre 0 y n

Consultas

RETURN define la salida de una query

- **RETURN identificador[.propiedad]:** Devuelve nodo/propiedad/relación
- **RETURN *:** Devuelve todos los elementos
- **RETURN DISTINCT:** Resultados únicos
- **RETURN expresión:** evaluar expresión

Escritura

CREATE: Permite crear nuevos patrones. Se pueden crear nodos,relaciones, …

MATCH CREATE: Permite crear patrones a partir de patrones existentes

MERGE: asegura que un patrón existe, si no, lo creo

ON CREATE: Permite asignar se crea el patrón

ON MATCH: Permite asignar si el patrón existe

MATCH MERGE: Como MATHC CREATE pero con merge

Lectura

MATCH: Permite realizar consultas que cumplan con un patrón

OPTIONAL MATCH: Permite incluir patrones adicionales

WHERE: Permite incluir restricciones en la consulta

ORDER BY: Especifica el orden de mostrado de los resultados

LIMIT: Limita el numero de elementos a mostrar

SKIP: Especifica desde que fila empezar a mostrar el resultado

WITH: Permite especificar como se van a pasar elementos y como se van a pasar a la siguiente parte de la consulta

UNWIND: Expande una colección a un conjunto de filas

UNION: Combina resultados de varias consultas

Agregación

count(): Permite contar el número de elementos

sum(): Suma todos los valores

avg(): Media de todos lo valores

max(): Valor máximo

min(): Valor mínimo

collect(): Permite crear una colección con los elementos resultantes

DISTINCT: Elimina duplicados

RETURN k, agregación: Permite agregar la clave k de agregación

Funciones

Predicados:

- **all():** Comprueba si todos los elementos de una colección cumplen un predicado
- **any():** Comprueba si alguno de los elementos cumple un predicado
- **none():** Comprueba si ninguno de los elementos un predicado
- **single():** Cumple si un elemento de una colección cumple un predicado
- **exists():** Comprueba si el patron existe

NOMBRE_FUNCION (identificador): Funciones escalares. Devuelven un solo valor

- **length:** Longitud de un camino
- **size:** Longitud de una lista o string.
- **type:** Tipo de relación del identificador especificado.
- **head, last:** Devuelve el primer o último elemento de una colección.
- **timestamp:** Devuelve el timestamp.
- **toInt, toFloat, toString:** Convierte una variable en entero, real o
- texto.
- **startNode, endNode:** Devuelve el nodo de comienzo o fin de una
- relación.
- **coalesce:** Devuelve el primer valor distinto de NULL de una lista.
- **elementId:** id de un nodo o relación.
- **properties:** Devuelve las propiedades de un nodo o relación.

NOMBRE_FUNCION (identificador): Colecciones. Devuelven colecciones

- **nodes:** Devuelve todos los nodos de una ruta.
- **relationships:** Devuelve todas las relaciones de una ruta.
- **labels:** Listado de todas las etiquetas de un nodo.
- **keys:** Listado con todas las claves de las propiedades de un nodo o relación.
- **reduce:** Devuelve el resultado acumulado al aplicar una expresión a todos los elementos de una colección.
- **tail:** Devuelve todos los elementos de una colección excepto el primero.
- **range:** Devuelve todos los elementos de una colección en un intervalo especificado y con un paso especificado.

Actualización y Eliminación

SET: permite actualizar las etiquetas de los nodos y propiedades

Ejemplos:

- **Nueva Etiqueta:** MATCH (a:e1) SET a:e2
- **Nuevo Valor:** MATCH (a:e1) SET a.nombre= “Juan”

FOREACH: Permite actualizar datos en colecciones de datos y resultados agregados

DELETE: Permite eliminar nodos/relaciones

REMOVE: Permite eliminar etiquetas/propiedades

Índices y Restricciones

CREATE INDEX: Crea un índice para una propiedad de una definición de una etiqueta

SHOW INDEXES: Muestra los índices

DROP INDEX: Elimina un índice

CREATE CONSTRAINT: Crea una restricción del tipo indicado a la propiedad indicada

DROP CONSTRAINT: Elimina una restricción del tipo indicado a la propiedad indicada

Cassandra y Big Data

Uso de columnas para almacenar información

- **Clave:** Nombre único de la columna
- **Valor:** Contenido
- **Timestamp:** Fecha en la que el documento fue insertado o actualizado para comprobar la validez del dato

Cassandra es una base de datos diseñada para soportar grandes volúmenes de datos distribuidos. Estas son algunas de sus características:

- Descentralización (Alta Disponibilidad)
- Replicación
- Tolerancia a fallos
- Escalabilidad

Arquitectura

Nodo: Instancia de Cassandra. 1 o mas en un servidor para desarrollo. 1 por servidor para producción

- **Nodo Semilla:** Nodo al que un nodo nuevo se comunica por primera vez cuando se conecta a un cluster para conocer su estructura y papel
- **Coordinador:** Cada uno de los nodos del cluster que recibe una petición del cliente
    - Encargado de recibir la petición y devolver el resultado
    - Elegido de forma aleatoria en función de la disponibilidad
    - Si cae, se encarga el siguiente
- **Driver:** Interfaz para conexión con cliente
    - Se encarga de decidir que nodo coordina cada petición
    - Implementa Round-Robin
- **Nodo Virtual:** Se comportan como nodos normales pero se les puede asignar de forma dinámica y automática los segmentos del token ring
    - **Bootstrap:** Proceso cuando un nuevo nodo es levantado para reasignar parte del token ring
    - **Decommision:** Proceso cuando un nodo es eliminado, se reasigna su parte del token ring

Partición: Unidad básica de información

- **Proceso de Particionado:** Se encarga de asignar un token a cada unidad de datos a partir de su clave
    - Cada clave de partición (token) se utiliza para determinar el emplazamiento de los datos
    - Cada nodo tiene un rango de particiones
- **Particionador:** Encargado de realizar el proceso de particionado

- **Token Ring:** Conjunto de claves de partición existentes en un clúster.
    - Esta dividido en segmentos y cada segmento este asignado a un nodo
    - En función de la clave de partición generada por el particionador, la replica de una partición será dirigida por el coordinador a un nodo

Rack: Conjunto lógico de nodos

Data Center: Conjunto lógico de racks

Cluster: Conjunto completo de nodos

Replicación: Permite ofrecer una alta disponibilidad y tolerancia a fallos

- Cada partición tiene una o varias replicas en función de la configuración de replicación en su keyspace
- **Factor de Replicación:** Determina cuantos nodos deben realizar una petición de escritura. Como mínimo 1 y como máximo todos los nodos.
- **Estrategia de Replicación:** Determina como se distribuyen las réplicas.
    - **SimpleStrategy:** Se distribuyen de forma secuencial
    - **NetworkTopologyStrategy:** Cada data center tiene su propio factor de replicación y su coordinador remoto para realizar la replicación

Nivel de Consistencia: Determina cuantos nodos deben responder a una petición antes de que el coordinador devuelva la respuesta

- **Escribiendo:** Cuantos han almacenado la partición
- **Leyendo:** Cuando han devuelto la partición
- Se definen distintos niveles en función del número de nodos con replica deben enviar acknowledge
    - **ONE, TWO, THREE:** El primero, los 2 primeros, …
    - **ANY:** Cualquier Nodo
    - **ALL:** Todos
    - **QUORUM:** Al menos el 51% del factor de replicación
    - **LOCAL_ONE:** El primer nodo en el data center
    - **LOCAL_QUORUM:** Quorum en el data center
    - **EACH_QUORUM:** Quorum de cada data center

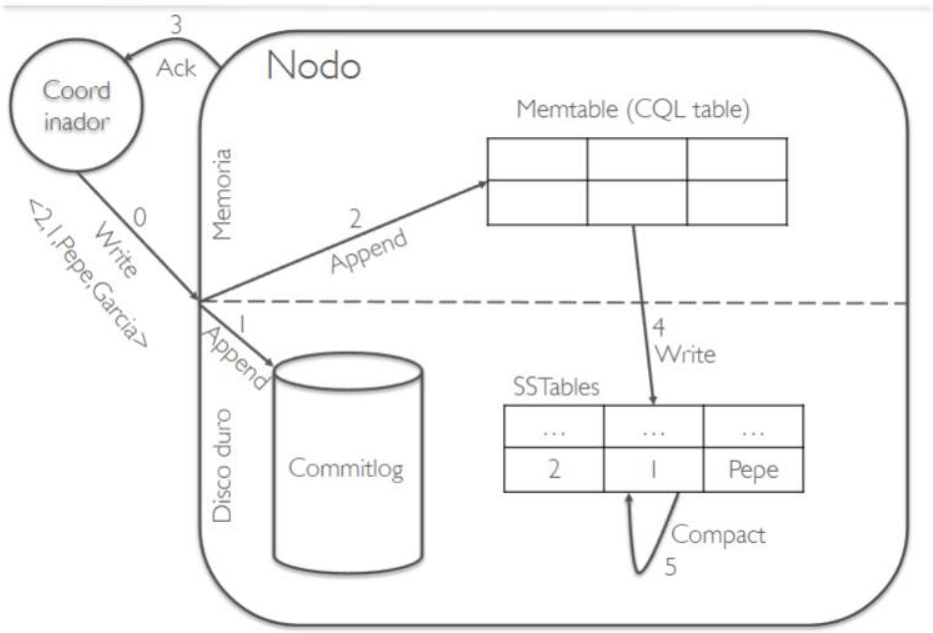
Hinted Handoff: Registra caídas y fallos en los nodos

- Permite gestionar las operaciones de escritura en nodos caídos
    - En caso de que el nodo este caído, el coordinador guarda un hint que especifica la réplica a escribir
    - Cuando el nodo vuelve a esta disponible el coordinador realiza la operación de escritura
    - No se guardan hints cuando el nodo esta mas de 3 horas caido

Flujo de Escritura

El proceso está compuesto por 4 etapas que aseguran durabilidad y consistencia de los datos

- Escritura en Memtables
- Escritura en Commitlog
- Escritura en las SSTables
- Compactación de Datos en SSTables



Commitlog

Introduce cada nueva consulta de escritura en disco duro para que si existe una caída de servidor, este pueda actualizar las memtables con consultas pendientes a ejecutar

La escritura en disco se lleva a cabo cuando las consultas en el commitlog sin escribir en las SSTables alcancen un tamaño determinado

Cuando se escriben en las SSTables se marcan como flushed

Hasta que no esta escritas en disco duro no se devuelve el acknowledge al coordinador

Memtable

Se crea un memtable por cada tabla CQL de cada keyspace

Se van acumulando peticiones de escritura en BBDD. Es posible varias entradas para un dato

Permite consultas de lectura aunque no haya sido escrito aun

Todo el contenido de las Memtables se transfiere a las SSTables periódicamente

SSTable

Tablas inmutables

Se crea una por cada memtable cuando se hace un flush. Existiran varias SSTables por CQL table

Se va compactando la información para que solo existe un SSTable

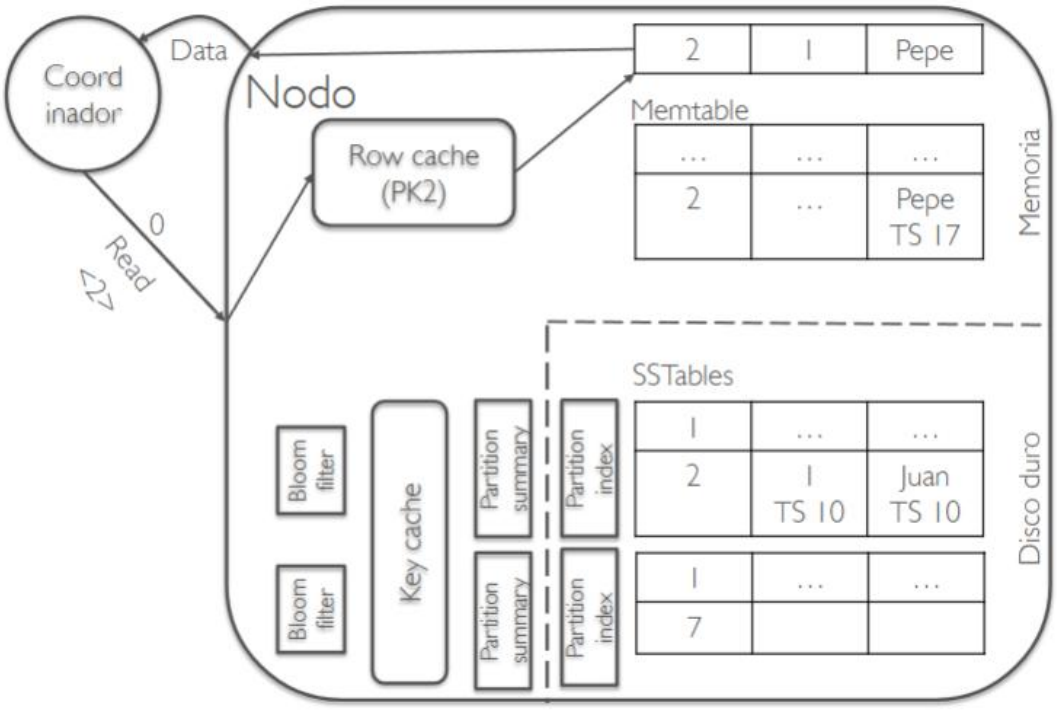
Flujo de Lectura

Se busca tanto en las memtables como las SSTables las entradas para la partición

Una vez localizadas, se devuelve al coordinador mediante un proceso llamado merge el conjunto de campos con el timestamp mas reciente

Existen filtros y caches intermedias para acelerar el proceso

read_repair_chance(10%): Probabilidad con que se lanza un proceso de comprobación de consistencia del dato en todos los nodos



Row Cache

Permite almacenar los resultados de las ultimas consultas para devolver el dato sin necesidad de merge

Opcional. Por defecto desactivado

Se guarda periódicamente el contenido a disco para poder recuperar rápidamente en caso de reinicio

Bloom Filter

Estructura de datos que indica si un primary key esta en su SSTable asociada

Reduce coste de búsqueda

Positivo si cree que la primary key esta el SSTable y negativo si sabe que no esta

- Hay falsos positivos, pero no falsos negativos

Se puede regular el porcentaje de falsos positivos

Key Cache

Permite almacenar las posiciones de las ultimas consultas realizadas

Se guarda periódicamente en disco

Partition Summary e Index

Partition Summary:

- Indice del partition inedx que permite acceder al segmento donde se encuentra la posición en la SSTable del partition key
- El partition summary se encuentra en memoria para mejorar el rendimiento

Partition Index:

- Contiene el conjunto completo de posiciones para cada partition key en su SStable
- Se encuentra en el disco duro

Bases de Datos Vectoriales

Conceptos Básicos

Embedding: Representación vectorial densa que captura el significado de un objeto en un espacio métrico

- **Densos:** Todos sus componentes son reales
- **Normalización:** Evita sesgos por magnitud y permite usar coseno
- **Espacio Métrico:** La similitud se mide con métricas especificas

Métricas de Similitud

Similitud del Coseno: $cos\ (\theta) = \frac{x \cdot y}{||x||\ ||y||}$

- Ideal para vectores normalizados
- Captura dirección e ignora magnitud

Producto Interno:

- Útil si el modelo fue entrenado para maximizarlo
- Se alinea con coseno en vectores normalizados

Distancia Euclidea (L2):

- Menos común en texto pero mas cuando la magntidud aporta información

Generación de Embeddings

APIs: Como OpenAI

- **Ventajas:** Alta calidad, fácil uso
- **Inconvenientes:** Coste, privacidad

Open-source: sentence-tranformers

- **Ventajas:** Control, Fine-tuning
- **Incovenientes:** Infraestructura, mantenimiento

Modelo de Datos

Estructura Tipica:

- **Identificador Estable:** doc_id o chunk_id
- **Vectores con nombre o cuerpo:** y métrica
- **Metadatos Descriptivos:** Tipo, idioma, categorías, …
- **Trazabilidad mínima:** Modelo de embeddings, versión, fecha de generación
- **Vinculo al origen:** URL para reconstruir el resultado

Como filtrar:

- Los filtros deben reducir de verdad el conjunto candidato
- Si el motor lo soporta, se pueden indexar los campos de filtro
- En colecciones grandes, particionar por un metadato muy usado

Estrategias de Búsqueda

Búsqueda Exacta (FLAT): Comprar la consulta contra todos los vectores y garantiza top k verdadero

- Coste lineal en número de elementos
- Funciona muy bien con filtros selectivos por metadatos
- Común como segunda etapa
- Usar cuando:
    - La colección es pequeña o esta prefiltrada
    - Necesitas exactitud total
    - Implementas un re-ranking

Búsqueda Aproximada (ANN): Recorre solo una parte bien elegida para reducir latencia y coste

- Usa índices especializados
- Compomiso latencia
- Requiere memoria para el índice y tiempo de construcción
- Natural cuando hay millones de vectores
- Usar cuando:
    - Escalas a millones de vectores con baja latencia y alto QPS
    - Aceptas un pequeño error y optimizas latencia/coste
    - Puedes tunera el índice

Evaluación

- **Intrínseca:** Analiza el espacio vectorial
    - **Separabilidad:** Vectores como clases distintas bien separadas
    - **Agrupamiento:** Métricas como Silhouette Score
- **Extrínseca:** Impacto en tareas reales
    - **Recall@k:** relevante recuperados en top-k / relevantes totales. Prioritario en búsqueda semantica
    - **Precision@k:** relevantes en top-k / k. Valora la pureza del ranking
    - **MRR:** $MRR = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{posicion\ de\ primer\ relevante}$
    - **nDCG:** Considera relevancia multiple y posición
    - **Hit Rate:** Porcentaje de consultas con al menos un resultado relevante