MONGO DB

1. CONSTRUCTS OF ITS DATA MODEL

MongoDB è un noSQL Database costituito da una collezione di documenti, ciascuno dei quali è un insieme di dati chiave valore.

Opensource è stato progettato nel 2008 ed è un database NoSQL (Ovvero non utilizza tabelle come i tradizionali database relazionale) scalabile, flessibile e ad alte prestazioni. È stato progettato per gestire l’archiviazione orientata ai documenti ed utilizza un formato BSON per l’archiviazione dei dati.

A differenza delle tabelle SQL, non ci sono limiti strutturali a ciò che si può memorizzare in MongoDB, ovvero gli schemi dei dati non vengono applicati. In altre parole, non ci sono tabelle predefinite per memorizzare dati in modo che inserendoli, si deve seguire uno schema ben preciso; si può invece memorizzare quello che si vuole, il che lo rende ideale per le strutture dati più organiche e disordinate. Però attenti che non ci sono controlli come in SQL.

Caratteristiche di Mongo DB

* Flessibile: La sua flessibilità proviene dal fatto che i campi in mongo DB non hanno necessariamente gli stessi dati, i campi all’interno di una stessa collezione possono contenere dati diversi (di natura diversa) e questa flessibilità lo rende ideale per applicazioni in cui gli schemi dei dati possono cambiare frequentemente.
* Si adatta a molti contesti, in particolare è utile quando si devono gestire grandi quantità di dati eterogenei senza schema fisso. Non è invece opportuno quando si devono gestire relazioni tra gli oggetti oppure garantire vincolo di integrità referenziale tra essi.
* È possibile aggiungere, cancellare o modificare campi nei documenti senza modificare lo schema di collezione, il che consente di adattare facilmente la struttura dei dati ai bisogni dell’applicazione.

MongoDB è costituito principalmente da 3 componenti: databases, collezioni, documenti

Il database è al primo livello della gerarchia, dopo di che vengono le collezioni e poi i documenti.

Il database fornisce un container per memorizzare? (stocker) e organizzare i dati e ciascun database contiene una o più collezioni, ciascuna delle quali a sua volta contiene zero o più documenti.

Un database può contenere più collezioni ma una stessa collezione non si può estendere su più database. Similmente, una collezione può contenere più documenti ma un documento non si può estendere su più collezioni

Un documento è un insieme di coppie chiave-valore, dove i valori possono essere di diversi tipi di dati, inclusi altri documenti nidificati, array e tipi di dati scalari. La struttura di base di un documento è rappresentata in formato BSON (Binary JSON) che è un formato binario che estende JSON per includere tipi di dati aggiuntivi come i tipi di dati binari.

Esempio: Supponiamo di avere un database chiamato negozio con due collezioni: prodotto e clienti.. i termini di gerarchia, potrebbe apparire:

DataBase: “Negozio”

Collezioni: “Prodotti”, “Clienti”

Documenti in prodotti:

{ "\_id": 1, "nome": "Laptop", "prezzo": 1000, "categoria": "Elettronica" }

{ "\_id": 2, "nome": "Libro", "prezzo": 20, "categoria": "Libri" }

Documenti in clienti:

{ "\_id": 101, "nome": "Alice", "età": 30, "indirizzo": { "via": "Via Roma", "città": "Roma" } }

{ "\_id": 102, "nome": "Bob", "età": 25, "indirizzo": { "via": "Via Milano", "città": "Milano" } }

MongoDB vs SQL:

Molte componenti presenti in mongoDB hanno il corrispondente in SQL

Database -> database

Tabelle -> collezioni

Righe -> documenti

Colonne -> campi

MongoDB non implementa il concetto di vincolo di integrità referenziale (Che nei database relazionali assicura che le relazioni tra le tabelle siano valide e che ogni chave esterna in una tabella punti a un’esistente chiave primaria in un’altra tabella), ma offre un’approccio diverso per modellare le relazioni tra i documenti:

* **Embedding Documents:**
  + MongoDB consente di incorporare documenti all’interno di altri documenti, il che è noto come “embedding”, questo può semplificare l’accesso ai dati correlati, specialmente se i dati correlati sono sempre recuperati insieme al documento principale.
  + Questa struttura prevede di incorporare un documento collegato in un campo del documento che lo referenzia.
  + { titolo: 'La commedia',

autori: [{ nome: 'Dante', cognome: 'Alighieri' }],

edizioni: [

{ isbn: 'xxxxxxx', anno: 2014, casaEditrice: 'Edizioni X', prezzo: 35.50, ultima: true },

{ isbn: 'yyyyyyy', anno: 2013, casaEditrice: 'Edizioni X', prezzo: 33.20 }

]} => abbiamo incorporato l’autore e le edizioni (2 documenti), il primo ha relazioni di molti a molti, il secondo ha una relazione uno a molti.

Vantaggi:

* L’atomicità. Per mongoDB le uniche operazioni atomiche sono quelle che coinvolgono un singolo documento, non è possibile effettuare transazioni che coinvolgono due o più documenti. Quindi salvando tutto il documento, avremo la garanzia che la nostra modifica è effettivamente avvenuta senza il rischio che salvataggi concorrenti abbiano messo il nostro database in uno stato non coerente.

Ad esempio abbiamo la garanzia che, anche in caso di letture/ scritture concorrenti, ci sarà una sola edizione con il campo ‘ultima’ impostato a ‘true’

* Numero di letture: per avere le informazioni necessarie su un elemento del nostro database, ci basterà una sola lettura, il che è comodo soprattutto in caso di sharding, ossia quando avremo suddiviso le nostre collezioni e distribuite in diversi nodi di mongoDB.. in tal caso, non dovremo interrogare diversi nodi per mettere insieme le informazioni su un’entità della nostra applicazione.
* Lo schema deve essere strutturato in modo che l’applicazione possa ottenere i suoi dati in una unica operazione di lettura.

Svantaggi:

* Ridondanza: nel caso di relazioni molti a molti. Se volessimo ad esempio inserire un’altra opera dello stesso autore nel database, dovremmo replicare tute le informazioni sull’autore. Se tali informazioni sono poche, il danno è minimo. Se invece sono tante, potremmo far crescere rapidamente il nostro database. Se invece volessimo aggiornare uno dei dati di un autore, dovremmo effettuare un aggiornamento massimo su tutti i documenti che incorporano l’autore da aggiornare… potrebbe richiedere molto tempo se il database ha dimensioni grandi.

/\* Utile quando si hanno dati che sono sempre recuperati insieme e non cambiano frequentemente.

Esempio: supponiamo di avere una collezione chiamata “azienda”, vogliamo memorizzare informazioni sui dipendenti all’interno dei documenti aziendali. Invece di utilizzare un riferimento ‘\_id’ per collegare i documenti aziendali in una collezione separata, possiamo incorporare direttamente i dati dei dipendenti nei documenti aziendali.

// Documento Azienda

{

"\_id": ObjectId("5f8b2b5a7647d235873e3b6a"),

"nome": "ABC Corporation",

"settore": "Tecnologia",

"dipendenti": [

{

"nome": "Alice",

"ruolo": "Sviluppatore",

"stipendio": 80000

},

{

"nome": "Bob",

"ruolo": "Progettista",

"stipendio": 90000

}

]

}

Accesso semplificato: quando vengono recuperati i dati dell’azienda, si ottengono automaticamente i dati dei dipendenti senza la necessità di eseguire operazioni di join.

Dati correlati sempre disponibili: se i dati dei dipendenti sono statici e non cambiano spesso, incorporarli può migliorare le prestazioni poiché i dati correlati sono sempre disponibili. \*/

* **Referencing Documents**:
  + Si può utilizzare il concetto di “referencing” per gestire le relazioni in MongoDB. Questo implica l’uso di riferimenti tra documenti, dove un documento fa riferimento all’\_id di un altro documento. Tuttavia, mongoDB non fa automaticamente riferimenti o verifica l’integrità referenziale in modo nativo
  + Le relazioni tra documenti possono essere rappresentate anche tramite l’indicazione di una chiave o un identificativo che permette di collegarli. Ciò significa che avremo più documenti che condividono dei riferimenti tra loro. Considerando l’esempio precedente,

{

titolo: 'La commedia',

autori: [ 'dante' ],

edizioni: [ 'xxxxxxx', 'yyyyyyy' ]

}

{ \_id: 'dante', nome: 'Dante', cognome: 'Alighieri' }

{ \_id: 'xxxxxxx', anno: 2014, casaEditrice: 'Edizioni X', prezzo: 35.50, ultima: true },

{ \_id: 'yyyyyyy', anno: 2013, casaEditrice: 'Edizioni X', prezzo: 33.20 }

Ha senso suddividere questi documenti in tre collezioni: opere, autori ed edizioni.

Vantaggi:

* Scritture e ridondanza: a differenza del modello precedente, qui il vantaggio maggiore si ha con le relazioni molti a molti. Infatti non abbiamo nessuna ridondanza sui dati degli autori. Quindi per modificare i dati di un autore, faremo una sola scrittura e avremo la garanzia che tale modifica sarà valida per tutte le opere di quell’autore.

Svantaggi:

* Numero di letture e consistenza: per ricostruire i dati di un documento coinvolto in una relazione, dobbiamo effettuare tante letture quanti sono i documenti referenziati. I campi ‘edizioni’ ed ‘autori’ hanno semplicemente un array di valori: MongoDB non è consapevole del fatto che si riferiscano a documenti effettivamente esistenti in altre collezioni. Ciò perché non esiste il concetto di relazione ed integrità referenziale.

Riepilogando, non è possibile fornire una regola generale di modellazione dei dati, ma bisogna valutare caso per caso, in base all’utilizzo che si farà del database ed alle esigenze delle applicazioni. La scelta del modello dipende da diversi fattori, inclusi esigenze specifiche dell’applicazione e della natura dei dati e le prestazioni desiderate.

Però si può dire che:

* L’utilizzo di documenti incorporati è più indicato in caso di relazioni uno-a-molti
* L’utilizzo di documenti referenziati è più indicato in caso di relazioni molti-a-molti.

**Validazione di schema:** Si possono utilizzare le regole di validazione di schema in MongoDB per garantire che i documenti rispettino determinati criteri (Ad esempio i tipi di dati e gli intervalli di valori consentiti), anche se questo non è un sostituito diretto del vincolo di integrità referenziale.

Siccome i documenti in una raccolta non devono necessariamente avere gli stessi campi o tipi di dati per impostazione predefinita, una volta stabilito uno schema dell’applicazione, è possibile utilizzare la validazione dello schema per garantire che non vi siano modifiche involontarie dello schema o tipi di dati impropri.

Esempi di applicazione:

**Per una raccolta di utenti**: assicurarsi che il campo ‘password’ sia archiviato solo come stringa. Impedisce cosi agli utenti di salvare la propria password come un tipo di dati imprevisto (immagine)

MongoDB controlla la convalida durante gli aggiornamenti e inserisce nella raccolta, se un documento non supera la convalida, l’operazione viene rifiutata e non avviene la scrittura del documento nella collezione (però si può configurare mongoDB per consentire la scrittura di documenti invalidi).

Come avviene?

* + Definizione delle regole: Per attivare la validazione dello schema, è necessario specificare le regole di validazione sotto forma di uno schema JSON. Questo schema definisce i campi consentiti, i tipi di dati, i vincoli, le regole di validità e altre restrizioni che i documenti devono rispettare.
  + Assegnazione dello schema ad una collection: Una volta definito lo schema di validazione, è possibile assegnarlo a una specifica collection. Questo schema viene quindi applicato ai documenti in quella collection.
  + Validazione in fase di inserimento o aggiornamento: Quando si tenta di inserire o aggiornare un documento in una collection con uno schema di validazione definito, MongoDB verifica che il documento rispetti le regole specificate nello schema. Se il documento non soddisfa le regole di validazione, l'operazione di inserimento o aggiornamento viene rifiutata, e il documento non viene memorizzato.

La validazione dello schema è particolarmente utile per garantire la coerenza e l'integrità dei dati, evitando l'inserimento di documenti con errori o dati non validi all'interno del database MongoDB.

1. QUERY LANGUAGE

MongoDB fornisce un ricco linguaggio di interrogazione implementato in JavaScript ed esposto attraverso la shell di MongoDB (MQL => Mongo Query Language). Utilizza una propria query language per interagire con il database, supportando operazioni CRUD (Create, Read, Update, Delete) utili per fornire le funzionalità necessarie per interagire con i dati.

MongoDB ha il proprio linguaggio chiamato MQL per eseguire operazioni basiche ed avanzate sui dati, oltre alle operazioni basiche (CRUD), MongoDB fornisce funzioni potenti di aggregazione, e differenti operatori per filtrare, ordinare, raggruppare and arrange data. È possibile scrivere queries dalla shell or in the UI via MongoDB Compass or Mongo Atlas

Questa query language è basata su JSON ed offre un insieme di operatori e comandi per eseguire varie operazioni (Ricerca, aggiornamento, cancellazione e aggregazione dei dati), tra cui alcuni sono:

● $and (unisce le clausole della query con un and logico e restituisce tutti i documenti che corrispondono alle condizioni di entrambe le clausole)

● $not (inverte l’effetto della query e torna i documenti che non rispettano la query)

● $nor (unisce le clausole della query con un nor logico e restituisce tutti i documenti che non corrispondono a entrambe le clausole.)

● $or (unisce le clausole della query con un or logico, restituisce tutti i documenti che corrispondono alle condizioni una delle due clausole)

● $eq (corrisponde a valori che sono uguali a un valore specifico)

● $gt (corrisponde a valori che sono maggiori a un valore specifico)

● $gte (corrisponde a valori che sono maggiori o uguali a un valore specifico)

● $in (corrisponde a uno qualsiasi dei valori specificati in un array)

● $lt (corrisponde a valori che sono minori a un valore specifico)

● $lte (corrisponde a valori che sono minori o uguali a un valore specifico)

● $nin (not in => non corrisponde a nessuno dei valori specificati in un array)

I comandi principali in mongoDB sono:

● Find: Il comando find in MongoDB viene utilizzato per recuperare documenti da una singola collezione. È essenzialmente l'equivalente di una query SQL SELECT in un database relazionale. Con find, è possibile specificare un criterio di ricerca per trovare i documenti che corrispondono a un certo set di condizioni. Find() permette di selezionare documenti in una collezione e ritornare un cursore ai documenti selezionati (cursore = struttura che permette di scorrere i record restituiti da una query)

● Aggregate: L'operazione aggregate in MongoDB è molto più potente e flessibile rispetto a find. Consente di eseguire una serie di operazioni di aggregazione sui dati recuperati dalla collezione. Queste operazioni di aggregazione possono includere la trasformazione dei dati, il raggruppamento, il calcolo di statistiche, il filtro avanzato e molto altro. È particolarmente utile quando si desidera eseguire operazioni complesse sui dati come la creazione di report o la generazione di dati aggregati. Aggregate elabora più documenti e restituisce risultati calcolati. È possibile usare operazioni di aggregazione per:

* Raggruppare insieme i valori di più documenti
* Eseguuire operazioni sui dati raggrupati per restituire un singolo risultato
* Analizzare i cambiamenti dei dati nel tempo.

Aggregate offre una gamma più ampia di operatori per gestire operazioni più complesse rispetto a find, possiamo vedere tra essi:

* $match: Utilizzato per filtrare i documenti all'interno di una raccolta, consentendo di selezionare solo quelli che soddisfano determinate condizioni
* $group: Consente di raggruppare i documenti in base a un campo specifico e di eseguire operazioni di aggregazione sui dati raggruppati. È utile per eseguire calcoli come somme, medie, conteggi, ecc.
* $project: Utilizzato per selezionare specifici campi da un documento e per creare nuovi campi che rappresentano trasformazioni dei dati esistenti
* $sort: utilizzato per ordinare i risultati di una query
* $limit: Utilizzato per limitare il numero di documenti restituiti da una query
* $lookup: Consente di eseguire un'operazione di join tra due collezioni, collegando i documenti in una raccolta con i documenti in un'altra raccolta in base a una condizione specifica.
* $min, $max. $avg, $sum: vengono usati all’interno di group per eseguire operazioni di aggrregazione. In particolare sono utili quando si vuole calcolare minimo, massimo, media o somma di un campo all’interno di un gruppo di documenti.

1. Architettura distribuita

MongoDB supporta l’architettura distribuita attraverso il concetto di replication e sharding, i quali consentono di distribuire i dati su più nodi per migliorare le prestazioni, la disponibilità e la scalabilità del sistema.

Sharding: Consente di distribuire i dati su più nodi (shards) in modo da gestire volumi di dati più elevati e carichi di lavoro intensi.

Replication: è un gruppo di nodi che contengono copie identiche di dati.

SHARDING

È un processo che consente di distribuire i dati su più nodi (shards) per migliorare le prestazioni e la scalabilità del database, lo sharding viene utilizzato quando la dimensione dei dati supera la capacità di memorizzazione o di elaborazione di un singolo server.

Lo sharding coinvolge diversi componenti che collaborano per distribuire i dati su più nodi in cluster. I componenti principali di uno sharding system sono:

* Shards: rappresentano i nodi fisici o virtuali che contengono una parte dei dati, ogni shard è un server mongoDB autonomo o un insieme di server MongoDB replicati (replica set). Gli shard possono essere aggiunti o rimossi per gestire la crescita del carico di lavoro.
* Configuration server: mantengono i metadati dello sharding, memorizzano informazioni sulla distribuzione dei dati su vari shard, i chunk(porzione di dati distribuita su uno shard) di dati ed altre configurazioni di sharding
* Router(mongos): funge da punto di accesso per le applicazioni client, i router indirizzano le operazioni di lettura e scrittura ai nodi shard appropriati in base alle regole di sharding e alle chiavi di sharding

Ogni replica set è costituito da un nodo primario e uno o più nodi secondari. I nodi primari gestiscono operazioni di scrittura e mantiene un log delle operazioni (oplog) che viene utilizzato dai nodi secondari per mantenere la sincronizzazione, mentre le operazioni di lettura possono coinvolgere sia il nodo primario che i nodi secondari. In caso di guasto del nodo primario, il sistema eseguirà automaticamente un’elezione per selezionare un nuovo nodo primario tra i nodi secondari disponibili.

“Il cluster MongoDB è costituito da una serie di set di repliche.

Ogni set di repliche è costituito da un minimo di 3 o più istanze mongo. Un cluster partizionato può essere costituito da più istanze di shard mongo e ciascuna istanza di shard funziona all'interno di un set di repliche di shard. L'applicazione interagisce con **Mongos,** che a sua volta comunica con **gli shard** . Pertanto nello Sharding le applicazioni non interagiscono mai direttamente con i nodi shard. Il router di query distribuisce i sottoinsiemi di dati tra i nodi di shard in base alla chiave di shard.”

DISTRIBUZIONE DEI DATI TRA I VARI SHARDS

Viene utilizzata la shard-key per determinare come verrano distribuiti i dati tra i vari shard. La scelta della shard-key è fondamentale per garantire una distribuzione equa dei dati ed ottenere prestazioni ottimali nel distribuited system. È composta da uno o più campi all’interno di un documento e quando si configura una collezione, è necessario specificare una shard-key per quella collezione. Quando si esegue una modifica all’interno di un documento, il sistema determina il valore della shard-key associato a quel documento. Utilizzando la shard-key, il sistema decide a quale shard assegnare il documento. L’obiettivo è di distribuire uniformemente i dati tra i vari shard.

La shard-key può essere di diversi tipi:

1. Hashed shard-key: un hash della shard-key viene calcolato ed utilizzato per distribuire i dati tra i vari shard
   1. Migliora la distribuzione uniforme dei dati, ma rende difficile la query basata sulla shard key stessa
2. Range shard-key: i dati sono distribuiti in base all’intervallo dei valori della shard-key
   1. Consentono query più efficienti sulla shard key, ma richiedono attenzione nella scelta per evitare uno shard overload.

NB: la scelta della scelta dovrebbe distribuire uniformemente i dati tra gli shard per evitare squilibri del carico.

Quindi la distribuzione dei documenti tra gli shard può essere fatta in 2 modi:

* Range-based: ad ogni shard viene assegnato un intervallo specifico di valori di shard-key. Ad esempio, è possibile dividere i dati in un intervallo di valori di dati o intervallo di ID. in questo caso, una serie di shard-key i cui valori sono consecutivi o vicini hanno maggiori probabilità di risiedere nello stesso chunk. Questo consente di eseguire operazioni mirate, in quanto si possono indirizzare le operazioni solo agli shard che contengono i dati richiesti. Qui può essere coinvolto uno shard.
* Hash-based: i documenti sono distribuiti in base ad una funzione hash applicata alla shard-key. Questo approccio è utile per distribuire uniformemente i dati su tutti i nodi, evitando uno sbilanciamento. Tuttavia, potrebbe essere meno efficiente per query di intervallo specifico poiché i dati correlati possono essere distribuiti su nodi diversi anche se i valori delle shard-key possono essere vicini. Qui possono essere coinvolti tutti gli shard.

Durante la vita del sistema, il carico di lavoro può cambiare e la politica di sharding iniziale potrebbe non essere più valida, sbilanciando cosi il cluster.

Per questo motivo, mongoDB valuta periodicamente l’equilibrio degli shard ed esegue operazioni di ribilanciamento se necessario.

L’unità di riequilibrio è chiamata shard chunk(in genere 64 MB di dimensione). Ogni chunk contiene valori contigui di chiavi shard. Un processo in background noto come “balancer” migra automaticamente i chunk tra gli shard per garantire che ogni shard abbia sempre lo stesso numero di chunk.

REPLICAZIONE

È un processo che permette di mantenere più copie dello stesso data in diversi server. Fornisce una alta ridondanza e forte disponibilità dei dati replicando un set di dati su più nodi. In questo modo, se il server è down o indisponibile per una ragione o l’altra, il dato è sempre disponibile in un altro nodo e l’applicazione può essere on line.

I dati possono essere replicati tra i nodi del cluster utilizzando il replica set che è costituito da un nodo primario e due o più nodi secondari.

Il nodo primario riceve tutte le operazioni di scrittura, mantiene tutti i cambiamenti nel log di operazioni (oplog)

I nodi secondari leggono l’oplog dal nodo primario e applicano i cambiamenti nei loro datasets. In caso di fallita del nodo primario, un nodo secondario viene scelto come nuovo nodo primario

L’impostazione di default è che tutte le scritture arrivano al nodo primario, e sono propagate in modo asincrono sui secondari.

I nodi secondari non sono quindi sempre allineati con quelli primari. Il nodo primario memorizza le informazioni sulle modifiche ai documenti in una raccolta all'interno del suo database locale, chiamata oplog.

Queste modifiche vengono continuamente applicate ai nodi secondari che appartengono all'insieme di replica. I nodi primari non sono fissi, ma sono determinati da un'elezione che coinvolge tutti i nodi.

Per diventare primario, un nodo deve essere in grado di contattare più della metà del set di repliche. I membri di un replica set comunicano frequentemente tramite messaggi heartbeat che consentono di reagire a situazioni di guasto.

Se un primario si accorge di non essere più in grado di ricevere messaggi heartbeat da più della metà dei secondari, allora rinuncia al suo status di primario e viene indetta una nuova elezione.

Di default tutte le operazioni di scrittura vanno al primary e l’operazione è conclusa quando sono concluse sul primary; le altre sono asincrone.

Se il chiamante vuole invece rimanere bloccato fino a che tutte le copie hanno avuto l’aggiornamento si può fare una blocking call; è il chiamante a doverlo specificare e accettare il ritardo che ne consegue. Per impostazione predefinita, i client leggono dalla primaria; tuttavia, i client possono specificare una preferenza di lettura per inviare le operazioni di lettura alle secondarie.

La replica asincrona alle secondarie significa che le letture dalle secondarie possono restituire dati che non riflettono lo stato dei dati sulla primaria.

CONSISTENCY MODEL

Vogliamo evitare che nodi replicati si trovino in differenti stati. Per farlo abbiamo 3 principali modi:

* Strong consistency: every read request returns the most update value
  + Quando avviene una scrittura, finché il dato non viene replicato sugli altri server, le prossime operatìzioni di letture vengono bloccate.
  + alta disponibilità (a volte il sistema non è disposto a rispondere a richieste di lettura) // lower performance/availability
* Eventual consistency: once data is updated, eventually read requests return the most update value
  + Non vengono bloccate operazioni di lettura. Nel caso in cui al momento della lettura il dato non era stato replicato sugli altri server, viene letto il valore vecchio.
  + Consente una bassa disponibilità // sometiems seeing stale values

Write concern: rappresenta il livello di durabilità e successo che un’operazione deve raggiungere prima che venga considerata completata. Alcuni valori che può assumere sono:

* W:
  + ‘1’: l’operazione di scrittura deve essere confermata dal nodo primario. Questa è la configurazione predefinita se non viene specificato alcum “write concern”
  + ‘0’: l’operazione di scrittira è completata a livello locale senza aspettare la conferma dai nodi si replica o altri server. Non c’è garanzia di durabilità e consistenza oltre al nodo locale
  + ‘majority’: l’operazione di scrittura deve essere confermata dalla maggioranza dei nodi di replica. Questo è spesso utilizzato per garantire una maggiore durabilità

Read concern: rappresenta il livello di consistenza delle operazioni di lettura, determina fino a che punto un’operazione di lettura può vedere le modifiche apportate da operazioni di scritture non confermate da tutti i nodi dei set di repliche. Alcuni suoi valori comuni sono:

* Local: un’operazione di lettura può vedere solo modifiche confermate sul nodo in cui è stata eseguita l’operazione di lettura
* Majority: un’operazione di lettura può vedere solo le modifiche confermate dalla maggioranza dei nodi di replica.
* Linearizable: garantisce una lettura lineare e ritorna solo dopo che tutte le operazioni di scrittura che precedono la lettura sono state confermate dalla maggioranza dei nodi.