BD NoSQL

Duma Amalia Diana
Grupa 1, Anul 1

Inginerie Software

Cuprins

1 Introducere

Odată cu creșterea rapidă a volumelor de date și diversificarea cerințelor de scalare și performanță, bazele de date NoSQL au devenit o alegere populară pentru gestionarea datelor. Termenul NoSQL vine de la "Not only SQL". Bazele de date NoSQL au fost concepute pentru a aborda nevoile specifice ale aplicațiilor moderne, care implică gestionarea unor volume mari de date, structuri de date complexe și cerințe de scalabilitate orizontală. Acestea reprezintă o alternativă la bazele de date relaționale tradiționale, oferind flexibilitate și performanță. În contextul dezvoltării aplicațiilor, bazele de date NoSQL aduc mai multe beneficii. Ele permit dezvoltatorilor să modeleze datele într-un mod mai flexibil, fără constrângerile schemei fixe impuse de bazele de date relaționale. Acest lucru înseamnă că aplicațiile pot adapta și evolua structurile de date în timp real, fără a fi nevoie de modificări complexe în schema bazei de date.

Alegerea acestui subiect vine din dorința de a înțelege mai bine avantajele și provocările asociate cu utilizarea bazelor de date NoSQL, precum și impactul lor asupra dezvoltării aplicațiilor și a industriei software în ansamblu. Prin explorarea acestui domeniu, se are în vedere obținerea de cunoștințe și perspective noi, care să faciliteze construirea și gestionarea aplicațiilor mai eficiente și mai scalabile în viitorul digital în continuă evoluție.

Prima secțiune introduce bazele de date NoSQL și prezintă conceptele fundamentale. Se explorează avantajele oferite de acestea, cum ar fi modelele de date flexibile, scalabilitatea orizontală și interogările rapide, precum și dezavantajele asociate utilizării lor. În a doua secțiune sunt explorate diferitele tipuri de baze de date NoSQL: bazele de date bazate pe perechi cheie-valoare, care sunt eficiente în gestionarea datelor simplificate, bazele de date orientate pe documente, care sunt ideale pentru structuri de date complexe, bazele de date grafice, care se concentrează pe relațiile dintre date, și bazele de date orientate pe coloane, care sunt optimizate pentru interogări rapide pe date voluminoase. Caracteristicile distinctive și utlizările acestora sunt evidențiate.

Secțiunea trei introduce MongoDB, una dintre cele mai populare baze de date NoSQL, recunoscută pentru flexibilitatea sa și capacitățile de scalare. Sunt examinate operațiile CRUD (Create, Read, Update, Delete) precum și importanța indexurilor în accelerarea interogărilor și asigurarea performanței sistemului. De asemenea, se discuta despre conceptele de atomicitate, consistență și operații distribuite în contextul utilizării MongoDB. În final, ultima secțiune analizează Apache Cassandra, o altă soluție de bază de date NoSQL populară, cu accent pe scalabilitate și disponibilitate. Sunt discutate operațiile CRUD, indexurile și funcționalitatea motorului de stocare, care permite performanțe de înaltă disponibilitate și rezistență la eșecuri.

2 Bazele de date NoSQL: Concepte Fundamentale

Termenul de NoSQL a fost utilizat pentru prima dată în 1998, când Carlo Strozzi a lansat o bază de date open-source numită "Strozzi NoSQL". Strozzi a folosit termenul pentru a sublinia faptul că baza sa de date nu folosea SQL (Structured Query Language) [4]. De-a lungul timpului, "NoSQL" a devenit un cuvânt generic pentru a desemna o gamă largă de tehnologii de stocare a datelor care se concentrează pe abordări non-relaționale. În prezent, cea mai populară traducere a termenului este "Not only SQL".

Bazele de date NoSQL au apărut la sfârșitul anilor 2000 pe măsură ce costul stocării a scăzut dramatic, astfel cantitatea de date pe care aplicațiile trebuiau să le stocheze și să le interogheze a crescut. Aceste date veneau în toate formele: structurate, semi-structurate și polimorfice [16]. Ca urmare, definirea schemelor în avans a devenit aproape imposibilă. Bazele de date NoSQL constituiau soluția la aceasta problemă prin eliminarea necesității unei scheme, datele stocându-se într-o singură structură de date, cum ar fi documentul JSON.

NoSQL este, de asemenea, un tip de bază de date distribuită, ceea ce înseamnă că informațiile sunt copiate și stocate pe diverse servere, care pot fi locale sau la distanță. Acest lucru asigură disponibilitatea și fiabilitatea datelor. Dacă unele dintre date devin indisponibile, restul bazei de date poate continua să funcționeze [9].

2.1 Avantaje a bazelor de date NoSql

Bazele de date NoSQL (Non-SQL) oferă numeroase avantaje față de bazele de date relaționale. Acestea au modele de date flexibile, scalabilitate orizontală, interogări incredibil de rapide și sunt ușor de utilizat pentru dezvoltatori.

2.1.1 Modelele de date flexibile

Bazele de date NoSQL precum MongoDB, Cassandra sau Couchbase folosesc adesea scheme dinamice. Spre deosebire de schemele predefinite ale bazelor de date relaționale, acetea pot adapta diferite structuri de date în cadrul aceleiași baze de date sau chiar aceleiași colecții. Această flexibilitate permite dezvoltatorilor să stocheze date fără a defini mai întâi structura acestora, facilitând evoluția modelului de date în timp.

Unele baze de date NoSQL, în special cele orientate pe documente, cum ar fi MongoDB, adoptă un design fără schema. Acest lucru înseamnă că fiecare document dintr-o colecție poate avea propria sa structură unică. Ca rezultat, dezvoltatorii pot adăuga sau elimina câmpuri, pot schimba tipurile de date sau pot modifica schema fără a afecta datele existente sau a necesita migrări complexe [15].

Astfel, această flexibilitate a modelelor vine în sprijinul practicilor de dezvoltare Agile. Dezvoltatorii pot itera rapid, se pot adapta la cerințele de business aflate în schimbare și pot integra continuu noi funcționalități fără a fi constrânși de scheme rigide. Ca urmare, ciclurile de dezvoltare sunt mai scurte și capacitatea de a livra rapid beneficii se îmbunătățește.

2.1.2 Scalarea orizontală

Bazele de date NoSQL sunt proiectate să utilizeze scalarea orizontal, adică distribuirea sarcinii de lucru pe mai multe servere sau noduri în locul unui singur server puternic [19].

Această abordare permite organizațiilor să gestioneze volumele de date aflate în creștere și traficul prin simpla adăugare de hardware la clusterul existent.

Această arhitectură distribuită permite fiecărui nod din cluster să proceseze independent interogările și să efectueze o parte din sarcina totală de lucru. Astfel, sistemul poate scala orizontal fără a crea blocaje. Un alt avantaj al scalării orizontale îl reprezintă elasticitatea infrastructurii [7]. Organizațiile pot să ajusteze resursele dinamic în funție de cerere. Atunci când este necesară capacitate suplimentară, noi noduri pot fi adăugate la cluster fără probleme, iar nodurile existente pot fi eliminate sau înlocuite fără a întrerupe serviciul. Ca și consecință, utilizarea resurselor este optimizată și costurile operaționale sunt reduse.

Spre deosebire de scalarea verticală, care implică trecerea la servere mai mari și mai scumpe, scalarea orizontală permite organizațiilor să scaleze incremental prin adăugarea de servere mai mici și mai accesibile, pe măsură ce este necesar. Această abordare este rentabilă pentru procesarea datelor la scară mare și a sarcinilor de lucru cu un nivel de concurență ridicată.

2.1.3 Interogări rapide

În bazele de date SQL, normalizarea este un proces folosit pentru a organiza eficient structura unei baze de date. Acest proces implică descompunerea unei tabele mari în tabele mai mici și definirea relațiilor între ele [11]. Odată ce baza de date este normalizată, interogările comune se bazează adesea pe operații de îmbinare pentru a aduce împreună datele relaționate din tabele diferite. Această abordare asigură că informațiile pot fi extrase și manipulate într-un mod coerent și eficient, fără a pierde din acuratețe sau performanță. Scopul normalizării este de a reduce redundanța și dependența în date.

Bazele de date NoSQL sunt concepute pentru a optimiza modelele de acces la date în funcție de cerințele aplicației. De exemplu, în MongoDB există o regulă de baza care spune că datele frecvent accesate împreună trebuie să fie stocate împreună [15]. Această optimizare asigură că interogările pot prelua datele necesare printr-o singură operație de citire, fără costurile suplimentare ale îmbinării mai multor tabele. Astfel, bazele de date NoSQL rămân alegerea populară pentru aplicațiile care prioritizează răspunsuri rapide și scalabilitate.

2.1.4 Ușor pentru dezvoltatori

În cadrul sistemelor NoSQL se folosesc adesea modele de date bazate pe documente, în care datele sunt stocate în documente flexibile, similare formatului JSON. JSON este un format de schimb de date care este folosit pe scară largă în dezvoltarea web și este susținut de majoritatea limbajelor de programare. Ca și urmare, se simplifică procesul de stocare și preluare a datelor pentru dezvoltatori.

Prin reducerea necesității transformării datelor și evitarea pașilor de serializare/deserializare, dezvoltatorii pot manipula direct obiectele bazei de date folosind aceleași structuri de date și tipuri de date pe care le folosesc în codul aplicației lor. Acest lucru conduce la o consistență între schema bazei de date și modelul de date al aplicației, ceea ce face codul mai ușor de întreținut.

În plus, modificările aduse modelului de date pot fi propagate cu ușurință între codul aplicației și schema bazei de date, reducând riscul de introducere a erorilor în timpul actualizărilor sau refactorizărilor. Astfel, dezvoltatorii pot lucra mai eficient și pot gestiona mai

bine evoluția aplicației lor în timp.

2.2 Dezavantaje a bazelor de date NoSql

Unul dintre cele mai importante dezavantajele este că aceste baze de date nu suportă tranzacții ACID când vine vorba de mai multe documente. ACID este un acronim care face referire la un set de patru proprietăți care definesc o tranzacție: Atomicitate, Consistență, Izolare și Durabilitate. Atomicitatea asigură faptul că fiecare instrucțiune dintr-o tranzacție este executată sau niciuna. Această proprietate previne pierderea și coruperea datelor. Consistența asigură că tranzacțiile fac modificări în tabele în moduri predefinite și previzibile astfel încât baza de date să rămână consistentă înainte și după tranzacție. Izolarea asigură că tranzacțiile concurente nu interferă sau nu se afectează reciproc. Durabilitatea asigură că modificările aduse datelor de către tranzacțiile executate cu succes vor fi salvate, chiar și în caz de eșec al sistemului [10].

Bazele de date NoSQL pun accentul pe "Disponibilitate și Performanță". Construirea unei baze de date care să ofere proprietăți ACID este dificilă, așadar "Consistența și Izolarea" sunt adesea sacrificate, rezultând în aplicarea majorității principiilor BASE. BASE este un acronim care înseamnă "Basically Available, Soft state, Eventually consistent". Unul dintre conceptele de bază din spatele acestui set de proprietăți este că responsabilitatea datelor revine dezvoltatorului și nu ar trebui gestionată de baza de date [2]. Alegerea între proprietățile ACID și BASE depinde de cerințele specifice ale aplicației și de compromisurile pe care dezvoltatorii sunt dispuși să le facă în ceea ce privește consistența, disponibilitatea și toleranța la partiționare.

Sistemele NoSQL, în special cele adaptate pentru arhitecturi distribuite, sunt mai dificil de întreținut. Una dintre provocările semnificative este gestionarea distribuției datelor pe mai multe noduri sau partiții. Distribuirea datelor este esențială pentru a asigura o scalabilitate orizontală adecvată și pentru a menține performanța sistemului în timp ce volumul de date și numărul de utilizatori cresc. Gestionarea distribuției datelor necesită planificare meticuloasă și optimizare continuă. [17]. Un aspect crucial al gestionării distribuției datelor este asigurarea unei distribuții uniforme a acestora între noduri sau partiții. O distribuție neuniformă poate duce la suprasolicitarea unor noduri sau partiții, ceea ce poate afecta negativ performanța generală a sistemului. Astfel, este important să se utilizeze strategii de fragmentare și de distribuire a datelor care să echilibreze încărcătura între noduri sau partiții.

3 Tipuri de baze de date NoSQL

Bazele de date NoSQL se împart în patru categorii principale: DB bazat pe perechi cheie-valoare, DB orientat pe documente, DB grafic și DB bazat pe coloane. Bazele de date NoSQL se împart în patru categorii principale: DB bazat pe perechi cheie-valoare, DB orientat pe documente, DB grafic și DB bazat pe coloane. În următoarele subsecțiuni se va vorbi despre fiecare categorie oferindu-se două exemple pentru fiecare tip, ilustrând varietatea și versatilitatea acestor sisteme.

3.1 Baze de date bazate pe perechi cheie-valoare

Cea mai simplă formă a unei baze de date NoSQL este bazată pe perechi cheie-valoare. Fiecare element este identificat printr-o cheie unică, iar valoarea asociată cu acea cheie poate fi orice, cum ar fi un șir de caractere, un număr, un obiect sau chiar o altă structură de date. Bazele de date bazate pe perechi cheie-valoare sunt extrem de flexibile deoarece nu impun o schemă fix. Fiecare cheie și valoare pot avea o structură diferită, permițând stocarea datelor heterogene în același sistem de baze de date.

Această metodă de stocare este similară cu tabelele hash în care cheile sunt utilizate ca indici. Prin urmare, modelul de date este simplu: o hartă sau un dicționar care permite utilizatorului să solicite valorile în funcție de cheia specificată [13]. Abordarea preferă scalabilitatea ridicată în detrimentul consistenței, astfel join-urile și operațiile de agregare sunt omise. Avantajele oferite sunt: căutări rapide, opțiuni pentru stocare masivă și concurență ridicată.

Unul dintre cele mai populare cazuri de utilizare pentru acest tip de bază de date este stocarea în cache. Prin stocarea în memorie a datelor accesate frecvent se poate îmbunătăți semnificativ performanța aplicațiilor web. Datele stocate în cache pot include fragmente HTML, răspunsuri API sau orice alte date costisitoare din punct de vedere computațional care pot fi precalculate și reutilizate. De asemenea, ele excelează și în cadrul aplicațiilor care implică analiză în timp real și urmărirea evenimentelor. Date precum interacțiunile utilizatorului, citirile senzorilor sau intrările de jurnal pot fi stocate ca perechi cheie-valoare cu marcaje temporale. Prin stocarea și interogarea eficientă a acestor date, organizațiile pot obține informații valoroase despre comportamentul utilizatorilor, performanța sistemului și valorile operaționale în timp real. Ca și exemple, avem următoarele baze de date:

1. Amazon Dynamo DB - o bază de date fără server care suportă modelele de date cheie-valoare și document. Datorită schemei flexibile, fiecare element poate avea multe atribute diferite. Dynamo DB include suport pentru atomicitate, consistență, izolare și durabilitate pentru aplicații care au o logică de afaceri complexă. De asemenea, oferă suport nativ, la nivel de server, pentru tranzacții, simplificând experiența dezvoltatorului în efectuarea de modificări [1]. Dezvoltatorii pot folosi Amazon DynamoDB pentru a construi aplicații moderne fără server, care pot începe mic și se pot scala global. Amazon DynamoDB se poate scala pentru a susține tabele de aproape orice dimensiune, cu scalare orizontală automată [3].

2. Berkeley DB - o bază de date oferă caracteristici avansate, inclusiv stocarea datelor valorice, replicarea pentru disponibilitate ridicată, accesul concurent și adaptabilitate la eșecuri non criticale într-o bibliotecă software independentă. Baza de date vine cu o

colecție de tehnologii fundamentale, care pot fi configurate pentru a satisface orice necesitate a aplicației, de la soluții de stocare locale la cele distribuite la nivel mondial. [8].

3.2 Baze de date orientate pe documente

Un document este o înregistrare într-o bază de date de tip document. De obicei, un document stochează informații despre un singur obiect și orice metadate asociate acestuia. Documentele stochează date în perechi cheie-valoare. Valorile pot fi o varietate de tipuri și structuri, inclusiv șiruri de caractere, numere, date, tablouri sau obiecte. Documentele pot fi stocate în formate precum JSON, BSON și XML.

Datele sunt stocate în documente flexibile în comparație cu structura rigida a tabelelor. Aceste documente, adesea în format JSON sau BSON, seamănă foarte mult cu obiectele din limbajele de programare populare. Documentele se mapează la obiectele din cod, astfel încât să se poată lucra cât mai natural cu ele. Se elimină descompunerea datelor în tabele, rularea join-urilor costisitoare și integrarea unui strat separat de mapare relațională cu obiecte (ORM) [12]. Datele care sunt accesate împreună sunt stocate împreună, astfel încât dezvoltatorii au mai puțin cod de scris, iar utilizatorii finali obțin performanțe mai mari.

Schema unui document este dinamică și auto-descriptivă, astfel încât dezvoltatorii nu trebuie să o predefinească mai întâi în baza de date. Câmpurile pot varia de la document la document. Dezvoltatorii pot modifica structura în orice moment, evitând migrarea disruptivă a schemei. Ca și exemple, avem următoarele baze de date:

1. MongoDB - o bază de date dezvoltată de 10gen folosind C++ și lansată în 2009. Scopul este de a gestiona și stoca colecții de documente similare JSON. Această bază de date orientată pe documente asigură că datele pot fi interogate și indexate. Un volum mare de date poate fi accesat la viteze foarte mari. Datele care depășesc 50 GB pot fi accesate de 10 ori mai rapid decât în MySQL obișnuit. Datorită acestei viteze, multe proiecte cu date în creștere tind să treacă la MongoDB [8]. O altă caracteristică importantă a acestei baze de date este limbajul său de interogare și capacitățile de indexare. Limbajul oferă o sintaxă bogată pentru interogarea și manipularea datelor, permițând dezvoltatorilor să efectueze operațiuni complexe cu ușurință. În plus, MongoDB suportă diverse tipuri de indexuri, inclusiv indexuri pe câmp unic, compus și geospatial.

2. Couch DB - o bază de date dezvoltată de Apache Software Foundation folosind C++ și lansată în 2005. Folosește documente JSON pentru a stoca datele și oferă o interfață RESTful HTTP pentru a crea și actualiza documente, iar ca și limbaj de programare se folosește JavaScript. În adiție, oferă o aplicație web încorporată numită FULTON, care poate fi utilizată pentru administrare [13]. CouchDB folosește o strategie de rezolvare a conflictelor bazată pe istoricul revizuirilor documentelor, permițându-i să rezolve automat conflictele și să mențină integritatea datelor pe toate replicile. În plus, se utilizează un mecanism de logare înainte de scriere (WAL), care asigură că modificările datelor sunt mai întâi scrise într-un jurnal înainte de a fi aplicate în fișierul principal al bazei de date.

3.3 Baze de date grafice

Bazele de date grafice sunt baze de date care stochează datele sub forma unui graf alcătuit din noduri și muchii. În noduri se stochează obiectele de date, iar fiecare nod poate avea un număr nelimitat și tipuri de relații. Muchiile reprezintă relațiile între noduri. Ele pot reprezenta atât relații de tip unu-la-multe, cât și de tip multe-la-multe. O muchie are întotdeauna un nod de început, un nod de final, un tip și o direcție [18].

Baza de date utilizează o tehnică numită adiacență fără index, ceea ce înseamnă că fiecare nod conține un pointer direct care indică către nodul adiacent. Un alt avantaj îl reprezintă stocarea eficientă și fără schema a datelor semistructurate. Interogările sunt exprimate ca traversări, făcând bazele de date grafice mai rapide decât cele relaționale. În adiție, acest tip e conform cu proprietățile ACID și oferă suport pentru rollback [13]. Modelul de tip graf este o alegere bună pentru aplicațiile care furnizează recomandări datorită modului în care se pot stoca relațiile între categoriile de informații, cum ar fi interesele clienților, prietenii și istoricul de achiziții.

Un alt avantaj al bazelor de date grafice este flexibilitatea lor. Acestea pot gestiona eficient datele cu structuri în schimbare și pot fi adaptate la noi scenarii de utilizare fără a necesita modificări semnificative ale schemei bazei de date. Această caracteristică le face deosebit de utile pentru aplicațiile cu structuri de date în schimbare rapidă sau cerințe de date complexe. Ca și exemple, avem următoarele baze de date:

1. Neo4j - o bază de date dezvoltată de Neo Technology folosind Java și lansată în 2007. Este bazată pe un model de date graf de tip proprietate, care cuprinde noduri și relații împreună cu proprietățile lor. Este fiabilă, conformă cu proprietățile ACID și scalabilă. Oferă o interfață REST și un API Java destul de ușor de utilizat. De asemenea, poate fi încorporată în fișiere jar. Folosește CYPHER ca limbaj de interogare [13]. În timp ce adiacența fără indexare a Neo4j oferă performanțe excelente pentru multe operațiuni legate de graf, există situații în care indexarea tradițională poate fi în continuare benefică, cum ar fi căutarea de noduri specifice pe baza proprietăților sau atributelor. În aceste cazuri, Neo4j poate folosi Apache Lucene, o bibliotecă puternică de căutare a textului, pentru a crea și utiliza indici pentru interogări eficiente [14].

2. Titan DB - o bază de date dezvoltată de Aurelius Labs folosind Java și lansată în 2012. Un aspect semnificativ al bazei de date este abilitatea sa de a gestiona grafuri masive, utilizând mecanisme eficiente de indexare și căutare. Ca urmare, permite accesul rapid la date și explorarea relațiilor complexe din cadrul grafurilor. În plus, este conformă cu proprietățile ACID, asigurând consistența și fiabilitatea datelor [13].

3.4 Baze de date orientate pe coloane

O bază de date orientată pe coloane este un tip de bază de date NoSQL care își stochează datele în coloane, în locul organizării bidimensionale coloană-rând a unei baze de date relaționale tradiționale. Într-o bază de date SQL, toate informațiile despre o intrare sunt stocate sub forma unei înregistrări pe un rând, împărțită în coloane. Toate datele pentru acea înregistrare sunt stocate împreună în memorie. Într-o bază de date orientată pe coloane, toate informațiile pentru un atribut sunt stocate împreună în memorie. Ca și consecință, coloanele care nu sunt relevante

pentru o interogare particulară sunt ignorate în timpul căutării, ceea ce face interogarea mai rapidă [13].

Acest tip de bază de date excelează în scenarii care implică un flux mare de date și în care analiza în timp real și scalabilitatea sunt cruciale, cum ar fi serviciile financiare pentru analiza riscurilor, comerțul electronic pentru recomandări personalizate, big data, aplicațiile care colectează și analizează date în serie temporală. etc. Ca și exemple, avem următoarele baze de date:

1. Big Table - o bază de date dezvoltată folosind limbajele de programare C și C++ și lansată în 2005. Oferă consistență, toleranță la erori și persistență. Implementarea Bigtable are trei componente majore: o bibliotecă care este legată la fiecare client, un server principal și mai multe servere tabletă. Serverele tabletă sunt folosite pentru a gestiona un set de tablete, echivalentul tabelelor într-un RDBMS. Serverul principal gestionează modificări ale schemei, efectuează sarcini precum asignarea tabletelor către serverele de tablete, echilibrarea încărcăturii serverului de tablete, colectarea gunoiului etc. Big Table este disponibil doar ca parte a motorului de aplicații Google, fiind folosit de mai multe aplicații precum Gmail, YouTube și Google Earth [6].

2. Cassandra - o bază de date dezvoltată de Apache Software Foundation folosind Java și lansată în 2008. Se bazează atât pe modelul Dynamo dezvoltat de Amazon, cât și pe cel dezvoltat de Google, Big Table. Astfel, implică concepte atât de stocare pe cheie-valoare, cât și de stocare pe coloane. Oferă caracteristici precum disponibilitate ridicată, toleranță la partiționare, persistență și scalabilitate ridicată. Poate fi folosită pentru o varietate de aplicații precum site-uri de socializare, bănci și finanțe, analiză de date în timp real, retail online etc. Dezavantajul acestei baze de date constă în faptul că citirile sunt comparativ mai lente decât scrierile [5].

4 MongoDB

MongoDB este o bază de date orientată pe documente proiectată pentru ușurința dezvoltării aplicațiilor și scalarea acestora. O înregistrare în MongoDB este un document reprezentând o structură de date compusă din perechi: atribut-valoare. Aceste documente stocate sub format BSON, reprezentarea binară a formatului JSON. Valorile atributelor pot include alte documente, tablouri și tablouri de documente. Fiecare document are un atribut numit ID care reprezintă un identificator și este folosit ca și cheie primară. Câmpurile dintr-un document BSON sunt ordonate. Atunci când se compară documente, ordinea câmpurilor este semnificativă. Pentru executarea eficientă a interogărilor, motorul de interogare poate reordona câmpurile în timpul procesării interogării. Această abordare a folosirii documentelor prezintă avantaje precum corespondența cu tipurile de date native din multe limbaje de programare, reducerea necesității de îmbinări costisitoare prin documente încorporate și tablouri, și susținerea polimorfismului fluent prin schema dinamică.

Facilitatea de replicare oferită de MongoDB este numită "replica set". Acest replica set reprezintă un grup de procese mongod care mențin același set de date și, de asemenea, oferă redundanță și disponibilitate ridicată. Mongod este procesul principal de tip daemon pentru sistemul MongoDB. Se ocupă de solicitările de date, gestionează accesul la date și efectuează operațiuni de management în fundal. Replicarea oferă redundanță și crește disponibilitatea datelor. Cu mai multe copii ale datelor pe diferite servere de baze de date, replicarea oferă un nivel de toleranță împotriva pierderii unui singur server de bază de date. Un replica set conține mai multe noduri care poartă date și opțional un nod arbitru. Dintre aceste noduri, unul este considerat nod primar, iar celelalte noduri secundare. Nodul primar primește toate operațiile de scriere. Nodurile secundare replică acțiunile nodului primar și aplică operațiile în seturile lor de date astfel încât seturile lor de date să reflecte setul de date al primarului. Dacă nodul primar nu este disponibil, un nod secundar eligibil va organiza o alegerie pentru a se alege singur drept noul nod primar. În mod implicit, clienții citesc de la nodul primar. Acestă preferință se poate schimba pentru a trimite operațiile de citire la nodurile secundare. Dezavantajul acestei abordări este adus de replicarea asincronă: citirile de la nodurile secundare pot returna date care nu reflectă starea datelor de pe nodul primar.

MongoDB oferă scalabilitate orizontală prin fragmentare, o arhitectură de bază de date care partitionează datele după intervaluri de chei și distribuie datele între două sau mai multe instanțe de bază de date. Un cluster fragmentat MongoDB constă din fragmente, mongos și servere de configurare. Fiecare fragment conține un subset al datelor împrăștiate și este implementat ca un replica set. MongoDB distribuie sarcinile de citire și scriere în întreaga configurație de fragmente a clusterului fragmentat, permițând fiecărui fragment să proceseze un subset al operațiunilor din cluster. Atât sarcinile de citire, cât și cele de scriere pot fi scalate orizontal în întregul cluster prin adăugarea de fragmente suplimentare. Implementarea serverelor de configurare și a fragmentelor ca replica set oferă o disponibilitate crescută. Chiar dacă unul sau mai multe replica set devin complet indisponibile, clusterul poate continua să efectueze citiri și scrieri parțiale. Asta înseamnă că, în timp ce datele de pe fragmentele indisponibile nu pot fi accesate, citirile sau scrierile îndreptate către fragmentele disponibile pot reuși.

În MongoDB se pot stoca date geospațiale ca obiecte GeoJSON sau ca perechi de coordonate legacy și, ca urmare, baza de date suportă operațiuni de interogare pe date geospațiale. Pentru

a specifica date GeoJSON, se utilizează un document cu un câmp numit **type** care specifică tipul obiectului GeoJSON și un câmp numit **coordinates** care specifică coordonatele obiectului. Dacă se specifică coordonate de latitudine și longitudine, longitudinea trebuie listată prima și apoi latitudinea. Valorile valide ale longitudinii sunt între -180 și 180, iar ale latitudinii între -90 și 90. Interogările geospațiale se calculează pe o sferă, utilizând sistemul de referință WGS84. WGS84 este prescurtarea pentru Sistemul Geodezic Mondial 1984. Este un sistem de referință geodezic standard utilizat în scopuri de poziționare și navigație. WGS84 definește un sistem de coordonate și un elipsoid de referință pentru Pământ, oferind o modalitate consistentă de a reprezenta locațiile de la suprafața planetei în termeni de latitudine, longitudine și altitudine. [20].

4.1 Operații CRUD

Operațiile de creare sau inserare adaugă documente noi într-o colecție. Dacă colecția nu există, operațiile de inserare vor crea colecția. MongoDB oferă următoarele metode pentru a insera documente într-o colecție:

```
db.collection.insertOne()
db.collection.insertMany()
```

În MongoDB, operațiile de inserare, actualizare și ștergere vizează o singură colecție. Toate operațiile de scriere în MongoDB sunt atomice la nivelul unui singur document. Operațiile de citire recuperează documente dintr-o colecție. MongoDB oferă următoarele metode pentru a citi documente dintr-o colecție:

```
db.collection.find()
```

Operațiile de actualizare modifică documentele existente dintr-o colecție. MongoDB oferă următoarele metode pentru actualizarea documentelor dintr-o colecție:

```
db.collection.updateOne()
db.collection.updateMany()
db.collection.replaceOne()
```

Operațiile de ștergere elimină documente dintr-o colecție. MongoDB oferă următoarele metode pentru ștergerea documentelor dintr-o colecție:

```
db.collection.deleteOne()
db.collection.deleteMany()
```

MongoDB oferă posibilitatea de a efectua operații de scriere în bloc. Operațiile de scriere în bloc afectează o singură colecție. Aceste operații pot fi ordonate sau neordonate. În cazul unei liste ordonate de operații, MongoDB execută operațiile în serie. Dacă apare o eroare în timpul procesării uneia dintre operațiile de scriere, MongoDB va returna fără a procesa alte operații de scriere rămase în listă. În cazul unei liste neordonate de operații, MongoDB poate executa operațiile în paralel, însă acest comportament nu este garantat. Dacă apare o eroare în timpul procesării uneia dintre operațiile de scriere, MongoDB va continua să proceseze restul

operațiilor de scriere rămase în listă. Executarea unei liste ordonate de operații pe o colecție fragmentată va fi în general mai lentă decât executarea unei liste neordonate, deoarece cu o listă ordonată, fiecare operație trebuie să aștepte ca operația anterioară să se termine. De asemenea, driverele MongoDB permit automat reîncercarea unor operațiuni de scriere (retryable writes) și citire (retryable reads) în cazul în care întâmpină erori de rețea sau în cazul în care nu se poate găsi un nod principal în replica set sau în clusterul fragmentat.

4.2 Indexuri

Indexurile sunt structuri de date B-tree care stochează o mică parte din setul de date al unei colecții într-o formă ușor de parcurs. Ordinea înregistrărilor din index susține potriviri eficiente pe bază de egalitate și operații de interogare bazate pe interval. În plus, MongoDB poate returna rezultate sortate folosind ordinea din index. Fără indexuri, MongoDB trebuie să scaneze fiecare document dintr-o colecție pentru a returna rezultatele interogării. Dacă există un index potrivit, acesta se folosește pentru a limita numărul de documente care trebuie să fie scanate. Deși indexurile îmbunătățesc performanța interogării, adăugarea unuia are un impact negativ asupra performanței operațiunilor de scriere. Pentru colecțiile cu un raport mare de scriere față de citire, indexurile sunt costisitoare deoarece fiecare inserare trebuie să le și actualizeze. În timpul creării unei colecții, se va adăuga automat un index unix care nu poate fi șters pe câmpul **id**. MongoDB oferă următoarele tipuri de indexuri:

1. Index pe un singur câmp - colectează și sortează date dintr-un singur câmp din fiecare document dintr-o colecție. Este foarte folositor în cazul în care se fac multe interogări repetate pe același câmp. De asemenea, ordinea de sortare (ascendentă sau descendentă) a cheii indexului nu contează deoarece MongoDB poate traversa indexul în ambele direcții.

2. Index compus - colectează și sortează datele din două sau mai multe câmpuri din fiecare document dintr-o colecție. Datele sunt grupate după primul câmp din index și apoi după fiecare câmp ulterior. Este foarte folositor în cazul în care se fac multe interogări repetate pe mai multe câmpuri. Un index compus poate conține până la 32 de câmpuri. Ordinea câmpurilor indexate afectează eficacitatea deoarece indexurile compuse conțin referințe către documente în funcție de ordinea câmpurilor. Prefixele de index sunt submulțimile de la început ale câmpurilor indexate. Sunt suportate interogări pe toate câmpurile incluse în prefixul indexului deoarece câmpurile sunt analizate în ordine, iar dacă o interogare omite un prefix de index, nu poate folosi niciunul dintre câmpurile indexului care urmează după acel prefix.

3. Index multi-cheie - colectează și sortează date din câmpuri care conțin valori de tip array cu valori scalare sau documente încorporate. Atunci când un index se creează pe un câmp care conține o valoare de tip array, MongoDB setează automat indexul respectiv să fie un index multi-cheie. Atunci când un filtru de interogare specifică o potrivire exactă pentru un array ca întreg, MongoDB poate folosi indexul multi-cheie pentru a căuta prima element din array-ul interogat, dar nu poate folosi scanarea indexului multi-cheie pentru a găsi întregul array.

4. Index geospațial - îmbunătățesc performanța interogărilor pe date geospațiale. MongoDB oferă două tipuri de indexuri geospațiale: Indexuri 2dsphere, care interpretează geometria pe o sferă și indexuri 2d, care interpretează geometria pe o suprafață plană.

5. Index text - Indexurile de text susțin interogări de căutare a textului pe câmpuri de tip șir de caractere. Indexurile de text îmbunătățesc performanța atunci când se caută cuvinte sau expresii specifice. O colecție poate avea doar un singur index de text, dar acel index poate acoperi mai multe câmpuri. Construirea unui index de text este similară cu construirea unui index multi-cheie mare, dar durează mai mult decât construirea unui index simplu ordonat (scalar) pe aceleași date. Aceste indexuri stochează cuvintele individuale dintr-un șir de text. Ele nu stochează expresii sau informații despre proximitatea cuvintelor în documente. Ca rezultat, interogările care specifică mai multe cuvinte rulează mai rapid când întreaga colecție se încadrează în RAM.

6. Index hash - colectează și stochează hash-urile valorilor câmpului indexat. Acestea susțin fragmentarea folosind chei de fragmentare hash. Fragmentarea bazată pe hash utilizează un index hash al unui câmp ca și cheie de fragmentare pentru a partiționa datele în întregul cluster fragmentat. Indexurile hash trunchiază numerele cu virgulă mobilă la numere întregi de 64 de biți înainte de a le hashui, ceea ce poate duce la coliziuni. Funcția de hash nu suportă indexurile multi-cheie, astfel nu se poate crea un index hash pe un câmp care conține un array.

4.3 Atomicitate, consistență și operații distribuite

Pentru a controla proprietățile de consistență și izolare ale datelor citite din seturile de replici și clusterele fragmentate există opțiunea **readConcern**. Indiferent de nivelul preocupării de citire, cele mai recente date de pe un nod pot să nu reflecte cea mai recentă versiune a datelor din sistem. Sunt disponibile următoarele niveluri de preocupare pentru citire:

1. **local** - O interogare cu acest nivel returnează date de la instanțe fără garanția că datele au fost scrise pe majoritatea membrilor setului de replici, putând fi revocate. Disponibil pentru utilizare cu sau fără sesiuni și tranzacții. În cazul creării explicite a unei colecții sau a unui index, tranzacția trebuie sa utilizeze acest nivel de preocupare.

2. **available** - O interogare cu acest nivel funcționează la fel ca una cu nivelul **local**. Pentru clusterele fragmentate, acest nivel oferă cei mai reduse timpi de citire posibili între diferitele preocupări de citire. Cu toate acestea, acest lucru vine cu costul consistenței, deoarece preocuparea de citire **available** poate returna documente orfane atunci când se citește dintr-o colecție fragmentată.

3. **majority** - Pentru operațiile de citire care nu sunt asociate cu tranzacții multi-document, acest nivel garantează că datele citite au fost recunoscute de o majoritate a membrilor setului de replici (adică documentele citite sunt durabile). În cazul tranzacțiilor multi-document, aceste garanții rămân valabile doar daca tranzacția are nivelul de preocupare de scriere **majority**. În caz contrat, nu se oferă nicio garanție despre datele citite în tranzacții.

4. **linearizable** - O interogare cu acest nivel returnează date care reflectă toate scrierile de majoritate recunoscute ca fiind reușite și finalizate înainte de începerea operației de citire. Interogarea poate aștepta ca scrierile executate în mod concurent să se propage către o majoritate a membrilor setului de replici înainte de a returna rezultatele.

5. **snapshot** - O interogare cu acest nivel returnează date majoritar comise așa cum apar pe fragmente de la un punct specific în timpul recent. Acest nivel oferă garanții doar dacă tranzacția are preocuparea de scriere **majority**. Acest nivel este disponibil pentru tranzacțiile multi-document.

Write concern descrie nivelul de confirmare solicitat de la MongoDB pentru operațiile de scriere către un mongod, replica seturi sau clustere fragmentate. În clusterele fragmentate, instanțele mongos vor transmite preocuparea de scriere către fragmente. Pentru tranzacții multi-document, preocuparea de scriere se setează la nivelul tranzacției, nu la nivelul operației individuale. Seturile de replici și clustere fragmentate suportă setarea unei preocupări de scriere globale implicite. Operațiile care nu specifică o preocupare explicită de scriere moștenesc setările implicite de preocupare de scriere globale. Write concern-ul poate include următoarele câmpuri:

```
{ w: <value>, j: <boolean>, wtimeout: <number> }
```

Opțiunea **w** este pentru a solicita confirmarea că operația de scriere s-a propagat către un număr specificat de instanțe mongod sau către instanțe mongod cu etichete specificate, **j** se setează pentru a solicita sau nu confirmarea că operația de scriere a fost scrisă în jurnalul de pe disc, iar **wtimeout** pentru a specifica un timp limită pentru a preveni blocarea operațiilor de scriere pe termen nelimitat.

În MongoDB, o operație de scriere este atomică la nivelul unui singur document, chiar dacă operația modifică mai multe documente încorporate într-un singur document. Atunci când o singură operație de scriere modifică mai multe documente, modificarea fiecărui document este atomică, dar operația în ansamblu nu este atomică. În cazul operațiilor de scriere a mai multor documente, fie printr-o singură operație de scriere sau mai multe operații de scriere, alte operațiuni pot interveni. Pentru situațiile care necesită atomicitatea citirilor și scrierilor la mai multe documente, MongoDB suportă tranzacții distribuite, inclusiv tranzacții pe seturi de replici și clusteri fragmentați. În cele mai multe cazuri, o tranzacție distribuită implică un cost mai mare de performanță față de scrierile la un singur document.

5 Apache Cassandra

Apache Cassandra este o bază de date NoSQL distribuită și open-source. Implementează un model de stocare coloană largă și partitionată cu semantici eventual consistente. Cassandra a fost inițial proiectată la Facebook folosind o arhitectură etapizată și orientată pe evenimente (SEDA). Această proiectare inițială a implementat o combinație între tehnicile de stocare și replicare distribuită de la Amazon Dynamo și modelul de date și stocare Bigtable de la Google. Apache Cassandra a fost dezvoltată ca o combinație de clasă superioară a ambelor sisteme pentru a satisface cerințele de stocare la scară largă, atât în ceea ce privește volumul de date, cât și volumul de interogări. Pe măsură ce aplicațiile au început să solicite replicare globală completă și operarea în orice moment a citirilor și scrierilor cu latență scăzută, a fost necesar un nou tip de model de bază de date pentru a satisface aceste noi cerințe.

Setările de configurare Apache Cassandra sunt configurate în fișierul **cassandra.yaml**, care poate fi editat manual sau cu ajutorul uneltelor de gestionare a configurațiilor. Unele setări pot fi manipulate în timp real folosind o interfață online, dar altele necesită repornirea bazei de date pentru a intra în vigoare.

Cassandra oferă unelte pentru gestionarea unui cluster. Comanda **nodetool** interacționează cu interfața de control live a lui Cassandra, permițând manipularea în timpul rulării a multor setări din **cassandra.yaml**. **Auditlogviewer** este utilizat pentru a vizualiza jurnalele de audit. **Fqltool** este utilizat pentru a vizualiza, reda și compara jurnalele de interogări complete. În plus, Cassandra suportă funcționalitatea de snapshot atomic încorporată, care prezintă un snapshot la un punct în timp (PIT) al datelor Cassandra pentru integrare ușoară cu multe unelte de backup. Cassandra suportă, de asemenea, backup-uri incrementale în care datele pot fi backup-uite pe măsură ce sunt scrise.

Apache Cassandra se bazează pe mai multe tehnici din sistemul distribuit de stocare cheie-valoare al lui Amazon Dynamo. Fiecare nod din acest sistem are trei componente principale: Coordonarea cererilor peste un set de date partiționate, membru al inelului și detectare a eșecurilor și un motor local de persistență (stocare). Cassandra obține scalabilitatea orizontală prin partiționarea tuturor datelor stocate în sistem folosind o funcție de hash. Fiecare partiție este replicată către mai multe noduri fizice. Deoarece fiecare replică poate accepta independent mutații pentru fiecare cheie pe care o deține, fiecare cheie trebuie să aibă o versiune. Spre deosebire de lucrarea originală Dynamo în care versiunile deterministe și ceasurile vectoriale erau folosite pentru reconcilierea actualizărilor concurente ale unei chei, Cassandra folosește un model mai simplu de ultima scriere câștigă, în care fiecare mutație este marcată cu un timestamp (inclusiv ștergerile), iar apoi cea mai recentă versiune a datelor este valoarea "câștigătoare".

Cassandra utilizează marcarea temporală a mutației pentru a garanta coerența eventuală a datelor. Mai exact, toate mutațiile care intră în sistem sunt marcate cu o marcă temporală furnizată fie de ceasul clientului, fie, în absența unei mărci temporale furnizate de client, de ceasul nodului coordonator. Actualizările se rezolvă conform regulii de rezoluție a conflictelor a ultimei modificări câștigătoare.

5.1 Operații CRUD

Cassandra oferă limbajul de interogare Cassandra (CQL), similar cu SQL-ul, pentru a crea, modifica și șterge schema bazei de date, precum și pentru a accesa datele. CQL permite

utilizatorilor să organizeze datele într-un cluster de noduri Cassandra folosind:

- Keyspace - Definește modul în care un set de date este replicat, per centru de date. Replicarea reprezintă numărul de copii salvate per cluster. Keyspace-urile conțin tabele.
- Tabel - Tabelele sunt compuse din rânduri și coloane. Coloanele definesc schema pentru un singur datum într-o tabelă. Tabelele sunt partiționate în funcție de coloanele furnizate în cheia de partiționare. De asemenea, se pot adaugă noi coloane în mod flexibil fără să fie nevoie de timp de inactivitate.
- Partiție: Definește partea obligatorie a cheii primare pe care toate rândurile trebuie să o aibă pentru a identifica nodul dintr-un cluster în care este stocat rândul. Toate interogările performante furnizează cheia de partiționare în interogare.
- Rând: Conține o colecție de coloane identificate printr-o cheie primară unică formată din cheia de partiționare și, opțional, chei de grupare suplimentare.
- Coloană: Un singur datum cu un tip care aparține unui rând.

Interogarea datelor din baza de date se realizează folosind instrucțiunea **SELECT**. Aceasta citește una sau mai multe coloane pentru unul sau mai multe rânduri dintr-o tabelă. Returnează un set de rezultate al rândurilor care corespund cererii, fiecare rând conținând valorile selecției corespunzătoare interogării. O instrucțiune **SELECT** conține cel puțin o clauză de selecție, care determină ce coloane vor fi interogate și returnate, și numele tabelei pe care se execută selecția. Deoarece CQL nu execută perații de tip join sau sub-interogări, acestă instrucțiune se aplică doar unei singure tabele. Se poate adăuga o clauză **WHERE** care poate restrânge și mai mult rezultatele interogării, iar clauzele suplimentare pot ordona sau limita rezultatele.

```
select_statement::= SELECT [ JSON | DISTINCT ] ( select_clause | '*' )
    FROM ‘table_name‘
    [ WHERE ‘where_clause‘ ]
    [ GROUP BY ‘group_by_clause‘ ]
    [ ORDER BY ‘ordering_clause‘ ]
    [ PER PARTITION LIMIT (‘integer‘ | ‘bind_marker‘) ]
    [ LIMIT (‘integer‘ | ‘bind_marker‘) ]
    [ ALLOW FILTERING ]
select_clause::= ‘selector‘ [AS ‘identifier‘](',' ‘selector‘ [AS ‘identifier‘])
selector::== ‘column_name‘
    | ‘term‘
    | CAST '(' ‘selector‘ AS ‘cql_type‘ ')'
    | ‘function_name‘ '(' [ ‘selector‘ ( ',' ‘selector‘ )_ ] ')'
    | COUNT '(' '_' ')'
where_clause::= ‘relation‘ ( AND ‘relation‘ )*
relation::= column_name operator term
        '(' column_name (',' column_name )* ')' operator tuple_literal
        TOKEN '(' column_name# ( ',' column_name )* ')' operator term
```

```
operator::= '=' | '<' | '>' | '<=' | '>=' | '!=' | IN | CONTAINS | CONTAINS KEY
group_by_clause::= column_name ( ',' column_name )*
ordering_clause::= column_name [ASC|DESC] (',' column_name [ASC|DESC])*
```

Inserarea datelor pentru un rând se realizează folosind instrucțiunea **INSERT**. Această instrucțiune scrie una sau mai multe coloane pentru un rând dat într-o tabelă. Deoarece un rând este identificat de cheia sa primară, trebuie specificată cel puțin o coloană. Lista coloanelor de inserat trebuie furnizată folosind sintaxa **VALUES**. Spre deosebire de SQL, **INSERT** nu verifică în mod implicit existența anterioară a rândului. Rândul este creat dacă nu exista anterior și actualizat în caz contrar. În plus, nu există mijloace de a ști care acțiune a avut loc.

```
insert_statement::= INSERT INTO table_name ( names_values | json_clause )
        [ IF NOT EXISTS ]
        [ USING update_parameter ( AND update_parameter )* ]
names_values::= names VALUES tuple_literal
json_clause::= JSON string [ DEFAULT ( NULL | UNSET ) ]
names::= '(' column_name ( ',' column_name )* ')'
```

Actualizarea unui rând se realizează folosind instrucțiunea **UPDATE**. Această instrucțiune scrie una sau mai multe coloane pentru un rând dat într-o tabelă. Clauza **WHERE** este folosită pentru a selecta rândul de actualizat și trebuie să includă toate coloanele cheii primare. Coloanele care nu sunt chei primare sunt setate folosind cuvântul cheie **SET**. Într-o instrucțiune **UPDATE**, toate actualizările în cadrul aceleiași chei de partiționare sunt aplicate atomic și izolat. Spre deosebire de SQL, instrucțiunea nu verifică în mod implicit existența anterioară a rândului. Rândul este creat dacă nu exista anterior și actualizat în caz contrar.

```
update_statement ::= UPDATE table_name
                     [USING update_parameter ( AND update_parameter )*]
                     SET assignment( ',' assignment )*
                     WHERE where_clause
                     [IF ( EXISTS | condition ( AND condition)*)]
update_parameter ::= ( TIMESTAMP | TTL ) ( integer | bind_marker )
assignment: simple_selection'=' term
                 '| column_name'=' column_name ( '+' | '-' ) term
                 | column_name'=' list_literal'+' column_name
simple_selection ::= column_name
                     | column_name '[' term']'
                     | column_name'.' field_name
condition ::= 'simple_selection operator term
```

Ștergerea rândurilor sau a părților din rânduri se realizează folosind instrucțiunea **DELETE**. Această instrucțiune șterge coloane și rânduri. Dacă numele coloanelor sunt furnizate imediat după cuvântul **DELETE**, doar acele coloane sunt șterse din rândul indicat de clauza **WHERE**. În caz contrar, sunt eliminate întregi rânduri. Clauza **WHERE** specifică care rânduri trebuie șterse. Mai multe rânduri pot fi șterse cu o singură instrucțiune folosind un operator **IN**. Un interval de rânduri poate fi șters folosind un operator de inegalitate (cum ar fi >=).

```
delete_statement::= DELETE [ simple_selection ( ',' simple_selection ) ]
        FROM table_name
        [ USING update_parameter ( AND update_parameter# )* ]
        WHERE where_clause
        [ IF ( EXISTS | condition ( AND condition)*) ]
```

Se pot executa mai multe operații inserare, actualizare și ștergere într-o singură instrucțiune prin gruparea lor printr-o instrucțiune **BATCH**. Această instrucțiune economisește călătoriile de rețea între client și server prin gruparea actualizărilor. Toate actualizările dintr-un grup care aparțin unei anumite chei de partiționare sunt efectuate în izolare. În mod implicit, toate operațiile din lot sunt efectuate ca fiind înregistrate, pentru a asigura finalizarea tuturor mutațiilor în cele din urmă (sau niciuna).

```
batch_statement::= BEGIN [ UNLOGGED | COUNTER ] BATCH
                   [USING update_parameter(AND update_parameter)*]
                   modification_statement (';' modification_statement)*
                   APPLY BATCH
modification_statement ::= insert_statement | update_statement |
                           delete_statement
```

5.2 Indexuri

Datele stocate în tabelele CQL pot fi interogate prin diverse metode. Metoda principală utilizează cheia de partiționare definită pentru o tabelă și se numește indexare primară. Adesea, totuși, o interogare trebuie să utilizeze o altă coloană a unei tabele pentru a selecta rândurile dorite, iar indexarea secundară este necesară. Indexarea secundară utilizează căutarea rapidă și eficientă a datelor care se potrivesc unei anumite condiții. După ce orice index este creat, datele pot fi interogate folosind acel index. Apache Cassandra are următoarele tipuri de indexare disponibile:

1. Index primar - Indexul primar este cheia de partiționare în Apache Cassandra. Motorul de stocare folosește cheia de partiționare pentru a stoca rândurile de date, iar căutarea cea mai eficientă și rapidă a datelor se potrivește cu cheia de partiționare. În cadrul unei tabele, un rând este identificat în mod unic de cheia sa primară, și de aceea toate tabelele trebuie să definească o singură cheie primară. O cheie primară este compusă din una sau mai multe dintre coloanele definite în tabel. Sintactic, cheia primară este definită cu fraza PRIMARY KEY urmată de o listă separată prin virgulă a numelor coloanelor în paranteze. Ordinea coloanelor în definiția cheii primare definește cheia de partiționare și coloanele de clusterizare.

2. Index secundar (2i) - Indexarea secundară este indexarea integrată originală scrisă pentru Apache Cassandra. Aceste indexuri sunt toate indexuri locale, stocate într-un tabel ascuns pe fiecare nod al unui cluster, separat de tabelul care conține valorile care sunt indexate. Indexul trebuie citit din nod. Această metodă de indexare este recomandată numai atunci când este utilizată împreună cu o cheie de partiționare. Deoarece indexurile sunt construite local pe fiecare nod Apache Cassandra dintr-un cluster, utilizarea 2i pentru interogări duce la performanțe slabe.

3. Indexarea atașată stocării (SAI) - SAI folosește indexuri pentru coloanele non-partiționate și atașează informațiile de indexare la SSTables care stochează rândurile de date. Indexurile sunt localizate pe același nod ca și SSTable și sunt actualizate atunci când SSTable-ul este actualizat. SAI este metoda de indexare cea mai potrivită pentru majoritatea scenariilor de utilizare. Indexarea atașată stocării (SAI) este un index cu scalabilitate ridicată și distribuit global pentru bazele de date Cassandra. Principalele avantaje față de indexurile existente pentru Apache Cassandra sunt: permite căutarea vectorială pentru aplicațiile de inteligență artificială, partajează datele comune de indexare în mai multe indexuri pe aceeași tabel, ameliorează problemele de scalabilitate la scriere, reducere semnificativă a utilizării discului, performanțe mari pentru intervalul numeric, streaming fără copiere a indexurilor. SAI depășește orice altă metodă de indexare disponibilă pentru Apache Cassandra. SAI oferă mai multă funcționalitate decât indexarea secundară (2i), folosind doar o fracțiune din spațiul pe disc și reducând costul total de proprietate (TCO) pentru disc, infrastructură și operațiuni. Pentru performanța în calea de citire, SAI funcționează cel puțin la fel de bine ca celelalte metode de indexare în ceea ce privește throughput-ul și latența. SAI este de asemenea complet compatibil cu streamingul fără copiere (ZCS). Astfel, atunci când se pornesc sau decomisionează noduri într-un cluster, indecșii sunt complet transmiși împreună cu SSTable-urile și nu sunt serializați sau reconstruiți la capătul nodului care primește.

5.3 Motor de stocare

Apache Cassandra procesează datele în mai multe etape pe calea de scriere, începând cu înregistrarea imediată a unei scrieri și terminând cu o scriere a datelor pe disc: înregistrarea datelor în jurnalul de commit, scrierea datelor în memtable, descărcarea datelor din memtable, stocarea datelor pe disc în SSTables. Când are loc o scriere, Cassandra scrie datele într-un jurnal de angajament local pe disc. Această acțiune asigură o durabilitate configurabilă prin înregistrarea fiecărei scrieri efectuate pe un nod Cassandra. În cazul în care apare o închidere neașteptată, jurnalul de angajament furnizează scrieri permanente și durabile ale datelor. Jurnalul de angajament este partajat între tabele. Toate mutațiile sunt optimizate pentru scriere în stocare în segmente de jurnal de angajament, reducând numărul de căutări necesare pentru a scrie pe disc. Segmentele de jurnal de angajament sunt limitate de opțiunea **commitlog_segment_size**. Odată ce dimensiunea definită este atinsă, este creat un nou segment de jurnal de angajament.

Când are loc o scriere, Cassandra scrie de asemenea datele într-un memtable. Memtable-urile sunt structuri în memorie în care Cassandra tamponează scrierile. În general, există un singur memtable activ per tabel. Memtable-ul este un cache de scriere în memorie a partițiilor de date pe care Cassandra le caută după cheie. Memtable-urile pot fi stocate complet în memorie sau parțial în afara memoriei, în funcție de memtable_allocation_type. Memtable-ul stochează scrierile în ordine sortată până când se atinge un limită configurabilă. Când limita este atinsă, memtable-urile sunt scrise pe disc și devin SSTable-uri imutabile. Când apare un eveniment declanșator, memtable-ul este plasat într-o coadă care este scrisă pe disc. În timpul procesului de flush, datele sunt scrise pe disc în ordinea sortată a memtable-ului. De asemenea, este creat un index de partiție pe disc care mapează token-urile către o locație pe disc.

Fisierele SSTable sunt fisiere de date imutabile pe care Cassandra le folosește pentru a păstra datele pe disc. SSTables sunt menținute pe tabel. SSTables sunt imutabile și nu sunt rescrise niciodată după ce memtable-ul este golit. Prin urmare, o partiție este stocată în mod tipic în mai multe fișiere SSTable, pe măsură ce datele sunt adăugate sau modificate. Fiecare fișier SSTable este compus din mai multe componente stocate în fișiere separate:

- Data.db - Datele reale, adică conținutul rândurilor.
- Partitions.db - Fișierul de indexare a partițiilor mapează prefixe unice ale cheilor de partiție decorate către locațiile fișierelor de date, sau, în cazul partițiilor late indexate în fișierul de indexare a rândurilor, către locații în fișierul de indexare a rândurilor.
- Rows.db - Fișierul de indexare a rândurilor conține doar intrări pentru partițiile care conțin mai mult de un rând și sunt mai mari de un bloc de index. Pentru toate aceste partiții, stochează o copie a cheii de partiție, un antet al partiției și un index al separatorilor de blocuri de rânduri, care mapează fiecare cheie a rândului în primul bloc în care poate fi găsit orice conținut cu o cheie de rând egală sau mai mare.
- Index.db - Un index de chei de partiție către poziții în fișierul Data.db. Pentru partițiile late, acest lucru poate include, de asemenea, un index către rânduri în cadrul unei partiții.
- Summary.db - Un eșantion (implicit) la fiecare a 128-a intrare din fișierul Index.db.
- Filter.db - Un filtru Bloom al cheilor de partiție din fișierul SSTable.
- CompressionInfo.db - Metadate despre deplasările și lungimile fragmentelor de compresie din fișierul Data.db.
- Statistics.db - Stochează metadate despre SSTable, inclusiv informații despre marcajele temporale, mormanele, cheile de grupare, compactarea, repararea, compresia, TTL-urile și altele.
- Digest.crc32 - Un digest CRC-32 al fișierului Data.db.
- TOC.txt - O listă de fișiere componente pentru SSTable în format text simplu.
- SAI*.db - Informații de indexare pentru indexurile atașate stocării. Prezent doar dacă SAI este activat pentru tabel.

6 Concluzie

Examinarea bazelor de date NoSQL din acest referat evidențiază rolul lor semnificativ în abordarea nevoilor de scalabilitate și flexibilitate ale aplicațiilor moderne care se ocupă cu volume mari de date. Bazele de date NoSQL se împart în baze bazate pe perechi cheie-valoare, baze de date orientate pe documente, baze de date grafice și baze de date orientate pe coloane. Acestea oferă avantaje distincte față de sistemele tradiționale de baze de date relaționale, în special în manipularea unor volume mari de date nestructurate și pentru a permite scalabilitate rapidă.

Principalele avantaje includ modele flexibile de date, capacitatea de a scala orizontal și facilitarea interogărilor rapide, care împreună îmbunătățesc dezvoltarea și implementarea aplicațiilor în medii care necesită o disponibilitate ridicată și prelucrarea datelor în timp real. Cu toate acestea, aceste sisteme introduc și anumite compromisuri, cum ar fi lipsa garanțiilor puternice de consistență furnizate de tranzacțiile ACID tradiționale, care sunt adesea atenuate prin adoptarea consistenței finale pentru a optimiza disponibilitatea și toleranța la partiționare.

MongoDB esteo bază de date NoSQL orientată pe documente care stochează informațiile în format BSON, o reprezentare binară a JSON-ului, permițând structuri de date complexe precum documente încorporate și tablouri. Fiecare document este identificat în mod unic printr-un câmp ID, servind ca și cheie primară a acestuia. În MongoDB, operațiile pentru crearea, citirea, actualizarea și ștergerea datelor sunt cuprinzătoare și sunt concepute pentru a facilita manipularea ușoară a datelor. MongoDB este proiectată să asigure performanțe robuste chiar și în timp ce aplicațiile se extind. Această performanță este realizată prin utilizarea de indexi, care susțin operațiile eficiente de interogare și contribuie la menținerea unei performanțe ridicate pe măsură ce volumele de date cresc. Tipurile de indexi în MongoDB includ indexi pentru un singur câmp și indexi compuși, care facilitează recuperarea rapidă a datelor prin pre-sortarea acestora în moduri specificate.

Apache Cassandra se evidențiază ca o bază de date NoSQL extrem de scalabilă și distribuită, proiectată special pentru a gestiona cantități mari de date pe multe servere de uz comun. Acest design robust oferă întreprinderilor o disponibilitate ridicată și o toleranță la defecțiuni, care sunt cruciale pentru aplicațiile care necesită scalabilitate masivă și timp de funcționare continuu. Cassandra operează pe baza unui model de stocare cu coloane largi, permițându-i să gestioneze volume mari de date cu eficiență ridicată în scriere și citire. Datele în Cassandra sunt organizate în tabele care conțin rânduri și coloane, și este optimizată în special pentru interogări pe seturi mari de date răspândite pe mai multe servere. Arhitectura bazei de date suportă un design de schemă flexibil, care permite adăugarea de coloane la orice rând fără a afecta alte rânduri, oferind o flexibilitate semnificativă în comparație cu bazele de date relaționale tradiționale.

Referatul subliniază faptul că, în timp ce bazele de date NoSQL oferă beneficii semnificative în anumite cazuri de utilizare, ele nu sunt universal superioare bazelor de date relaționale. Alegerea între utilizarea unei baze de date NoSQL sau a unei baze de date relaționale ar trebui să fie determinată de cerințele specifice ale aplicației, incluzând natura datelor manipulate, viteza necesară pentru preluarea și procesarea datelor și nevoile de scalabilitate.

Bibliografie

[1] Amazon. Amazon dynamodb features. https://aws.amazon.com/dynamodb/features/.

[2] Deka Ganesh Chandra. Base analysis of nosql database. *Future Generation Computer Systems*, 52:13–21, 2015.

[3] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220, 2007.

[4] Keith D. Foote. A brief history of non-relational databases. https://www.dataversity.net/a-brief-history-of-non-relational-databases/, 2018.

[5] Apache Software Foundation. What is apache cassandra? https://cassandra.apache.org/_/index.html.

[6] Google. Bigtable overview. https://cloud.google.com/bigtable/docs/overview/.

[7] Adity Gupta, Swati Tyagi, Nupur Panwar, Shelly Sachdeva, and Upaang Saxena. Nosql databases: Critical analysis and comparison. In *2017 International conference on computing and communication technologies for smart nation (IC3TSN)*, pages 293–299. IEEE, 2017.

[8] Abdul Haseeb and Geeta Pattun. A review on nosql: Applications and challenges. *International Journal of Advanced Research in Computer Science*, 8(1), 2017.

[9] IBM. What is a nosql database? https://www.ibm.com/topics/nosql-databases/.

[10] Avneet Kaur. Acid properties in dbms. https://www.geeksforgeeks.org/acid-properties-in-dbms/, 2023.

[11] Henning Köhler and Sebastian Link. Sql schema design: foundations, normal forms, and normalization. In *Proceedings of the 2016 International Conference on Management of Data*, pages 267–279, 2016.

[12] Inc. MongoDB. What is a document database? https://www.mongodb.com/document-databases, 2023.

[13] Ameya Nayak, Anil Poriya, and Dikshay Poojary. Type of nosql databases and its comparison with relational databases. *International Journal of Applied Information Systems*, 5(4):16–19, 2013.

[14] Neo4j. Overview: What is neo4j (graph database)? https://www.graphable.ai/software/what-is-neo4j-graph-database/.

[15] Lauren Schaefer. Nosql vs. sql databases. https://www.mongodb.com/nosql-explained/nosql-vs-sql#differences-between-sql-and-nosql.

[16] Lauren Schaefer. What is nosql? https://www.mongodb.com/nosql-explained.

[17] Aaron Schram and Kenneth M Anderson. Mysql to nosql: data modeling challenges in supporting scalability. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, pages 191–202, 2012.

[18] Amazon Web Services. What is a graph database? https://aws.amazon.com/nosql/graph/.

[19] Surya Narayanan Swaminathan and Ramez Elmasri. Quantitative analysis of scalable nosql databases. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 323–326. IEEE, 2016.

[20] VirtualSurveyor. What is wgs84? https://support.virtual-surveyor.com/support/solutions/articles/1000261351-what-is-wgs84-, 2022.