

Univerzitet u Nišu, Elektronski fakultet

Katedra za računarstvo

Niš, 2022

Interna struktura i organizacija skladišta podataka Neo4j baze

Seminarski rad

**Mentor**: **Student**:

Prof. dr Aleksandar Stanimirović Vladeta Manić 1429

**Sadržaj**

Uvod ………………………………………………………………………………………………………………………………………. 4

Sistemi zaupravljanje bazama podataka …………………………………………………………………………………. 4

Graf baze podataka ………………………………………………………………………………………………………………… 5

Zasto graf baze podatak …………….…………………………………………………………………………………………… 6

Prednost i upotreba graf baza podataka .………………………………………………………………………………. 7

Neo4j ..……………………………………………………………………………………………………………………………………. 8

Primena Neo4j baze podataka ………………………………………………………………………………………………… 9

Arhitektura Neo4j baze ……………………………………………………………………………………………………….… 10

Arhitektura sistema za upravljanje bayom podataka ……………………………………………………………… 10

Interna struktura Neo4j baze …………………………………………………………………………………………………. 12

Neo4j API ………………………………………………………………………………………………………………………………. 15

Neo4j podaci na disku …………………………………………………………………………………………………………... 15

Indeksi kod Neo4j baze ………………………………………………………………………………………………………….. 17

Zaključak ..................................................................................................................................... 21

Literatura .................................................................................................................................... 22

**Uvod**

Baza podataka predstavlja skup međusobno povezanih podataka, smeštenih u spoljašnjoj memoriji rađunara. Podaci su istovremeno dostupni raznim korisnicima i aplikacionim programima. Podaci su organizovani pomoću tabela, šema, grafova,…

Baze podataka predstavljaju viši nivo rada s podacima u odnosu na klasične programske jezike. Reč je o tehnologiji koja je nastala s namerom da se uklone slabosti tradicionalne “automatske obrade podataka”. Ta tehnologija osigurala je veću produktivnost, kvalitet i pouzdanost u razvoju aplikacija koje se svode na pohranjivanje i pretraživanje podataka u računaru. Prednosti baze podataka u odnosu na fajl sistem su velike, neki od problema koji se prevazilaze korišćenjem baza podataka su:

* Redudantnost podataka, pojava istih podataka na više mesta
* Nekonzistentnost podataka, sve kopije nekog podatka nisu iste
* Otežan pristup podacima
* Neovčašćeni pristup podacima
* Nepostojanje mehanizma za oporavak od greške

**Sistemi za upravljanje bazama podataka**

Sistem za upravljanje bazama podataka predstavlja softverski proizvod koji se nalazi između korisnika i fizičkih podataka u bazi. On omogućava efikasno formiranje, korišćenje i menjanje baze podataka.

Neke od usluga koje obezbeđuje SUBP su:

* Zaštita korisnika baze od detalja na hardverskom nivou. Korisnik ne sme da trpi nikave posledice u slučaju promene fizičke strukture podataka.
* Upravljanje sekundarnim skladištem. SUBP obično uključuje usluge za brzi pristup određenim podacima pomoću indeksa, grupisanje srodnih podataka i obezbeđivanje efikasnog korišćenja bafera koji u memoriji računara na kom se izvršava SUBP zadržava podatke kojima se nedavno pristupalo.
* Kontrola uporednog pristupa. SUBP poseduje određene zaštitne mehanizme kako bi se osiguralo održavanje integriteta baze u slučaju konkurentnog višekorisničkog rada. Podsetimo se, dva posla se izvršavaju konkurentno ako pri radu upotrebljavaju iste resurse (procesor, memoriju, disk, podatke,...). Za ove poslove nije unapred poznata njihova međusobna vremenska lociranost (jedan posao može da počne i završi se pre početka drugog, može da počne pre a završi se posle završavanja drugog posla itd.). Ako postoji period vremena u kome su dva posla (doslovno) aktivna, onda kažemo da se izvršavaju paralelno.
* Rekuperacija. SUBP mora da bude fleksibilan, odnosno tolerantan na greške u softveru i otkazivanje hardvera. Zbog toga sečesto radi privremena replikacija podataka na više mesta kako bi se omogućila rekuperacija baze, odnosno, automatsko pronalaženje i ispravljanje različitih vrsta grešaka.
* Bezbednost. SUBP obezbeđuje mehanizme koji omogućavaju administratoru baze podataka da kontroliše pristup bazi (ko i gde može da pristupi kao i šta može da uradi u bazi podataka).
* SUBP je obično zasnovan na nekom teorijskom modelu i podržava programske jezike za: definisanje strukture i uslova integriteta baze podataka (DDL, Data Definition Language)

**Graf baze podataka**

Relacione baze podataka su dobre za transakcijske podatke i još uvek preovladavaju kada je u pitanju skladištenje podatak. Međutim, sa porastom broja relacija među podacima( npr. kod društvenih mreža), grafovi postaju sve češći izbor kada je u pitanju čuvanje i predstavljanje novih tipova podataka. Trenutne platforme moraju da upravljaju velikom količinom podataka i međusobno povezanih informacija. U poslednjih desetak godina veliku popularnost dobile su NoSQL baze podataka posebno zbog potreb da se manipuliše velikom količinom podataka koji su međusobno povezani. Posebna vrsta NoSQL baza podataka koja ispunjava ovu potrebu su graf baze podataka.

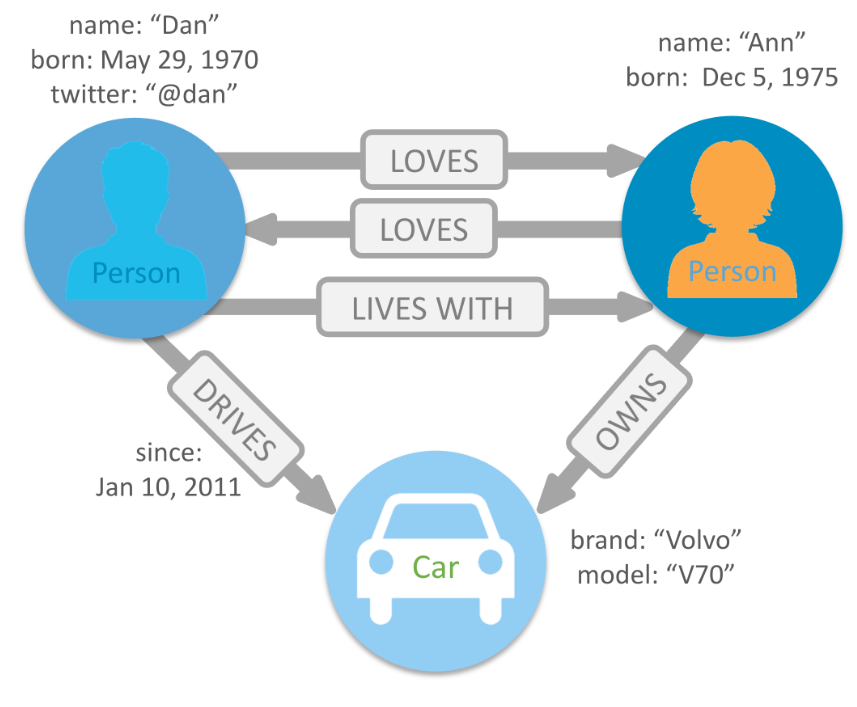
Graf baze podataka su tip nerelacionih baza podataka koje omogućavaju efektivno i efikasno rešenje za čuvanje informacija kod kojih je veza među podacima od jako velikog značaja. Graf baze podataka se mogu definisati kao baze podataka koje koriste strukturu grafa sa čvorovima, potezima i propertijima za čuvanje podataka. Interesovanje za model grafa je poraslo u poslednjih nekoliko godina, zbog mogućnosti njihove primene kod semantičkog web-a ili kod analize socijalnih mreža. Glavna prednost je izuzetno brz pristup kompleksnim podacima, koji se mogu naći na društvenim mrežama, kod mrežnih sistema, sistema za preporučivanje...

Ovaj tip baza podataka je jednostavan za razumevanje zato što je njegov koncept baziran na teoriji grafova. Ova teorija je zasnovana na grafovima, koji su matematička struktura koja se koristi za predstavljanje veza između objekata. U ovom kontekstu, graf je struktura koja je organizovana pomoću čvorova( predstavljaju entitete), potega( relacija između čvorova) koji su predstavljeni pomoću linija između čvorova i pomoću properti( predstavljaju osobine čvorova ili potega). Tako da se graf baze podataka mogu jednostavno opisati kao način za predstavljanje i čuvanjepodataka korišćenjem strukture grafa: čvorova, potega i propertija. Jednostavnost skladišta se ogleda u njegovoj strukturi i jednostavnom pristupu podacima što čini graf baze veoma praktičnim za korišćenje. Graf baze su optimizovane da čuvaju i prikazuju podatke koji su struktuirani u obliku grafa.

Problem graf baza je to što ponekad nisu naročito efikasne u svim operacijama, na primer, predstavljnje podataka iz relacionog modela. One nisu zamena za relacione baze podataka ali su efikasno rešenje kada se radi sa velikom količinom međusobno povezanih podataka.

**Zašto graf baze podataka**

Živimo u svetu gde je sve povezano, i razumevanje većine domena zahteva obradu velike količine povezanosti da bi razumeli šta se zapravo dešava. Često nailazimo na slučaj da su veze između stvari podjednako bitne.



Relacione baze podataka mogu da pamte ove veze, ali da bi ih prikazale potrebno je izvršiti JOIN operaciju koja je jako „skupa“. Ispostavilo se da relacione baze podataka loše rukuju relacijama. Kod graf baza podataka ne postoji JOIN operacija. Relacije se pamte u podatke (čvorove) u mnogo fleksibilnijem formatu. Sve u vezi sistema je optimiyovano u cilju što bržeg prolaska kroz podatke, milioni konekcija po sekundi po jezgru.

Graf baze podataka se susreću sa velikim izazovima. Današnji problemi sa podacima su relacije „many-to-many“ odnosno više prema više sa heterogenim podacim koji zahtevaju:

* Prolazak kroz „duboku“ hijerarhiju
* Pronalaženje skrivenih veza između udaljenih stavki
* Otkrivanje međusobne povezanosti između stavki

Bilo da je u pitanju društvena mreža, platna mreža ili mreža puteva vidi se da sve to predstavlja međusobno povezan graf relacija. I kada želimo da postavimo pitanje u vezi nekih podataka, često se ta pitanja odnose na relacije među podacima nego na konkretne podatke.

**Prednost i upotreba graf baza**

Graf baze podataka su efikasne u nekoliko industrijskih grana, za telekomunikacije, za finansijski sektor, logistiku, ugostiteljstvo i zdravstvo. Graf baze podataka su jako bitne u dosta slučajeva, na primer za analizu ponašanja na društvenim mrežama, upravljanje podacima... Prema nekim istraživanjima graf baze podataka su najbrže rastuća kategorija što se tiče sistema za upravljanje bazama podataka. Glavne prednostii su:

- Pretraga informacija je mnogo optimizovanija u poređenju sa relacionim bazama podata, sobzirom da koristi prednosi blizine podataka jednom ili više čvorova.

- Veoma intuitivne zahvaljujući njihovoj prirodnoj formi prikazivanja podataka – grafovima.

- Izuzetno su agilne u procesu razvoja jer dozvoljavaju jednostavno prilagođavanje, bilo da je u pitanju upis informacija ili njihovo brisanje.

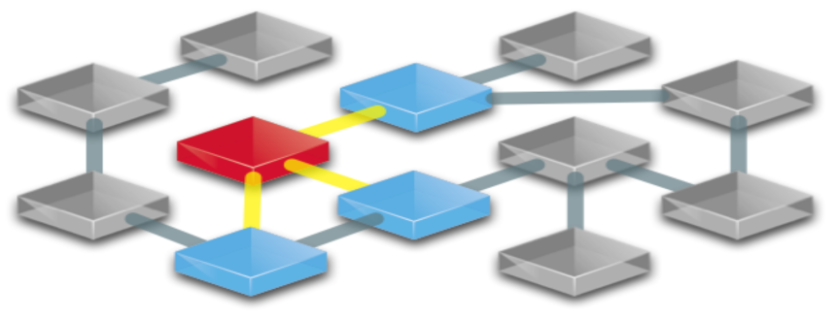
- Dozvoljavaju nove tipove podataka.

- Pogodne su za podatke koji su međusobno povezani.

- Optimizovane za data majning.

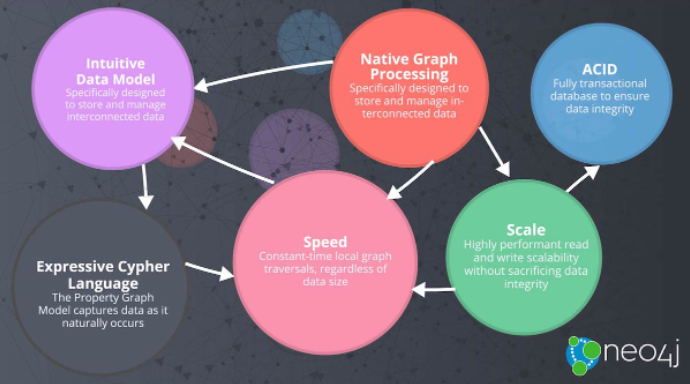
- Imaju visoke performanse u odnosu na relacione baze kada je u pitanju povlačenje podataka koji se nalaze na velikoj „dubini“.

**Neo4j**



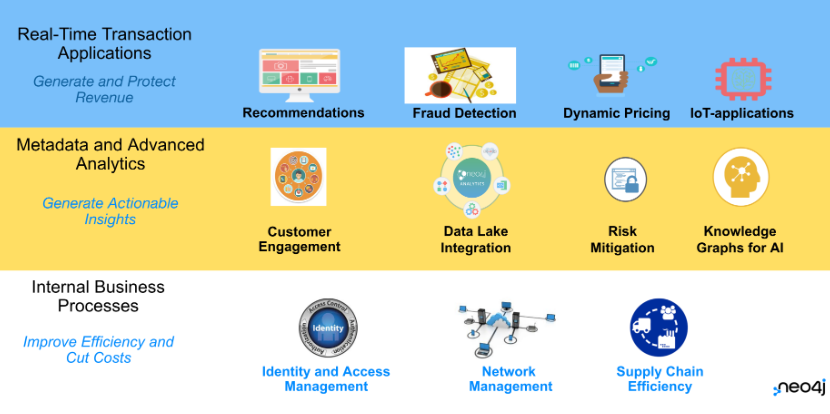
Neo4j je NoSQL graf baza podataka. Razvijena je tako da zadovoljava ACID svojstva transakcione baze podataka koja u osnovi ima graf za skladištenje i obradu podataka. Neo4j koristi model grafa na svim nivoima, sve do nivoa skladišta. Podaci nisu zapamćeni samo kao apstrakcija grafa pomoću neke druge tehnologije, već se pamte baš kao što ih zamišljamo. To je jako važno zato jer predstavlja jedan od glavnih razloga zašto Neo4j nadmašuje ostale graf baze i ostaje veoma fleksibilan. Pored toga što koristi graf kao osnovu, Neo4j pruža ono što se i očekuje od baze podataka, ACID transakcije, podršku za klastere, podršku za prevazilaženje grešaka. Stabilnost i pouzdanost celog sistema je razlog zašto Neo4j svoju primenu nalazi uvelikim kompanijama koje imaju veliku proizvodnju. Glavne karakteristike Neo4j baze koje je čine najjednostavnijom graf bazom za rad su:

* Cypher, jezik za pisanje upitan sličan SQL-u, ali je optimizovan za rad sa grafovima. Koriste ga i druge baze kao sto su SAP i HANA graph i Redis graph.
* Vreme obilaska je konstantno kod velikih grafova bilo da se radi o obilasku po širini ili po dubini zahvaljujući efikasnom predstavljanju čvorova i relacija. Omogućavaju skaliranje na čak milijardu čvorova na srednje jakom hardveru.
* Fleksibilnost šeme grafa koja se može menjati i prilagođavati tokom vremena, sto omogućava dodavanje novih relacija da bi se ubrzao i skratio proces u slučaju da postoji potreba za tim
* Drajveri za popularne programske jezike kao što su JavaScript, .NET, Java, Python i drugi.



**Primena Neo4j baze**

Danas se Neo4j nalazi široku primenu, u različitim područjima rada, kao što su energetika, tehnologija, prodaja, proizvodnja, finansijski sektor, vlada. Od inovativnih tehnologija do već razvijenih poslova, korisnici nalaze primenu i stvaraju novi prihod i poboljšavaju ukupnu efikasnost poslovanja zahvaljujući graf bazi.



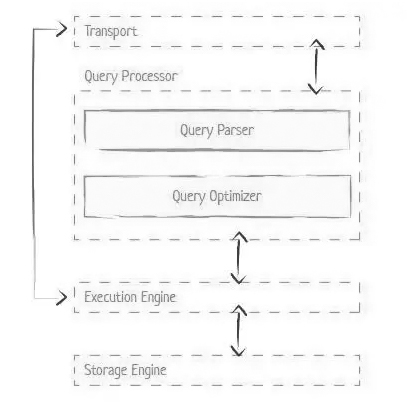
**Arhitektura Neo4j-a**

Neo4j je počeo kao Java biblioteka za kreiranje i papmćenje podataka koji imaju struktura grafa, i tokom vremena je postao samostalni sistemem graf baze podataka sa inicijalnim fokusom na transakcijske performanse, čak u slučaju real-time obrade. Tokom vremena i sa pojavom analitike grafova, fokus je počeo da se pomera na razvijanje novih mogućnosti vezanih za nauku o graf podacima i analitiku, što je Neo4j svrstalo u kategoriju sistema za hibridnu obradu transakcija i analitike.

Organizacije nastavljaju da razvijaju potrebu za transakcionim sistemima, bilo da se radi o klasičnim povezanim slučajevima korišćenja ili mnogim drugim slučajevima koji bi mogli imati značajnu korist od graf baze podataka. Ovi slučajevi korišćenja često zahtevaju i zagarantovanu konzistentnost podataka s jedne strane, ali i stalnu, ugrađenu analitiku s druge strane, bez potrebe za kopiranjem i restrukturiranjem podataka za kasniju analizu (kao kod RDBMS sistema). Neo4j je bio rani lider kada je u pitanju spajanje transakcionih i analitičkih slučajeva upotrebe u jedan HTAP (hybrid transaction and analytics processing) sistem, omogućavajući elegantnu, kombinovanu arhitekturu unutar jedne baze podataka. Kada razumemo kako različiti delovi funkcionišu zajedno, možemo da poboljšavamo performanse Neo4j platforme. Prvo, treba da razumemo arhitekturu visokog nivoa sistema za upravljanje bazom podataka, a zatim da pogledamo kako je Neo4j napravljen za rukovanje različitim radnim opterećenjima grafova, posmatrajući sve od indeksiranja do skladištenja.

**Arhitektura sistema za upravljanje bazama podataka**

Pre razumevanja arhitekture Neo4j-a neophodno je da razumemo kako generalno sistemi za upravljanje bazama podataka funkcionišu. Sistemi za uprvljanje bazama podataka funkcionišu po modelu klijent/server, gde baza podataka predstavlja server a aplikacije ili interfejsi funkcionišu kao klijenti. Sistemi za upravljanje bazama podataka još uvek koriste fajlove za skladištenje podataka ali umesto da se oslone na klasični pristup sa folderima, oni uređuju fajlove pomoću specifičnih formata svojstvenih za baze podataka. Na slici ispod prikazane su najčešće komponente sistema za upravljanje bazom podataka:

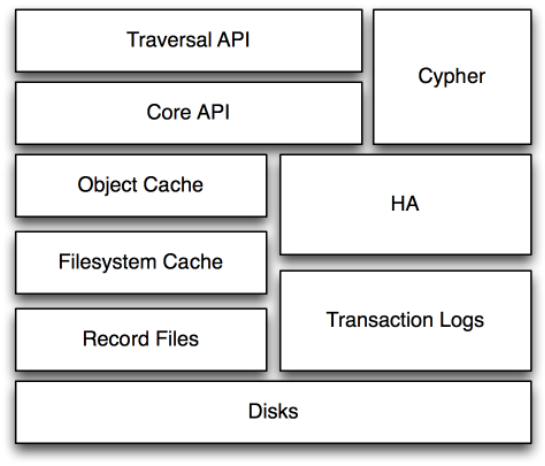


Transport: na početku bilo koje interakcije da bazom podataka upit klijnta stiže kroz transportni podsistem koji ga zauzvrat predaje procesoru upita. Isto funkcioniše i Neo4j.

Query processor: Obrada upita započinje parsiranjem upita da bi se interpretirao i validairao, nakon čega se vrši optimizacija upita tako što se nalzi najefikasniji način za pribavljanje podataka. Neo4j koristi Cypher planer upita, koji kao i većina planera upita vriši optimizaciju na osnovu interne statistike, indeksa, položaja podataka i tako dalje.

Excecution Engine: Izlaz iz procesora upita bilo kog sistema baze podataka plan izvršenja, koji predstavlja sekvencu operacija koj treba izvršiti. Plan izvršenja se predaje execution engine-u koji izvršava plan i spaja rezultate. Neo4j razlaže upite na manje elemente koji se nazivaju operacije, one se kombinuju u strukturu stabla koja predstavlja plan izvršenja. Takođe kod Neo4j-a poslednji čvotovi u stablu odnosno listovi se razrešavaju u zapis i izlaz se prenosi uz stablo, dok ostali čvorovi zahtevaj izlaze iz drugih grana.

**Interna struktura Neo4j**



Osnova Neo4j Arhitekture je fizički disk koji Neo4j koristi. Uvek je bolje da se podaci povlače iz memorije ali Neo4j će imati potrebu da čita sa diska. Zbog toga je najbolje da se koriste diskovi sa što većom brzinom upisa i čitanja, kao na primer SSD, kad je to moguće.

**Keširanje**

U ćilju smanjenja kašnjenja, Neo4j obezbeđuje dva nivoa keširanja:

* Filesystem Cache
* Object Cache

Filesystem cache se nalazi u slobodnom delu memorije, ili RAM memorije na operativnom sistemu koji nije dodeljen ni jednom procesu, i pristupa mu se preko memorijski mapiranog U/I. Kada proces zatraži fajl, ili deo fajl, fajl se učitava u memoriju i to u filesystem cache.

Naknadna čitanja se opslužuju iz filesystem keša pre nego sa diska, čak se i upisi vrše u filesystem keš pa se tek onda prebacuju na disk. Neo4j koristi snagu Java Native U/I da bi operativni sistem efikasnije učitava, čitao, i upisivao podatke sa i na fizički disk.

Filesystem keš se može konfigurisati u neo4j.propertioes fajlu:

neostore.nodestore.db.mapped\_memory=22M

neostore.propertystore.db.mapped\_memory=66M

neostore.propertystore.db.string.mapped\_memory=1.4G

neostore.propertystore.db.arrays.mapped\_memory=0M

neostore.relationshipstore.db.mapped\_memory=1.0G

* nodestore pamti sve čvorove
* propertystore pamti svojstva čvorova i potega, gde su primitive definisane prvim svojstvom, stringovi drugim svojstvom a nizovi su definisani u poslednjem svojstvu
* relationship store, pamti sve relacije

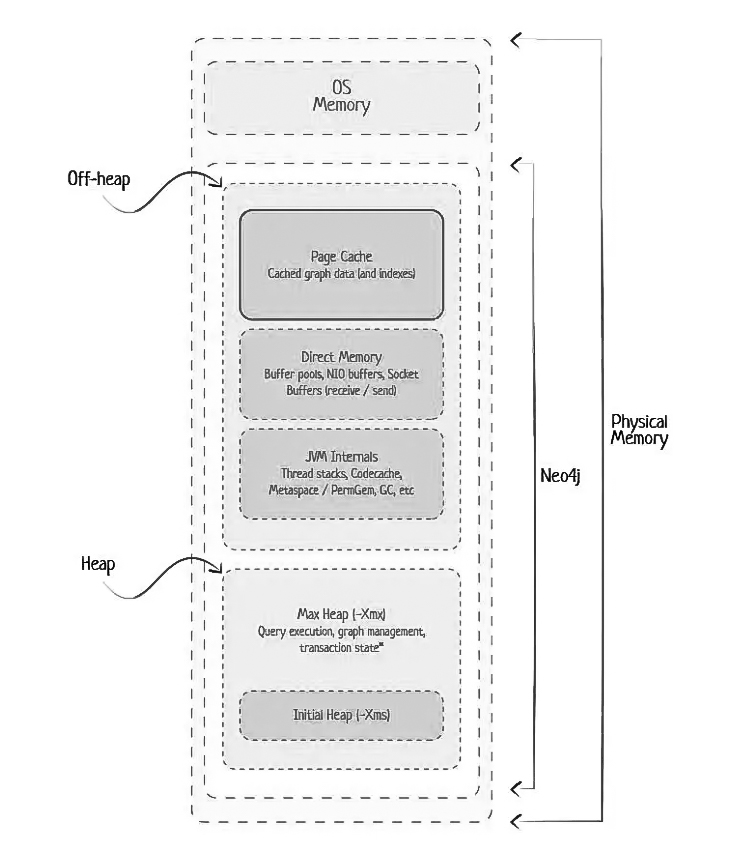
Druga vrsta keša koji Neo4j obezbeđuje je object keš. Zbog toga Neo4j radi u JVM, on održava čvorove i potege kao java objekte u memoriji. Postoje dve memorijske konfiguracije u Neo4j-u:

* JVM red konfiguracija
* Izbor tipa keša

Konfiguracija reda je dosta složena ali najjednostavnije je reći da može da se podesi inicijalna i maksimalan veličina reda, sta se dešava u slučaju pada reda i može se podesiti strategija garbage kolektora. Veliki red znači da možemo drzati veću količinu podataka ali sto je veći red to je duze vreme rada garbage kolektora.

Kod Neo4j je moguće odabrati tip keša, a tipovi su:

* „None“ odnosno bez korišćenja keša
* Soft(inicijalni tip): obezbeđuje optimalno korišćenje dostupne memorije, dobar je za obilaske visokih performansi ali može doći do problema sa garbage kolektorom ukoliko je veliko opterećenje i ako se često korišćeni delovi grafa ne mogu držati u memoriji
* Weak(slabi): obezbeđuje kratki životni vek keširanog objekta, dobar je za aplikacije velikog protoka gde se veliki deo često korišćenog grafa može čuvati u memoriji.
* Strong(jaki): čuva sve učitane podatke u memoriji i nikad ih ne otpušta, dobar je za male setove podataka.
* HPC(high performance cache): keš viskih performansi koji omogućava da se određeni deo memorije rezerviše za učitavanje čvorova irelacija, ova opcija je dostupna samo u enterprise verziji Neo4j-a.



**Transaction logs**

Neo4j pruža podršku za ACID transakcije za svoje operacije, gde ACID označava atomičnost, konzistentnost, izolovanost i izdržljivost, baš kao i relacione baze podataka. ACID transakcije osiguravaju da podaci ostaju konzistentni i pružaju funkcionalnost povratka na osnovu uspeha ili neuspeha transakcija.

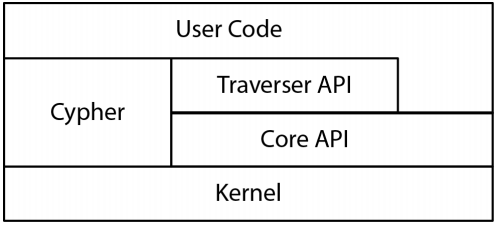
Neo4j podržava ACID transakcije implementacijom evidencije unapred za upisivanje (WAL), u kojoj upisuje promene u aktivnu datoteku evidencije transakcija i fizički spušta te promene na disk pre nego što se primene na skladištene fajlove.

Upisivanje logova omogućava Neo4j-u da zadrži svoju podršku za ACID transakcije. Logovi transakcija se pamte u direktorijumu graf baze podataka, pod nazivom nioneo\_logical.log.\*.

HA(high availability) modul je dostupan samo u enterprise verziji Neo4j-a, i podržava stvari kao što su grupisanje i deljenje podataka.

**Neo4j API**

Da bi programeri mogli da rade sa Neo4j sistemom, Neo4j obezbeđuje nekoliko api-ja koji omogućavaju manipulaciju nad podacima u grafu. Na slici ispod su prikazani ti API-ji koje Neo4j nudi:



Kernel API obezbeđuje funkcije za obradu događaja tokom transakcije. Na taj način je korisniku omogućeno da prati transakcije i reaguje kad se dese željeni događaji.

Core API je Java API koji korisnicima obezbeđuje pristup strukturama koje graf koristi. Koristi lazy lading odnosno pročitane podatke evaluira tek kaf budu zatraženi, prilikom upisa vodi računa o poštovanju ACID principa.

Traversal API korisnicima omogućava da zaštite delove grafa odnosno da deklarišu ograničenja koja npr. onemogućavaju obilazak nekih delova, da odrede koja se ti obilaska koristi da li po dubini ili po širini, ograničavanje relacija koje mogu biti obiđene, definisanje funkcije za evaluaciju prilikom posete čvora.

Cypher je deklarativni jezik sličan SQL-u ali prilagođen graf bazama i ima sintaksu koja omogućava pretragu relacija.

**Neo4j podaci na disku**

Pošto je Neo4j baza podataka bez šeme, koristi fiksne dužine zapisa da bi se sačuvali podaci i pratilo odstupanja u ovim datotekama, da bismo znali kako da preuzmemo podatke da bismo odgovorili na upite. Sledeća tabela ilustruje fiksne veličine koje Neo4j koristi za tip Java objekata koji se čuvaju:

|  |  |  |
| --- | --- | --- |
| Store File | Record size | Contents |
| neostore.nodestore.db | 15 B | Nodes |
| neostore.relationshipstore.db | 34 B | Relationships |
| neostore.propertystore.db | 41 B | Properties for nodes and relationships |
| neostore.propertystore.db.strings | 128 B | Values of string properties |
| neostore.propertystore.db.arrays | 128 B | Values of array properties |
| Indexed Property | 1/3\*AVG(X) | Each index entry is approximately 1/3 of the average property value size |

Karakteristike propertija(svojstva):

* Jedno svojstvo zauzima 32 bajta, taj blok je podeljen u četiri manja bloka od po 8 bajta. Svako polje može da čuva ključ ili vrednost ili oba i ključ i vrednost.
* Ključ i tip zauzimaju 3,5 bajta (ključ 4 bita i tip 24 bita)
* boolean, byte, short, int, char, float se smeštaju u preostale bajtove istog bloka
* small long se takođe smešta u isti blok
* big long ili double se pamte u posebnom bolu(što znači da takvo svojstvo koristi dva bloka)
* referenca na string skladište ili skladište nizova se smešta u isti blok kao i ključ

Podaci koji se čuvaju na disku luvaju se koa ulančane liste sa fiksnom veličinom elementa. Svojstva se pamte kao ulančana lista svojstava, gde svaka sadrži ključ i vrednost i pokazivač na sledeće svojstvo. Svaki čvor i relacija referenciraju svoje prvo svojstvo. Čvorovi takođe referenciraju i prvu relaciju u svom lancu relacija. Svaka relacija referencira početni i krajnji čvor. Takoše referencira i sledeću relaciju za početni i krajnji čvor redom.

Primer preračunavanja neophodnog prostora na disku:

* Pamti se 4M čvorovam svaki od njih ima po 3 svojstva, pamti se i 2M relacija od kojih svaka ima po jedno svojstvo.
* Čvorovi : 4.000.000x15B = 60.000.000B (60MB)
* Relacije: 2.000.000x34B = 68.000.000B (68MB)
* Svojstva: 14.000.000x41B = 574.000.000B (574MB)
* UKUPNO: 703MB

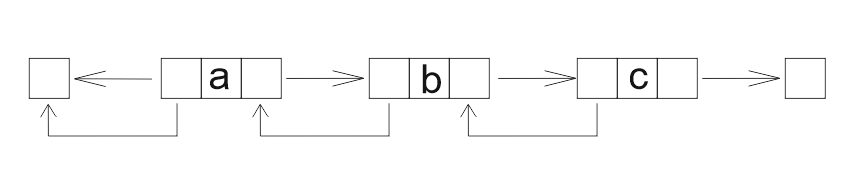
**Indeksi kod Neo4j baze podataka**

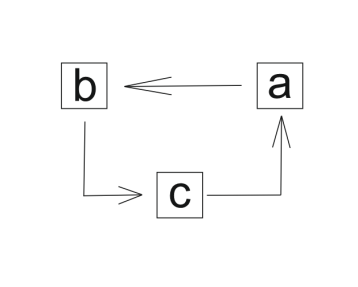
Za DBMS sisteme indeksiranje predstavlja krucialnu stavku kada su u pitanju preformanse. Indeks je struktura koja organizuje podatke na disku tako da njihovo pronalaženje bude lakše, tako što mapira ključeve na posebnoj lokaciji u skladištu. Prilikom rasta baze podataka, povećavaja se I indeks, sto dovodi do pogoršanja performansi kada je upis podataka u pitanju I mogu zauzeti veliki deo prostora u skladištu.

Prednost indeksa je to što ukoliko pretražujemo tabelu po indeksiranoj koloni, kao npr. SELECT \* FROM user WHERE age >= 21, a kolona age je indeksirana kolona, u tom slučaju baza može brzo pronaći odgovarajući red pomoću indeksa I vratiće odgovarajuće redove iz tabele kao odgovor na upit. Nedostatak je to što se za svaku indeksiranu kolonu mora menjati indeks kada ubacujemo, menjamo ili brišemo podatke.

Neo4j indeksiranje se ne oslanja na tradicionalni princip indeksiranja, jer bi to dovelo do značajnog smanjenja performansi. Umeto toga koristi se princip index-free adjacency, što znači da svaki čvor referencira svog suseda direktno, što funkcioniše kao mikro-indeks koji se čuva unutar čvora. Potezi tj. veze između čvorova su dvostruko ulančane liste koje ne referenciraju samo sledeće čvor nego I prethodni.

Korišćenje index-free adjacency je ono što definišeobradu kod nativnih graf baza. Takođe je važno za upite visokih performansi pošto se pomoću tog pristupa postiže konstantne brzina bez obzira na veličinu podataka, u poređenju sa tradicionalnim indeksiranjem kod kog se brzina smanjuje sa porastom veličine podataka. Slika ispod prikazuje dvostruko ulančane liste koje predstavljaju veze između čvorova i čvorove odnosno ulančanu listu gde svaki čvor referencira svog suseda.





Način obilaska grafa kod Neo4j-a:

* Kada pristigne zahtev, pokazivač se postavlja na prvi element u globalnom indeksu
* Nakon toga se računa u bajtovim pomeraj tako što se množi veličina u bajtovima početnog čvora ili identifikator relacije sa veličinom skladišta čvorova da bi se našla početna adresa čvora
* Posle toga u zahtevu traži veličinu relacija i svojstava, koji su povezani sa čvorom

Relacija u sebi čuva adresu početnog čvora, krajnjeg čvora, i povezane relacije i svojstva , u oba smera i sledeći i prethodni. Referenciranjem podataka direktno u memoriji korišćenjem pokazivača, i iteriranjem kroz ulančanu listu pomeranjem sa pokazivača na pokazivač (referenciranjem), redukuje se broj čitanja sa diska i značajno se ubrzava izvršenje upita.

Tipovi indeksa

Neo4j koristi tri tipa indeksa to su:

* B-stablo, indeksi koji mapiraju svojstva prema čvoru ili vezi i izvršavaju tačnu pretragu indeksa na svim tipovima. Indeski b-stabla su jedna od najpopularnijih struktura, česta su kod drugih baza podataka, i poznati su po tome da obezbeđuju efikasno korišćenje hardvera i izvršavanje operacija. Teže tome da smanje veličinu stabla, tako što omogućavaju uređeni sekvencijalni pristup. Kod Neo4j-a su dizajnirani tako da mapiraju tačnu vrednost svojstva na čvor ili relaciju i da tako omoguće brži prolazak kroz graf.
* Indeksi koji koriste tokene, kreiraju se za sve čvorove i relacije po difoltu. Često je bolje koristiti indekse b-stabla ali tokeni su dostupni po difoltu. Ovi indeksi omogućavaju pronalaženje čvora pomoću njegovih oznaka što zaobilazi skeniranje i filtriranje svih oznaka čvora. Uprkos činjenici da se performanse poboljšavaju kada se odbace token indeksi, u nekim situacijama oni mogu biti od koristi.
* Tekst indeksi koriste se kod pretraživanja tekstova.

Neo4j obezbeđuje sledeće mehanizme za indeksiranje podataka, a to su:

* Ručno upravljanje indeksima
* Šema indeksa
* Automatsko indeksiranje

Za pretragu Neo4j koristi Apache Lucene i omogućava programerima pristup metodama koje obezbeđuju ručno upravljanje pretragom indeksa. Neo4j definiše indekse po imenu tako da programeri imaju slobodu da povežu indeksirane podatke sa kojim god imenom zele. Na primer, indeks za korisnike može biti „users“ i može mu se pristupati automatski pomoću IndexManagera koji ima graf baza podataka:

*IndexManager IndexManage = graphDB.index()*

*Index userIndex= indexManager.forNodes(„users“);*

Sa ovakvim indeksom, programer može da kreira čvor i onda da doda index za čvor.

*Node person = graphDB.createNode();*

*person.setProperty( “name”, “Steven” );*

*person.setProperty( “email”, “steve@here.com” );*

*userIndex.add( person, “email”, “steve@here.com” );*

Ovaj primer prikazuje kreiranje osobe, podešavanje imena i email-a, i nakon toga dodavanje indeksa za osobu na osnovu email-a. Nakon kreiranja indeksa pronalaženje čvora je dosta brže neko prilikom provere svakog elementa u grafu.

IndexHits indexHits = userIndex.get( “email”, “steve@here.com” );

Node user = indexHits.getSingle();

Ovim upitom je zatraženo da baza podataka vrati sve čvorove čija je email adresa „steve“here.com“. Nakon poziva ove funkcije vraćena je instanca IndexHits pomoću koje se pristupa čvorovima sa traženom email adresom. U konkretnom slučaju se očekuje samo jedan rezultat pa se poziva funkcija getSingle() da bi se pristupilo čvoru.

Izazov kod ručnog upravljanja indeksima je to što programer mora da vodi računa o kreiranju indeksa, promene indeksa, brisanja. To ostavlja dosta odgovornosti na aplikaciji, što nije uvek dobro rešenje.

Automatsko indeksiranje može biti šematsko indeksiranje ili auto indeksiranje.

Šematsko indeksiranje dozvoljava automatsko kreiranje indeksa za čvor i za set propertija. Na primer, ako imamo labelu „USER“ onda se može podesiti Neo4j tako da kreira šemu indeksi za sve čvorove koji sadrže labelu „USER“ na osnovu propertija „email“. Nakon toga će Neo4j upravljati indeksima slično kao kod relacione baze podataka.

Kada se doda novi čvor koji ima labelu „USER“ dodaće se novi indeks za taj properti email tog čvora, ako se izmeni email čvora Neo4j će izmeniti indeks, ako se obriše čvor Neo4j će automatski obrisati indeks.

Auto indeksiranje, može automatski da kreira indekse za čvorove ili za relacije i za specifična svojstva. Na primer, Neo4j se može podesiti tako da automatski kreira indekse za čvorove koji imaju name svojstvo. Nedostatak je to što trenutno ne postoji način da se zada šema na osnovu koje će automatski da se kreiraju indeksi, pa će automatsko indeksiranje mešati čvorove svih oznaka. Na primer, ako imamo čvorove koji su označeni kako MOVIE i čvorove koji su označeni kao USER, oba imaju atribut ime, automatsko indeksiranje će pomešati zajedno filmove i korisnike.

Da bi se postigle što bolje performanse potrebno je koristiti odgovarajuće indeksiranje sa odgovarajućom arhitekturom skladišta.

**Zaključak**

Neo4j je jedna od najpopularnijih graf baza podataka. U ovom radu je data osnova funkcionisanja i strukture samog sistema što je dovoljno da se stekne jasna slika kako funkcioniše Neo4j. Predočene su prednosti korišćenja graf baze podataka umesto klasične relacione za rešavanje konkretnih domenskih problema, performanse prolaska kroz graf su dosta bolje u odnosu na SQL JOIN operaciju. Iako u nekim slučajevima NoSql baze predstavljaju mnogo bolje rešenje u odnosu na klasične relacione, nije moguće totalno izbaciti iz upotreba relacione baze podataka, ipak se najčešće sreće zajedničko korišćenje oba tipa baza podataka u nekim sistemima.

**Literatura**

[1]<https://www.graphable.ai/blog/neo4j-performance/>

[2]<https://neo4j.com/docs/getting-started/current/graphdb-concepts/>

[3]<https://neo4j.com/blog/native-vs-non-native-graph-technology/>

[4]<https://en.wikipedia.org/wiki/Database>

[5]https://en.wikipedia.org/wiki/Graph\_database

[6]https://www.graphable.ai/blog/neo4j-performance/

[7]https://blog.turbonomic.com/blog/on-technology/devs-are-from-venus-ops-are-from-mars-big-data-neo4j-part-3