**SEMINARSKI RAD:**

**Sistemi za upravljanje bazama podataka**

**Tema:** **Oporavak Neo4j baze podataka**

Student:

Aleksandar Cenic 1062

S А D R Ž А Ј

**Strana**

[1. U](#_Toc191445171)vod [3](#_Toc191445171)

[2.](#_Toc191445172) Transakcije [4](#_Toc191445172)

[2.1.](#_Toc191445173) Transakcije kod Neo4j-a [4](#_Toc191445173)

2.1.1. Stanje transakcije u memori 4

2.1.2. HTTP API 4

[2.2.](#_Toc191445174) Menadžment transakcija [8](#_Toc191445174)

2.2.1. Nivo izolacije 8

2.2.2. Deadlocks 9

2.3. Log transakcije 9

[3.Back-up i oporavak](#_Toc191445175) 11

[3.1.Full back-up](#_Toc191445176) 12

[3.2.](#_Toc191445177) Incremental backup [17](#_Toc191445177)

[3.3.](#_Toc191445177) Oporavak kod Neo4j-a …..……………………………………………………………18

[4. Zaključak 19](#_Toc191445180)

[Literatura 2](#_Toc191445183)0

# 

# 1. uvod

Čuvanje informacija i važnih podataka, predstavljalo je ljudima veliki izazov još od davnih dana. Kako su se one čuvale u knjigama i raznim starim spisima, tako je vremenom hartija gubila na kvalitetu i polako se raspadala. Zbog očuvanja informacija iz starih knjiga i spisa ljudi su ih prepisivali u druge knjige i spise. Na žalost, neke knjige i spisi nisu prepisani i njihovi ostaci su u jako lošem stanju, pa se arheolozi koriste raznim metodama restauracije ne bi li uspeli da rekonstruišu dokumente. U današnje vreme, informacije se zapisuju na drugačiji način, digitalno. Digitalni zapis informacija u bazama podataka je dosta pouzdani, ali nije maksimalno siguran. Kod baza podataka se može dogoditi pad baze iz više razloga, tako da analogno prepisivalju knjiga i spisa iz davnih vremena, baze podataka koriste sigurnosne kopije (backup). Analogno restauraciji dokumenata, baze podataka koriste metode oporavka.

U ovom radu biće obrađena Neo4j baza podataka, backup i oporavak.

# 2. Transakcije

Transakcija nad bazom podataka se sastoji od jedinice rada izvršene u okviru sistema za upravljanje bazama podataka (ili sličnog sistema), takve da se tretira na koherentan i pouzdan način, nezavisna od drugih transakcija. Transakcije nad bazama podataka imaju dve glavne svrhe:

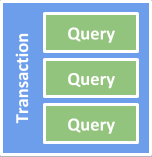
1. Da omoguće pouzdano izvršavanje jedinica rada kod takvih da je moguć oporavak od greški, i da baza podataka ostaje konzistentna čak i u slučaju sistemskih grešaka, kada se izvršavanje prekine (u potpunosti ili delimično) i mnoge operacije nad bazom ostanu nedovršene ili sa nejasnim statusom.
2. Da omoguće izolaciju među programima koji istovremeno pristupaju bazi podataka. Bez izolacije, rezultati programa mogu da budu pogrešni.

Transakcija nad bazom podataka po definiciji mora da bude atomična (engl. atomic), konzistentna (engl. consistent), izolovana (engl. isolated) i trajna (engl. durable). Skraćenica koja označava ovaj skup osobina je ACID (od početnih slova engleskih reči za ove četiri osobine).

Transakcije funkcionišu po principu sve ili ništa, što znači da će se jedinica rada koja čini transakciju ili u potpunosti izvršiti nad bazom podataka ili neće imati nikakvog efekta. Štaviše, sistem mora da izoluje svaku transakciju od ostalih transakcija, rezultati moraju da budu u skladu sa postojećim ograničenjima u bazi podataka, i transakcije koje su uspešno izvršene moraju da budu zabeležene u trajnoj skladišnoj memoriji.

**2.1. Transakcije kod Neo4j-a**

Neo4j baza podataka za upite koristi Cypher upitni jezik. Svaki izraz u Cypher upitnom jeziku se eksplicitno izvršava unutar transakcije. Kao što smo već rekli, transakcija je atomična jedinica izvršenja koja sadrži jedan ili više Cypher upita.



Slika:1

Transakcije mogu sadržati upite za čitanje i upis podataka. Upiti koji su zaduženi samo za čitanje podataka, uvek će se izvršiti uspešno. Sa druge strane, za upite koji ažuriraju podatke, neuspešno izvšenje se može desiti iz nekog razloga. Na primer, ako upit pokušava da prekrši neko ograničenje, u tom slučaju se cela transakcija vraća nazad i nema promena u grafu.

Kako se svaki Cypher izraz izvršava unutar konteksta same transakcije, ništa od podataka neće biti prisutno na disku sve dok se transakcija ne potvrdi. Pošto nije moguće izvršiti Cypher upit van transakcije, moguće je izvršiti više upita kroz jednu jedinu transakciju u pomoć sekvence operacija:

1. Otvoranje transakcije.
2. Pokretanje više ažurirajućih upita.
3. Potvrđenje svih upita.

**2.1.1. Stanje transakcije u memori**

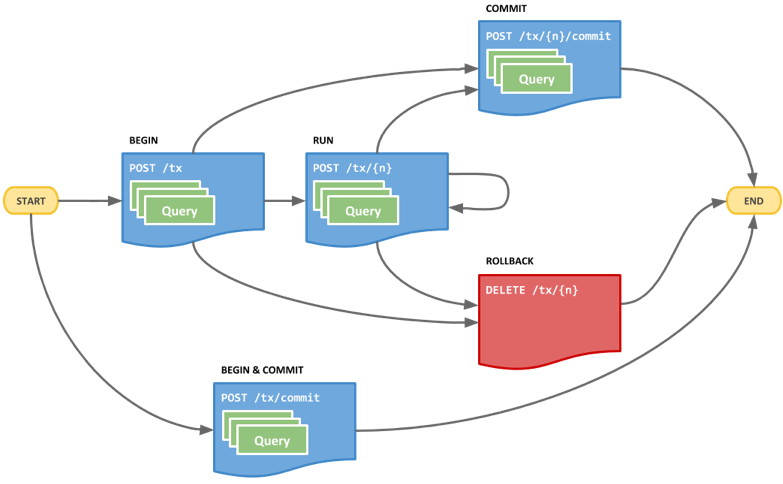
Transakcija će sve promene držati u memori sve dok upit ili ceo set upita ne bude stigao do kraja sa izvršavanjem. Takav tip podatka koji transakcija drži u memori naziva se stanje transakcije. Upiti koji vrše samo čitanje podataka, ne zahtevaju alokaciju memorije za stanje transakcije.

Po default-u, stanje transakcije je alocirano „off-heap“. Kada je stanje transakcije alocirano off-heap, maksimalna veličina stanja transakcije može se definisati uz pomoć parametra: **dbms.tx\_state.max\_off\_heap\_memory**. Memorija u kojoj se nalazi stanje transakcije nije unapred alocirana, već će rasti i skupljati se u skladu sa aktivnostima baze podataka. Kada se stanje transakcije drži off-heap, to predstavlja benificiju za aplikacije koje karakteriše intenzivni upisi velikog broja podataka.

Stanje transakcije se može alocirati i na heap-u setovanjem parametra: **dbms.tx\_state.memory\_allocation** na ON\_HEAP. Ako je konfigurisano da se stanje transakcije čuva na heap-u, ne može se specificirati njegova maksimalna veličina.

**2.1.2. HTTP API**

Transakcijama Cypher upitnog jezika se upravlja preko nekoliko različitih URI- ova, koji su dizajnirani za korišćenje kroz propisani obrazac. Obezbeđeni su objekti koji vrše celi transakcioni ciklus kroz jedan ili više HTTP zahteva.

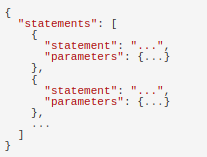
****



Slika:2

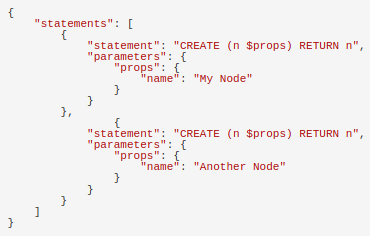
Na predhodnoj slici prikazan je tok transakcije, gde svaki od blokova predstavlja zasebni HTTP zahtev. Stanje svake transakcije je pohranjeno na serveru na kome je započeta. Period života transakcije automacki se žavršava nakon isteka vremena neaktivnosti. Po default-u, vreme neaktivnosti transakcije iznosi 60 sekundi. Ako bi smo želeli da održimo transakciju u životu, bez potvrđenja novih upita, ili sa „praznim zahtevom“ to radimo kroz /tx/{n} URI.

Payload koj se, kao JSON objekat, šalje uz HTTP POST zahtev ima strukturu prikazanu na slici 3.



Slika:3

Konkretan primer JSON objekta koji se šalje uz zahtev prikazan je na slici 4.



Slika:4

Parametri su predstavljeni kao ključ-vrednost parovi, gde je svaka od vrednosti parametara mapirana na određeni tip podatka.

Kao rezultat upita JSON objekat ima strukturu prikazanu na slici 5.



Slika:5

Izvršenje upita: **UNWIND range(0,2,1) as number RETURN number** bi vratilo rezultat prikazan na slici 6.



Slika:6

Detalnije informacije o Neo4j HTTP API-ju nalaze se u dokumentaciji na sledećoj web adresi: <https://neo4j.com/docs/http-api/4.0/actions/>.

**2.2. Menadžment transakcija**

Transakcije kod Neo4j baze podataka su jedno nitne, ograničene i nezavisne, tako da više transakcija se može započeti u jednoj niti i one će biti potpuno nezavisne jedna od druge.

Interakcioni ciklus rada sa transakcijama se odvija na sledeći način:

1. Početak transakcije.
2. Izvršenje operacija nad bazom podataka.
3. Potvrđenje ili vraćanje transakcije.

Neo4j deklariše transakciju kao jedan od resursa, potom startuje transakciju i pokušava da izvrši operacije nad grafom. Poslednja operacija u **try** bloku treba da potvrdi ili odbaci (roll back) transakciju. U ovakvom scenariju, **try-with-resources** mehanizam se koristi kako bi štitio protiv neočekivanih izuzetaka i kako bi osigurao da se transakcija zatvori bez obzira na to šta se dešava unutar transakcije. Sve nepotvđene transakcije biće vraćene kao deo resursa za čišćenje na kraju transakcije. U slučaju da je transakcija eksplicitno potvrđena ili vraćena, čišćenje resursa se neće zahtevati, već će transakcija biti zatvorena kao prazna operacija.

**2.2.1. Nivoi izolacije**

Transakcije kod Neo4j baze podataka koristie pročitani-potvrđeni (read-committed) nivo izolacije, što znači da će se podaci videti skoro po potvrđenju i neće se videti podaci u drugim transakcijama sve dok ne budu potvrđene. Ovakav tip izolacije je slabiji u odnosu na serijalizaciju, ali pruža značajnu prednost u performansama.

Neo4j Java API omogućava eksplicitno zaključavanje čvorova i relacija. Korišćenje lock-ova daje priliku simulacije višeg nivoa izolacije kroz dobijanje i oslobađanje lock-a. Na primer, ako je lock zaključan na zajedničkom čvoru ili zajedničkoj relaciji, onda će sve transakcije biti serijalizovane na tom lock-u, dajući efekat serijalizaciskog izolacionog nivoa.

Kod Cypher upitnog jezika moguće je dobiti lock kako bi simulirali poboljšani nivo izolacije u pojedinim slučajevima. Razmotrimo slučaj gde više konkurentnih Cypher upita inkrementira vrednost property-ja. Zbog ograničenja read-committed nivoa izolacije, inkrementiranje možda neće tezultirati očekivanom vrednošću. Ako postoji direktna zavisnost, onda će Cypher automacki dobiti lock pre čitanja. Direktna zavisnost se pojavljuje kada **SET** klauzula u izrazu sa svoje desne strane ima zavisno čitanje property-ja ili u vrednosti ključ-vrednost para u literal mapi.

Na primer, ako se sledeći upit bude pokrenuo od 100 konkurentnih klijenata, vrlo je verovatno da neće inkrementirati vrednost za 100 osim ako ne dobije lock pre čitanja vrednosti property-ja. Ovo se dešava zato što će svi upiti pročitati vrednost **n.prop** u njihovoj transakciji i neće videti inkrementiranu vrednost od bilo koje druge transakcije koja nije još potvrđena. Navešćemo nekoliko primera ovakvih upita.

1. U ovom primeru se zahteva lock tako da će Cypher automacki dobiti jedan:  
 **MATCH (n:X {id: 42})**

**SET n.prop = n.prop + 1**

2.I u ovom primeru imamo istu situaciju:  
 **MATCH (n)**

**SET n += { prop: n.prop + 1 }**

**2.2.2. Deadlocks**

Kako su u upotrebi lock-ovi sasvim je moguće da se desi deadlock. Neo4j će detektovati bilo kakav deadlock (prouzrokovam dobijanjem locka) pre nego što se dogode i baciti izuzetak. Pre nego se baci izuzetak, transakcija će biti označena kao roll back. Svi lock-ovi dodeljeni transakciji bivalju zadržani sve dok transakcija ne završi, po završetku će biti oslobođeni. Kada se lock-ovi oslobode, transakcije koje čekaju na određene lock-ove drzane od strane završene transakcije mogu se procesuirati.

Doživljavanje učestalih deadlock-ova daje indikaciju da se dešavaju konkurentni zahtevi za upisom. U tom slučaju nije ih moguće izvršiti a da se ne ispune planirana izolacija i delatnost. Solucija je da se osigura dešavanje konkurentnih update-ova na razuman način. Na primer, ako imamo dva jedinstvena čvora A i B dodavanje ili brisanje relacije na oba čvora po bilo kom redosledu, za svaku transakciju rezultiraće deadlock-om ako postoje dve ili više transakcija koje to rade konkurentno. Jedna od opcija da se osigura da će se update-ovi dešavati uvek u istom redosledu (prvo A nakon toga B). Druga opcija je da se osigura da svaka od transakcija nema konfliktnih upisa ka čvoru ili relaciji. Ovo se može postići puštanjem jedne niti da obavi sve update-ove.

**2.3. Log transakcije**

Log transakcije čuva sve operacije upisa u bazu podataka. To uključuje sva dodavanja i modifikacije nad podacima, kao i dodavanje i modifikacija indeksa i ograničenja. Log-ovi transakcija su „izvor istine“ u slučajevima kada je potreban oporavak bazi podataka. Oni se koriste kako bi obezbedili inkrementalni back-up kao i za klaster operacije.

Po defaultu, log-ovi transakcija se nalaze na lokaciji: **<neo4j-home>/data/transactions/<database-name>**. Svaka baza podataka sadrži svoj direktorijum sa log-ovima transakcija. Lokacija root direktorijuma sa folderima je konfigurisan od: **bms.directories.transaction.logs.root**. Za maksimalne performanse preporučljivo je da se konfiguriše da transakcioni log-ovi budu smešteni na pouzdanom uređaju.

Log zamena se konfiguriše parametrom **dbms.tx\_log.rotation.size**. Po default-u, log zamena se dešava kada veličina loga nadmaši 250MB.

Postoji nekoliko različitih načina za kontrolu zadržane količine transakcionih log-ova koristeći parametar **dbms.tx\_log.rotation.retention\_policy**. Ovaj parametar se može konfigurirati na dva različita načina:

1. **dbms.tx\_log.rotation.retention\_policy=<true/false>**

Ako je ovaj parametar setovan na true transakcioni logovi biće sačuvani na određeno vreme. Ova operacija nije preporučvljia zbog efikasnog neograničenog skladištenja. Stari log-ovi transakcije ne mogu se sigurno arhivirati ili ukloniti sa eksternim poslovima od sigurnog rezanja log fajlova, jer zahtevaju znanje o najčešćim uspešnim checkpoint-ima.

Ako je ovaj parametar setovan na false samo najčešće korišćeni transakcioni log-ovi biće sačuvani. Ova operacija nije preporučljiva u production Enterprise Edition okruženju kao inkrementalni back-up, oslanja se na transakcione logove iz predhodnog back-up-a.

2. **dbms.tx\_log.rotation.retention\_policy=<amount> <type>**

Na ovaj način možemo definisati koliku količinu log fajlova želimo da zadržimo i koji tip fajla.

Rezanje transakcionog log-a odnosi se na sigurno automacko rezanje ili uklanjanje nepotrebnih transakcionih log fajlova. Transakcioni log se može odrezati ili ukloniti kada jedan ili više fajlova ispadne iz konfigurisane politike zadržavanja. Dve stvari su potrebne kako bi fajl bio ukloljen:

1. Fajl mora biti zamenjen (rotiran).
2. Mora bar jedan checkpoint da se dogodi u skorašnjem log fajlu.

Opažanje da ima dosta više logova nego što je očekivano, to je najverovatnije zato što se checkpoint-i ne dešavaju dovoljno učestalo ili traju predugo.Ovo je privremeni uslov i jaz između očekivanog i osluškivanog broja log fajlova koji će biti zatvoreni kod sledećeg uspešnog checkepoint-a. Interval između checkpoint-a može biti konfigurisan koristeći **dbms.checkpoint.interval.time** i **dbms.checkpoint.interval.tx**

# 3. back-up i oporavak

U IT industriji, back-up (sigurnosna kopija) ili proces back-up-a se odnosi na kopije ili kopiranje podataka sa svrhom da se iz tih kopija mogu vratiti podaci u slučaju kada se originali oštete i postanu nedostupni. Back-up kod Neo4j klijenta se može može izvršavati u dva neznato različita moda:

1. Full back-up.
2. Incremental backup.

Full back-up se uvek zahteva inicijalno za baš prvi back-up na odabranoj lokaciji. Naknadni back-up će pokušati da koristi inkrementalni mod gde se samo koristi delta od transakcionih log-ova dok je poslednji back-up prebačen i prihvaćen na odabranoj lokaciji. Ako zahtevani transakcioni log-ovi se ne nalaze na serveru, onda će klijent pribeći izvođenju full back-up-a osim ako parametar **--fallback-to-full** nije uključen.

Nakon što je back-up uspešno izveden proveravač koenzistentnosti biva ubačen po default-u. Proveravanje koenzistentnosti kod back-up-a predstavlja jednu od glavnih operacija koja može upotrebiti značajne računarske resurse kao što je memorija, CPU I/O.

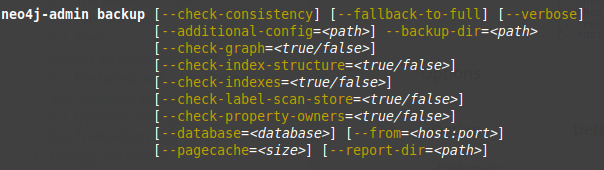
Veoma je obeshrabrujuće pokretati back-up klijenta na live Neo4j serveru, pogotovo zajedno sa proverom koenzistentnosti. Takva radnja može negativno uticati na server. Da bi se izbegao negativan uticaj na pokrenuti server sa zahtevima resursa od strane back-up klijenta, preporučuje se da se back-up i provera koenzistentnosti radi na stabilnoj mašini koja ima dovoljno slobodnih resursa da izvrši proveru koenzistentnosti. Druga alternativa je razdvajanje back-up-a od provere koenzistentnosti i rasporediti da se taj deo rada obavi kasnije na stabilnoj mašini. Vrednosti provere koenzistentnosti kod back-up ne bi trebalo zanemariti, one su vitalne za sigurno čuvanje i osiguraće kvalitet podataka.

Transakcioni log fajlovi su smenjeni i odrezani u zavisnosti od konfiguracije. Na primer, postavljajući **dbms.tx\_log.rotation.retention\_policy**=3 files zadržaće 3 transakciona log fajla u back-up-u.

Konfiguracija memorije se postiže setovanjem promenjljive okruženja HEAP\_SIZE pre startovanja back-up programa. Ako nije postavljena promenjljiva HEAP\_SIZE, onda će java Virtualna Mašina da izabere vrednost baziranu na resursima samog servera. HEAP\_SIZE konfiguriše maksimalnu veličinu alociranu na heap-u potrebnu za back-up proces.

Veličina page cache-a se može odlučiti za back-up proces koristeći opciju –**pagecache**. Ako nije eksplicitno definisano veličina page cache-a će biti 8MB.

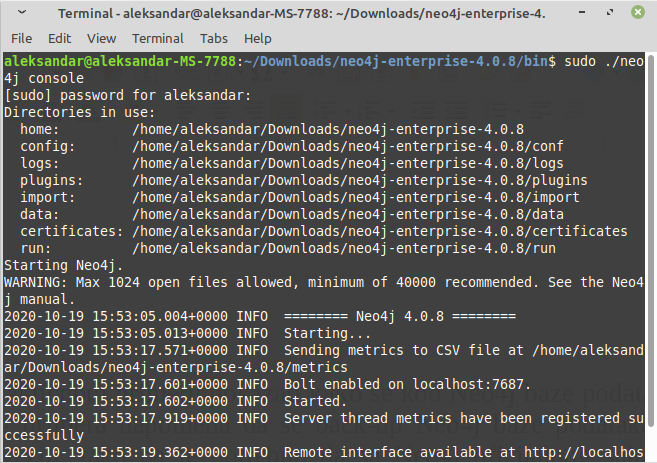
Kod Neo4j baze podataka može se napraviti sigurnosna kopija (back-up) u online modu uz pomoć **backup** komande skripte **neo4j-admin**.



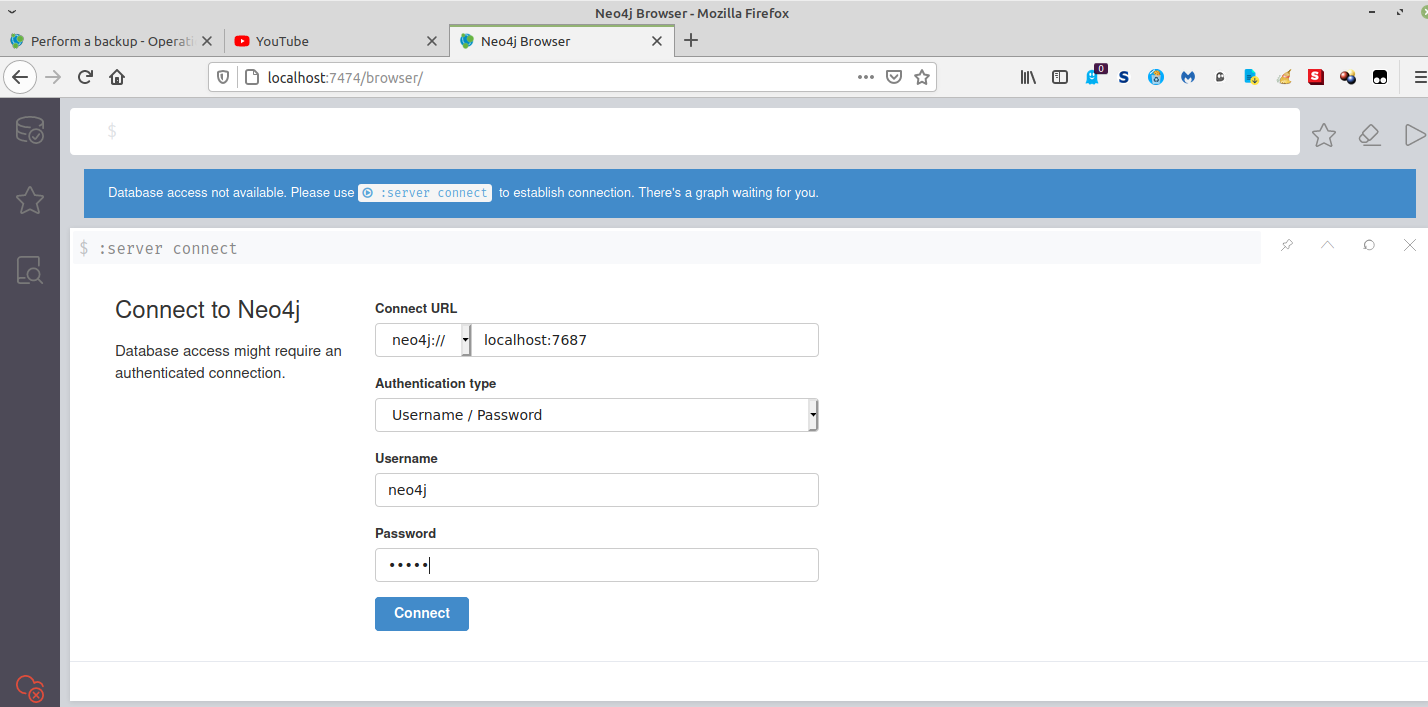
Slika:7

## 3.1. Full back-up

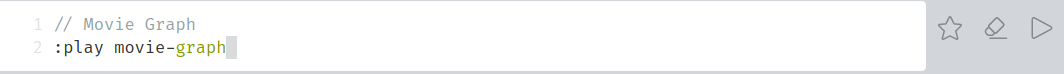
U ovom pod poglavlju ćemo na praktičnom primeru pokazati kako se kod Neo4j baze podata izvršava Full back-up. Pre samog primera napomena da se back-up Neo4j baze podataka može vršiti samo u **Enterprise** veziji, tako da je potrebno da sa Neo4j oficijalnog sajta skinemo i pokrenemo **Enterprise** verzijubaze.

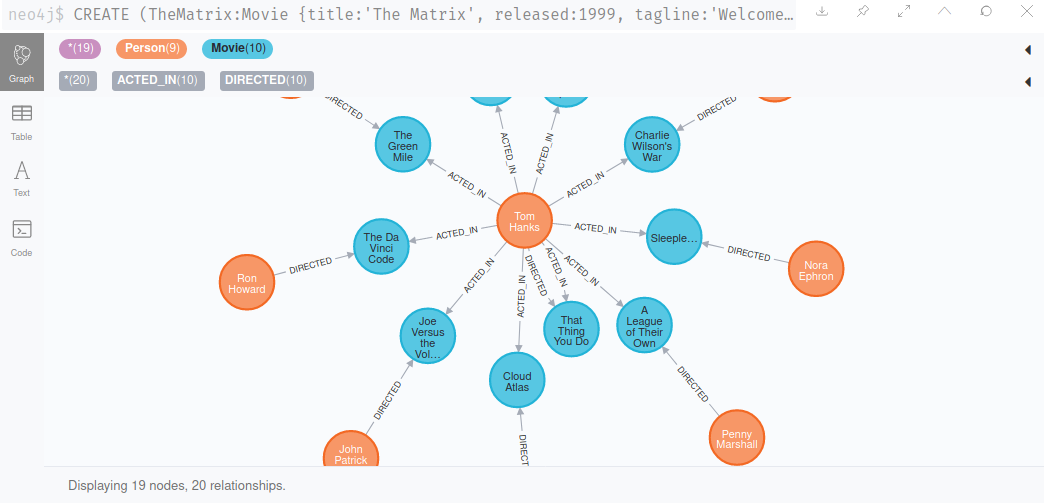
Slika:8

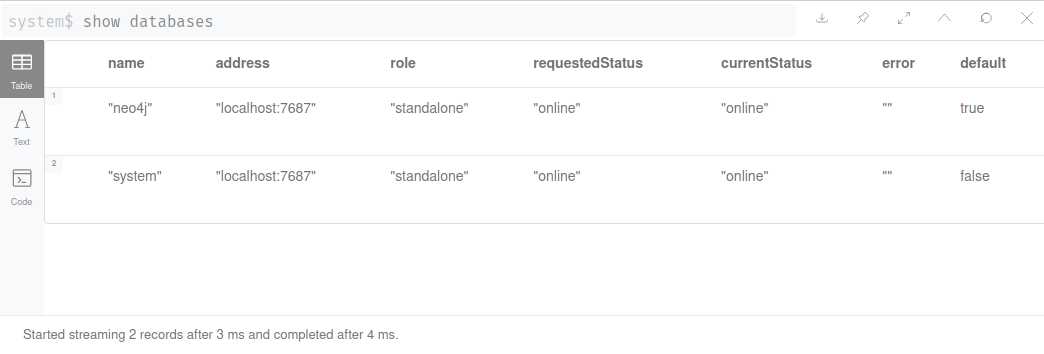
Kada smo pokrenuli Neo4j možemo pristupiti korisničkom interfejsu kroz naš web pretraživač putem adrese: <http://localhost:7474/>.

Slika:9

Za praktičan primer back-up-a Neo4j baze uzećemo Movie graf koji dolazi uz sam Neo4j kao pokazni primer grafa.

Slika:10

Slika:11

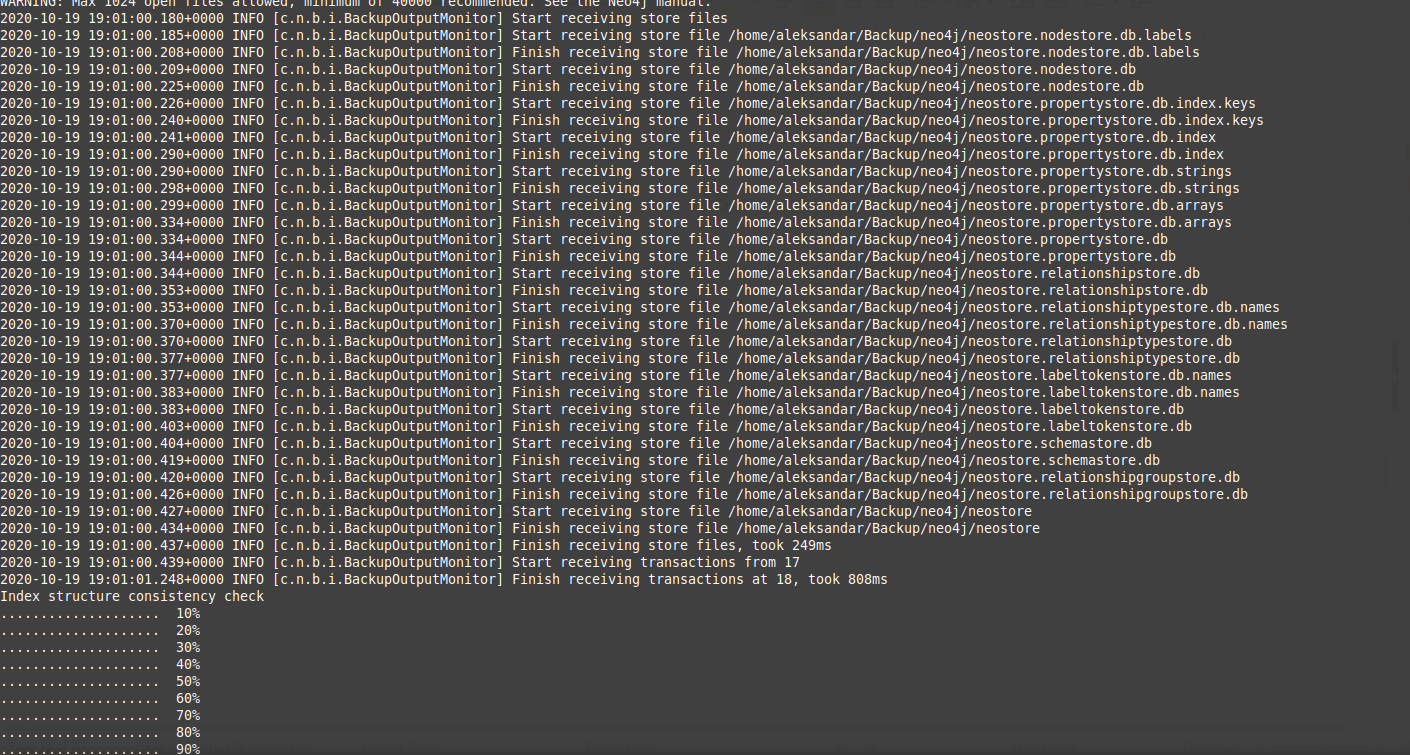
Slika:12

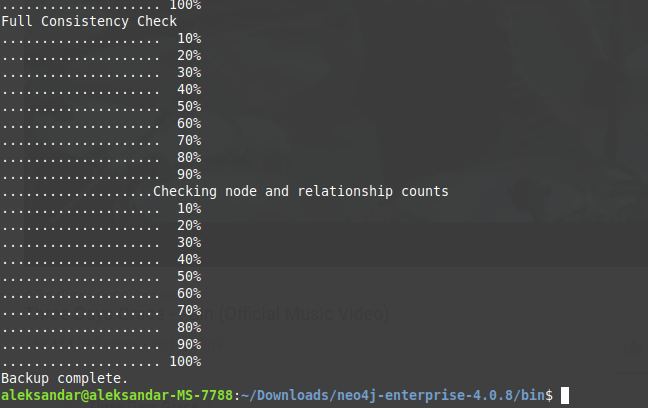
Sada ćemo izvršiti backup nad bazom podataka. Kako se back-up vrši po prvi put automacki je to biti full-back-up. Back-up vršimo komandom: **neo4j-admin backup**.

Izvršavamo back-up komandom: ./neo4j-admin backup --backup-dir=/home/aleksandar/Backup/ --from=localhost:6362 –database=neo4j

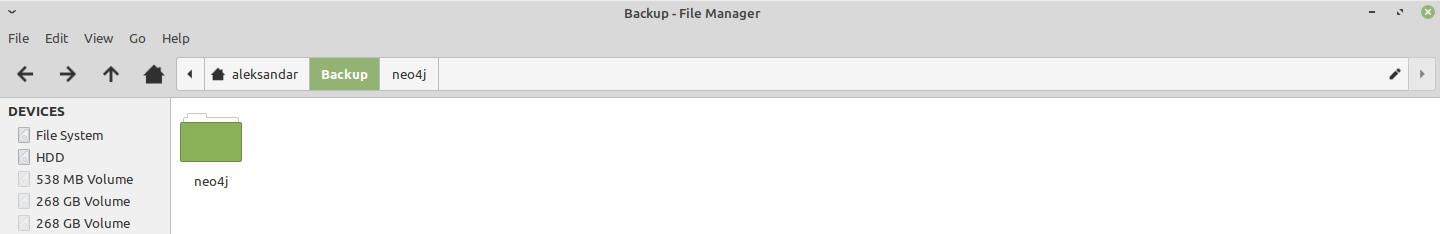
Slika:13

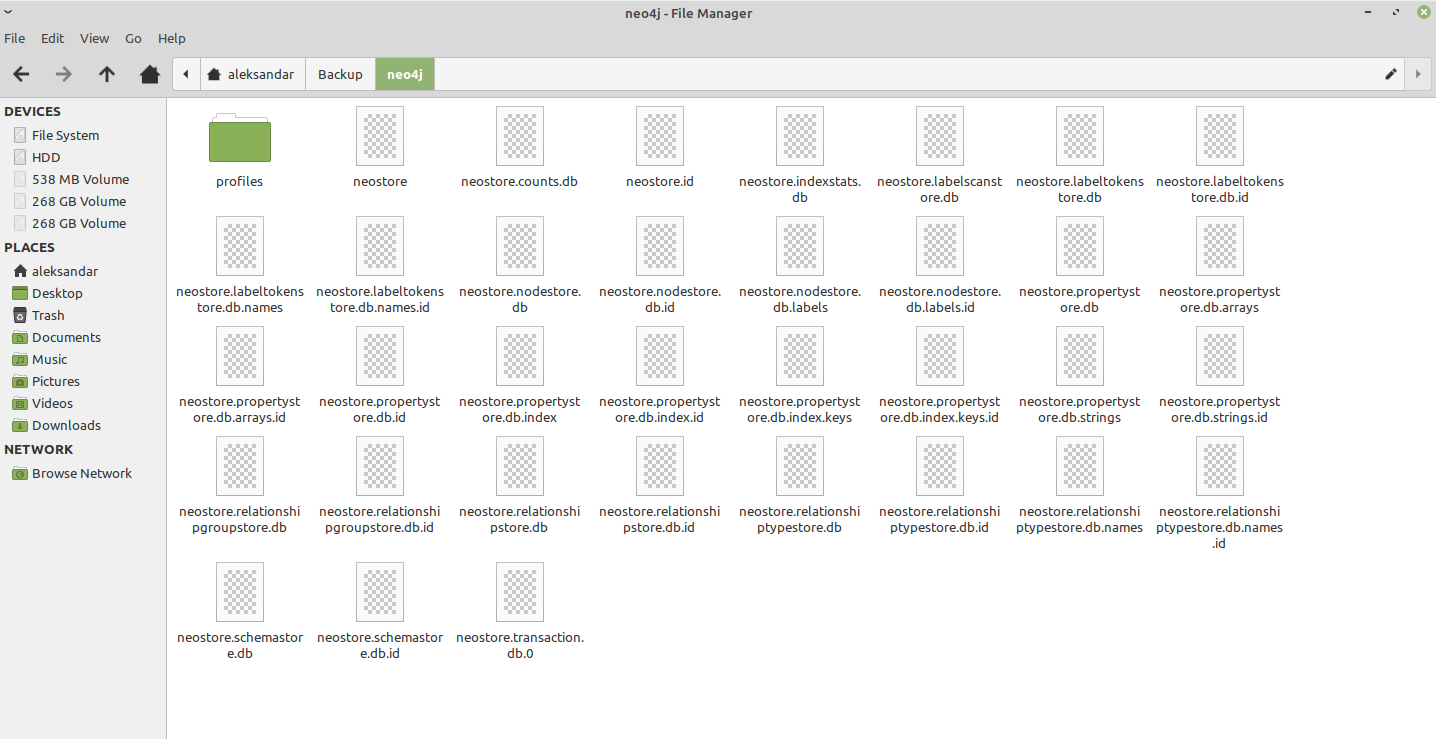
Kao izlaz dobijamo sledeće:

Slika:14

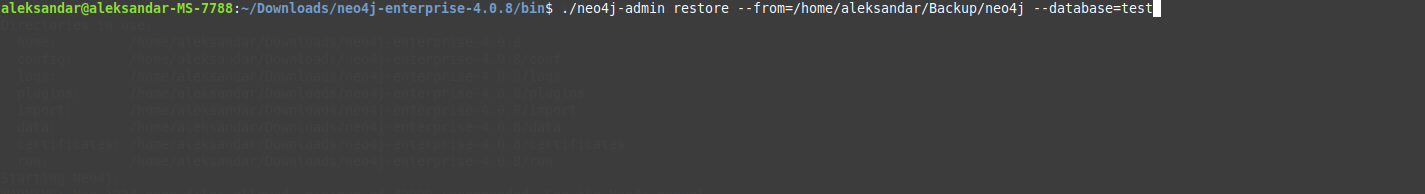
Slika:15

Da bi smo se uverili da je back-up uspešno završen proverimo backup folder tu bi trebao da se nalazi folder pod imenom neo4j sa podacima.

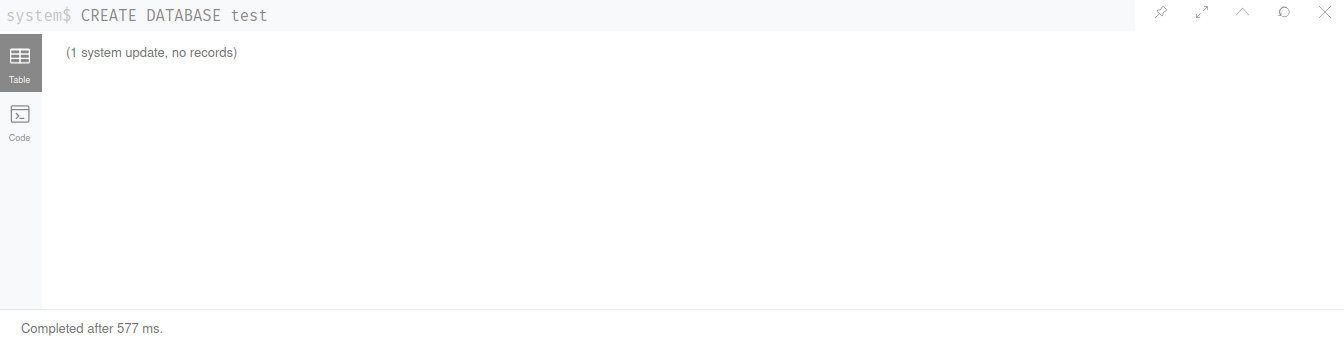
Slika:16

Slika:17

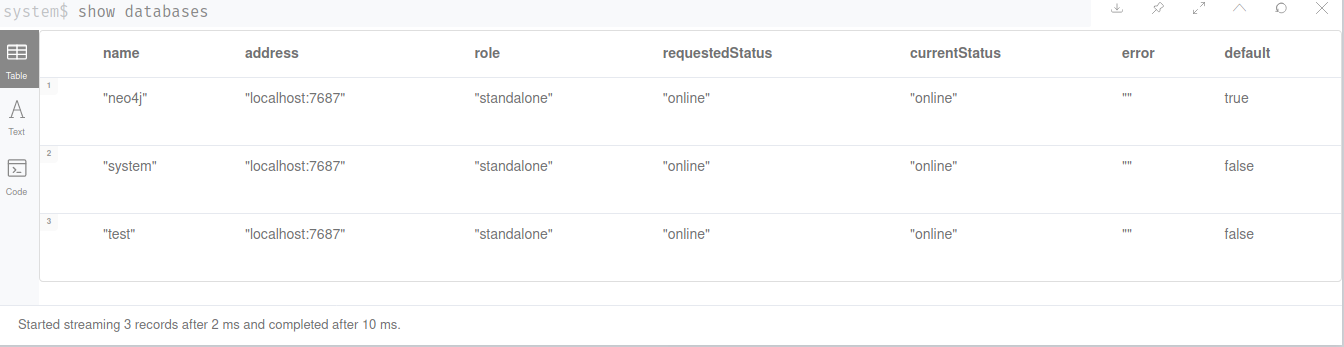
Sada ćemo izvršiti povratak podataka iz back-up-a. To možemo da učinimo komandom **neo4j-admin restore**. Izvršavamo komandu: ./neo4j-admin restore --from=/home/aleksandar/Backup/neo4j –database=test . Neo4j server prilikom procesa povratka podataka treba biti isključen!

Slika:18

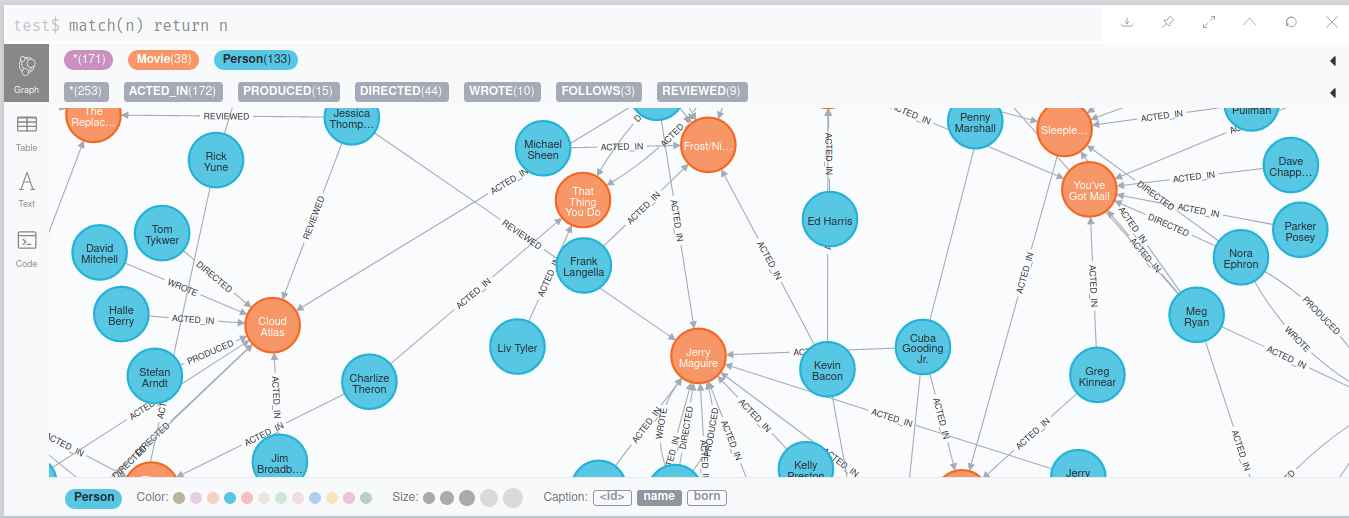
Podaci iz back-up-a su smešteni u novoj bazi pod nazivom test. Povratak podataka iz back-up-a ne kreira automacki bazu podataka, već ćemo morati da mi kreiramo bazu pod istim imenom test.

Slika:19

Možemo videti da sada baza test postoji u DBMS.

Slika:20

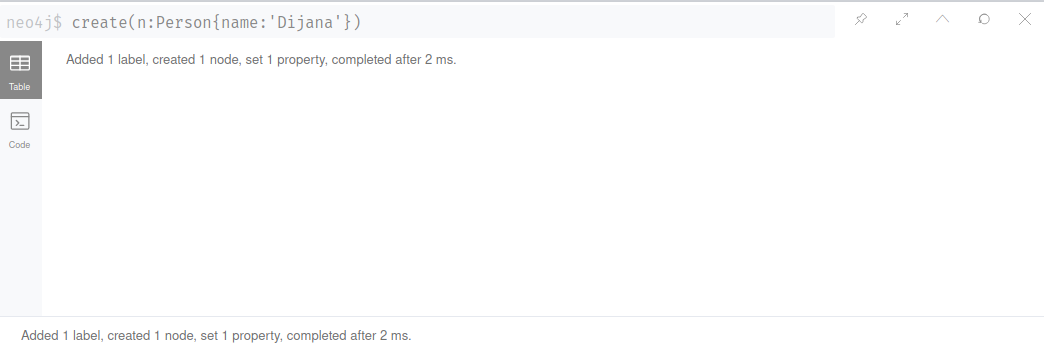
Proverimo sada da li baza sadrži podatke.

Slika:21

Uspešno smo izvršili povratak podataka iz back-up-a.

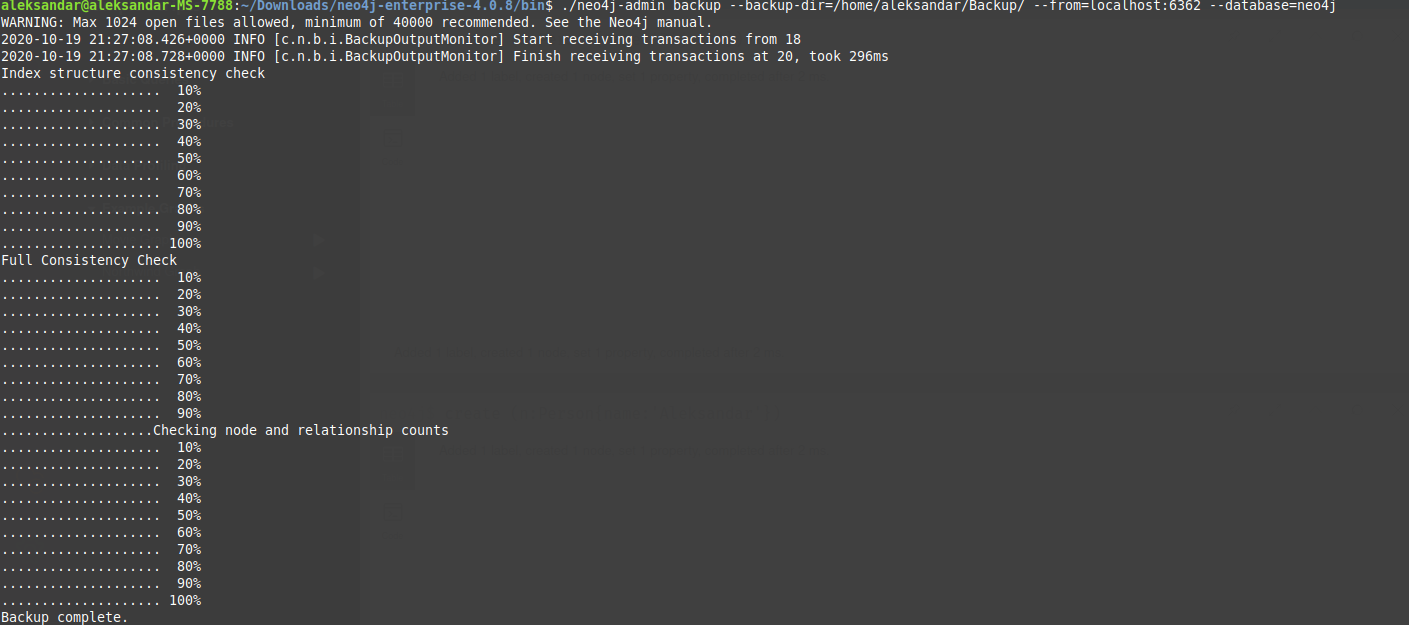
## 3.2. Incremental backup

Inkrementalni back-up, kako smo ranije napomenuli vrši se nakon uspešnog full back-up-a, odnosno vrši nadogradnju već postojećeg back-up-a. Neo4j za back-up u inkrenemtalnom modu koristi transakcione logove. Kako bi smo praktično odradili inkrementalni back-up, dodaćemo dva nova čvora u predhodno korišćenom Movie grafu.

Slika:22

Slika:23

Stopiraćemo Neo4j server i izvršiti back-up ka istom direktorijumu kao i predhodni back-up.

Slika:24

## 3.3. Oporavak kod Neo4j-a

U poglavlju o transakcijama videli smo da svaka transakcja ima ciklus koji se sastoji od počinjanja transakcije, izvršenja i potvrđenja ili vraćanja transakcije u zavisnosti od uspešnosti izvršenja. Proces checkpoint-ing-a je markiranje transakcionih log-ova. Neo4j vrši oporavak transakcija koje nisu potvrđene od strane DBMS-a tj. Onih koje su vraćene pa se moraju ponoviti u procesu oporavka kako bi se uspešno potvrdile. Sama Neo4j baza vrši oporavak pri samom pokretanju servera. Čita potrebne podatke iz transakcionih log-ova, kako bi uspešno izvršila oporavak. Ukoliko nedostaju neki od log-ova transakcija koje se trebaju ponovo izvršiti prilikom oporavka server Neo4j baze se neće pokrenuti!

# 4. zaključak

U ovom radu opisali smo funkcionisanje transakcija, back-up-a i oporavka kod Neo4j baze podataka. Izveli smo praktičan primer back-up-a u oba moda full i incremental. Sve predhodno navedeno dovodi nas do zaključka da se oporavak podataka može izvršiti putem back-up-a, a da oporavak od pada transakcije vrši sam DBMS. Prilikom pokretanja serveravlaa, čita log-ove transakcija i na osnovu toga ponavlja izvršenje transakcije. U današnje vreme oporavak baze igra veliku ulogu jer je veliki broj podataka koj se u njima čuva i njihov gubitak bi mogao izazvati katastrofu za firmu ili organizaciju.

# literatura

[1. ] Članak sa wikipedije: Transakcija (baze podataka)

[2. ] Članak iz Neo4j dokumentacije: Transactions Link: <https://neo4j.com/docs/cypher-manual/current/introduction/transactions/>

[3. ] Članak iz Neo4j dokumentacije: Memory configuration Link: <https://neo4j.com/docs/operations-manual/current/performance/memory-configuration/>

[4. ] Članak iz Neo4j dokumentacije Trasaction flow Link: <https://neo4j.com/docs/http-api/current/actions/transaction-flow/>

[4. ] Članak iz Neo4j dokumentacije Query and result format Link: <https://neo4j.com/docs/http-api/current/actions/query-result/>

[5. ] Članak iz Neo4j dokumentacije Transaction Menagement Deadlocks, Isolatoin level Link: <https://neo4j.com/docs/http-api/current/actions/query-result/>

[6. ] Članak iz Neo4j dokumentacije Transaction loging Link: <https://neo4j.com/docs/operations-manual/current/monitoring/logging/#transaction-logging>

[7 ] Članak iz Neo4j dokumentacije Preform a backup Link:

<https://neo4j.com/docs/operations-manual/current/backup/performing/>

[8 ] Članak iz Neo4j Checkpointing and Log Pruning interactions Link:

<https://neo4j.com/developer/kb/checkpointing-and-log-pruning-interactions/>

Датум предаје: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Комисија:

Председник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Испитивач \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члан \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Коментар:

Датум одбране: \_\_\_\_\_\_\_\_\_\_\_\_\_ Оцена\_\_\_\_\_\_\_\_\_\_ (\_\_\_)