



UNIVERZITET U NIŠU  
ELEKTRONSKI FAKULTET  
Katedra za računarstvo



# **CouchDB, interna struktura i organizacija indeksa**

Sistemi za upravljanje bazama podataka

Student:

Andrija Petrović 1387

Mentor:

Doc. dr Aleksandar Stanimirović

# Sažetak

NoSQL baze podataka karakteriše fleksibilno, ili bolje rečeno, skladištenje podataka bez šeme, što je posebno korisno u ranim fazama razvoja aplikacije, gde je struktura podataka često podložna promenama. Ipak u kasnijim fazama, nedostatak prisustva šeme, može stvoriti mnoštvo problema programeru. Različite verzije podataka se moraju uzeti u obzir u svakom koraku procesa razvoja. Ovaj rad, baziran je na internoj strukturi CouchDB baze podataka, koja upravo pripada NoSQL porodici. Nakon detaljnog objašnjenja unutrašnje strukture, sledi detaljan prikaz organizacije indeksa, kroz teoriju, ali i praktične primere. Za interakciju sa *CouchDB* v. 3.1.1 bazom podataka, koja je prethodno instalirana u vidu servera na *Ubuntu 21.10 (impish)* operativnom sistemu, korišćen je Python programski jezik u sprezi sa Bash skriptama.

# Sadržaj

<b>Sažetak</b>	<b>1</b>
<b>Sadržaj</b>	<b>2</b>
<b>1 Uvod</b>	<b>3</b>
1.1 DBMS	3
1.2 SQL vs NoSQL	4
1.2.1 SQL	4
1.2.2 NoSQL	4
<b>2 CouchDB odlike</b>	<b>8</b>
2.1 Uvod	8
2.2 Po čemu se CouchDB izdvaja?	8
<b>3 Interna struktura</b>	<b>10</b>
3.1 Princip skladištenja podataka	10
3.2 Tehnički detalji	11
3.2.1 HTTP	11
3.2.2 JSON	12
3.2.3 Spisak promena	13
3.2.4 Replikacija	14
3.2.5 Automatsko upravljanje konfliktima	15
3.2.6 Upiti	16
<b>4 Organizacija indeksa</b>	<b>20</b>
4.1 Uvod	20
4.2 CouchDB indeksi	20
4.2 CouchDB značaj indeksa	24
<b>5 Zaključak</b>	<b>27</b>
<b>Reference</b>	<b>28</b>

# 1 Uvod

## 1.1 DBMS

Baze podataka postale su esencijalne za svaki vid poslovanja. Prilikom posete bilo koje web lokacije, baze podataka su te koje omogućavaju serviranje potraživanih informacija korisniku. Nalaze se u srži mnogobrojnih naučnih istraživanja, ali i korporacija, kojima na taj način omogućavaju održavanje celokupne evidencije, koja je od vitalnog značaja. Koriste se za skladištenje podataka koje su prikupili astronomi, istraživači ljudskog genoma ili bio-hemičari, koji istražuju svojstva proteina, između mnogih drugih naučnih aktivnosti.

Moć baza podataka potiče od skupa znanja i tehnologije koja se razvijala tokom nekoliko decenija i oličena je u specijalizovanom softveru koji se naziva sistemom za upravljanje bazom podataka, ili DBMS. DBMS je veoma moćan alat za kreiranje i upravljanje velikim količinama podataka, na efikasan način, pritom omogućavajući opstanak istih tokom dugog vremenskog perioda, na bezbedan način. DBMS predstavlja jedan od najkompleksnijih tipova softvera.

U suštini baza podataka nije ništa drugo do veliki skup informacija koje postoje tokom veoma dugog vremenskog perioda, često i više decenija. Prosto rečeno, termin baza podataka se odnosi na kolekciju podataka kojom upravlja DBMS. Sistem za upravljanje bazom podataka obezbeđuje mehanizam za skladištenje i preuzimanje podataka. Postoje tri glavna tipa sistema za upravljanje bazama podataka, a to su RDBMS (relacioni sistemom za upravljanje bazom podataka), OLAP (Sistemi za analitičku obradu na mreži) i NoSQL.

Od DBMS-a se očekuje da:

1. Dozvoli korisnicima kreiranje nove baze podataka uz odabir sopstvene šeme tj. logičke strukture podataka, koristeći specijalizovani jezik za definiciju podataka.
2. Omogući postavljanje upita o podacima i modifikaciju podataka, koristeći odgovarajući jezik, koji se često naziva jezik upita ili jezik za manipulaciju podacima.
3. Podržava skladištenje velikih količina podataka, tokom dužeg vremenskog perioda, omogućavajući efikasan pristup podacima za upite i modifikaciju baze podataka.
4. Posедуje izdržljivost, tj. mogućnost oporavka baze podataka u slučaju kvarova, raznih grešaka ili namernih zloupotreba.
5. Kontrolišite pristup podacima od strane više korisnika istovremeno, pritom ne dozvoljavajući neočekivane interakcije među korisnicima (izolacija).

## 1.2 SQL vs NoSQL

Kada je u pitanju izbor baze podataka, najveća odluka je odabir relacione (SQL) ili nerelacione (NoSQL) strukture podataka. Iako su obe baze podataka održive opcije, ipak postoje određene ključne razlike između njih, koje korisnici moraju imati na umu pri donošenju odluke.

### 1.2.1 SQL

Najrasprostranjenije u poslovanju su relacione baze podataka sa SQL-om, kao jezikom upita. Relacioni model je naučno zasnovan još 1960-ih, kontinuirano se razvijao i standardizovao tokom godina, tako da danas postoje različite, veoma zrele implementacije.

SQL je moćan jezik upita za analizu i izdvajanje velikih količina podataka iz relacionih tabela koje prate fiksnu šemu SQL baze podataka. SQL je jedna od najsvestranijih i najrasprostranjenijih dostupnih opcija, što ga samim tim čini sigurnim izborom posebno za velike složene upite. Ipak sa druge strane, takva rigidnost, može biti itekako restriktivna. SQL zahteva korišćenje unapred definisane šeme, zarad preciznog određivanja strukture podataka, pre nego što rad sa podacima zapravo i otpočne. Pritom, svi podaci moraju imati identičnu strukturu. Ovo može zahtevati značajnu prethodnu pripremu, što znači da bi bilo kakva buduća promena strukture, bila izuzetno teška, ako ne i nemoguća, zbog čega bi zasigurno došlo do remećenja postojećeg sistema.

U gotovo svim situacijama, SQL baze podataka su vertikalno skalabilne. To u suštini znači, da je moguće povećati opterećenje na jednom serveru, modernizacijom, ili uvećanom količinom esencijalnog hardvera, poput RAM memorije, procesora ili SSD diska za skladištenje podataka. SQL baze podataka su zasnovane na tabelama, i prate ACID svojstva (atomičnost, konzistentnost, izolacija i izdržljivost). Ovo čini relacione SQL baze podataka, boljom opcijom za aplikacije koje zahtevaju transakcije u više redova, kao što je primer računovodstvenog sistema, ili za stare sisteme koji su od samog starta, izgrađeni sa relacionim strukturama na umu.

### 1.2.2 NoSQL

NoSQL baze podataka predstavljaju baze podataka, koje se razlikuju od krutog, relacionog pristupa. Nastale su iz potrebe da se čuvaju velike količine heterogenih podataka. Relacione baze podataka, ne uspevaju posebno kod web aplikacija, koje često uključuju nekoliko miliona operacija čitanja i pisanja od strane različitih korisnika, istovremeno. Karakteriše ih izuzetno dobra horizontalna skalabilnost. Pored skalabilnosti, takođe poseduju, visok nivo performansi, pristupačnosti, ali i fleksibilnosti.

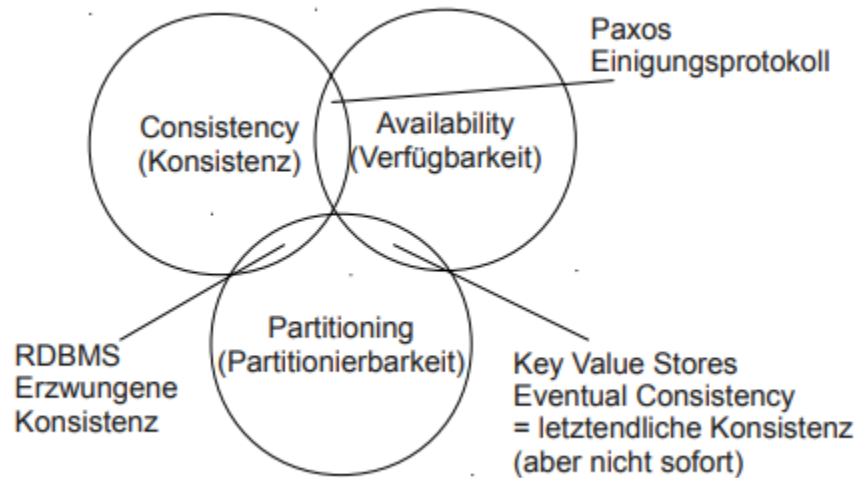
NoSQL baza podataka, poseduje dinamičku šemu za nestrukturirane podatke. Podaci se čuvaju na mnogo načina, što zapravo znači, da mogu biti orijentisani na dokumente, kolone, zasnovani na grafikonima ili organizovani kao

ključ/vrednost skladište. Baš zbog te fleksibilnost moguće je kreiranje dokumenata, bez potrebe za prethodnim definisanjem interne strukture. Pritom, svaki dokument može imati svoju sopstvenu, ili jedinstvenu strukturu. Sintaksa se razlikuje od baze do baze podataka, a polja mogu biti dodavana u toku razvojnog procesa. S obzirom na odličnu horizontalnu skalabilnost, može se zaključiti, da se sa uvećanjem saobraćaja, može upravljati deljenjem ili dodavanjem više različitih servera. Zaključno sa time, NoSQL može na kraju postati daleko veći i moćniji, čineći pritom NoSQL baze podataka, preferiranim izborom za velike ili stalno promenljive skupove podataka.

Pored parova ključ-vrednost, gde su upiti mogući samo za jedinstvene ključeve, pošto svi objekti koje sistemi dalje ne tumače mogu biti uskladišteni u oblasti vrednosti, NoSQL baze podataka, mogu biti zasnovane i na skladištenju dokumenata. Skladišta dokumenata, takođe čuvaju parove ključ/vrednost, međutim, za razliku od baza podataka ključnih vrednosti, vrednosti su uvek dokumenti, koji omogućavaju ugneždene vrednosti i samim tim složenije strukture podataka. Dokument može da sadrži i druge dokumente. JSON se koristi kao format podataka, zbog toga što prirodno podržava koncept dokumenata. U bazama podataka dokumenata, pretraga podataka, ne mora biti vezana isključivo samo za ključeve, već se može vršiti i za objekte, koji poseduju, određene attribute od interesa. MongoDB i CouchDB baze podataka, predstavljaju najrasprostranjenije primere skladišta dokumenata. U proširivim skladištima zapisa podataka, koje predstavljaju treći tip skladištenja podataka NoSQL baza, podaci se čuvaju u redovima i kolonama. Skalabilnost se postiže deljenjem ovih redova i kolona. Prilikom podele redova, opsezi bivaju razdvojeni isključivo, prema primarnim ključevima, a potom raspoređeni na različite servere. Kolone su grupisane zajedno, u grupe kolona, kojima se učestalo pristupa. Google Big Table, HBase i Cassandra su najpoznatiji predstavnici ovog tipa skladištenja podataka. NoSQL baze prate *Brevers CAP teoremu* (konzistentnost, dostupnost i tolerancija particije).

Danas postoji toliko mnogo NoSQL sistema da je teško dobiti brzi pregled glavnih kompromisa koji su uključeni prilikom procene relacionih i nerelacionih sistema u okruženjima bez jednog servera. Postoje tri glavna svojstva koja moraju biti uravnotežena, kada se vrši izbor sistema za upravljanje podacima, a to su: doslednost, dostupnost i tolerancija particija. Konzistentnost znači da svaki klijent uvek ima isti pogled na podatke. Dostupnost znači da svi klijenti uvek mogu da čitaju i pišu, a tolerancija particija, znači da sistem dobro funkcioniše na fizičkim mrežnim particijama.

Prema [CAP teoremi](#), moguće je odabrati samo dva svojstva, od pomenutih. CAP teorema, jedna je od tvrdnji doktora Erica A. Brevera, da su od tri moguća svojstva sistema baza podataka — konzistentnost, dostupnost i particionost — samo dva ikada ostvariva. Na slici 1 postoje preseči između 2 svojstva, ali u sredini nema tačke gde se sva svojstva preklapaju. Centralni konsenzus teorije je sledeći: Relacione baze podataka su konzistentne i particione – ali po ceni dostupnosti. Particionisani sistem kao što su HBase mogu postići, isključivo konzistentnost ili dostupnost. CouchDB, pak, prema ovoj teoremi, zaključno sa slikom 2, može postići, dostupnost i toleranciju na particionisanje.

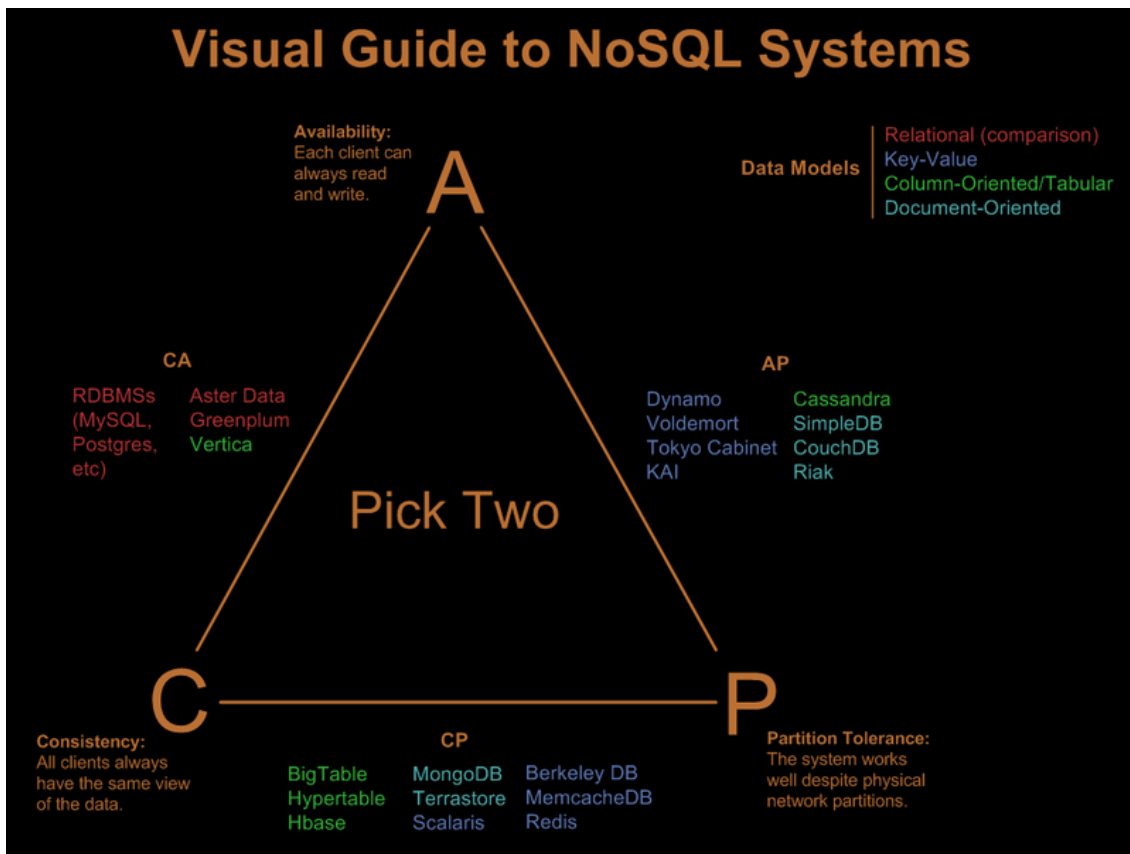


*Slika 1. CAP teorema*

Jedan od primarnih ciljeva NoSQL sistema je jačanje horizontalne skalabilnosti. Kako bi bilo moguće skalirati horizontalno, neophodna je, veoma jaka tolerancija mrežnih particija, koja zahteva odustajanje ili od konzistentnosti ili od dostupnosti. NoSQL sistemi to obično postižu opuštanjem relacionih sposobnosti i/ili labavljenjem transakcione semantike.

Pored CAP konfiguracija, još jedan značajan parametar u odabiru sistema za upravljanje podacima je prema modelu podataka koji koriste: relacioni, ključ-vrednost, orijentisan na kolone ili orijentisan na dokument (postoje i drugi, ali ovi su glavni).

Na slici 2, prikazan je vizuelni vodič svojstava, kojim se korisnik može voditi, prilikom izbora baze podataka. Prikazane su neke od danas, najkorišćenijih baza podataka, prema svojstvima, koja su u mogućnosti da ostvare. Prema tome, jednostavno je uočiti svojstvene mogućnosti baze podataka od interesa. Poređenja radi, prikazane su i neke baze podataka iz SQL porodice, pa je s obzirom, na njihovo mesto na grafikonu, lako razgraničiti svojstva SQL od NoSQL predstavnika.



Slika 2. CAP teorema, vizuelni vodič



## 2 CouchDB odlike

### 2.1 Uvod

CouchDB (“Cluster Of Unreliable Commodity Hardware”) je baza podataka otvorenog koda (open-source), razvijena od strane softverske fondacije Apache. Razvijena je, kao projekat zajednice, kojem doprinosi, nekolicina komercijalnih entiteta, podržavajući na taj način dalji razvoj. CouchDB je napisana na Erlang-u, funkcionalnom programskom jeziku sa fokusom na pisanje robusnih, tolerantnih na greške i visoko konkurentnih aplikacija.

Fokus je na jednostavnosti korišćenja. U osnovi koristi HTTP protokol, kao svoj glavni programski interfejs i JSON za skladištenje podataka. Posедуje sofisticiranu funkciju replikacije, koja je zapravo, glavni razlog upotrebe vrednosti, ove NoSQL baze podataka. Prvi put objavljena 2005. godine, postala je Apache projekat 2008. godine. U 2007. godini, CouchDB dodaje sistem za indeksiranje i upite podataka, zasnovan na MapReduce konceptu. Koristi se širom sveta, od nezavisnih entuzijasta, do industrijskih giganta. Projekat poseduje zdravu, programersku zajednicu i podršku, uz neprestano rastuću bazu obožavalaca.

Arhitektonski dizajn, čini CouchDB izuzetno prilagodljivom prilikom particionisanja i skaliranja podataka na više čvorova. Podržava horizontalno particionisanje i replikaciju, što omogućava balansiranje opterećenja čitanja i pisanja tokom postavljanja baze podataka.

CouchDB ima veoma izdržljiv i pouzdan mehanizam za skladištenje, napravljen od temelja za infrastrukturu sa više oblaka i više baza podataka. Kao NoSQL baza podataka, CouchDB je veoma prilagodljiva i otvara vrata za razvoj predvidljivih aplikacija, zasnovanih na performansama bez obzira na obim podataka ili broj korisnika.

### 2.2 Po čemu se CouchDB izdvaja?

CouchDB je više nalik git-u nego MySQL bazi podataka. Kod git-a, interakcije su trenutne. Zašto je to tako? Sve operacije su lokalne. Kopija skladišta, uključuje sve informacije koje udaljeni server ima i stoga nema potrebe da se svaka interakcija odvija preko mreže, do nekog servera. CouchDB funkcioniše kao git, jer se može imati kopija svih podataka na lokalnoj mašini, kao i na udaljenom serveru, ili više servera, čime je omogućeno kontinuirano podešavanje integracije, uz rasprostranjenost po čitavom svetu. CouchDB funkcija replikacije, predstavlja pokretačku snagu opisane funkcionalnosti.

Dok MongoDB nudi master-slave tip replikacije sa ugrađenim automatskim izborom, CouchDB nudi master-master i master-slave tipove replikacije. Master-slave pristup, automatskim izborom, promovise sekundarnu bazu podataka kao

primarnu, u slučaju nedostupnosti. Sa skupovima replika u MongoDB-u, može postojati jedna primarna baza podataka, sa više repliciranih baza podataka koje imaju sekundarnu ulogu. Pomenuta dva tipa replikacije, kod CouchDB baze podataka, omogućavaju pristup podacima sa malim kašnjenjem bez obzira na lokaciju. CouchDB će početi da šalje sve promene, koje se dese na izvoru, ciljnoj bazi podataka. Ovo je jednosmerni proces. U slučaju potrebe dvosmernog procesa, mora se pokrenuti replikacija na odredišnom serveru, tako da on bude izvor, a udaljeni server odredište.

Recimo, da imamo kancelariju u Srbiji, koja koristi CouchDB. Osoblje nema nikakvih problema, jer pristupa podacima putem jako brze lokalne mreže. Ako bismo zatim otvorili kancelariju u Australiji, gde osoblje koristi isti softver i potražuje iste podatke, kao i osoblje u Srbiji, suočavamo se sa problemom velike latencije, jer slanje zahteva, širom sveta dodaje značajno kašnjenje svakom zahtevu. Pored toga, ako mrežna veza, u srpskoj kancelariji ili bilo koja mreža između njih ima problema, kancelarija u Australiji biva praktično odsečena od pristupa podacima, koji su im potrebni za obavljanje posla sa svojim klijentima. Srećom, znamo, da CouchDB ima ugrađenu replikaciju, što omogućava postavljanje baze podataka i aplikacije u kancelariji u Australiji. Zaposlenima u Australiji, sada je omogućen pristup podacima, korišćenjem lokalne LAN mreže. U pozadini, obe CouchDB instance, mogu replicirati izmene jedna na drugu, tako da podaci dodati u Srbiji uvek stižu do Australije i obrnuto. Replikacija se postiže veoma jednostavno, što je lako uočljivo u kodu koji sledi.

```
curl -X POST https://srpska_kancelarija/_replicate -d  
'{"source":"https://australijska_kancelarija/app", "target":"app","continuous":true}'
```

#### *Kod 1. Replikacija*

Korišćenjem ugrađene replikacije, pospešuje se korisničko iskustvo, kroz pristup korisničkim podacima bez kašnjenja, pri čemu je korišćenje aplikacija u velikoj meri nezavisno od eksterne mreže.

## 3 Interna struktura

U ovom poglavlju biće predstavljeni interni strukturni elementi CouchDB baze podataka, uz pojašnjenja, kako se implementiraju različite funkcije unutar CouchDB-a i kako njihova kombinacija čini otporan, brz i efikasan sistem baze podataka.

### 3.1 Princip skladištenja podataka

CouchDB je sistem za upravljanje bazom podataka koji može da upravlja bilo kojim brojem logičkih baza podataka. Jedina ograničenja su raspoloživi prostor na disku i deskriptori datoteka iz operativnog sistema. U stvari, nije neuobičajeno postaviti arhitekturu baze podataka sa CouchDB-om gde svaki korisnik dobija sopstvenu bazu podataka. CouchDB može da se nosi sa rezultirajućim potencijalno stotinama hiljada ili milionima baza podataka.

Svaka baza podataka je podržana jednom datotekom u sistemu datoteka. Svi podaci koji ulaze u bazu podataka skladište se u toj datoteci. Indeksi za poglede (views) koriste istu osnovnu mehaniku skladištenja, ali svaki pogled dobija sopstvenu datoteku u sistemu datoteka koja je odvojena od glavne datoteke baze podataka.

Datoteke baze podataka i datoteke indeksa pregleda koriste se u append-only (isključivo dodavanje) modu. To znači da se svi novi podaci koji dođu u bazu podataka dodaju na kraj datoteke. Čak i kada se dokumenti izbrišu, to je informacija koja ide na kraj datoteke. Rezultat je izuzetna otpornost na gubitak podataka: jer kada se podaci predaju datoteci i ta datoteka se isprazni u osnovni hardverski disk, CouchDB nikada neće pokušati da u potpunosti ili delimično prepíše te podatke. To znači da u bilo kom scenariju greške (pad softvera, pad hardvera, nestanak struje, itd.) CouchDB garantuje da su prethodno pohranjeni podaci i dalje netaknuti. Jedini način na koji se problemi mogu uvući u sistem, je kada osnovni disk ili sistem datoteka pokvare podatke, pa čak i tada CouchDB koristi kontrolne heš sume da otkrije ove greške.

Bezbednost podataka je jedan od velikih dizajnerskih ciljeva CouchDB-a, što je i obezbeđeno upravo opisanim dizajnom, a zarad maksimalanog stepena otpornosti. Pored toga, dodavanjem podataka uvek na kraju datoteke, omogućava osnovnom sloju skladištenja da radi u velikim i praktičnim grupama, bez mnogo traženja, što umanjuje opterećenje HDD i SSD diskova.

Kompromis koji CouchDB upražnjava, sastoji se u potrebi za procesom koji se zove sažimanje (compaction). Ovaj proces namenjen je čišćenju dodatnog prostora baze podataka i starih revizija dokumenata. Sažimanje prati izvor promena baze podataka od početka i kopira sve najnovije verzije dokumenata u novu datoteku. Po završetku, proces atomski zamenjuje staru i novu datoteku, a zatim briše staru datoteku.

Baze podataka i indeksi prikaza koriste varijantu B+ stabla, za upravljanje skladištenjem podataka. B+ stabla su veoma široka i plitka. U slučaju CouchDB-a, drvo visine 3 može da obradi preko 4 milijarde dokumenata. Ne postoji gornja granica za broj dokumenata ili količinu podataka uskladištenih u jednom od B+ stabala. Prednost širokog stabla je radna brzina. Gornji slojevi stabla ne sadrže nikakve stvarne korisničke podatke i uvek se uklapaju u keš sistema datoteka. Dakle, za bilo koje čitanje ili pisanje, čak i sa stotinama milijardi dokumenata, CouchDB-u je potreban samo jedan disk za traženje podataka, ili mesta, za pisanje novog dokumenta.

CouchDB poseduje REST API zasnovan na HTTP protokolu, koji pomaže da se lako komunicira sa bazom podataka. Jednostavnu strukturu HTTP resursa i metoda (GET, PUT, DELETE) je lako razumeti i koristiti. Pošto podatke čuvamo u fleksibilnoj strukturi zasnovanoj na dokumentima, nema potrebe brinuti o strukturi podataka. Korisnicima je obezbeđeno moćno mapiranje podataka, koje omogućava ispitivanje, kombinovanje i filtriranje informacija. CouchDB obezbeđuje replikaciju laku za korišćenje, pomoću koje možete da kopirate, delite i sinhronizujete podatke između baza podataka i mašina. Baza podataka je najudaljenija struktura podataka/kontejner u CouchDB-u. Svaka baza podataka je zbirka nezavisnih dokumenata. Svaki dokument održava sopstvene podatke i samostalnu šemu. Metapodaci dokumenta sadrže informacije o reviziji, što omogućava spajanje razlika koje su nastale dok su baze podataka bile van konekcije. CouchDB implementira kontrolu konkurentnosti više verzija, kako bi se izbegla potreba da se zaključa polje baze podataka tokom upisivanja. Ažuriranja dokumenata (dodavanje, uređivanje, brisanje) prate atomičnost, odnosno biće sačuvani u potpunosti ili neće biti sačuvani uopšte. Baza podataka neće imati delimično sačuvane ili uređene dokumente.

JavaScript se koristi, kao jezik za zadatke pisanja skripti u bazi podataka. CouchDB poseduje dodatke (plugins), koji omogućavaju povećanje osnovnih karakteristika i semantike na način koji korisnik zahteva. U ovom trenutku, dodaci moraju biti napisani na Erlang-u. Dodatak, na primer, može da bude sekundarni mehanizam za upite, kao što je GeoCouch, dvodimenzionalni mehanizam za indeksiranje i upite koji funkcioniše slično pogledima, ali je optimizovan za geo-prostorne upite.

## 3.2 Tehnički detalji

U osnovi CouchDB baze podataka su HTTP kao glavni pristupni protokol i JSON kao format za skladištenje podataka u dokumente.

### 3.2.1 HTTP

Glavni način da se bilo šta uradi sa CouchDB bazom, jeste preko HTTP protokola. Kreiranje baze podataka, unos podatka, upit kreiranih podataka, podešavanje replikacije ili konfigurisanje baze podataka, predstavljaju akcije koje se izvršavaju kreiranjem HTTP zahteva. Sledeći HTTP zahtev će kreirati novu bazu podataka.

```
curl -X PUT http://127.0.0.1:5984/baza
```

### *Kod 2. Kreiranje baze podataka*

Za razliku od relacionih baza podataka, koje čuvaju tabele i redove, CouchDB skladišti podatke u dokumente unutar baze podataka. Dokument sadrži i strukturu i podatke. Zbog toga se CouchDB često klasifikuje kao baza podataka orijentisana na dokumente. Jednostavan način razmišljanja o dokumentu iz perspektive programera je objekat ili serijalizacija objekta.

### 3.2.2 JSON

CouchDB dokumenti su enkodovani u JSON formatu. Ono što JSON čini tako pogodnim, je to što je on podskup svih izvornih tipova koji su zajednički svim programskim okruženjima: brojevi, logičke vrednosti, stringovi, liste i hešovi. Ovo čini JSON odličnim formatom za razmenu podataka između različitih sistema, jer sve što je neophodno jeste JSON parser, koji prevodi JSON u izvorne tipove programskog jezika, što je znatno jednostavnije od slojeva za prevođenje, koji bi mapirali sve vrste sofisticiranih stvari između različitih okruženja. Pored toga, JSON je izvorni tip za veb programiranje, prilično je koncizan i dobro se kompresuje, tako da je prirodan izbor za programiranje mobilnih aplikacija. Sledi primer kreiranja dokumenta, sa podacima u JSON formatu.

```
curl -X PUT http://127.0.0.1:5984/baza/dokument -d '{"ime": "Mišel", "godine": 19}'
```

### *Kod 3. Kreiranje dokumenta*

CouchDB će pohraniti svaki JSON objekat. U tom smislu, CouchDB je u osnovi, baza podataka bez šema. Ovo pomaže u izradi prototipova aplikacija, jer nije neophodno trošiti bezbroj sati unapred definišući model podataka, već je moguće odmah početi sa programiranjem i skladištenje JSON objekata u CouchDB. Ovo takođe eliminiše srednji sloj poznat kao Object Relational Mappers (ORMs). Površno govoreći, ORM pretvara relacionu bazu podataka u nešto što deluje prirodno objektno orijentisanim programerima. Sa CouchDB-om dobijate isti interfejs, ostavljajući čitavu klasu problema iza sebe od samog početka. Pored toga, izvorni kod mnogih popularnih ORM-ova je veći od izvornog koda CouchDB-a.

CouchDB takođe podržava skladištenje binarnih podataka. Mehanizam se naziva prilog (attachments) i funkcioniše kao prilozi e-pošte: proizvoljni binarni podaci se čuvaju pod imenom i sa odgovarajućim tipom sadržaja u stavci `_attachments` unutar dokumenta. Ne postoji ograničenje veličine za dokumente ili priloge.

Mogućnost skladištenja proizvoljnih podataka je naravno blagoslov kada počinjete sa programiranjem, ali u slučaju da kasnije u razvojnem ciklusu aplikacije, želite da budete sigurni da baza podataka dozvoljava samo pisanje dokumenata koji imaju svojstva koja očekujete, CouchDB podržava opcionu primenu šeme. Forsiranje šeme se postiže JavaScript funkcijom koja odlučuje da li je dokument u skladu sa očekivanim standardom ili ne.

```
function(dokument) {  
  if(!dokument.ime) {  
    throw({  
      "forbidden":  
      "document must have a name property"  
    });  
  }  
}
```

#### *Kod 4. Opciona šema*

Ova funkcija se pokreće svaki put kada pokušate da napišete novi dokument ili ažurirate postojeći dokument u bazi podataka. Takođe je moguće učitati podržane biblioteke, koje primenjuju više deklarativne šeme koristeći JSON šemu.

### 3.2.3 Spisak promena

Jedna od uzbudljivijih karakteristika CouchDB-a je spisak promena (Changes Feed). Može se poistovetiti sa git dnevnikom (log). Predstavlja listu svih dokumenata u bazi podataka sortiranih prema poslednjoj nedavnoj promeni. Čuva se u brznoj strukturi indeksa i može vrlo efikasno da odgovori na pitanje „Šta se od tada dogodilo?“ za bilo koji vremenski opseg istorije baze podataka. Bilo od početka, ili samo poslednjih nekoliko promena, koje su napravljene u bazi podataka. Spisak promena je dostupan preko HTTP-a u nekoliko različitih modifikacija i omogućava tri veoma zanimljive upotrebe:

- **Kontinuirani režim:** kontrolna tabla može da otvori konekciju sa spiskom promena, a CouchDB će ostaviti otvorenu konekciju sve dok se ne dogodi promena u bazi podataka. Zatim šalje jedan red JSON-a sa informacijama o dokumentu koji je upravo napisan na kontrolnu tablu. Kontrolna tabla tada može da ažurira svoje interne strukture podataka i prikaže ih krajnjem korisniku. Spisak promena bi mogao da se koristi za implementaciju e-pošte ili u mobilnoj aplikaciji, koja služi za razmenu poruka.
- **Kontinuirano od:** pored informacija o novim dokumentima, promenama ili brisanju dokumenata iz baze podataka, spisak promena takođe uključuje redni broj koji je kao ceo broj sa automatskom inkrementacijom, koji se ažurira svaki put kada dođe do promene baze podataka. Spisak promena je indeksiran ovim identifikatorom sekvence, tako da je zahtev upućen CouchDB bazi, za slanje dokumenata od trenutka poslednje

komunikacije veoma efikasna operacija. Sve što treba da uradite je da zapamtite poslednji ID sekvence koji ste dobili od CouchDB-a i da ga koristite kao parametar za svoj sledeći zahtev. Na taj način možete da održavate kopiju podataka u vašoj bazi podataka, koje omogućavaju efikasna delta ažuriranja, gde treba da zahtevate samo količinu podataka koja se promenila od poslednjeg puta kada ste komunicirali sa CouchDB-om. Pored toga, ova arhitektura je veoma otporna, jer u slučaju da se komunikacija prekine ili sruši iz bilo kog razloga, može da se nastavi odakle je stala. ID-ovi sekvenci se razlikuju od ID-ova dokumenata ili ID-ova revizije, po tome što se održavaju po bazi podataka i povećavaju se svaki put kada se izvrši promena u bazi podataka.

- **Mašina stanja dokumenta:** uobičajeni obrazac je i korišćenje dokumenta za praćenje stanja operacije u više koraka, recimo slanje e-pošte. Koraci bi mogli biti: 1. krajnji korisnik inicira proceduru slanja e-pošte; 2. backend je primio korisničku nameru i detalje e-pošte; 3. podsistem odgovoran za slanje e-pošte rezerviše e-poštu za slanje (tako da drugi paralelni podsistemi ne šalju imejl dva puta); 4. podsistem odgovoran za slanje e-pošte pokušava SMTP isporuku; 5. stanje zapisa podsistema (uspeh ili neuspeh); 6. frontend preuzima stanje slanja e-pošte i u skladu sa tim ažurira korisnički interfejs. Svi ovi diskretni koraci mogu koristiti jedan dokument da bi se osigurala konzistentnost operacije, a spisak promena se može koristiti za labavo povezivanje svih podsistema potrebnih za obavljanje celog postupka.

### 3.2.4 Replikacija

Revizije omogućavaju CouchDB replikaciju. Tokom replikacije, ako cilj već ima dokument sa istim ID-om, CouchDB može da uporedi revizije i zaključi da li su identične, ili je jedna revizija potomak druge. Pri kreiranju dokumenta CouchDB šalje sledeći odgovor:

```
{
  "ok": true,
  "id": "dokument",
  "rev": "1-e7eef66f242a6s2d9342a6627eak19s"
}
```

*Kod 5. Odgovor pri kreiranju dokumenta*

Revizija (rev) označava određenu verziju dokumenta. U našem slučaju to je prva verzija ( 1- ) i heš nad sadržajem tog dokumenta. Sa svakom promenom dokumenta dobijamo novu reviziju. Da bismo ažurirali naš dokument, moramo da obezbedimo postojeću reviziju kako bismo dokazali da znamo šta menjamo. Dakle, tehnički, revizija je MVCC token koji osigurava da nijedan klijent ne može slučajno da prepíše podatke koje nije nameravao. U osnovi, replikacija je operacija koja uključuje izvornu i ciljnu bazu podataka. Podrazumevani režim za operaciju je da uzme sve dokumente koji se nalaze u izvornoj bazi podataka i da ih replicira u ciljnu bazu podataka.

```
curl -X POST http://127.0.0.1:5984/_replicate
-d '{"source":"baza", "target":"kopija"}
```

#### *Kod 6. Replikacija*

Postoje razni drugi režimi. Ako su se izvor i cilj replicirali ranije, replikacija je dovoljno pametna da replicira samo dokumente koji su dodati, promijenjeni ili izbrisani na strani klijenta, od poslednje replikacije.

Replikacija takođe može biti kontinuirana, tada se ponaša kao obična replikacija, ali umesto da se zaustavi kada se završi na kraju replikacije svih dokumenata iz izvora, ona nastavlja da osluškuje spisak promena izворne baze podataka i replicira ih onako kako se pojavljuju.

Filtrirana replikacija omogućava definisanje Java-Script funkcije, koja odlučuje da li dokument treba replicirati ili ne. Ova funkcija radi na spisku promena, tako da se može koristiti i van replikacije. U situaciji sa više primarnih baza podataka, replikacija može funkcionisati i unazad. Ovakav efekat se postiže startovanjem druge replikacije gde izvor i cilj menjaju mesta.

### 3.2.5 Automatsko upravljanje konfliktima

Jedno veliko pitanje pada na pamet, kada govorimo o sistemima baza podataka sa više primarnih baza podataka: „Šta je sa konfliktima?“.

CouchDB se ponosi podrškom za automatsko otkrivanje sukoba. Pokreće ga još jedna struktura podataka koju CouchDB drži, a koju još nismo istražili: stablo revizije za svaki dokument. Umesto da se čuva samo najnovija verzija dokumenta, takođe se čuva i lista revizija (samo N-Hash informacije) po redosledu njihovog pojavljivanja. Kada repliciramo izvor i cilj pri čemu postoji dokument na cilju koji ima isti ID kao jedan od dokumenata u izvornoj bazi podataka, CouchDB može jednostavno da uporedi stabla revizije da bi utvrdio da li je verzija na cilju predak onog na izvoru i da li može da izvrši jednostavno ažuriranje ili mora da prijavi konflikt. Ako CouchDB otkrije da je dokument u konfliktom stanju, dodaje novo polje u svoju JSON strukturu:

```
_conflicts: ['1-asa...', '1-asw...']
```

#### *Kod 7. Konflikti*

U praksi, CouchDB zadržava obe verzije dokumenta i obaveštava korisnika, koje revizije su u sukobu. Sa tim informacijama, korisnik može pokušati da reši konflikt dokumenta tako što će izabrati jednu od dve revizije i izbrisati

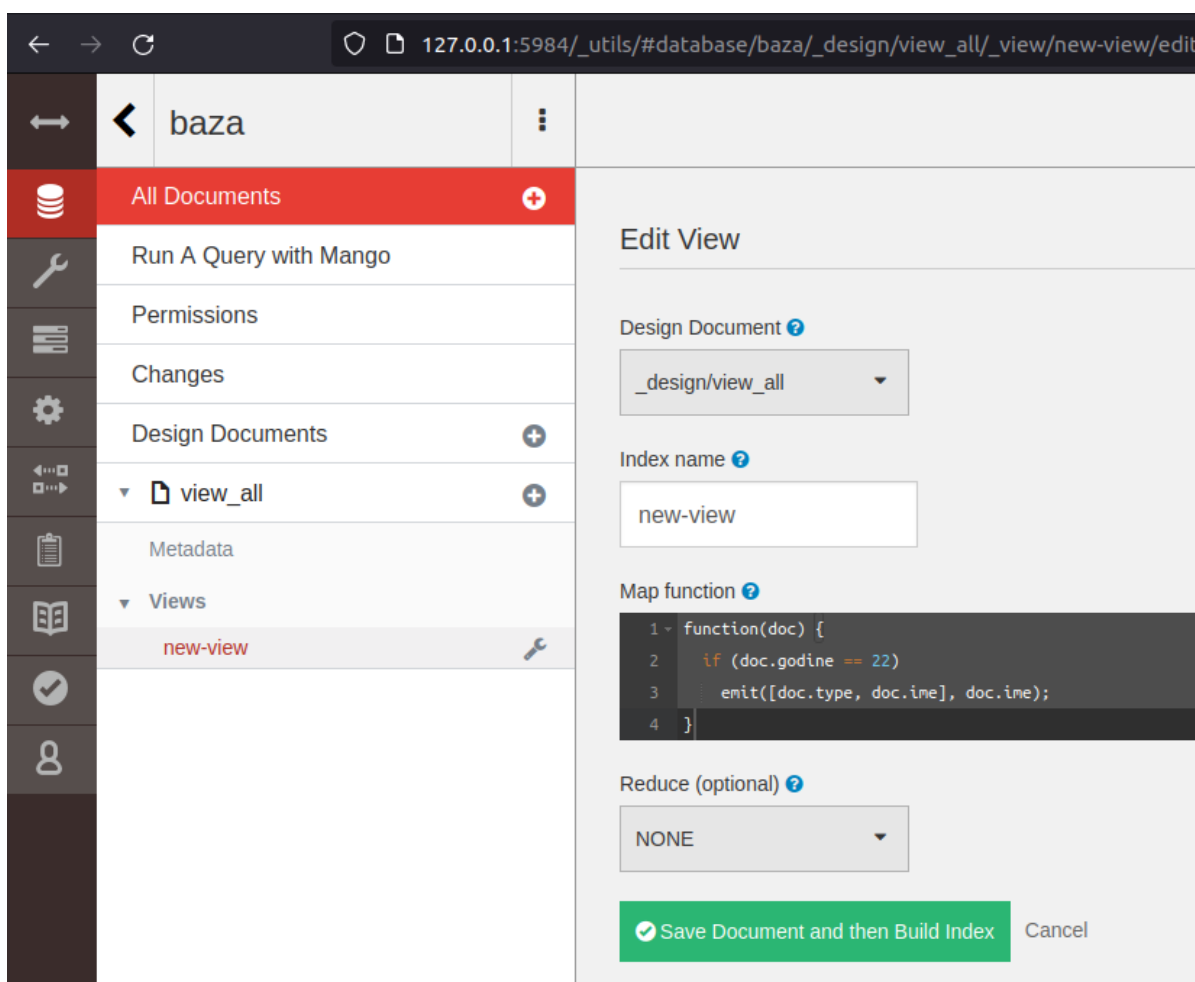


drugu, ili spojiti dve revizije u novu treću, rešenu verziju. Funkcioniše veoma slično konfliktima u sistemima kontrole verzija, gde se dobijaju >>>>>>>HEAD i <<<<<<<VERSION markeri koji moraju biti rešeni, pre nego što se nastavi sa radom. Za razliku od sistema kontrole verzija, gde su konflikti označeni tako da ih nijedan kompajler ne bi slučajno dozvolio, CouchDB će proizvoljno i deterministički izabrati pobedničku reviziju, koja će biti vraćena podrazumevano, ako klijent to zatraži. Nakon što se konflikt replicira kroz ceo klaster, svi čvorovi će odgovoriti istom podrazumevanom revizijom, obezbeđujući konzistentnost podataka čak i kada su u pitanju konflikti. Konfliktno stanje je prirodno stanje u CouchDB-u koje nije više ili manje strašno od bilo kojeg drugog, ali sa njim se mora pozabaviti, jer će zadržavanje velikog broja konflikata učiniti CouchDB manje efikasnim tokom vremena. Očekuje se da će klijentski softver dodati osnovni mehanizam za rešavanje sukoba. U teoriji, CouchDB bi ovde mogao da obezbedi nekoliko podrazumevanih slučajeva, ali pošto od aplikacije zavisi kako treba rešavati konflikte, programeri su se do sada klonili ovoga. Najgori scenario je da aplikacije i korisnici izgube podatke, za koje su pretpostavljali da su bezbedni, a to nije nešto što se uklapa u filozofiju CouchDB-a.

### 3.2.6 Upiti

Baza podataka koja je samo skladište ključ-vrednost je relativno jednostavna za izgradnju, ali je ograničena njena primenljivost. Da bi takav DBMS bio koristan za širok spektar aplikacija, trebalo bi da podržava neki mehanizam za postavljanje upita. Relacione baze podataka su u potpunosti zasnovane na modelu SQL upita, a CouchDB na sistemu upita zvanom pogledi (Views), koji se nalazi na vrhu osnovne baze podataka i koristi MapReduce paradigmu u sprezi sa JavaScript funkcijama, za kreiranje indeksa, koji obezbeđuju pristup podacima na načine koji su korisniji za aplikacije nego samo osnovno skladište podataka. Korišćenje MapReduce funkcija omogućava da model upita bude grupisan sa osnovnim podacima. CouchDB upit je dvofazna operacija. Najpre kreiramo definiciju indeksa, a zatim pravimo zahteve prema tom indeksu.

Definicije pogleda (Views), su specificirane na posebnim dokumentima, koje CouchDB naziva projektnim (design) dokumentima. Funkcije koje indeksiraju selektovane podatke u pogled, se mogu definisati u *Fauxton* CouchDB editoru.



Slika 3. Fauxton

Jedina stvar koja ih razlikuje od običnih dokumenata, je to što njihov identifikacioni kod počinje sa *\_design/*. Da bismo kreirali indeks koji je sortiran po ciljanom broju godina osobe, moramo da napišemo sledeću Java-Script funkciju:

```
function(doc) {
  if (doc.godine == 22)
    emit([doc.type, doc.ime], doc.ime);
}
```

Kod 8. Indeks sortiran po imenu

Ova funkcija se zove funkcija mape jer se pokreće tokom dela „Map“ MapReduce funkcije. Ona se skladišti u dokumentu dizajna pod proizvoljnim imenom *osobe/po\_imenu*. Nakon uskladištenja, moguće je izvršiti CouchDB upit, uz pregršt opcija za ograničavanje skupa rezultata, na ono što nam je tačno potrebno.

```
curl http://127.0.0.1:5984/baza/_design/dokument/_view/po_imenu
{"offset": 0, "rows": [
  {"key": "Mišel", "value": null, "id":"0f87bab4"}
]}
```

#### *Kod 9. CouchDB upit*

Ekvivalentan upit sortiranja po imenu, napisan u SQL-u bi izgledao ovako:

```
SELECT ime FROM dokument ORDER BY ime
```

#### *Kod 10. SQL sortiranje po imenu*

Ono što se dešava ispod haube kada napravite prvi HTTP zahtev za prikaz, nakon što ga definišete u dokumentu dizajna je sledeće:

CouchDB-ov mehanizam za preglede primećuje da još uvek ne postoji indeks koji bi odgovorio na ovaj pogled, tako da zna da treba da ga kreira. Da bi to uradio, otvara spisak promena baze podataka u kojoj je uskladištena definicija pogleda i čita rezultate iz spiska promena, jedan po jedan. Za svaki rezultat, preuzima odgovarajući dokument iz same baze podataka i primenjuje ga na funkciju mape koju smo definisali. Svaki put kada se pozove funkcija *emit(ključ, vrednost)*, engine za prikaz će kreirati par ključ-vrednost u indeksu prikaza. Indeks se čuva u datoteci na disku koja je odvojena od glavne datoteke baze podataka. Kada se završi sa izgradnjom indeksa, otvara se indeksna datoteka i čita se od vrha do dna uz povratni rezultat, kao što vidimo iznad. Sada za svaki sledeći zahtev za prikaz, engine za pregled može samo da otvori indeks i pročita rezultat nazad klijentu. Međutim, pre nego što to uradi, on proverava, koristeći spisak promena baze podataka, da li ima novih ažuriranja ili brisanja u bazi podataka od poslednjeg puta kada je indeks upitan, i ako je to slučaj, engine za prikaz će ugraditi sve nove promene u indeks, pre vraćanja novog rezultata indeksa klijentu.

To znači da se CouchDB indeksi grade na lenj način, kada se postavljaju upiti, umesto kada se novi podaci ubacuju u bazu podataka, kao što je uobičajeno u drugim sistemima baza podataka. Prednost je ovde dvostruka:

1. postojanje velikog broja indeksa ne usporava umetanje novih ažuriranja u bazu podataka
2. masovno ažuriranje indeksa novim promenama je mnogo više prostorno, vremenski i računarski efikasnije.

Ovo takođe znači da izvršenje prvog upita za prikaz, koji je kreiran u bazi podataka koja već ima milione dokumenata u sebi, može potrajati.

Gornji primer prikazuje samo deo „Map“ iz MapReduce-a. Deo „Reduce“ omogućava izvršenje proračuna nad rezultatima mapiranja. Pisanje Reduce funkcije nije obavezujuće, jer ako funkcija mapiranja radi ono što je neophodno, nema potrebe da se dodaje Reduce funkcija. Jednostavan primer je *\_count* Reduce funkcija, koja jednostavno broji sve redove rezultata prikaza. Moguće je napisati mnogo sofisticiranije Reduce funkcije.

## 4 Organizacija indeksa

Indeksiranje podataka postaje jako korisno, u momentu kada baza podataka toliko naraste da se podaci ne mogu preuzeti u razumnom vremenskom roku, putem upita koji skenira celokupnu bazu podataka.

### 4.1 Uvod

Većinu vremena nije poželjno vršiti upite svih uskladištenih podataka odjednom, već se koriste filteri, kako bi se ograničio izbor podataka. Ovaj proces se obično izvršava preko klauzule *where* u SQL-u, metode *find* u MongoDB-u ili metode *selector* u CouchDB-u. Korisnik može potraživati preuzimanje podataka određenim redosledom, tako što će ih sortirati ili izabrati samo određena polja koja će biti vraćena. Iako je moguće obavljati takve zadatke na strani klijenta, oni se generalno delegiraju serveru baze podataka kako bi se osigurala skalabilnost. Uostalom, to i jeste posao servera! Međutim, ne možete očekivati da baza podataka automatski, sama po sebi, obrađuje ogromnu količinu podataka na savršeno odgovarajući način za vaše potrebe. Tu nastupaju indeksi. Indeksi se koriste da kažu bazi podataka kako da organizuje podatke, na takav način da se upiti mogu efikasno pokrenuti, iako baza podataka sadrži ogromnu količinu elemenata.

Indeks je način da organizujete dokumenta, kako biste omogućili brže preuzimanje. Jednostavno poređenje bez analogije sa bazom podataka, bila bi knjiga recepata, koja može imati indeks, koji navodi sve recepte po njihovom nazivu sa brojem njihove stranice. Ako čitalac traži određeni recept, ovaj indeks omogućava izbegavanje pretraživanja ili čitanja, knjige u celini, samo da bi pronašao ono što ga interesuje. Princip je isti za indekse baza podataka. Glavni cilj jeste da se izbegne skeniranje celokupne baze podataka, svaki put kada tražimo određene podatke. Iako su računari mnogo brži od ljudi pri skeniranju podataka, ovaj zadatak može biti dugotrajan kada se radi sa hiljadama ili milionima dokumenata. U formalnijim terminima, vremenska složenost upita koji se pokreće na bazi podataka sa  $n$  dokumenata biće  $O(n)$  bez indeksa, jer zahteva da se svi dokumenti skeniraju uzastopno. U literaturi postoji mnogo vrsta indeksa. Jedna od najčešćih struktura indeksa je B-stablo: ono je podrazumevano u PostgreSQL-u, kao i u MongoDB-u. Ova struktura indeksa performantna je u  $O(\log n)$  u proseku, što garantuje veoma brzo vreme pristupa čak i sa velikim količinama podataka.

### 4.2 CouchDB indeksi

Indeksi omogućavaju da se upiti nad bazom podataka pozivaju bez potrebe da se ispituje svaki red sa svakim upitom, što ih čini bržim i efikasnijim. Obično su indeksi napravljeni za kriterijume upita, koji se često pojavljuju, što omogućava efikasnije ispitivanje podataka. Da bi se iskoristila glavna prednost CouchDB-a – mogućnost obavljanja bogatih upita prema JSON podacima – indeksi nisu neophodni, ali se toplo preporučuju zarad boljih performansi.

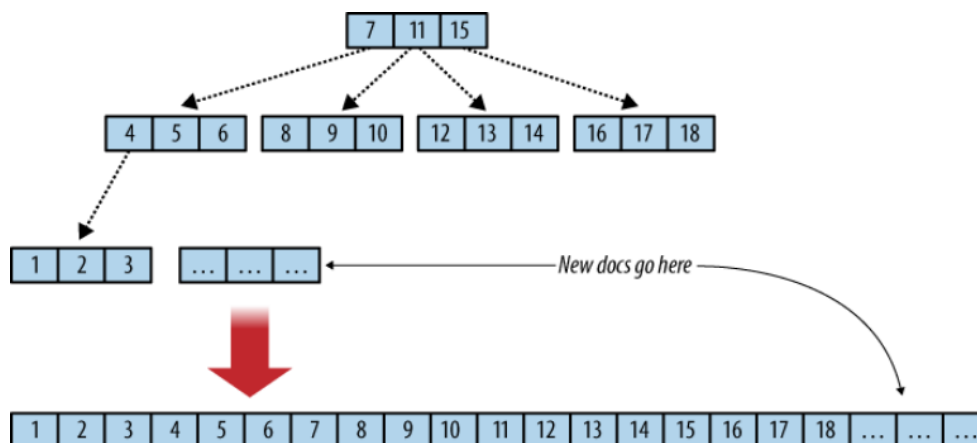
Takođe, ako je sortiranje potrebno u upitu korisnika, CouchDB zahteva izgradnju indeksa koji bi uključio sva sortirana polja.

Sekundarni indeksi u CouchDB-u se ne ažuriraju tokom operacija pisanja dokumenta. Da bi se izbegla velika kašnjenja pri čitanju indeksa nakon velikog bloka upisa, CouchDB automatski pokreće pozadinske poslove kako bi sekundarne indekse održavao „toplim“. Demon odgovoran za ovaj proces interno je poznat kao *ken* i može se konfigurisati korišćenjem podešavanja.

CouchDB poseduje interni podsistem, koji može dati prioritet IO operacijama, povezanim sa određenim klasama operacija. Ovaj podsistem se može konfigurisati da ograniči resurse posvećene pozadinskim operacijama kao što su interna replikacija, zbijanje ili sekundarno indeksiranje.

Kod CouchDB baze podataka, indeksi su zapravo B+ stabla, što predstavlja strukturu, izvedenu iz B-stabala. Indeksirani podaci su zapravo polja dokumenata. Kada su indeksirani, postavljeni su ključevi i organizovani su kao stablo, sa dokumentima uskladištenim u listovima čvorova. Dakle, pronalaženje određenog dokumenta podrazumeva pretraživanje stabla, od korena do lista.

Ključevi su organizovani prema njihovoj vrednosti. Ako pretpostavimo da čvor poseduje dva ključa  $k_1$  i  $k_2$  i tri podređena čvora, krajnji levi podređeni čvor mora imati sve svoje ključeve inferiorne u odnosu na  $k_1$ , krajnji desni podređeni čvor mora imati sve svoje ključeve superiornije u odnosu na  $k_2$ , a ključevi srednjeg podređenog čvora moraju biti između vrednosti ključeva  $k_1$  i  $k_2$ . Na slici ispod, pozajmljenoj iz [vodiča za CouchDB](#), može se uočiti, da svaki čvor ima tri ključa, ali u CouchDB implementaciji, čvor B+ stabla zapravo može da uskladišti više od 1600 ključeva.



Slika 4. B-stablo

Ovaj dizajn je veoma efikasan za upite iz željenog opsega. U gornjem primeru, upit koji bi tražio dokumente sa indeksiranim poljem između 1 i 3 će pretražiti stablo da pronađe prvu poziciju dokumenta (sa ključem 1), a zatim će dobiti sledeća dva susedna dokumenta. Indeksiranje podataka nije tako lako kao što izgleda. Ne postoji univerzalan način za indeksiranje. Organizacija indeksa mora zavisiti od korisničkog modela podataka, ali i upita koje treba pokrenuti.

Kreiranje indeksa pretrage u CouchDB bazi podataka, postiže se dodavanjem Java-Script funkcije u dokumentu dizajna u bazi podataka. Indeks se gradi nakon obrade jednog zahteva za pretragu ili nakon što server detektuje ažuriranje dokumenta. Funkcija indeksa prihvata sledeće parametre:

1. Field name - ime polja koje želite da koristite kada postavljate upit za indeks. Ako ovaj parametar postavite na podrazumevani, onda se ovo polje ispituje ako nijedno polje nije navedeno u sintaksi upita.
2. Data - podaci koji želite da indeksirate, na primer *dokument.adresa*.
3. Opcioni treći parametar uključuje sledeća polja: *boost*, *facet*, *index*, i *store*.

Broj redova koji DBMS vraća korisniku, može se promeniti korišćenjem parametra *limit*. Svaki odgovor uključuje polje *bookmark*, čija vrednost može biti iskorišćena u budućim upitima. Primer dizajnerskog dokumenta koji definiše indeks pretrage:

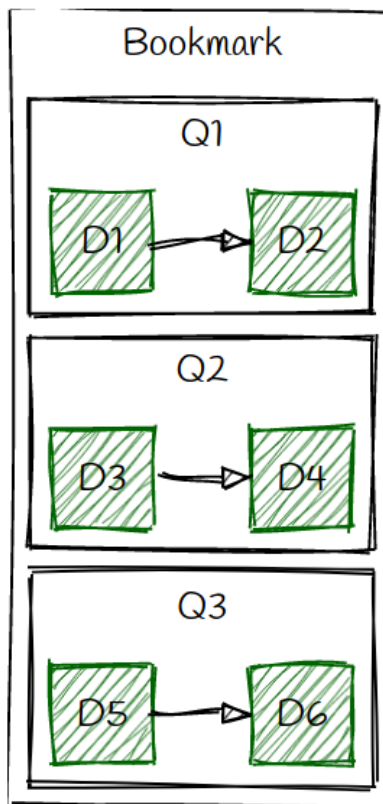
```
{
  "_id": "_design/primer_pretrage",
  "indexes": {
    "adresa": {
      "index": "function(doc){ ... }"
    }
  }
}
```

*Kod 11. CouchDB indeks pretrage*

Kada ima previše podataka za rukovanje odjednom na strani klijenta, dobro je primeniti paginaciju. Ovo se sastoji od podele Q upita na nekoliko Qi podupita tako da klijent može progresivno da preuzima podatke i ažurira korisnički interfejs u skladu sa tim. Iz perspektive korisničkog interfejsa, postoje različite strategije za implementaciju paginacije. Obično, ako se podaci prikazuju kao lista, možete imati dugme „Učitaj više“ na dnu ili automatski otkriti potrebu za preuzimanjem više podataka, kada je korisnik na kraju liste.

U CouchDB-u mogu da se koriste obeleživači (bookmarks) koji se mogu tumačiti kao pokazivač stabla, čime možete da naznačite gde da započnete upit. Ovo je jednostavno string koji vraća svaki Qi podupit, koji možete proslediti kao

parametar za sledeći upit. Na taj način, svaki  $Q_i$  podupit počinje na kraju prethodnog upita, izbegavajući nepotrebno skeniranje:



*Slika 5. Paginacija*

Ovaj sistem obeleživača ne postoji u MongoDB ili PostgreSQL bazama podataka. Međutim, i dalje možete imati koristi od efikasne paginacije korišćenjem poslednjeg ključa dokumenta koji je vratio prethodni upit. Kako je stablo organizovano po ključevima,  $Q_i$  podupit će biti ispravno pozicioniran u stablu, odnosno na kraju prethodnog upita.

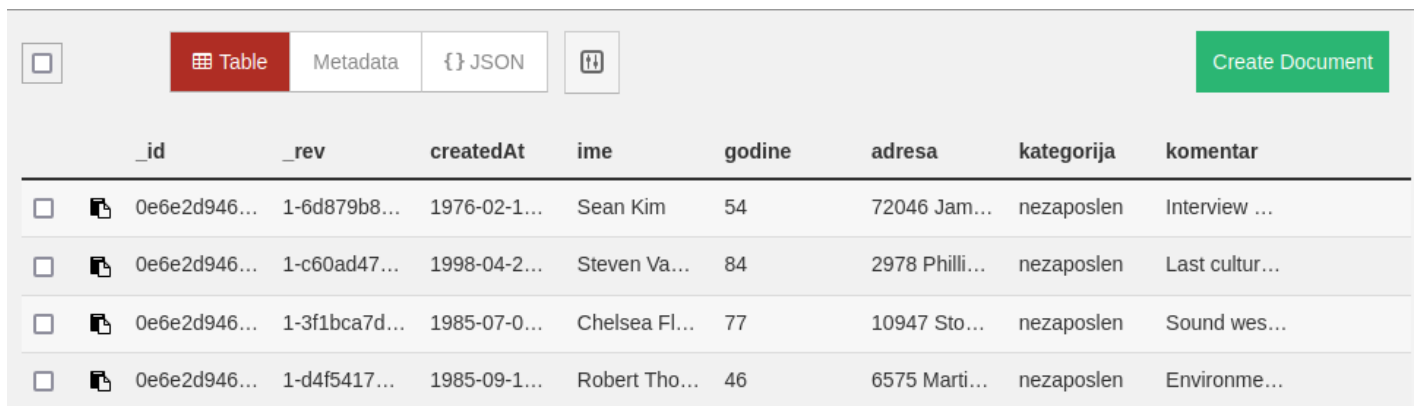
Prilikom dizajniranja indeksa, može pomoći da se ima na umu kako je B-stablo strukturirano, posebno kada se radi o indeksima sa više polja. Ako treba da postavite upit za nekoliko polja istog indeksa, postoji opasnost od generisanja suboptimalnih upita. Da bi ovaj upit bio efikasan, potrebno je napraviti indeks koji će organizovati podatke u susednim redovima.





Prilikom kompleksnijih upita, gde indeksiranje u susednim redovima nije od pomoći, pribegava se korišćenju parcijalnih indeksa. Koriste se za izražavanje predikata koji će biti evaluirani u trenutku indeksiranja. Podaci se tada indeksiraju samo ako se poklapaju sa predikatima. Ovakvo indeksiranje je korisno i kod upita sa konstantnim vrednostima.



## 4.2 CouchDB značaj indeksa

Zarad demonstracije i provere značaja indeksa, korišćenjem Python programskog jezika, kreirana je velika baza podataka sa tačno 419 563 nasumično generisanih dokumenata. Kreiranje ovako velike baze (veličina na disku 200MB) je trajalo oko 2 sata na *AMD Ryzen 7 4800H* procesoru sa 8 fizičkih jezgara i 32GB DDR4 RAM memorije, što nam pokazuje koliko je upis u bazu “skup”. Pritom su korišćene Python biblioteke [couchdb](#) i [faker](#). Faker je paket koji omogućava generisanje nasumičnih lažnih podataka, zbog čega omogućava brzo i efikasno pohranjivanje baze podataka. Svaki dokument poseduje sledeću strukturu:



		_id	_rev	createdAt	ime	godine	adresa	kategorija	komentar
<input type="checkbox"/>		0e6e2d946...	1-6d879b8...	1976-02-1...	Sean Kim	54	72046 Jam...	nezaposlen	Interview ...
<input type="checkbox"/>		0e6e2d946...	1-c60ad47...	1998-04-2...	Steven Va...	84	2978 Philli...	nezaposlen	Last cultur...
<input type="checkbox"/>		0e6e2d946...	1-3f1bca7d...	1985-07-0...	Chelsea Fl...	77	10947 Sto...	nezaposlen	Sound wes...
<input type="checkbox"/>		0e6e2d946...	1-d4f5417...	1985-09-1...	Robert Tho...	46	6575 Marti...	nezaposlen	Environme...

*Slika 6. Nasumično generisani dokumenti*

Najpre je neophodno ostvariti konekciju sa lokalnom CouchDB bazom podataka.

```
def db_connect() -> couchdb.Server:
    try:
        couch_server = couchdb.Server()
        couch_server.resource.credentials = ('aback', 'admin231')
        print("\nConnection to CouchDB server successful!")
        return couch_server
    except Exception as e:
        print("\nDB server connection error! {}".format(e))
```

*Kod 12. CouchDB lokalna konekcija*

Zatim, jednostavnom iteracijom, željeni broj puta, generišemo i skladištimo lažne podatke u vidu dokumenata. Svaki novo-generisani red podataka, predstavlja JSON objekat, koji se čuva u vidu novog dokumenta unutar CouchDB baze podataka.

```
def db_fake_populate(db_server, size) -> None:
    try:
        db = db_server.create("baza")
        for _ in range(size):
            my_dict = {'createdAt': str(FAKE.date_time_between(start_date='-50y',
end_date='now')), 'ime': FAKE.name(), 'godine': random.randint(18, 90), 'adresa':
FAKE.address(), 'kategorija': KATEGORIJA[random.randint(0, 2)], 'komentar':
FAKE.text()}
            print("\n{}".format(my_dict))
            db.save(my_dict)
    except Exception as e:
        print("\nDB already created! {}".format(e))
```

*Kod 13. CouchDB generisanje podataka*

CouchDB omogućava upit podataka na više načina. U ovom radu ćemo uporediti brzinu izvršenja upita bez prethodnog indeksiranja, sa brzinom nakon indeksiranja u pogled (View). Kreiramo oba upita sa proverom godina starosti osobe, od 77, tako da podaci bivaju filtrirani samo na osobe sa toliko godina. Za potrebe testa brzine upita, bez indeksiranja, kreirana je funkcija koja koristi princip mango selektora. Definicija povećanja limita je obavezna, jer u suprotnom dobijamo jako mali broj povratnih rezultata.

```
def db_query_simple(db_server) -> None:
    db = db_server["baza"]
    mango = db.find({"selector": {
        "godine": { "$eq": 77 }
    },
    "limit": 1000000
    })
    results = mango
    for row in results:
        print("\n {} {} {} {}".format(row['ime'], row['godine'], row['kategorija'],
row['createdAt']))
```

*Kod 14. CouchDB mango selektor*

Kreiranje željenog indeksa u pogled, je nešto kompleksnije. Najpre definišemo Java-Script funkciju koja će kreirati željeni pogled, u vidu dokumenta spremnog za čitanje:

```
function(doc) {
  if (doc.godine == 77)
    emit([doc.ime, doc.godine, doc.kategorija, doc.createdAt]);
}
```

*Kod 15. CouchDB Java-Script kreiranje pogleda*

Nakon kreiranja pogleda pozivamo sledeću Python funkciju, pomoću koje čitamo rezultate:

```
def db_get_view(db_server) -> None:
    db = db_server["baza"]
    results = db.view('_design/view_all/_view/new-view')
    for row in results:
        print(row['key'])
    print("\nView size: {}".format(len(results)))
```

*Kod 16. CouchDB View Query*

Upit bez prethodnog indeksiranja se izvršavao znatno duže od upita pogleda koji je indeksiran. Vreme izvršenja upita korišćenjem mango selektora je oko 25 sekundi. Vreme potrebno za izvršenje ostaje nepromenjeno prilikom svakog sledećeg izvršenja identičnog upita. Broj filtriranih rezultata je jednak kod oba upita i iznosi 5 815 dokumenata.

```
Katherine Turner 77 nezaposlen 2019-08-31 21:14:48
Hannah Chang 77 student 1980-04-11 16:54:48
Mark Pittman 77 radnik 1995-03-26 23:14:25
Sandra Baker 77 radnik 1992-12-18 05:50:00
real    0m25,386s
```

*Slika 7. Mango selektor vreme izvršenja*

Vreme potrebno za izvršenje indeksiranog upita je višestruko kraće, pri čemu nije neophodno inkrementirati parametar *limit*, jer ne postoji nikakvo ograničavanje povratnih rezultata. Izvršenje upita je trajalo svega 0,5 sekundi.

```
['Zachary Salazar', 77, 'radnik', '2009-08-11 00:31:16']
['Zachary Thompson', 77, 'student', '1983-05-14 22:22:11']
['Zachary Young', 77, 'nezaposlen', '2019-03-07 13:01:20']
['Zoe Gonzalez', 77, 'radnik', '2010-12-06 17:58:38']
['Zoe Quinn', 77, 'student', '1972-09-23 01:52:31']

View size: 5815
real    0m0,507s
```

*Slika 8. View Query vreme izvršenja*

## 5 Zaključak

CouchDB se razlikuje od svih drugih baza podataka, ali pozajmljuje dobar broj dobro poznatih i dobro shvaćenih paradigmi, kako bi svojim korisnicima pružio jedinstveni skup funkcija. Osnovno skladištenje podataka u JSON-u i preko HTTP-a čine fleksibilno i moderno skladište podataka sa fleksibilnim mehanizmom upita.

Replikacija omogućava podacima da egzistiraju, tamo gde su potrebni, bilo da su u grupi koja obuhvata više centara podataka ili na telefonu krajnjeg korisnika. MapReduce paradigma, omogućava manipulaciju ogromnom količinom podataka, na decentralizovan i efikasan način, čime daje replikaciji još više na značaju.

Dizajniran oko bezbednosti podataka, CouchDB pruža veoma efikasno rešenje za skladištenje podataka koje je otporno na sistemske greške, greške u mreži kao i na iznenadne skokove u saobraćaju, a uz sve to je još i nezavisni projekat otvorenog koda sa liberalnom licencom.

# Reference

1. <https://docs.couchdb.org/en/stable/>
2. <https://blog.nahurst.com/visual-guide-to-nosql-systems>
3. <https://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis>
4. GAJENDRAN, Santhosh Kumar: A survey on nosql databases. In: University of Illinois (2012)
5. <http://www.odcms.org/wpcontent/uploads/2010/01/Cattell.Dec10.pdf>
6. Julian Browne. Brewer's CAP Theorem. <https://www.julianbrowne.com/article/brewers-cap-theorem>
7. <https://guide.couchdb.org/editions/1/en/btree.html>
8. <https://readthedocs.org/projects/couchdb-python/downloads/pdf/latest/>
9. <https://use-the-index-luke.com/no-offset>
10. <https://www.codementor.io/@arpitbhayani/fast-and-efficient-pagination-in-mongodb-9095flbqr#approach-2-using-id-and-limit>
11. Chris Anderson, Jan Lehnardt, Noah Slater - CouchDB The Definitive Guide
12. Bradley Holt - Scaling CouchDB Replication, Clustering, and Administration
13. Bradley Holt - Writing and Querying MapReduce Views in CouchDB
14. Martin C Brown - Getting started with CouchDB