Matematički fakultet
Univerzitet u Beogradu

Kristina Stefanović

Razmatranje nerelacionih karakteristika SUBP MariaDB na primeru razvoja veb aplikacije za razmenu poruka i dokumenata

Master rad

Beograd, 2018.

Matematički fakultet
Master rad

Autor:	Kristina Stefanović
Naslov:	Razmatranje nerelacionih karakteristika SUBP MariaDB na primeru razvoja veb aplikacije za razmenu poruka i dokumenata
Mentor:	dr Saša Malkov, Matematički fakultet
Članovi komisije:	dr Gordana Pavlović-Lažetić, Matematički fakultet, dr Nenad Mitić, Matematički faklutet
Datum:	20.09.2018.

Sadržaj:

1 Uvod 4
1.1 Baze podataka 4
1.2 Relacione baze podataka 4
1.3 Nerelacione baze podataka 6
2 *MariaDB* 9
2.1 Osnovne karakteristike sistema *MariaDB* 9
Podsistemi dostupni za integraciju sa sistemom *MariaDB* 9
Napredna pretraga teksta (engl. *full-text search*) 12
Rad sa geografskim i geometrijskim podacima 15
2.2 Relacione karakteristike sistema *MariaDB* 16
Transakcije 16
Strani ključevi 16
Virtualne kolone 16
2.3 Nerelacione karakteristike sistema *MariaDB* 17
Dinamičke kolone 18
Podsistem *Cassandra* 21
3 Aplikacija za razmenu poruka i dokumenata 22
3.1 Arhitektura aplikacije 22
3.2 Baza podataka 23
3.3 Implementacija baze podataka na sistemu MariaDB 25
4 Diskusija 31
4.1 Relaciono predstavljanje dinamičkih kolona 31
4.2 Simbioza relacionih i nerelacionih baza podataka 32
4.3 Rad sa dinamičkim kolonama 32
4.4 O specifičnostima sistema *MariaDB* 32
5 Zaključak 34
6 Reference 35

1 Uvod

Baze podataka su sastavni deo svakog softverskog rešenja koje ima potrebu da čuva i obrađuje podatke. Sami podaci i način na koji je bilo potrebno upravljati njima su u velikoj meri uticali na razvoj baza i sistema za upravljanje bazama podataka. Kako su se podaci i način na koji se upravlja podacima menjali, menjale su se i potrebe za svojstvima sistema za obradu baza podataka. Na primer, nekada se podrazumevalo u svim slučajevima da podaci budu konzistentni u svakom trenutku, da strukture podataka budu poznate u trenutku pravljenja sheme, da nema duplikata i slično. Međutim, kako je tokom godina količina podataka koji se svakodnevno generiše značajno porasla, pojavila se potreba za sistemima čija će svojstva i način rada biti drugačiji. Duplikati podataka se više ne izbegavaju. Jedna od poželjnih osobina sistema je dinamička shema podataka. Ovo su samo neki od razloga zbog kojih je dalji razvoj sistema za upravljanje bazama podataka bio usmeren ka nerelacionom pristupu.

Sistem za upravljanje bazama podataka *MariaDB* je rešenje koje je između ova dva pristupa i koje ima svojstva i relacionog i nerelacionog pristupa.

Cilj ovog rada je istraživanje sistema *MariaDB*. U okviru drugog poglavlja izložene su osnovne i napredne karakteristike sistema *MariaDB*. Pored toga, sagledana su relaciona i nerelaciona svojstva koja su podržana sistemom. Kako bi se stekao potpuniji utisak o sistemu, u sklopu ovog rada, kombinovanjem relacionog i nerelacionog pristupa razvijena je i aplikacija za razmenu poruka i dokumenata. Cilj razvoja aplikacije je da pruži dodatne informacije o korišćenju sistema, da pruži uvid u to koliko je razvoj ovakvog sistema kompleksan u odnosu na korišćenje samo relacionog ili samo nerelacionog pristupa. Informacije dobijene prilikom izrade aplikacije izložene su u trećem poglavlju master rada, dok su utisci tome da li ovakav pristup olakšava ili otežava izradu aplikacije i koja su alternativna rešenja izloženi u četvrtom poglavlju master rada.

1.1 Baze podataka

U osnovi razvoja svakog softverskog rešenja je sistem za upravljanje bazama podataka. U odnosu na svojstva koje imaju možemo ih grubo podeliti u dve grupe: sisteme zasnovane na relacionom i sisteme zasnovane na nerelacionom modelu.

Sistem *MariaDB* je predstavnik nove ideje u kojoj su podržana i svojstva iz relacionog modela i svojstva iz nerelacionog modela i zbog toga ne pripada ni jednoj od navedenih kategorija.

1.2 Relacione baze podataka

Pre nastanka sistema za upravljanje bazama podatka, podaci su reprezentovani fajlovima u direktorijumima, a obrada podataka je podrazumevala pisanje programa koji će im pristupati i izvršavati operacije nad njima. Ovakav pristup je imao dosta nedostataka:

- Ponavljanje podataka: isti podaci se mogu javljati u više dirketorijuma

- Nekonzistentnost podataka: kako se isti podaci mogu javiti u više direktorijuma može se desiti da se promena podataka ne izvede dosledno, već da negde promena bude izostavljena
- Programi za obradu podataka zavise od načina struktuiranja podataka: programi su pisani u skladu sa organizacijom direktorijuma, stoga svaka izmena strukture podataka zahteva promenu programa
- Korišćenje istih podataka od strane više korisnika nije bezbedno: ukoliko više korisnika menja isti podatak, samo poslednja promena će biti zapamćena, bez ikakvog upozorenja drugih korisnika
- Nedostatak mehanizma za oporavak od greške: ukoliko dođe do kvara, moguće je da će se deo programa za obradu podataka izvršiti, a drugi deo programa ne, što nas dovodi do nekonzistentnog stanja podataka.

Relacioni sistemi za upravljanje bazama podataka implementiraju mehanizme koji sprečavaju gore navedene probleme i to na sledeći način:

- Dupliranje podataka se izbegava normalizacijom podataka. Normalne forme su zasnovane na funkcionalnoj zavisnosti podataka. Razlikujemo pet normalnih formi i normalnu formu Boyce-Codd.
- Kao upitni jezik koristi se *SQL* (engl. *Structured Query Language*)
- Kako bi podaci u svakom trenutku bili konzistentni implementirani su mehanizmi koji se bave integritetom podataka. Integritet podataka se zasniva na proveravanju uslova koji mogu biti definisani na nivou samog podatka, na nivou relacije i na nivou povezanosti relacija.
- Bezbedno korišćenje istih podataka od strane više korisnika je omogućeno mehanizmom transakcija. Osobine transakcija su atomičnost, konzistentnost, izolovanost i trajnost (*ACID*) i one prevode bazu iz jednog konzistentnog stanja u drugo konzistentno stanje. Ovo je omogućeno različitim načinima zaključavanja objekta, gde objekat može biti cela relacija, red, nekoliko redova i slično.
- Za oporavak od greške, bilo kojeg tipa, zadužen je upravljač oporavka. On u skladu sa situacijom koja je nastala sprovodi neophodne korake kako bi se omogućio rad sa poslednjim konzistentnim stanjem baze.

Prethodno navedena svojstva relacionih sistema za upravljanje bazama podataka zaslužna su za visok nivo pouzdanosti i stabilnosti. Za aplikacije koje to zahtevaju relacioni model je i dalje najbolje rešenje.

Međutim, kako se promenio obim podataka koji se svakodnevno generiše i kako se pojavila potreba za aplikacijama koje će te podatke koristiti, pred sisteme za upravljanje bazama podataka su stavljena dva nova uslova: sistem bi trebalo da sa istom lakoćom obrađuje veliku količinu podataka kao i da omogući dinamičko menjanje strukutre podataka. Na ova dva uslova relacioni sistemi podataka ne mogu odgovoriti na dobar način, odnosno dobrim performansama. Neki od razloga su to što je distribucija podataka otežana kako bi se zadržala konzistentnost, takođe zbog normalizacije i neredundantnosti podataka operacije spajanja podataka će se izvršavati sporo. Promena strukture podataka bi bila jako skupa, i implementiranje rešenja koje bi moglo da radi sa dinamičkom srukturom bilo bi vrlo neefikasno.

1.3 Nerelacione baze podataka

Nerelacioni pristup bazama podataka se zapravo pojavio prilično davno. On je prethodio relacionom modelu u vidu mrežnih ili hirerarhijskih modela podataka. Međutim, kada danas govorimo o nerelacionim bazama podataka, obično podrazumevamo da se radi o tzv. „postrelacionim“ sistemima, tj. novim generacijama, najčešće distribuiranim sistemima sa sledećim svojstvima:

- Promenljiva struktura podataka (horizontalno skalabilna)
- Ne insistiraju na integritetu podataka
- Ne koriste upitni jezik *SQL*
- Omogućavaju lako dodavanje čvorova (vertikalno skalabilna)
- Najčešće prave kompromis u konzistentnosti podataka

Prvi termin koji se koristio u kontekstu nerelacionih baza podataka je *NoSQL*. Prvi ga je upotrebio Karlo Strozi 1998. godine prilikom razvoja baze podataka. Međutim, baza je i dalje bila relaciona s tim što nije koristila *SQL* kao upitni jezik. Nakon toga, 2009. godine termin je upotrebio Erik Evans kada je organizovana diskusija na temu novog naleta nerelacionih baza podataka. Značenje termina tada je mnogo sličnije značenju koje danas podrazmevamo pod tim terminom nego u trenutku kada je sam termin nastao. Danas je prirodnije koristiti „ne samo SQL“ (engl. *Not only SQL*) zato što praktično svaka nova baza sadrži upitni jezik koji je nalik na SQL.

Kako su nerelacioni sistemi najčešće distribuirani sistemi, oni se oslanjaju na teoremu *CAP*. Teorema *CAP*, ili Bruerova teorema, nazvana po Eriku Brueru tvrdi da distribuirani sistem može zadovoljiti samo dva od tri navedena kriterijuma:

- Konzistentnost (engl. *Consistency*): Sva čitanja, na svim čvorovima moraju da daju isti podatak, odnosno čitanje podataka ne zavisi od čvora sa kojeg se podatak čita.
- Raspoloživost (engl. *Availability*): Svaki čvor koji je dostupan uvek izvršava upite, odnosno na svaki zahtev će u nekom trenutku dati neki odgovor.
- Prihvatanje podeljenosti (engl. *Partition tolerance*): Sistem mora da zadovolji dva gore navedena kriterijuma čak i u slučaju da su neki čvorovi nedostupni, to jest da je došlo do prekida komunikacije između čvorova.

Kako po teoremi *CAP* sistem može da ispunjava samo dva od tri gore navedena uslova, sistem pripada jednoj od sledeće tri kategorije:

- CA: podaci moraju biti konzistenti na svim čvorovima. Sve dok su svi čvorovi dostupni, korisnici mogu biti sigurni da će nakon operacija čitanja i pisanja podaci biti isti na svim čvorovima. Ovakvi sistemi rade dobro sve dok nema particionisanja čvorova. Oni su prihvatljivo rešenje samo ukoliko imamo samo jedan čvor, odnosno ako nema distribuiranosti. Na ovakav vid kompromisa se retko pristaje.
- CP: podaci su u svakom trenutku konzistentni. Ukoliko dođe do particionisanja čvorova, čvor koji je i dalje dostupan odbija zahteve kako ne bi došlo do narušavanja

konzistentosti. U ovom slučaju žrtvuje se raspoloživost sistema. Kako sistem koji nije raspoloživ najčešće nije koristan i ovaj vid kompromisa se obično izbegava.

- AP: čvorovi ostaju dostupni i kada ne mogu da komuniciraju jedni sa drugim, podaci će biti sinhronizovani onda kada je particionisanje eliminisano. U ovom slučaju se ne garantuje da će svi čvorovi imati iste podatke (usled prekida komunikacije čvorova). Ovde se pravi kompromis na račun konzistentnosti podataka.

Iako nije moguće napraviti distribuirani sistem koji zadovoljava sva tri uslova to ne znači da nije moguće napraviti sistem koji će zadovoljavati potrebe rešenja. Bitno je razumeti potrebe i na osnovu toga rešiti koji kompromis je najprihvatljiviji.

Ove osobine su u skladu sa skupom uslova koji se naziva *BASE*, koji je manje strog od *ACID* svojstava transakcija. Skup uslova *BASE* podrazumeva:

- Uglavnom raspoloživ sistem (engl. *Basically Available*): ukazuje na to da sistem garantuje dostupnost,
- Promenljivog stanja (engl. *Soft state*): kaže da se stanje može promeniti u toku vremena čak i bez bilo kakve transakcije.
- Konvergentna konzistencija (engl. *Eventually consistent*): znači da će sistem postati konzistentan u nekom trenutku, ali da će u međuvremenu i dalje raditi i davati potencijalno različite odgovore.

Kako danas postoji veliki broj nerelacionih baza podataka, teorema *CAP* može biti jedan od načina na koji te sisteme možemo kategorizovati po tome na koji se kompromis pristaje.

Drugi način da se izvrši klasifikacija jeste da na osnovu podataka koji se čuvaju i sa kojima se radi, pa na osnovu toga imamo četiri kategorije:

- ***Model ključ-vrednost***: na ovaj model se često referiše kao na model bez sheme podataka. Sastoji se od skupa ključ-vrednost podataka pri čemu su ključevi jedinstveni. Ključeve možemo ili napraviti sami ili mogu biti napravljeni od strane programa. Vrednost podatka može biti niska ili *JSON* objekat ili *BLOB*. Najveća prednost ovog modela je brzina i jednostavnost podataka. Pretraga se može vršiti samo nad ključevima podataka što predstavlja jedno od ograničenja ovog modela. Takođe, sa velikom količinom podataka, ukoliko sami određujemo vrednost ključeva, naći novu jedinstvenu vrednost ključa može predstavljati izazov.
- ***Model dokumenata***: ovaj model se takođe sastoji od ključa i vrednosti. Razlika u odnosu na prethodni model je ta što vrednost može sadržati struktuirane podatke pri čemu se za njihovo predstavljanje mogu koristiti *JSON*, *XML*, *BSON*. Vrednosti koje se smeštaju se mogu značajno razlikovati u strukturi. Velika prednost ovog modela ogleda se u tome što se po dodavanju podatka u bazu prave i metapodaci koji omogućavaju pisanje upita koji može vršiti pretragu i prema sadržaju koji je smešten u vrednost.
- ***Model orijentisan ka kolonama:*** ovaj model se zasniva na kolonama i ključevima. Svaki podatak ima svoj ključ koji ga identifikuje, a same vrednosti podataka se smeštaju u kolone. Postoje raličite vrste kolona. Prve su regularne kolone, slične atributima u relacionim bazama podataka koje sadrže jednu vrednost. Pored njih postoje i super kolone koje predstavljaju kolone koje mogu sadržati više regularnih kolona. Broj kolona koji može biti smešten u super kolonu je neograničen. Takođe postoje i familije kolona

koje konceptualno odgovaraju tabeli u relacionim bazama podataka. Kod ovog modela vrednosti kolona se smeštaju kontinualno na disk što vrlo povoljno utiče na brzinu izvršavanja upita koji pristupaju podacima ili sadrže agregatne funkcije. Kod većine relacionih baza podataka to nije slučaj. Sam red se smešta kontinualno na disk, ali ne i više redova.

- ***Model orijentisan ka grafovima***: svi podaci su opisani čvorovima i njihovim vezama sa drugim čvorovima. Čvor sadži sva svojstva nekog entiteta, ali čvorovi istog tipa ne moraju imati iste odlike. Veze sa drugim čvorovima se predstavljaju usmeravajućim granama. Takođe, vezi između dva čvora možemo dodeliti neka svojstva. Ovaj model ima dosta sličnosti sa relacionim bazama, ali ono što mu daje prednost u odnosu na relacione baze je to što lako možemo dobiti informacije za koje bismo u relacionom modelu morali da spajamo nekoliko tabela što je skupa operacija.

Sistem *MariaDB* je sistem koji radi sa dinamičkim kolonama. Iako se ne može svrstati ni u relacione ni u nerelacione baze podataka, model nerelacionih baza sa kojim deli najviše sličnosti je model orijentisan ka kolonama.

Nerelacione baze podataka su jako popularne u današnje vreme, ali to ne znači da one mogu zameniti relacioni model. Prilikom izbora sistema za upravljanje bazama podatka koji je potreban za razvoj softverskog rešenja treba dobro sagledati kakva svojstva rešenje treba da zadovolji i na osnovu toga se odlučiti za relacioni ili nerelacioni model.

2 *MariaDB*

MariaDB je sistem za upravljanje bazama podataka. Sistem *MariaDB* je sistem otvorenog koda, licenciran GNU javnom licencom.
Sistem *MariaDB* i sistem *MySQL* su dva binarno kompatibilna sistema, odnosno oba sistema mogu izvršavati isti izvršni kod. Oba sistema koriste iste nazive fajlova, iste putanje, protokole, interfejse, biblioteke za povezivanje sa različitim programskim jezicima, itd. Stoga, prelazak sa jednog sistema na drugi ne iziskuje puno vremena, dovoljno je samo instalirati servis *MariaDB* i deinstalirati servis *MySQL*.
Razvoj sistema je započet 2009. godine. Od tada, konstantno je rađeno na unapređivanju rešenja i uvođenju novih funkcionalnosti. Prvo objavljeno rešenje bilo je *MariaDB* 5.1. Poslednja objavljena verzija je *MariaDB* 10.3.

2.1 Osnovne karakteristike sistema *MariaDB*

Sistem *MariaDB* proširuje spektar funkcionalnosti sistema *MySQL* omogućavajući podršku za statističku obradu podataka, rad sa geografskim podacima, kao i različite funkcije za obradu teksta.

MariaDB je skalabilno rešenje koje može biti dobar izbor i u slučaju da softversko rešenje obrađuje veliku količinu podataka, ali isto tako i kada to nije slučaj. Ovo je opravdano time što *MariaDB* podržava i relacione i nerelacione koncepte sistema za upravljanje bazama podataka. Dobra osobina ovog sistema je sigurnost i bezbednost. Ona je postignuta saradnjom razvojnog tima sa organizacijom *Mitre*, koja se bavi otkrivanjem ranjivosti rešenja. Kod sistema *MariaDB* se koristi enkripcija u podrazumevanom režimu rada. S obzirom da je rešenje otvorenog koda ova činjenica dosta doprinosi bezbednosti rešenja.

Prilikom razvoja sistema *MariaDB* stalno se radilo na unapređivanju postojećeg stanja i uvođenju novih funkcionalnosti. Neke od novih funcionalnosti su: dinamičke kolone, virtualne kolone, omogućavanje rada sa geografskim i geometrijskim podacima. Mogućnost koja je podrazmevana od verzije 10.1 je integracija sa rešenjem *Galera*. *Galera* je alat koji omogućava rad sa klasterima i ono što ga razlikuje od servera *MySQL* je to što on omogućava master-master replikaciju, dok je *MySQL* podržavao master-slave replikaciju.

Podsistemi dostupni za integraciju sa sistemom *MariaDB*

MariaDB ima nekoliko integrisanih podsistema koji podržavaju rad sa bazom podataka. Podsistemi su odgovorni za izvršavanje upita, tj. oni preuzimaju upit od korisnika, izvršavaju isti nad bazom i vraćaju dobijeni rezultat. Čitanje i pisanje podataka u sistemu *MariaDB* se izvršava tako što se te naredbe delegiraju podistemima. Pored čitanja i pisanja podataka, podsistemi mogu pružiti i podršku za transakcije, keširanje podataka i indeksa i referencijalni integritet. Šta će biti podržano konkretnim podsistemom zavisi od njegove namene. Na primer, neki podržavaju rad sa transakcijama, drugi rad sa klasterima, itd.

Prilikom pravljenja tabele može se navesti koji podsistem će se koristiti za rukovanje podacima napravljene tabele. Način da se navede podsistem je korišćenje ENGINE klauzule. ENGINE klauzula je opciona i ukoliko se klauzula ne navede biće korišćen podrazumevani sistem

InnoDB. Pregled podsistema koji su podrazumevani u sistemu *MariaDB* se može dobiti izvršavanjem naredbe SHOW ENGINES.

U sklopu sistema *MariaDB* postoji nekoliko podsistema. Neki od njih se mogu koristiti odmah, dok je za druge nepohodna aktivacija. Pored njih postoje i drugi sistemi koji su nezavisni od sistema *MariaDB* i njih je potrebno instalirati i integrisati sa sistemom.

Za instaliranje podsistema koristi se naredba INSTALL SONAME "name", gde name označava ime podsistema, dok se deinstaliranje izvršava pokretanjem naredbe UNINSTALL SONAME "name" pri čemu name ima isto značenje kao u naredbi INSTALL.

MariaDB, kao i *MySQL* nudi fleksibilnost jer u okviru iste sheme možemo imati tabele napravljene različitim podsistemima, međutim to može dovesti do problema jer različiti podsistemimi mogu predstavljati podatke na različite načine pa usklađivanje podataka može biti skupo i neefikasno. Takođe, problem može nastati i prilikom rada sa transakcijama jer različiti podsistemi imaju drugačije mehanizme zaključavanja.

Sledi pregled podsistema:

- ***XtraDB*** je razvijan od strane kompanije *Percona* koja pruža podršku za *MySQL, MariaDB, MongoDB* i druge platforme. *XtraDB* podsistem je bio podrazumevani podsistem do verzije 10.1. posle koje se prešlo na *InnoDB* podsistem. *XtraDB* je verzija *InnoDB* kod koje su implementirane popravke za neželjena ponašanja koja je ispoljavao *InnoDB* podsistem. Pored toga rađeno je i na poboljšanju performansi.
- ***InnoDB*** je podrazumevani podsistem. On podržava transakcije, čuvanje međustanja prilikom izvršavanja transakcija, *XA*[1] transakcije, strane ključeve. Performanse ovog podsistema su jako dobre, pa je zato on jedan od najčešće korišćenih podsistema. Transakcije se vrše korišćenjem kompleksnog sistema zaključavanja i logova za poništavanje. Zaključavanje se vrši na nivou reda (ili nekoliko redova) koji su identifikovani preko indeksa. Logovi za poništavanje se koriste kada imamo potrebu za poništavanjem transakcije.
- ***TokuDB*** je proizvod kompanije *Tokutek*. Podržava transakcije, *XA* transakcije, indekse zasnovane na tekstu, ali ne i strane ključeve. Još jedna od njegovih prednosti je kompresija podataka. Koliki će nivo kompresije biti zavisi od podataka sa kojima se radi, ali generalno gledano, nivo kompresije je veći od nivoa kompresije koji pružaju drugi podsistemi. Ono što ovaj podistem donosi kao novinu je to što koristi fraktalna stabla kod indeksiranja. On je integrisan u sistem *MariaDB,* ali da bi se koristio mora se aktivirati.
- ***MyISAM*** je podsistem koji se obično koristi kod tabela koje nemaju veliki broj redova. Kod ove komponente katanci se stavljaju na nivou tabele, i najčešće se koristi u situacijama kada se uglavnom čitaju podaci. Ne podržava transakcije. Ne podržava strani ključ.
- ***Memory*** podsistem čuva sve podatke u memoriji i koristi se kada nam treba brza pretraga podataka. Jako je osetljiv na nestanak struje ili na otkazivanje hardvera. Po ponovnom pokretanju servera tabele će se rekreirati ali će biti prazne. Ovaj podsistem se sve ređe se koristi.

[1] XA transakcije su transakcije kod kojih je uključeno više resursa koji ne moraju nužno biti deo baze.

- ***CSV*** je podsistem čija je glavna odlika čitanje i pisanje podataka iz formata gde su podaci odvojeni zapetom. Ovaj podsistem se ređe koristi od kako se pojavio podsistem *CONNECT* koji pruža mnogo više mogućnosti.
- ***OQGraph*** je podsistem namenjen baratanju podacima koji imaju složenu, npr. stablo ili graf strukturu. Relacione baze podataka i *SQL* nisu najpogodniji za rad sa stablima, stoga ova komponenta prevodi *SQL* zahteve u zahteve koji odgovaraju složenoj strukturi podataka.
- ***SpinxSE*** je podsistem je koji daje podršku prilikom pretraživanja teksta. Predstavlja alternativu ugrađenoj opciji za pretragu teksta sistema *MariaDB*. Podsistem ne zavisi od sistema *MariaDB* i može se koristiti potpuno nezavisno.
- ***MROONGA*** je još jedan sistem koji je omogućuje pretragu teksta. On podržava kineski, japanski i korejanski jezik i njihove karaktere. Takođe podržava rad sa geometrijskim i geografskim podacima.
- ***Cassandra*** komponenta se koristi kada želimo da pristupamo podacima iz klastera *Cassandra*. Komponenta je dostupna od *MariaDB* 10.0 i njen cilj je da omogući integraciju između *SQL* i *NoSQL* sveta. Koristeći ovu komponentu možemo da vršimo *SQL* upite nad podacima, vršimo spajanje tabela, mapiramo podatke dobijene iz klastera u tabelu iz *MariaDB* sistema. Značajnu ulogu pri mapiranju imaju dinamičke kolone sistema *MariaDB,* jer se preko njih mogu mapirati super kolone iz sistema *Cassandra*.
- ***CONNECT*** komponenta omogućava sistemu *MariaDB* da pristupi lokalnim, eksternim ili podacima sa udaljenog čvora, pa čak i podacima iz drugih sistema za upravljanje bazama podataka. Izvori podataka mogu biti različitog oblika, mogu biti različiti formati kao što su *CSV, XML, HTML, JSON*, mogu biti različiti formati fajlova kao što su *INI, DOS, BIN*, ali i veze sa drugim sistemima kao što *JDBC, ODBC, MySQL, MongoDB*. Može se raditi i sa kompresovanim fajlovima formata *ZIP*. Ovaj podsistem je razvijen kao podrška sistemu *MariaDB*. Nakon njegovog razvoja, zbog bolje podrške neki od već postojećih podsistema se zanemaruju, kao što su na primer *CSV, FEDERATED*.
- ***SEQUENCE*** komponenta omogućava kreiranje rastućeg ili opadajućeg niza pozitivnih celih brojeva pri čemu se kao ulaz daju početna vrednost, inkrement i krajnja vrednost. Takođe, podsistem kreira privremene tabele kada su one potrebne, ali ne postoji način da se to odradi eksplicitno.
- ***ARCHIVE*** je podsistem koji se koristio pri radu sa kompresovanim tabelama. Ovaj podsistem je zastareo i umesto njega sve više koriste *TokuDB, InnoDB*.
- ***Spider*** podržava particionisanje, fragmentisanje, transakcije. Omogućava da se sa tabelama koje pripadaju različitim instancima *MariaDB* postupa kao da pripadaju istoj instanci. Podsistem *Spider* omogućava paralelno procesiranje velikog broja upita.
- ***MariaDB Column Store*** je komponenta koja je kreirana kao podrška za obradu ogromne količine podataka, linearnom skalabilnošću, odličnim perfomansama koja pritom daje odgovor u realnom vremenu pri postavljanju upita koji se bave analizom podataka.

Napredna pretraga teksta (engl. *full-text search*)

Naprednu pretragu teksta kod *MariaDB* je podržana u podrazumevanom podsistemu. Kao što je pomenuto u prethodnom poglavlju, podrazumevani podsistem je *InnoDB*, mada ovu funkcionalnost možemo dobiti i korišćenjnem drugih podsistema kao što su *MyISAM*, *Aria*, *Mroonga*.

Ovakav sistem pretrage teksta podrazumeva pretraživanje po svim rečima svih dokumenata sačuvanih u sistemu baza podataka. Dobre performanse ovog rešenja osigurane su korišćenjem indeksa izgrađenih nad tekstualnim kolonama. Tip takvog indeksa je *FULLTEXT* i može se izgraditi nad podacima koji su tipa *CHAR*, *VARCHAR* ili *TEXT*. Pravljenje samog indeksa možemo izvršiti na bilo koji od 3 načina. Naredbom *CREATE TABLE*, *ALTER INDEX* ili naredbom *CREATE INDEX* nad već postojećom tabelom kao što se može videti u sledećim primerima.

```
CREATE TABLE `posts` (
    `id` int NOT NULL AUTO_INCREMENT,
    `title` varchar(255) NOT NULL,
    `content` text,
    PRIMARY KEY (`id`),
    FULLTEXT (`content`)
) ENGINE=InnoDB;

ALTER TABLE `posts` ADD FULLTEXT(`content`);

CREATE FULLTEXT INDEX `content` ON `posts` (`content`);
```

Primer 1: pravljenje FULLTEXT indeksa nad kolonom

Svaki put kada se dodaju novi podaci u tabelu indeks se ažurira, stoga je u slučaju dodavanja velikog broja podataka preporučljivo prvo obrisati indeks, pa ga zatim ponovo napraviti. Kod standardne pretrage teksta kreiranje takvih indeksa nije moguće pa je zbog tog razloga za izvršavanje upita potrebno skenirati celu tabelu. Takvo rešenje je jako neefiksano i otuda interesovanje za ovakvom funkcionalnošću.

Funkcije preko kojih postavljamo upit mogu biti jako izražajne, na primer u njima možemo definisati da u rezultatu pretrage određena reč može da se pojavljuje ili da se ne pojavljuje uopšte ili čak da definišemo koliko nam je bitno da se neka reč pojavljuje. Da bi se postavio upit koriste se funkcije *MATCH* i *AGAINST*, gde funkciji *MATCH* prosleđujemo kao parameter nazive kolona nad kojima želimo da se izvrši pretraga, dok funkciji *AGAINST* prosleđujemo parametre pretrage. Rezultat pretrage je sortiran prema njegovom značaju, a sam značaj je definisan preko kriterijuma pretrage.

Postoje tri režima izvršavanja upita. Prvi režim, režim prirodnog jezika (engl. *natural language mode*) će izvršiti upit tako što će pretražiti dokumente koji imaju bar jednu od navedenih reči iz kriterijuma pretrage. Drugi, operatorski režim (engl. *boolean mode*) omogućava pisanje kompleksnijih upita korišćenjem različitih operatora koji mogu isključiti dokument iz rezultata ako sadrži određenu reč, dodati na važnosti nekoj reči ili vršiti pretragu na osnovu fraze. Treći režim se zove režim proširene pretrage (engl. *query expansion search*) i podrazumeva vršenje pretrage kroz dva upita. U prvom se vrši upit u režimu prirodnog jezika, a onda se u rezultatu pretražuju najrelevantnije reči, dodaju se pretrazi i upit se ponovo izvršava čime dobijamo više podataka kao rezultat pretrage.

Režim prirodnog jezika

U ovom režimu *MariaDB* će vratiti sve redove relevantne kriterijumu pretrage. Rezultati će biti sortirani prema relevantnosti, a sama relavantnost se računa na osnovu nekoliko kriterijuma. Na primer, neki od kriterijuma su: broj reči u samom redu, broj reči u tabeli, broj rezultata koji zadovoljavaju kriterijum pretrage, broj ponavljanja reči iz kriterijuma pretrage u samom redu. Relevantnost se predstavlja pozitivnim razlomljenim brojem, dok ukoliko nije ispunjen ni jedan uslov iz kriterijuma pretrage relevantnost je 0.

Ukoliko imamo tabelu sa albumima muzičkih izvođača sa svim njihovim pesmama, kao i rečima tih pesama upit kojim bismo pretražili da u nazivu pesme ili u tekstu pesme postoji reč "*Sunshine*" može izgledati na sledeći način:

```
SELECT * FROM songs
WHERE MATCH(title,lyrics) AGAINST('Sunshine');
```

Primer 2: pretraga kolona "title" i "lyrics" po parametru "Sunshine"

Kako su rezultati soritirani po relevantnosti, može nam biti od značaja da vidimo koliki je izračunati koeficijent relevantnosti za svaki od podataka iz rezultata. To možemo uraditi tako što u *SELECT* naredbi pored kolona koje želimo videti u rezultatu navedemo i podizraz koji se sastoji od obe funkcije kao jednu od kolona i u *WHERE* klauzuli navedemo kriterijume pretrage, kao što je to urađeno u sledećem primeru.

```
SELECT song_id, MATCH(title,lyrics) AGAINST('Sunshine') as relevance
FROM songs
WHERE MATCH(title,lyrics) AGAINST('Sunshine');
```

Primer 3: prikazivanje izračunatog koeficijenta relevantnosti rezultata

Relevantnost rezultata je veća što je ispunjenost uslova iz kriterijuma veća. Sistem *MariaDB* će tražiti rezultate za svaki kriterijum prosleđen kao parametar funkcije *AGAINST*, pa će tako oni redovi koji imaju najviše ispunjenih kriterijuma imati najveću relevantnost, dok će oni koji ispunjavaju kriterijume samo delimično imati manju relevantnost. Na primer, ukoliko navedemo još jedan parametar u *AGAINST* funkciji sa vrednošću "*Walking*" red čiji naziv (title) "*Walking on Sunshine*" bi bio značajniji od reda čiji je naziv "*Ain't No Sunshine*" ukoliko kao podatke imamo "*Walking on sunshine*", "*Ain't No Sunshine*", "*Let the Sunshine in*" i "*Everybody Loves the Sunshine*".

Operatorski režim

Ovaj režim doprinosi fleksibilnosti i većoj izražajnosti upita. Kako bismo dodali još neke uslove kriterijumu pretrage možemo koristiti neki od operatora iz liste:

Operator	Značenje
+	reč je obavezna u redu rezultata
-	reč se ne sme pojavljivati u redu rezultata
<	reč koja sledi ovaj operator ima manji faktor relevantnosti od drugih reči
>	reč koja sledi ovaj operator ima veći faktor relevantnosti od drugih reči
()	se koristi da bi grupisala reči u podizraze
~	označava da reč koja ga sledi ima negativan faktor relevantnosti. On ne isključuje red koji sadrži kriterijum, niti doprinosi pozitivno ukupnoj relevantnosti reda
*	ukazuje na to da se nula ili više karaktera može naći na kraju reči
""	se koristi kada želimo da pronađemo redove koji sadrže frazu navedenu između navodnika

Tabela 1: Pregled dozovljenih operatora u operatorskom režimu rada

Da bi se izvršio upit u operatorskom režimu potrebno je nakon definisanih parametera u *AGAINST* funkciji navesti "*IN BOOLEAN MODE*". Treba imati na umu da u ovom režimu, podaci iz rezultata nisu sortirani po relevantnosti. Na primer, ako koristimo istu tabelu i podatke iz prethodnog režima, upit kojim želimo da vidimo sve pesme koje sadrže reč "*Sunshine*", ali ne i reč "*Everybody*" izgleda kao u navedenom kodu:

```
SELECT * FROM songs
WHERE MATCH(title,lyrics)
AGAINST('+Sunshine -Everybody IN BOOLEAN MODE);
```

Primer 4: Postavljanje upita u opertorskom režimu

Ovakav upit vratio bi: "*Walking on sunshine*", "*Ain't No Sunshine*", "*Let the Sunshine in*".

Prošireni režim

Prošireni režim je režim gde se prilikom korišćenja izvršavaju dva upita. Prvi upit koji se izvršava je upit sa zadatim parametrima u režimu prirodnog jezika. Kada je dobijen rezultat tog upita, postavlja se novi upit koji sadrži prosleđene parametre i parametre za koje je zaključeno da su relevantni na osnovu analize rezultata prvog upita. Dodatni parametri predstavljaju one reči koje se najčešće ponavljaju u rezultatu prethodnog upita. Ovaj režim može biti koristan u slučaju kada imamo mali broj prosleđenih parametera, jer na taj način možemo dobiti više podataka u samom rezultatu. Slično prethodnom režimu, da bi se izvršio upit u proširenom režimu u *AGAINST* funkciji upita nakon navedenih parametara treba navesti *WITH EXPANSION QUERY*.

Sledeći primer je preuzet iz zvanične dokumentacije sistema *MariaDB* koja je dostupna na internetu[3].

Imamo tabelu koja ima samo jednu kolonu i sledeće podatke:

```
CREATE TABLE titles(
    title TEXT,
    FULLTEXT(title)
) ENGINE=MyISAM;

INSERT INTO titles(title) VALUES
('MySQL vs MariaDB database'),
('Oracle vs MariaDB database'),
('PostgreSQL vs MariaDB database'),
('MariaDB overview'),
('Foreign keys'),
('Primary keys'),
('Indexes'),
('Transactions'),
('Triggers');
```

Primer 5: tabela "titles" sa podacima

Postavljanjem upita koji pretražuje podatke po kriterijumu "*database*" kao što je to opisano u sledećem primeru, dobili bismo rezultate koji slede:

```
SELECT * FROM titles WHERE MATCH(title) AGAINST('database' WITH EXPANSION
QUERY);
```

```
*************************************************************************

MySQL vs MariaDB database
Oracle vs MariaDB database
PostgreSQL vs MariaDB database
MariaDB overview
```

Primer 6: Postavljanje upita u proširenom režimu

Prva tri reda su dobijena korišćenjem režima prirodnog jezika. Analizom rezultata je ustanovljeno da je "*MariaDB*" bitan parametar i dodat je u skup parametara u pretrazi što je rezultiralo dodavanjem još jednog reda u rezultat.

Ograničenja napredne pretrage teksta

Napredna pretraga teksta čiju funkcionalnost omogućuju podsistemi kao što su *InnoDB, MyISAM, Aria* imaju sledeća ograničenja:

- reči čija je dužina manja od četiri karaktera se ne dodaju u indeks
- reči čija je dužina veća od osamdeset četiri karaktera se takođe ne indeksiraju
- svaki podsistem ima listu reči koje se jako često koriste i stoga se ne indeksiraju
- reči koje se pojavljuju u više od polovine redova smatraju se irelevantnim i ne utiču na rezutlate pretrage osim u slučaju korišćenja Operatorskog režima

Napredna pretraga teksta je dobro rešenje za sve jezike osim za japanski, korejanski i kineski. Razlog je to što ovi jezici ne koriste razmak za odvajanje reči. Takođe, nije redak slučaj da u ovim jezicima reč ima manje od četiri karaktera, što je jedno od ograničenja napredne pretrage teksta. Zbog tih razloga nastala su rešenja čiji je cilj rešavanje ovog problema. Neka od rešenja su podsistemi *Mroonga* i *SpinxSE*. Ovi podsistemi nisu deo sistema MariaDB stoga da bismo mogli da ih koristimo, moramo prethodno da ih integrišemo.

Rad sa geografskim i geometrijskim podacima

Sistem *MariaDB* podržava kolone sa "prostornim tipovima". Takve kolone su podržane u podsistemima *MyISAM, InnoDB* i *Archive*. Osnovni tip za ovakve podatke se naziva *GEOMETRY*, međutim u definisanju ovakvih kolona možemo biti i specifičniji pa tako postoje sledeći tipovi:

- POINT
- LINESTRING
- POLYGON
- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

Pretraga po ovakvim podacima je omogućena “prostornim indeksima” (engl. *SPATIAL INDEX*). Sintaksa je ista kao kod običnih indeksa sa dodatkom ključne reči *SPATIAL*. Kolone nad kojim želimo da kreiramo ovakav indeks ne smeju imati vrednost *NULL*.

2.2 Relacione karakteristike sistema *MariaDB*

Transakcije

Transakcije su jedno od najbitnijih svojstava relacionih sistema za upravljanje bazama podataka. Sistem *MariaDB* ne podržava transakcije sama po sebi, ali ima omogućenu podršku preko podsistema. Podsistemi koji se mogu integrisati sa sistemom *MariaDB* i koji podržavaju transakcije su *InnoDB*, *TokuDB* i *XtraDB*. Podrška koju ćemo dobiti u smislu rada sa transakcijama i njihovih specifičnosti zavise od konkretnog podsistema koji koristimo.

Strani ključevi

Isto pravilo koje važi za transakcije važi i za strane ključeve. Ukoliko postavljeni podsistem podržava stane ključeve sistemu *MariaDB* će biti omogućen rad sa istim, inače ne.

Virtualne kolone

Virtualne kolone su se pojavile sa verzijom 5.2. Nazivaju se još i generisane ili izračunljive jer njihove vrednosti dobijamo rešavanjem nekog determinističkog izraza preko kojeg se definišu.

Razlikujemo dva tipa ovakvih kolona: perzistentni tip kod kojeg su vrednosti sačuvane u tabeli i virtualni kod kojeg se vrednost svaki put ponovo izračunava kada se vrši upit nad tabelom. Podrazumevani tip jeste virtualni.

Razlike između dva tipa virtualnih kolona su sledeće:

- Kod perzistentnog tipa vrednosti su sačuvane u bazi tako što se po unosu reda vrednost izraza preračunava i upisuje u tabelu. Čitaju se kao pravi podaci.
- Kod virtualnog tipa vrednosti nisu sačuvane u bazi već se preračunavaju svaki put kada se vrednost zahteva upitom.
- Vrednosti onih kolona koje nisu zahtevane upitom se ne preračunavaju.

Naredba koju koristimo kako bismo napravili tabelu sa virtualnim kolonama je *CREATE TABLE*. Pre samog kreiranja bitno je da odredimo da li vrednost kolone želimo da sačuvamo u tabeli ili to nije slučaj. Način na koji to možemo da uradimo je prikazan sledećim odlomkom:

```
CREATE TABLE example_table(
    value int NOT NULL,
    name varchar(25),
    valueMod10 int AS (value mod 10) VIRTUAL,
    mulitpliedValue int AS (value *2) PERSISTENT
);
```

Primer 7: Pravljenje tabele sa virtualnim kolonama

U ovom primeru kolona “valueMod10” je virtualna kolona čija će se vrednost preračunavati svaki put pri čitanju te kolone, dok je kolona “miltipliedValue” sačuvana kolona, to jest vrednosti ove kolone će biti upisane u samu tabelu pri unošenju podataka. Vrednosti virtualnih ili perzistentnih kolona se ne mogu menjati. Ukoliko želimo eksplicitno da kažemo da treba biti postavljena podrazumevana vrednost možemo koristiti *NULL* ili *DEFAULT*.

Pravila koja važe za definisanje izraza su:

- nije dozvoljeno koristiti podizraze ili bilo koje vrednosti koje se ne nalaze u trenutnom redu,
- sačuvane funkcije takođe ne mogu biti korišćene, ugrađene mogu
- ne mogu se koristiti vrednosti koje su definisane nakon tekuće virtualne kolone kako bi se preračunala vrednost za tekuću. Ovo pravilo važi zbog toga što se virtualne kolone preračunavaju onim redom kojim su definisane.

Indeksi nad virtualnim kolonama su samo delimično podržani. Samo nad perzistentnim tipom virutalne kolone se može kreirati indeks. Kod virtualnog tipa, kreiranje indeksa nije dozvoljeno.

Virtualne kolone ne mogu biti primarni ključ niti deo primarnog ključa. Perzistentne kolone mogu biti deo stranog ključa s tim da imaju oraničenje da se ne mogu okidati operacije nakon menjanja ili brisanja reda koji referiše na te vrednosti u roditeljskoj tabeli.

Kako su se virtualne kolone pojavile u ranoj verziji *MariaDB*, kroz dalji razvoj i objavljivanjem novih verzija menjala su se i ograničenja vezana za virtualne kolone. Tako u verziji 10.1 izraz nije smeo biti konstanta. Sa verzijom 10.2.1 izraz nije smeo biti duži od 252 karaktera. Takođe, nisu se mogle koristiti vrednosti već preračunatih kolona za vrednosti drugih virtualnih kolona, isto važi i za neke promenljive korisnikovog okruženja. Funkcije koje je definisao korisnik takođe ne mogu biti korišćne. U ovoj verziji uslov da se mora koristiti deterministički izraz sveden je na perzistentan tip virutalnih kolona. Od verzije 10.2.3. perzistentni tip virtualnih kolona može biti indeksiran.

Virtualne kolone nemaju ograničenja za tip podataka nad kojim mogu biti korišćene, ali postoji ograničenje da se virtualne kolone mogu koristiti samo uz podsistem koji ih podržava i koji je integrisan u sistem *MariaDB*. Neki od podistema koji podržavaju virtualne kolone su *InnoDB*, *Aria*, *MyISAM*, *CONNECT* koji su opisani u prethodnom poglavlju.

2.3 Nerelacione karakteristike sistema *MariaDB*

Nerelacione karakteristike sistema *MariaDB* se ogledaju u dinamičkim kolonama i mogućnostima koje pružaju podsistemi kao što su na primer *Cassandra* i *CONNECT*.

Dinamičke kolone

Dinamičke kolone su odlika sistema *MariaDB*. Predstavljaju rešenje koje možemo koristiti pri radu sa nehomogenim podacima, pa samim tim i vezu između relacionog i nerelacionog modela.

Dinamičke kolone su sastavni deo sistema *MariaDB* od verzije 5.3. Predstavljaju način da se skup različitih kolona smesti u okviru jedne kolone. U samoj bazi predstavlja se kao podatak tipa *TINYBLOB*, *BLOB*, *MEDIUMBLOB*, *LONGBLOB*, a da bi se radilo sa njima koristi se definisani skup funkcija. Dinamičke kolone se najčešće koriste kada treba smestiti različite atribute koji nisu unapred poznati ili kada podaci imaju različita svojstva.

Sam rad sa dinamičkim kolonama nije kompleksan. Za pravljenje tabele koja sadrži dinamičku kolonu koristimo naredbu *CREATE TABLE* i postavljamo tip dinamičke kolone na *BLOB* kao u sledećem primeru. Ovim SQL upitom smo kolonu “tags” postavili kao dinamičku kolonu.

```
CREATE TABLE `messages` (
    `id` INT NOT NULL AUTO_INCREMENT,
    `sender` VARCHAR(20) NOT NULL,
    `recipient` VARCHAR(20) NOT NULL,
    `content` VARCHAR(200) NOT NULL,
    `time_posted` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
    ON UPDATE CURRENT_TIMESTAMP,
    `tags` BLOB NULL,
    PRIMARY KEY (`id`)
) COLLATE='utf8_general_ci'
ENGINE=InnoDB
AUTO_INCREMENT=73;
```

Primer 8: Pravljenje dinamičke kolone "tags"

Dodavanje novog reda se izvodi naredbom *INSERT*, pri čemu se vrednosti dinamičkih kolona postavljaju koristeći funkciju *COLUMN_CREATE* na slededći način:

```
INSERT INTO `messages`
(`sender`, `recipient`, `content`, `time_posted`, `tags`) VALUES
('kristina', 'kristina', 'A new message posted', '2017-03-23 15:39:14',
column_create('messageNum', 1));
```

Primer 9: Dodavanje reda u tabelu sa dinamičkom kolonom

Funkcija *COLUMN_CREATE* kao parametre dobija nazive kolona i njihove vrednosti. U ovom primeru to je kolona “messageNum” sa vrednošću 1.

Vrednost dinamičke kolone možemo pročitati uz pomoć funkcije *COLUMN_JSON* kojoj treba proslediti ime dinamičke kolone kao u sledećem primeru:

```
SELECT sender, recipient, content, time_posted, column_json(tags) as
tags
FROM messages;
```

Primer 10: Čitanje vrednosti iz tabele koja ima dinamičku kolonu pomoću funkcije column_json

Ono što je specifično za ovu funkciju jeste to što vraća rezultat u *JSON* formatu, što može biti jako korisno pri implementaciji softverskog rešenja.

Slično tome, za čitanje vrednosti iz dinamičke kolone možemo koristiti i *COLUMN_GET* funkciju, ali samo ako znamo naziv kolone čiju vrednost želimo da pročitamo. Ukoliko ne znamo naziv kolone čiju vrednost želimo pročitati onda možemo koristiti njen redni broj, ali moramo znati pod kojim rednim brojem je smeštena.

```
SELECT sender, recipient, content, time_posted, column_get(tags,
`messageNum` as char) as tags
FROM messages;
```

Primer 11: Čitanje reda iz tabele koja ima dinamičku kolonu pomoću funkcije column_get

Postavljanje upita koristeći naziv atributa omogućeno je od verzije 10.0.1, do tada su se koristili samo brojevi kolona, što je značilo da bi korisnik morao dobro da poznaje podatke kako bi pročitao željenu vrednost.

Ažuriranje vrednosti dinamičkih kolona se vrši korišćenjem naredbe *UPDATE* i neke od funkcija u skladu sa namerom upita. Tako za brisanje kolone iz dinamičke kolone koristimo funkciju *COLUMN_DELETE* koja kao parametre prima naziv dinamičke kolone i naziv kolone koju želimo da uklonimo iz dinamičke kolone.

```
UPDATE messages SET tags=column_delete(tags, "messageNum")
WHERE column_get(tags, 'messageNum' as char)='1';
```

Primer 12: Brisanje kolone iz dinamičke kolone sa imenom "messageNum" i vrednošću 1

Ukoliko ažuriranjem dinamičke kolone želimo istu da proširimo to možemo uraditi pomoću funckije *COLUMN_ADD* kojoj prosleđujemo ime dinamičke kolone, naziv nove kolone koju želimo da umetnemo i vrednost koju želimo upisati, kao u sledećem primeru.

```
UPDATE messages
SET tags=COLUMN_ADD(tags, 'status', 'succedded')
WHERE id=70;
```

Primer 13: Dodavanje nove kolone u dinamičku kolonu

Pored gore prikazanih funkcija: *COLUMN_CREATE*, *COLUMN_JSON*, *COLUMN_GET*, *COLUMN_ADD*, *COLUMN_DELETE*, postoje još i *COLUMN_EXISTS*, *COLUMN_LIST* i *COLUMN_CHECK* funkcije.

COLUMN_EXISTS funkciju koristimo kada želimo da proverimo da li postoji dinamička kolona sa prosleđenim imenom. Povratna vrednost funkcije je 1 ili 0.

COLUMN_CHECK funkcija je prava funkcija ukoliko želimo da validiramo sadržaj dinamičke kolone. Ukoliko je sadržaj validan povratna vrednost je 1, inače 0.

COLUMN_LIST funcija vraća sva imena kolona koje su sačuvane u dinamičkoj koloni.

COLUMN_CHECK, *COLUMN_JSON* dostupne od verzije 10.0.1. Kod svih ostalih funkcija moguće je proslediti ili ime kolone ili broj pod kojim se očekuje da bude smešten podatak.

Isto tako kada se prosleđuje vrednost može se proslediti i tip podatka, mada je ovaj tip podatka redundatan jer sistem *MariaDB* može sam da zaključi o kakvom se tipu podataka radi. Izuzetak je u slučaju *DATE* tipa, tu je neophodno proslediti tip ukoliko želimo da se taj podatak sačuva kao podatak tipa *DATE*.

Pregled funkcija za upravljanje dinamičkim kolonama:

```
COLUMN_CREATE(column_nr, value [as type], [column_nr, value [as
type]]...);
COLUMN_CREATE(column_name, value [as type], [column_name, value [as
type]]...);
COLUMN_ADD(dyncol_blob, column_nr, value [as type], [column_nr, value [as
type]]...);
COLUMN_ADD(dyncol_blob, column_name, value [as type], [column_name, value
[as type]]...);
COLUMN_GET(dyncol_blob, column_nr as type);
COLUMN_GET (dyncol_blob, column_name as type);
COLUMN_DELETE(dyncol_blob, column_nr, column_nr...);
COLUMN_DELETE(dyncol_blob, column_name, column_name...);
COLUMN_LIST(dyncol_blob);
COLUMN_CHECK(dyncol_blob);
COLUMN_EXISTS(dyncol_blob, column_nr);
COLUMN_EXISTS(dyncol_blob, column_name);
COLUMN_JSON(dyncol_blob);
```

Pregled ugrađenih funkcija za upravljanje dinamičkim kolonama

Kao što je navedeno u pregledu ugrađenih funkcija, pri korišćenju funkcija *COLUMN_CREATE* ili *COLUMN_ADD*, *COLUMN_GET*, *COLUMN_DELETE*, *COLUMN_EXISTS* za označavanje kolone koja pripada dinamičkoj koloni možemo koristiti njeno ime, ili njen indeks u nizu kolona dinamičke kolone.

Sistem *MariaDB* podržava i opciju ugnježđenja dinamičkih kolona uz ograničenja. Ograničenje se ogleda u tome da funkcija *COLUMN_JSON* može dekodirati podatke samo do 10. nivoa ugnježđenja. Ugnježđenje kolona se izvršava tako što se kao vrednost jedne kolone sadržane u dinamičkoj koloni postavi dinamička kolonu. Ovaj postupak je opisan sledećim upitom:

```
SELECT
COLUMN_JSON(COLUMN_GET(COLUMN_CREATE('test1',COLUMN_CREATE
('key1','value1','key2','value2','key3','value3')),
'test1' as BINARY));
```

Primer 14: Ugnježđenje dinamičke kolone u dinamičku kolonu

Treba imati u vidu da kao poslednji tip treba postaviti *BINARY* kako ne bismo naišli na probleme sa konverzijom karaktera, kao i da veliki broj ugnježđenja vodi do loših performansi.

Dinamičke kolone imaju dva ograničenja. Prvo ograničenje je to što maksimalni broj kolona ne može biti veći od 65536, a drugo je to što maksimalna dužina dinamičke kolone ne sme biti veća od 1GB.

Građenje indeksa nad dinamičkom kolonom nije moguće. Međutim postoji način na koji se može izgraditi indeks nad kolonom dinamičke kolone i on uključuje korišćenje perzistentnog tipa virtualne kolone. Potrebno je za tip kolone postaviti perzistentni tip, a za vrednost izraza koristiti funkciju *COLUMN_GET* kako bi se pročitala vrednost kolone. Na taj način bi vrednost kolone bila sačuvana pa samim tim bismo mogli napraviti i indeks nad njom. Sledeći kod opisuje način na koji se može napratiti indeks:

```
ALTER TABLE messages
ADD COLUMN location CHAR(10) AS (COLUMN_GET(messages, 'location' AS
CHAR(10))) PERSISTENT,
```

```
ADD INDEX index_location (location);
```

Primer 15: Pravljenje indeksa nad kolonom iz dinamičke kolone

Podsistem *Cassandra*

O podsistemu *Cassandra* je već govoreno u poglavlju o osnovnim konceptima sistema *MariaDB*, ali kako je njegov cilj da omogući rad sa podacima kojima upravlja sistem *Cassandra,* koja je jedna od najpopularnijih nerelacionih baza podataka, ovaj podsistem zavređuje više pažnje.

Podsistem *Cassandra* je integrisan u sistem *MariaDB,* ali da bi se koristio neophodno je prvo uraditi aktivaciju-instaliranje podsistema. Način na koji je moguće instalirati podsistem je objašnjen u 2. poglavlju.

Nerelaciona baza podataka *Cassandra* kao upitni jezik koristi *CQL*. Ovaj podsistem omogućava da se podacima iz *Cassandra* klastera pristupa koristeći *SQL*. Tu se postavlja pitanje da li onda možemo uvek koristiti *SQL* za rad sa podacima iz *Cassandra* klastera i odgovor je ne. Odgovor je ne iz prostog razloga što ovaj podsistem nije pogodan za izvršavanje upita koji vrše analitičku obradu podataka vellikog obima.

Da bi se podaci iz sistema *Cassandra* preslikali u tabelu u sistemu *MariaDB* potrebno je napraviti tabelu sa navođenjem podsistema *Casandra*. Nakon toga prostor ključeva bi trebalo postaviti na prostor kljčeva definisan u Cassandri i pomoću *COLUMN_FAMILY* bi trebalo specificirati ime familije kolona sa kojom želimo da radimo. Ime tabele može biti proizvoljno.

Primarni ključ u toj tabeli mora odgovarati ključu reda iz sistema *Cassandra*. Ime primarnog ključa mora da se podudara sa aliasom koji je postavljen u *Cassandri*, ili ukoliko alias nije definisan ime ove kolone mora biti postavljen na "*rowkey*". Takođe tipovi se moraju uskladiti. Za sada nisu podržani složeni primani ključevi, mada je podrška planirana u budućnosti.

Statičke kolone iz *Cassandra* klastera se mapiraju u kolone u tabeli sa odgovarajućim tipom podataka, dok se dinamičke kolone preslikavaju u dinamičke kolone sistema *MariaDB*. Razlika u odnosu na standardno kreiranje dinamičke kolone je u tome što se pored imena kolone i tipa *BLOB* mora navesti i atribut *DYNAMIC_STORAGE_SYSTEM=yes*. Imena svih kolona u tabeli moraju odgovarati imenima kolona iz *Cassandra* klastera.

Tipovi iz ova dva sistema se ne mogu mapirati 1-1 i pored toga mogu imati različita ograničenja, tako da pri mapiranju tipova treba to imati u vidu.

Operacije pravljenja, ažuriranja, čitanja i brisanja se mapiraju u odgovarajuće operacije sistema *Cassandra*.

Sve ovo zajedno nam omogućava upravljanje podacima kojima rukovodi sistem *Cassandra*.

3 Aplikacija za razmenu poruka i dokumenata

Kako bi se stekao još potpuniji utisak o korišćenju sistema *MariaDB* razvijena je aplikacija za razmenu poruka i dokumenata. Prilikom izrade aplikacije korišćena su i relaciona i nerelaciona svojstva sistema. Aplikacija je razvijana sa namerom da se stekne uvid u to kolika je kompleksnost korišćenja ovakvog sistema, koliko ovakav sistem može da olakša ili oteža razvoj aplikacije.

Zahtevi pri izradi aplikacije su sledeći:

- napraviti bazu podataka sa:
- (relacionim) osnovnim podacima o korisnicima
- porukama koje korisnici postavljaju na svoju "stranu" ili šalju ciljanim korisnicima i grupama, pri čemu su poruke tagovane atributima koje sami korisnici prave po želji.
- na primer, može da se napiše poruka koja ima obavezan tekst, datum, vreme, a onda korisnik može da doda i bilo koji drugi podatak, na primer: "predmet = dobp", bez ikakvog ograničenja.
- uz poruke mogu da se postavljaju dokumenti, koji imaju slične odlike kao poruke, ali moraju da podržavaju čuvanje verzija
- i poruke i dokumenti moraju da se reše nerelaciono
- aplikacija ne mora da ima deo za staranje o korisnicima, važno je samo da mogu da se
  - postave poruke i dokumenti
  - pregledaju poruke i dokumenti
  - pretražuju poruke i dokumenti

3.1 Arhitektura aplikacije

Prilikom razvijanja aplikacije korišćeno je nekoliko softverskih rešenja.

U izradi aplikacije korišena je verzija *MariaDB* 10.1. Kao klijent za bazu podataka korišćen je *HeidiSQL*, dok je za razvojno okruženje korišćena *IDEA Intellj* verzija 16.2. Rešenje je razvijeno tako da se sastoji iz serverskog i klijentskog dela aplikacije. Serverski deo aplikacije je razvijan u jeziku *Java*, kao *Maven* projekat koji koristi *REST* pozive za komunikaciju sa klijentskim delom aplikacije. Server koji je korišćen je *Apache Tomcat*. Za razvijanje klijentskog dela korišćen je *Javascript* okvir *Angular* 1.5.2.

Serverski deo aplikacije se sastoji od tri sloja.

- Prvi sloj, sloj kontrolera, služi za konfigurisanje *REST* pristupnih tačaka. U tom sloju prihavatamo zahteve klijentske aplikacije i prosleđujemo ih na dalju obradu posle koje vraćamo rezultata klijentskoj aplikaciji.
- Drugi sloj, sloj servisa, obrđuje zahtev koji smo dobili od prvog sloja, komunicira sa bazom podataka tako što poziva treći sloj, obrađuje rezultat koji smo dobili od baze i vraća odgovor sloju kontrolera.
- Treći sloj, sloj pristupa podacima iz baze podataka, komunicira sa bazom, prosleđuje joj upite i naredbe i vraća rezultat servisnom sloju.

Klijentski deo aplikacije je zaslužan za ceo prezentacioni sloj. U klijentskoj apikaciji takođe imamo tri dela.

- Prvi sloj je sloj prezentacije podataka. Tu ubrajamo sve *HTML*, *CSS* fajlove kao i *Angular* direktive.
- Takođe, drugi sloj je sloj kontrolera koji je spona između prikaza podataka i podataka dobijenih od servera. Pored toga ovaj sloj služi i za validaciju podataka.
- I treći sloj je sloj serivisa koji šalje zahteve aplikaciji na serveru, prosleđuje odgovor sloju kontrolera koji ih dalje prikazuje uz pomoć prezentacionog sloja.

3.2 Baza podataka

Prilikom razvoja aplikacije korišćena su i relaciona i nerelaciona svojstva sistema *MariaDB*. Podrazumevani podsistem u verziji sistema *MariaDB* korišćenoj za razvoj ove aplikacije je *InnoDB* daje podršku za relaciona svojsva kao što je referencijalni integritet, ali takođe i nerelaciona svojstva u vidu dinamičkih kolona. Koristeći se relacionim svojstvima napravljeni su entiteti Korisnika, Grupa i njihovih međusobnih odnosa. Nerelaciona svojstva su primenjena pri implementiranju eniteta Poruka i Dokumenta. Kod entiteta Poruka, tagovi kojima korisnici označavaju poruke predstavljeni su dinamičkom kolonom jer su nam tagovi koji će se koristiti nepoznati. Kod entiteta Dokument, u dinamičku kolonu su smeštene sve verzije sadržaja dokumenta koje imamo. Sledi shema baze podataka.

```
-- --------------------------------------------------------
-- Host:                 127.0.0.1
-- Server version:       10.1.25-MariaDB - mariadb.org binary distribution
-- Server OS:            Win64
-- HeidiSQL Verzija:     9.4.0.5125
-- --------------------------------------------------------

CREATE DATABASE IF NOT EXISTS `collab`;
USE `collab`;

CREATE TABLE IF NOT EXISTS `club` (
  `id` int(11) NOT NULL,
  `name` varchar(30) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

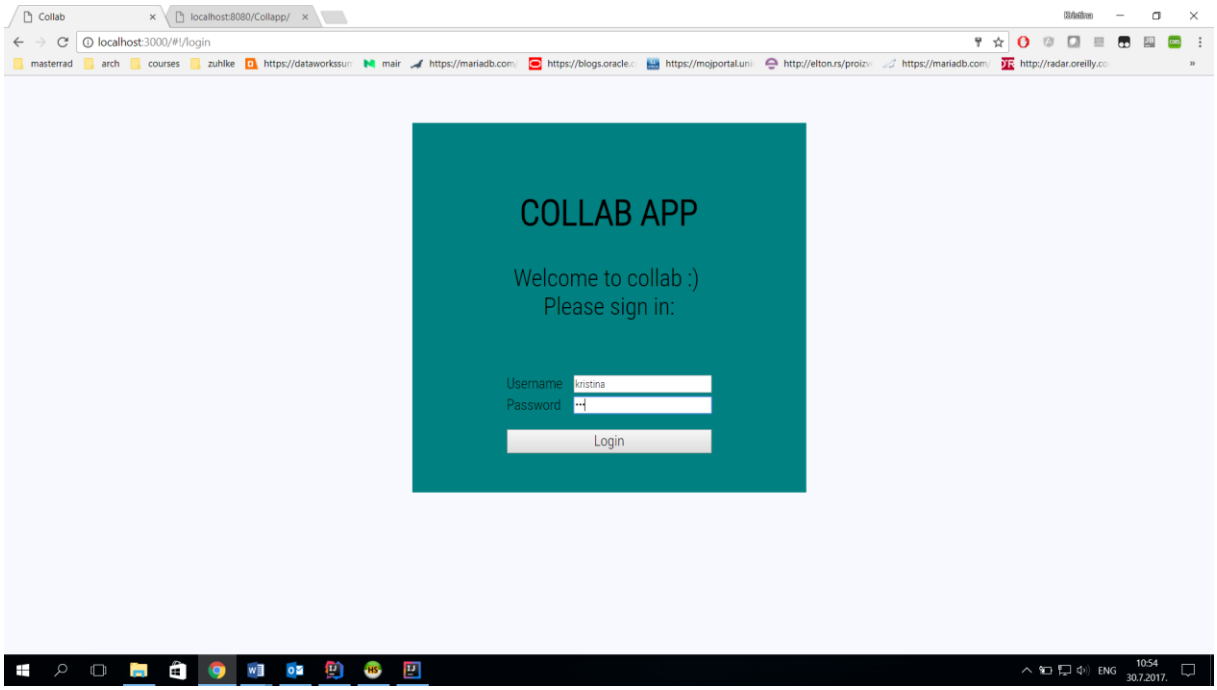
```
CREATE TABLE IF NOT EXISTS `documents` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `sender` varchar(20) NOT NULL,
  `recipient` varchar(20) NOT NULL,
  `last_modification_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `document` blob,
  `title` varchar(50) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_document_recipient` (`recipient`),
  KEY `fk_document_sender` (`sender`),
  CONSTRAINT `fk_document_sender` FOREIGN KEY (`sender`) REFERENCES `user`
(`username`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `member_of_club` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(100) NOT NULL,
  `club_id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_user_club` (`username`),
  KEY `fk_club_member` (`club_id`),
  CONSTRAINT `fk_club_member` FOREIGN KEY (`club_id`) REFERENCES `club`
(`id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fk_user_club` FOREIGN KEY (`username`) REFERENCES `user`
(`username`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `messages` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `sender` varchar(20) NOT NULL,
  `recipient` varchar(20) NOT NULL,
  `content` varchar(200) NOT NULL,
  `time_posted` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `tags` blob,
  PRIMARY KEY (`id`),
  KEY `fk_message_recipient` (`recipient`),
  KEY `fk_message_sender` (`sender`),
  FULLTEXT KEY `content` (`content`),
  CONSTRAINT `fk_message_sender` FOREIGN KEY (`sender`) REFERENCES `user`
(`username`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=102 DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `user` (
  `username` varchar(100) NOT NULL,
  `password` varchar(10) NOT NULL,
  `firstname` varchar(50) NOT NULL,
  `lastname` varchar(50) NOT NULL,
  `email` varchar(50) NOT NULL,
  `phoneNumber` varchar(50) NOT NULL,
  `address` varchar(50) NOT NULL,
  PRIMARY KEY (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Shema 1: Prikaz sheme korišćene u aplikaciji

3.3 Implementacija baze podataka na sistemu MariaDB

U skladu sa zahtevima dobijenim za razvoj aplikacije implementirano je sledeće:

Po pristupanju stranici dolazimo do *Login* strane. Korisnici aplikacije su već dodati u bazu podataka i kako ne postoji zahtev za posebnim registrovanjem korisnika ili za promenu šifre taj deo je zanemaren.



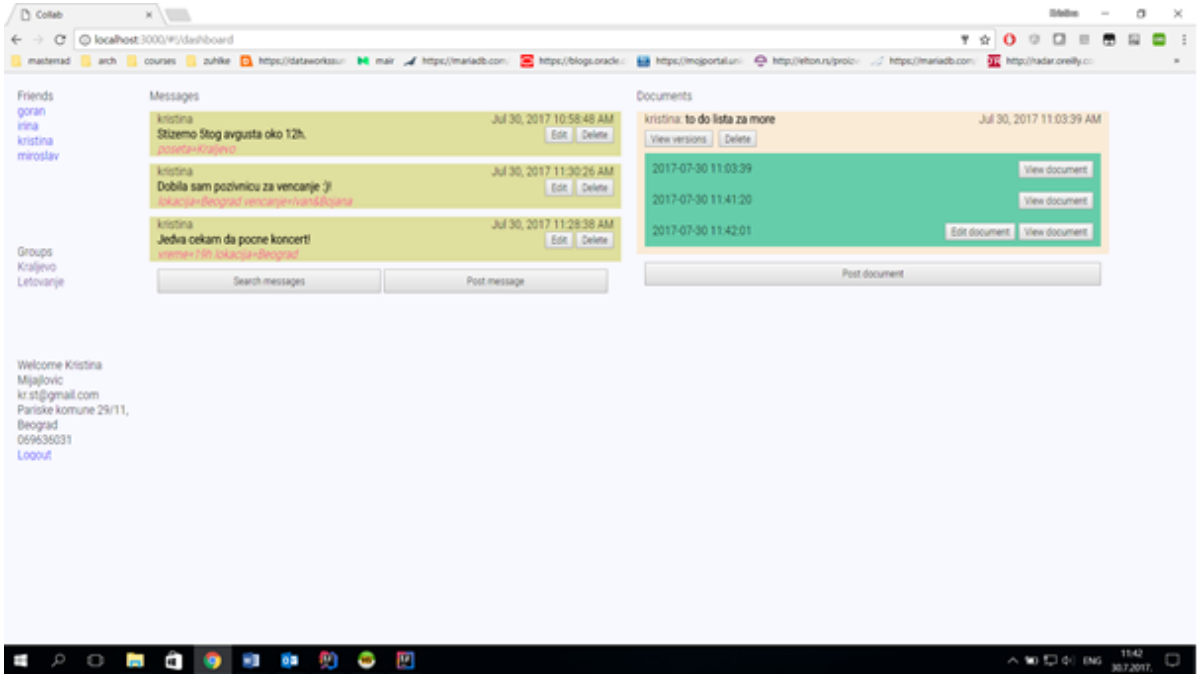
Slika 1: Stranica za logovanje korisnika

Po prijavljivanju korisnika, korisnik vidi stranicu na kojoj su prikazane sve informacije koje aplikacija nudi. Dakle, sa leve strane stranice prikazani su svi korisnici koji postoje u bazi, kojima korisnik može poslati poruku ili dokument. Ispod spiska korisnika nalazi se lista grupa u kojima je korisnik član. Ova relacija je modelovana u bazi i ne postoji deo aplikacije koji će prikazati sve moguće grupe i omogućiti korisniku da izabere neku u koju bi se učlanio ili iz koje bi se odjavio. Svi nazivi korisnika i grupa predstavljaju linkove ka njihovim stranicama.

Kada korisnik želi da pošalje nekom drugom korisniku ili grupi poruku ili dokument korisnik prvo mora izabrati korisnika ili grupu klikom na link i potom pristupiti pisanju poruke ili dokumenta.

Ispod spiska grupa prikazane su informacije sa podacima o profilu korisnika. U tom delu možemo videti informacije o adresi korisnika, broju telefona, email adresi, ali ne i lozinku. Pored informacija, imamo i link preko kojeg se možemo odjaviti iz aplikacije i vratiti na stranu za prijavljivanje.

U centralnom delu strane korisnik može videti svoju listu poruka i dokumenata.

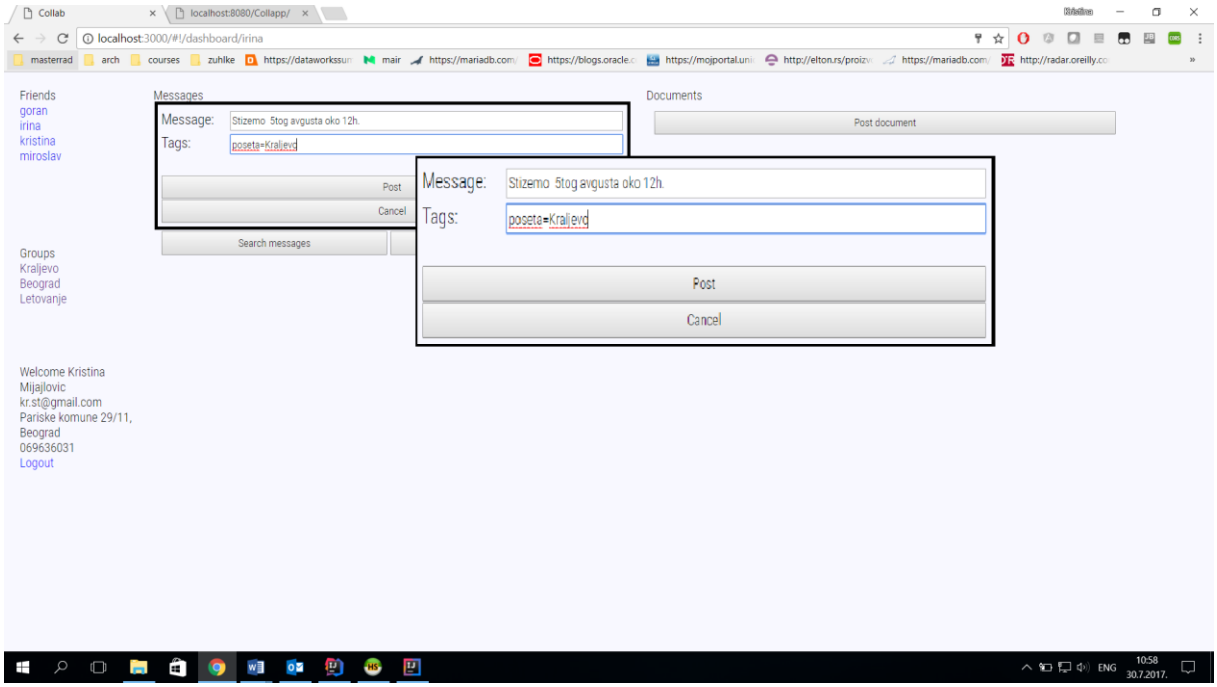


Slika 2: Osnovna strana korisnika

Kod svake poruke možemo videti ko je poslao poruku na stranicu, vreme u koje je poruka poslata, kao i tagove kojima je obeležena. Vlasnik stranice može obrisati ili izmeniti svaku poruku na svojoj stranici, nakon čega će sve poruke biti učitane ponovo. Iako je potreba za slučajem upotrebe gde vlasnik stranice menja poruku koju mu je poslao neko drugi upitna, ovde je to omogućeno kako bi se obuhvatio slučaj ažuriranja poruke i sadržaja dinamičke kolone. Prilikom ažuriranja vreme postavljanja će biti promenjeno na trenutno vreme. Kako su tagovi nepoznati u trenutku dizajniranja tabele oni su predstavljeni dinamičkom kolonom.

Pri postavljanju nove poruke dovoljno je samo uneti sadržaj poruke i tagove u formatu „tag1=vrednost1 tag2=vrednost2 ...“, pošiljilac, vreme i primalac će biti poslati u pozadini aplikacije. Dugme za slanje poruke će biti onemogućeno sve dok poruka nije unešena u pravom formatu, što znači dok nemamo postavljen sadžaj i tagove u gore navedenom formatu. U svakom trenutku je moguće odustati od poruke.

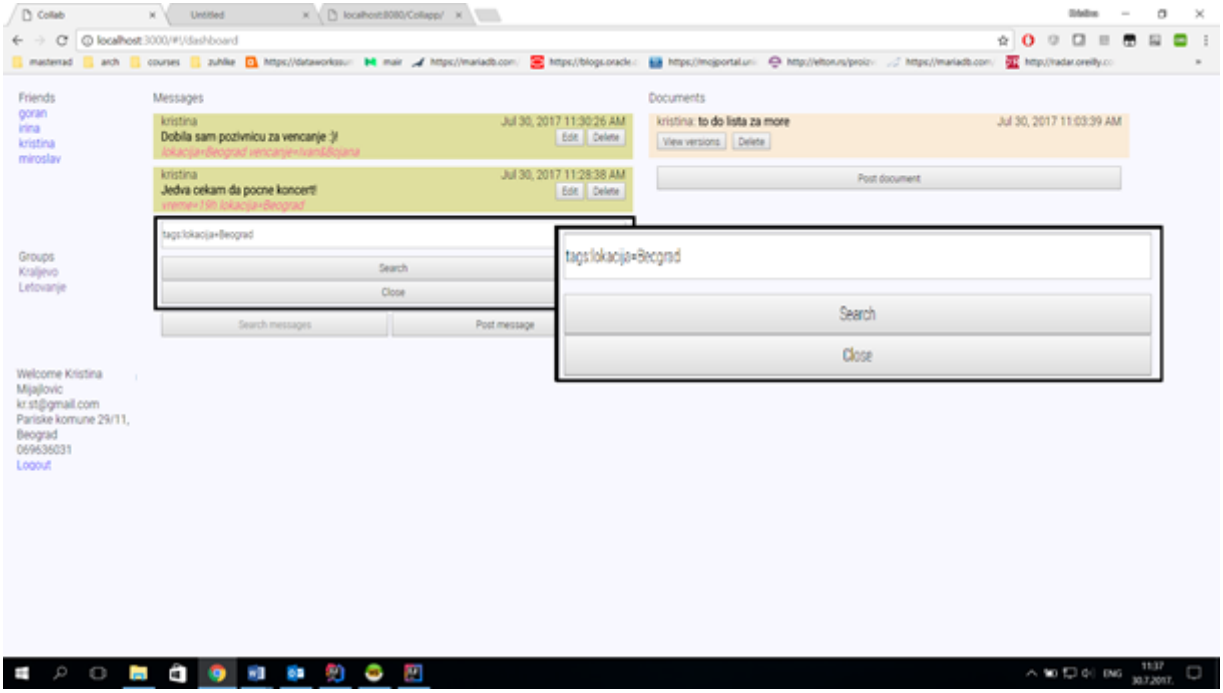
Postavljanje poruke preko implementirane aplikacije izgleda kao na priloženoj slici.



Slika 3: Postavljanje poruke na stranicu drugog korisnika

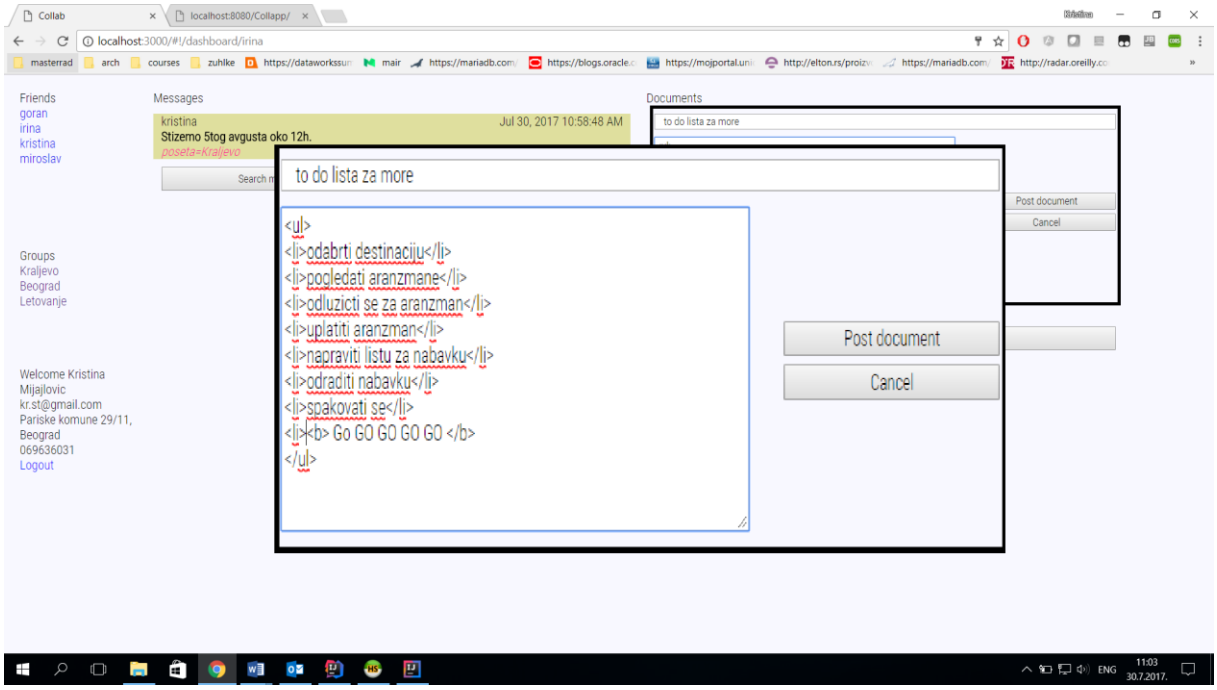
Pored postavljanja, menjanja i brisanja poruka omogućena je i pretraga poruka po 3 kriterijuma. Prvi kriterijum koji možemo postaviti je ime pošiljaoca, drugi kriterijum je sadržaj teksta poruke, i treći kriterijum po kojem možemo izvršiti pretragu je vrednost taga kojom je poruka tagovana.

U polje pretrage potrebno je upisati kriterijum i vrednost: “sender:Kristina” ili “content:message” ili “tags:status=succedded” po kojoj želimo da izvršimo pretragu. Ovde je bitno napomenuti da je prilikom upisa u polje pretrage očekivan unos kakav je naveden iznad. U slučaju drugog formata upit neće biti obrađen jer će parsiranje niske dati grešku.



Slika 4: Pretraživanje poruka

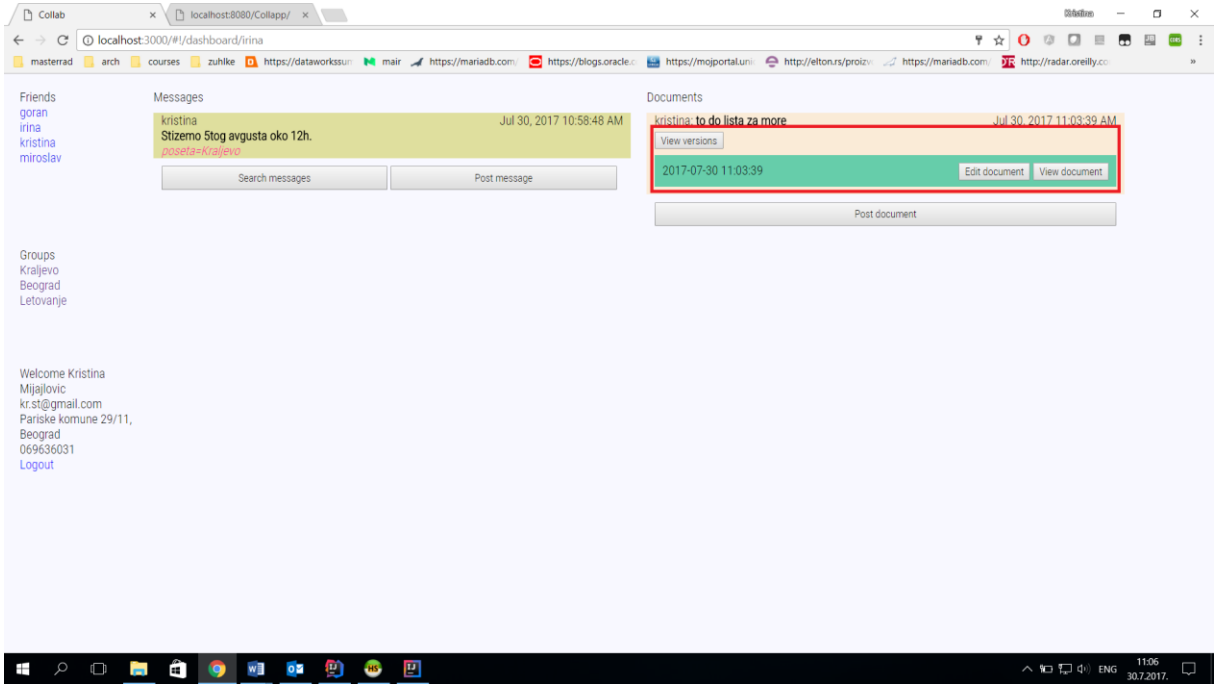
Kao i kod poruka moguće je postaviti, pregledati, menjati i brisati dokumente. Izgled kod dokumenata se razlikuje od izgleda poruka. Naime, tu ne postoji mogućnost postavljanja tagova.



Slika 5: Postavljanje dokumenta na stranici drugog korisnika

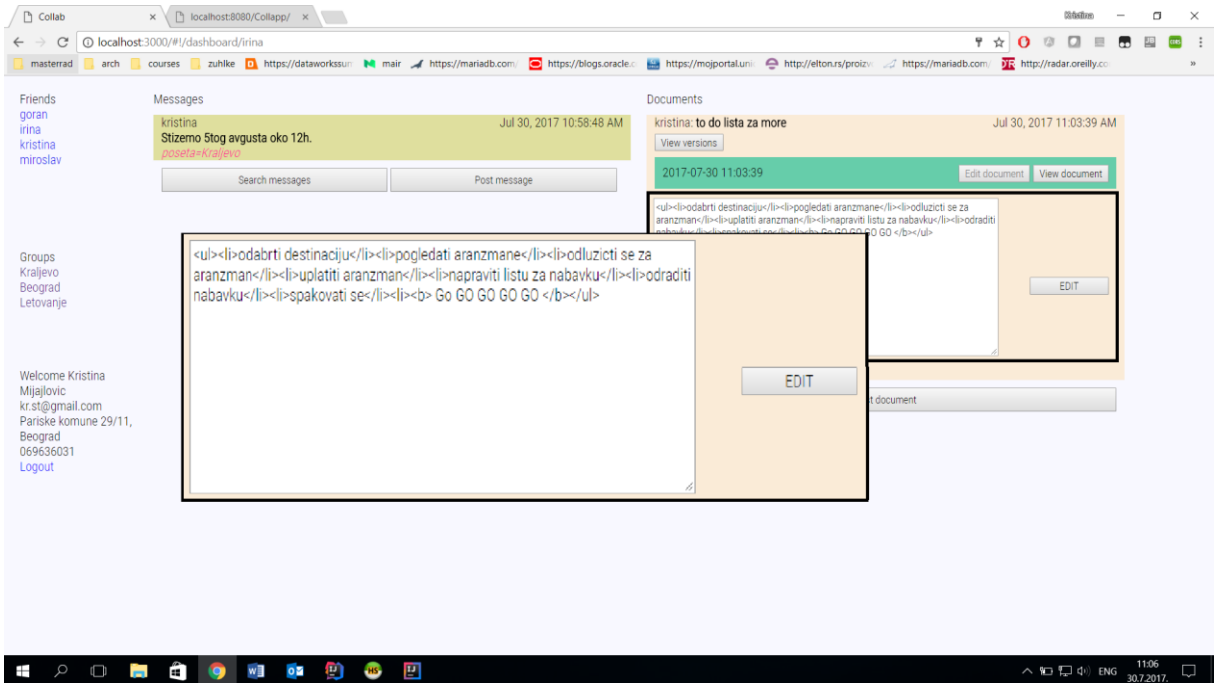
Pod dokumentom podrazumevamo *HTML* kod. Prilikom pregledanja dokumenta otvara se novi prozor i učitava *HTML* dokument.

Kod svakog dokumenta dostupne su nam sve njegove prethodne verzije. Svaku od verzija možemo pregledati klikom na dugme „*View document*“ dostupnim pored svake verzije dokumenta. Verzije dokumenata su sortirane po vremenu kada su nastale.



Slika 6: Pregled verzija dokumenata

Ažuriranje je moguće odraditi samo na poslednjoj verziji dokumenta. Klikom na dugme „*Edit*“ moguće je promeniti poslednju verziju dokumenta i sačuvati je u bazi.

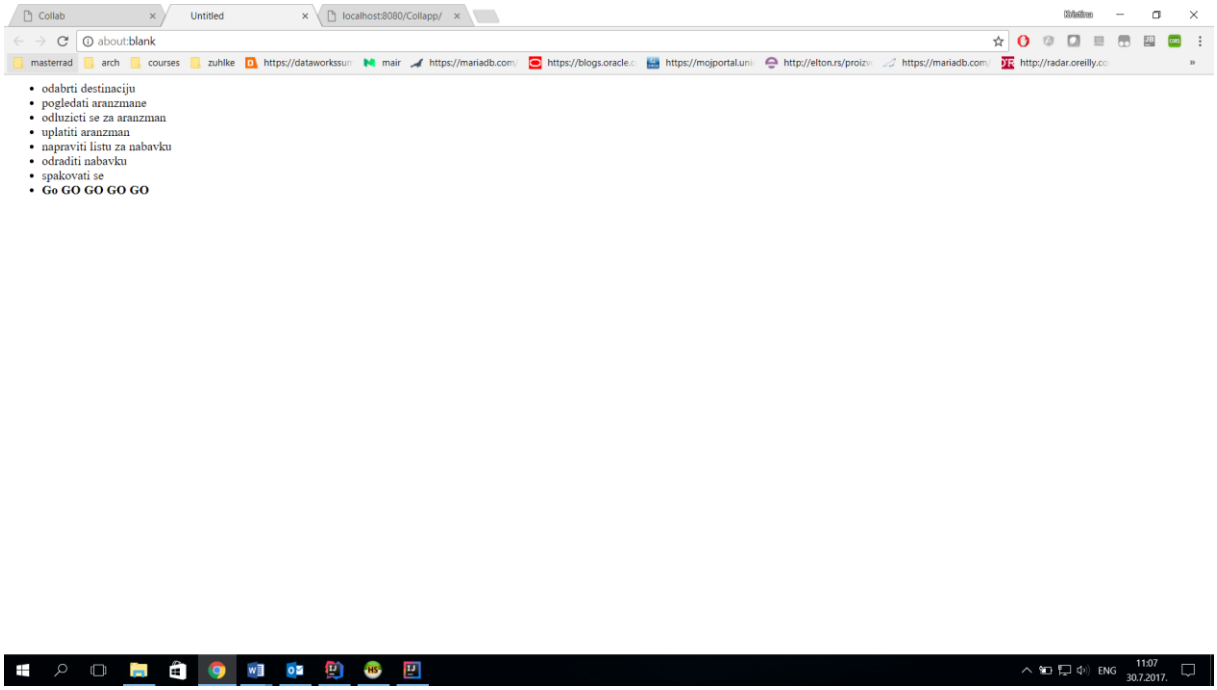


Slika 7: Menjanje sadžaja dokumenta

Dokumenti su u bazi predstavljeni takođe dinamičkom kolonom. Pri svakom ažuriranju sadržaja dokumenta dodaje se još jedna kolona u dinamičku kolonu čiji je ključ datum i vreme postavljanja, a vrednost sadržaj dokumenta. Na taj način je podržano verzionisanje dokumenta.

Klikom na „*Delete*“ dokument se uklanja iz baze.

Svaka verzija dokumenta se može pregledati klikom na dugme “*View document*”, pri čemu je sadržaj dokumenta prikazan u novom prozoru.



Slika 8: Pregled izabrane verzije dokumenta

4 Diskusija

Specifičnost i fleksibilnost sistema *MariaDB* ostavljaju mogućnost da se problemi rešavaju na različite načine. U narednim odeljcima su razmotrena neka od alternativnih rešenja.

4.1 Relaciono predstavljanje dinamičkih kolona

Jedno od najjednostavnijih rešenja za predstavljanje dinamičkih kolona relaciono jeste da se za svaki atribut doda nova kolona u postojećoj tabeli. Ovaj pristup bi podrazumevao prvo proveru svih kolona, gde bismo utvrdili da li atribut već postoji kao kolona. Ako postoji upisali bismo vrednost, a ako ne postoji morali bismo da ažuriramo tabelu tako da joj dodamo novi atribut u vidu kolone, postavimo podrazumevanu vrednost na *NULL*, a samo redu čije smo upisivanje radili upišemo vrednost tog atributa. Ovakvo rešenje je jako loše jer bi postojalo jako puno kolona sa *NULL* vrednostima, a samo po nekolicinu redova sa vrednostima atributa. Drugi problem koji imamo ovde jeste preveliki broj kolona, što je jako teško za održavanje i otežava čitljivost. Na primer, imamo red koji ima poruku i pošiljaoca i primaoca, ključ i onda ima 5 *NULL* vrednosti za kolone npr. cena, veličina, obim, površina, težina, i vrednost za kolonu prevoz. Isto tako, da bismo mogli da mapiramo podatke u našoj aplikaciji moramo znati sve kolone koje su nam u opticaju, a ako je njihov broj veliki onda je to već neodrživo.

Drugo rešenje jeste da napravimo jednu relaciju sa podacima koje znamo da ćemo imati u tabeli. U našem slučaju to bi bili identifikator, pošiljalac, primalac, sadržaj poruke. Zatim bi bilo potrebno napraviti i drugu tabelu koja bi služila za čuvanje svih atributa. Ona bi trebalo da ima identifikator reda, strani ključ koji bi bio primarni ključ prve tabele i dve kolone. Prva bi predstavljala atribut, a druga vrednost. Tako bismo recimo za poruku: koja ima dva taga npr boja=crvena i obim=32 imali u drugoj tabeli dva reda.

Id	Pošiljalac	Primalac	Tekst
2	Kristina	Ana	Ovu loptu treba da kupimo.

Tabela 2: Relacija Poruka koja sadrži atribute koji su obavezni

Id	Id_poruke	Atribut	Vrednost
0	2	Boja	Crvena
1	2	Precnik	32

Tabela 3: Relacija Atributi koja sadrži proizvoljne atribute

Ovakvo rešenje bi bilo lakše za održavanje od prvog jer podržava neograničeno dodavanje novih atributa bez menjanja strukture podataka, samim tim bi bilo i čitljivije i ne bi bilo problema sa mapiranjem podataka jer raspored i broj kolona već znamo. Međutim, više ne možemo dobiti sve tagove i poruku jednostavnim *SELECT* upitom već bi za dobijanje ovakvih informacija trebalo spojiti tabele i pored toga izmapirati sve redove iz tabele sa atributima u

odgovarajuću strukturu podataka. Dakle, za svaki upis u bazu morali bismo da dekomponujemo poruku i za svako čitanje da je sklopimo ponovo.

4.2 Simbioza relacionih i nerelacionih baza podataka

Postoje aplikacije koje koriste i relacione i nerelacione modele tako što deo aplikacije za koji im trebaju relaciona svojstva razvijaju i koriste neku od baza koji podrazumeva relacioni model, a za podatke koji se teško mogu predstaviti relacionim modelom, koriste neku od *NoSQL* baza.

Primer ovakvog modela je aplikacija koja ima svoj sistem plaćanja npr (nema neki integrisani sistem kao što je *paypal* ili slično) i pritom omogućava pisanje poruka ili/i postavljanje dokumenata i slično.

Ovo nije redak slučaj. Relaciona svojstva kao što su transakcije su ovde jako bitna, a pošto ne znamo strukturu podatatka i shemu pri postavljanju poruka i dokumenta trebaju nam svojstva i nerelacionih modela podataka.

MariaDB je jedan od predstavnika *NoSQL* pokreta, međutim, ona ne zanemaruje potrebe za relacionim svojstvima, već podržava i jedan i drugi model.

4.3 Rad sa dinamičkim kolonama

Dinamičke kolone su prilikom razvoja ove aplikacije bile izuzetno korisne, s obzirom da nismo znali za kojim i kakvim tagovima će postojati potreba. One su bile neophodno sredstvo u radu sa promenljivom strukturom podataka. Ono što može predstavljati problem prilikom korišćenja dinamičkih kolona prilikom razvoja aplikacije jeste to što prilikom pregledanja podataka preko klijenta za bazu (u ovom slučaju *HeidiSQL*), ne možemo imati uvid prostim pregledanjem podataka u to šta je smešteno u dinamičku kolonu jer je ona binarnog tipa. Potrebno je izvršiti upit koji bi transformisao podatke iz *BLOB* kolone da bi sačuvana vrednost mogla biti pročitana.

Još jedan problem pri radu sa dinamičkim kolonama jeste to što ukoliko se iz aplikacije prosledi upit ka bazi koji ima loše formatiranu nisku karktera ili ulaz za dinamičku kolonu, nećemo dobiti grešku. Greške nećemo biti ni svesni sve dok ne budemo imali potrebu da pročitamo vrednosti tog reda u našoj aplikaciji pri čemu ćemo dobiti grešku, jer čitanje tog binarnog niza neće dati valjane rezultate.

S obzirom na fleksibilnost koju dinamičke kolone pružaju mislim da su ovi problem zanemarljivi.

4.4 O specifičnostima sistema *MariaDB*

Sistem *MariaDB* sa svojim osnovnim svojstvima pruža veliku fleksibilnost. Sama činjenica da podrazumevani podsistem daje podršku za relaciona svojstva u vidu transakcija i referencijalnog integriteta i u isto vreme daje podršku za nerelaciona svojstva kao što su dinamičke strukture podataka daje prednost u odnosu na druge sisteme za upravljanje bazama podataka koji podržavaju samo relacione ili nerelacione koncepte.

Dinamičkim kolonama, sistem *MariaDB* rešava problem strogo definisane strukture, koja je bila jedna od uočenih slabosti relacionih sistema za upravljanje bazama podataka. One pružaju veliku fleksibilnost u tom pogledu. Pored toga, kako je kod dinamičkih kolona u *MariaDB* dozvoljeno i ugnježđenje kolona, vrednosti koje su smeštene u dinamičku kolonu mogu se posmatrati i kao dokumenti. Iz tog ugla gledanja, *MariaDB* bi mogla da se koristi u istim slučajevima kao i nerelacione baze podataka koje se koriste za skladištenje dokumenata, ali treba imati u vidu ograničenja dinamičkih kolona i pad performansi pri većem broju ugnježđenja.

Jedan od uočenih nedostatka nerelacionih sistema za upravljanje bazama podataka je nemogućnost izvršavanja transakcija tj. konvergentna konzistentnost. S obzirom da sistem *MariaDB* može koristiti i relacione koncepte u vidu transakcija nad dinamičkim kolonama ovaj nedostatak nerelacionih sistema se može nadomestiti.

Sve ovo ide u prilog sistemu *MariaDB* kao rešenju za upravljanje bazama podataka. Pored toga, dodatna prednost sistema *MariaDB* se ogleda u mogućnost integracije sa drugim podsistemima za rad sa kompleksnim strukturama podataka kao što je *OQGraph*, ali i podsistemima *Cassandra* i *CONNECT* koji doprinose većoj fleksibilnosti sistema.

Još jedna prednost sistema *MariaDB* je to što koristi *SQL* kao upitni jezik, koji je jezik relacionih sistema za upravljanje baza podataka pa je samim tim opšte privaćen i poznat inženjerima koji rade na razvoju aplikacija. Prelazak sa relacionog sistema rada na koncept koji podržava i jedan i drugi koncept je u značajnoj meri olakšan.

5 Zaključak

Cilj ovog rada je predstavljanje sveobuhvatnog utiska o sistemu *MariaDB*. On obuhvata istraživanje osnovnih i naprednih svojstava sistema kao i istraživanje na temu podrške koju sistem pruža relacionim i nerelacionim konceptima budući da sam sistem predstavlja spoj dva modela, relacionog i nerelacionog.

Istraživanje je predstavljeno u 2. poglavlju rada. Sveobuhvatni utisak je da je sistem *MariaDB* fleksibilno softversko rešenje. Fleksibilnost dolazi od toga što sistem dolazi sa većim brojem podrazumevanih podsistema koji pružaju određene funkcionalnosti. Pored podrazumevanih, sistem *MariaDB* ima mogućnost integracije sa drugim podsistemima. Tako na primer, ukoliko aplikacija treba da obezbedi naprednu pretragu teksta koji je pisan na kineskom jeziku, umesto ugrađenog podsistema možemo integrisati i koristiti podsistem *Mroonga*.

Podrška za relacione i nerelacione koncepte je omogućena podsistemima. Ukoliko aplikacija koju razvijamo čuva i obrađuje podatke za koje su nam potrebna svojstva transakcija ili referencijalni integritet, to možemo dobiti koristeći podsistem *InnoDB*, dok recimo *MyISAM* ne pruža podršku za iste.

Ukoliko je potrebna podrška za nerelacione koncepte kao što je dinamička struktura, ona je obezbeđena podrazumevanim podsistemom. Dinamička struktura podataka se ostvaruje korišćenjem dinamičkih kolona. Implementacija dinamičke sheme u relacionim rešenjima je skupa i neefikasna. Pored ovog nerelacionog svojstva sistem *MariaDB* omogućava saradnju sa drugim nerelacionim rešenjima kao što je *MongoDB* ili *Cassandra*.

Imajući u vidu sve navedeno, može se reći da sistem *MariaDB* prevazilazi granice relacionog i nerelacionog modela čineći pritom celinu koja pruža najbolje od oba modela.

6 Reference

1. Emilien Kenler, Federico Razzoli, **“MariaDB Essentials”**, *Packt Publishing Ltd Birmingham B3 2PB, UK.*, Oktobar 2015, ISBN 978-1-78398-286-8
2. Federico Razzoli, **“Mastering MariaDB”**, *Packt Publishing Ltd Birmingham B3 2PB, UK.*, Septembar 2014, ISBN 978-1-78398-154-0
3. Fondacija MariaDB, https://mariadb.com/kb/en/mariadb, datum: 05.04.2017.
4. Saša Malkov, **“Nerelacione baze podataka”**, link: http://poincare.matf.bg.ac.rs/~smalkov/files/dobp.r373.2014/public/predavanja/DOBP.07.2014.nerelacione.baze.pdf, datum: 05.09.2017.
5. Saša Malkov, **“Teorema CAP”**, link: http://poincare.matf.bg.ac.rs/~smalkov/files/dobp.r373.2014/public/predavanja/DOBP.03.2014.cap.projektovanje%20dbp.pdf, datum: 05.09.2017.
6. Anders Karlsson, **“Putting virtual columns good use”**, link: https://mariadb.com/resources/blog/putting-virtual-columns-good-use, datum: 11.08.2014
7. Stephane Combaudon, **“Virtual columns in Mysql and MariaDB”**, link: https://dzone.com/articles/virtual-columns-in-mysql-and-mariadb-mysql-perform, datum: 16.03.2016.
8. Russell J. T. Dyer, **“Dinamyc columns in MariaDB”**, link: http://radar.oreilly.com/2015/04/dynamic-columns-in-mariadb.html, datum: 29.04.2015.
9. Gordana Pavlović-Lažetić, **“Uvod u relacione baze podataka”**, link: http://poincare.matf.bg.ac.rs/~gordana//UvodRBP/FINALE.pdf, datum: 05.09.2017.
10. Peter Zaitsev, “**The Doom of Multiple Storage Engines**”, link: https://www.percona.com/blog/2010/05/08/the-doom-of-multiple-storage-engines/ datum: 08.05.2010