Performanse baza podataka i optimizacija upita

Šuprina, Lidija

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Lidija Šuprina

PERFORMANSE BAZA PODATAKA I OPTIMIZACIJA UPITA

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Lidija Šuprina

JMBAG: 0016129822

Studij: Informacijski i poslovni sustavi

PERFORMANSE BAZA PODATAKA I OPTIMIZACIJA UPITA

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Kornelije Rabuzin

Varaždin, rujan 2022.

Lidija Šuprina

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristila drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada jest performanse baza podataka i optimizacija upita. Performanse baze podataka mogu se definirati kao brzina kojom sustav za upravljanje bazom podataka (SUBP) pruža informacije korisnicima. Na performanse baza podataka utječu propusnost, resursi, radno opterećenje i optimizacija koja ubrzava izvedbu upita. Odgovori na isti upit mogu se dobiti na razne načine, a odabir postupka odgovaranja na upit na najefikasniji način zove se optimizacija upita. Na temelju istraživanja literature, objašnjeni su faktori utjecaja na performanse i iznesena je usporedba SQL i NoSQL sustava za upravljanje bazama podataka. Nad bazom podataka za autobusni kolodvor objašnjeni su ne-optimizirani i optimizirani upiti te planovi tih upita koje MySQL obrađuje.

Ključne riječi: baza podataka; performanse baza podataka; opterećenje; propusnost; resursi; optimizacija; upiti; optimizacija upita; planer upita

Sadržaj

1. Uvod

U ovom radu objašnjena je uloga i značaj performansi baza podataka te faktori koji utječu na performanse. Optimizacija upita, kao jedan od glavnih faktora koji utječu na performanse baza podataka, objašnjena je teorijski i demonstrirana na praktičnim primjerima.

Objašnjeni su osnovni pojmovi kao što su baza podataka, relacijska baza podataka i sustav za upravljanje bazama podataka te ERA dijagram u poglavlju Baza podataka. Predstavljen je i ERA dijagram po kojem je kreirana baza podataka nad čijim se tablicama demonstrira optimizacija upita.

Performanse baza podataka ne mogu se lako definirati jer postoji mnogo faktora koji utječu na performanse, drugim riječima sve utječe na performanse baza podataka – dizajn i arhitekture baza podataka, količina tablica i podataka, odabir sustava za upravljanje bazama podataka, hardverske mogućnosti i postavljanje upita. Postoje alati koji prate performanse baza podataka što čovjeku olakšava pronalaženje problema i brzo rješavanje istoga. Kako performanse ovise o sustavu za upravljanje bazama podataka, predstavljena je usporedba performansi SQL i NoSQL baza podataka.

Optimizaciju upita uglavnom radi sustav za upravljanje bazama podataka jer ima svoj alat (optimizator) koji radi planove upita i bira najefikasniji plan izvođenja te na temelju njega izvodi upit. Iako optimizator prolazi kroz faze optimizacija upita, odnosno radi na pronalaženju najboljeg puta izvođenja, čovjek također ima utjecaja na efikasnost upita načinom na koji postavi upit. Neki jednostavni i svi složeni upiti mogu se postaviti na više različitih načina, ali nisu svi optimizirani. Smjernice i primjeri za optimizaciju upita mogu se pronaći u poglavlju Optimizacija upita.

2. Metode i tehnike rada

Za pisanje prvog dijela ovog rada bilo je potrebno istražiti što su performanse baza podataka, koji su faktori utjecaja na performanse te kako se na njih može utjecati. Istraživanje je provedeno nad različitim materijalima iz knjižnice, stručnih izvora i s interneta. Napravljen je ERA dijagram u alatu Visual Paradigm Online na temelju kojeg je kreirana baza podataka u sustavu za upravljanje bazama podataka MySQL, a za MySQL koristila se aplikacija PhpMyAdmin koja dolazi u sklopu Wamp lokalnog poslužitelja. Testiran je alat za praćenje performansi baza podataka SolarWinds Database Performance Analyzer na online interaktivnom testu (https://database.demo.solarwinds.com/). Za optimizaciju upita bilo je potrebno istražiti kako sustav za upravljanje bazama podataka obrađuje upite te na koji se način upite može optimizirati. Za to su korišteni materijali iz knjižnice, stručnih izvora i s interneta, a za testiranje i demonstraciju upita i planova upita korišten je PhpMyAdmin.

3. Baza podataka

Pojam baze podataka ima mnogo definicija, ali svaka od njih na neki način govori da je baza podataka organizirani skup podataka kojim se može manipulirati. Neke od definicija baza podataka jesu:

> „Baza podataka je kolekcija podataka, ograničenja i operacija koja reprezentira neke aspekte realnog svijeta.“ [1]
>
> „Baza podataka je skup informacija koje su međusobno povezane, a koje se sustavno pohranjuju i organiziraju kako bi se olakšalo njegovo čuvanje, pretraživanje i uporaba.“ [2]
>
> „Baza podataka je velika količina informacija pohranjenih na računalu na način da h se lako može čitati ili mijenjati.“ [3]

Dakle, baza podataka jest organizirani skup strukturiranih podataka koji predstavlja dio realnog svijeta, a pohranjuje se i organizira na računalnom sustavu zajedno s ograničenjima i operacijama da bi se lako i jednostavno moglo manipulirati tim podacima, odnosno da bi se podatke moglo unositi, pretraživati, mijenjati i brisati.

3.1. Model podataka

Osnova sustava za oblikovanje i implementaciju baze podataka jest model podataka. Model podataka je način predstavljanja podataka koji se sastoji od strukturalne, integritetne i operativne komponente. [1] Strukturalna komponenta obuhvaća definiranje podataka, integritetna komponenta definiranje pravila integriteta podataka i operativna komponenta definiranje pravila manipulacije podacima i jezika koji se koristi za manipulaciju podacima. [4]

Prema modelu podataka, postoji više vrsta baza podataka. Neke od njih jesu: [5]

- Hijerarhijske baze podataka,
- Mrežne baze podataka,
- NoSQL baze podataka,
- Objektno-orijentirane baze podataka,
- Relacijske baze podataka.

Hijerarhijske baze podataka temelje se na hijerarhijskoj pohrani podataka u slogove koje se može smatrati skupom stabala. [4] Pojedino se stablo sastoji od čvorova (koji predstavljaju slog) i „roditelj-dijete“ veza pri čemu jedan slog roditelj može imati nula ili više slogova dijete, a jedan slog dijete mora imati jednog slog roditelja osim osnovnog sloga kojeg

se naziva korijenom stabla. Ovakav model predstavlja jednostavnu pohranu podataka po važnosti, ali mu je nedostatak ponavljanje podataka zbog pohrane podataka samo preko veza 1:M (jedan na više) ili 1:1 (jedan na jedan).

Mrežne baze podataka su hijerarhijske baze podataka s proširenjem koje omogućuje slogovima dijete da imaju više od jednog sloga roditelj što omogućuje stvaranje veza M:N (više na više) pri čemu struktura baze podataka nalikuje na mrežu. Prema ovom modelu, manje je ponavljanja podataka u odnosu na hijerarhijski model, ali je takav model vrlo složen što otežava kasnije mijenjanje strukture baze podataka. [5]

NoSQL baze podataka su baze podataka namijenjene za pohranu velike količine podataka koje su prisutne u modernim aplikacijama. [6] Omogućuju veliku skalabilnost i jednostavnu manipulaciju nad većom količinom podataka.

Objektno-orijentirane baze podataka su baze podataka temeljene na klasama i objektima na način da je jedan objekt instanca klase. Klasa može imati više instanci, odnosno objekata. Prednost ovog tipa baze podataka jest brza obrada velike količine podataka. [7]

Ovisno o tome kojim se podacima treba upravljati, bira se određeni model podataka, a zajedno s modelom podataka izabire se i sustav za upravljanje bazama podataka koji podržava odabrani model podataka. U ovom će se radu koristiti relacijska baza podataka.

3.2. Relacijske baze podataka

Relacijske baze podataka temelje se na prikazu podataka u obliku relacija, odnosno tablica. Relacija (tablica) se sastoje od atributa (stupaca) koji čine zaglavlje relacije i slogova (redaka) koji čine tijelo relacije. [4] Skup svih atributa jedne relacija naziva se relacijskom shemom, a shema baze podataka je skup svih relacijskih shema u bazi podataka. [1] [4]

Atribut je svojstvo nekog objekta iz realnog svijeta o kojem se prikupljaju podaci. Svi su podaci unutar jednog atributa istog tipa, a različiti atributi mogu sadržavati podatke različitog tipa, odnosno svaki atribut mora imati definiran tip podatka, a sve vrijednosti unutar tog atributa će biti istog tog tipa podatka. Redoslijed atributa u relaciji nije važan, ali svaki atribut mora imati jedinstven naziv unutar relacije. Ovisno o tipu podatka i ograničenjima koja se postave uz definiranje atributa dobiva se skup mogućih vrijednosti koje jedan atribut može imati, a taj skup naziva se domenom atributa. Broj atributa unutar relacije naziva se kardinalnošću relacije. [4]

Slog ili n-torka je jedan objekt iz realnog svijeta o kojem se prikupljaju određeni podaci, a ti podaci su svojstva koja postaju vrijednosti atributa. Što se tiče redoslijeda

slogova, niti on nije važan jer se relacija smatra matematičkim skupom u kojem redoslijed elemenata nije bitan. Također, unutar relacije ne smiju postojati dva sloga sa svim istim vrijednostima atributa, dakle mora im barem jedna vrijednost atributa biti različita. Upravo je u tome i važnost promatranja vrijednosti jednog atributa jednog sloga u odnosu na ostale atribute tog sloga kao i u odnosu na ostale slogove unutar relacije. [4]

Kako bi se osigurao ispravan rad sustava i zaštitili podaci, nad atributima je potrebno definirati ograničenja koja se nazivaju pravilima integriteta. Ta ograničenja postavljaju se na temelju ograničenja iz realnog svijeta za objekte iz realnog svijeta o kojima se prikupljaju podaci, odnosno instance entiteta [8]. Primjer ograničenja u relaciji *Osoba* bio bi da se za atribut *Masa* postavi ograničenje da vrijednost ne smije biti manja od 0 kg jer ne postoji osoba koja ima negativnu masu. Pravila integriteta su uglavnom različita za svaku bazu podataka, osim osnovnih pravila: pravilo integriteta primarnog ključa i pravilo referencijalnog integriteta. [4]

Za relacijske baze podataka vrlo je bitan pojam primarnog ključa. Primarni je ključ poseban atribut ili skup atributa koji služi kao jedinstveni identifikator. Vrijednosti primarnog ključa jednoznačno određuju svaki slog, stoga ne postoje dva ista sloga u relaciji. Relacija može imati samo jedan primarni ključ, a vrijednost primarnog ključa mora biti unesena te, po pravilu integriteta primarnog ključa, ne smije sadržavati *null*. [9]

Da bi se više relacija u bazi podataka povezale, važno je definirati i pojam vanjskog ključa. Vanjski je ključ atribut relacije u kojoj se sprema vrijednost primarnog ključa druge (moguće iste) relacije. Na taj se način stvara veza između dvije relacije. Relacija, također, može biti vezana uz samu sebe na način da uz atribut koji čini primarni ključ, sadrži atribut istog tipa podatka kao primarni ključ te je definiran kao vanjski ključ koji sadržava vrijednosti primarnog ključa iste relacije. Za razliku od primarnog ključa, vrijednosti vanjskog ključa mogu biti iste za više slogova, ali po pravilu referencijalnog integriteta atribut koji je definiran kao vanjski ključ ne smije sadržavati vrijednosti za koje ne postoji odgovarajuća vrijednost primarnog ključa u relaciji na koju se referencira. [10]

3.3. Sustav za upravljanje bazama podataka

Sustav za upravljanje bazama podataka jest skup programa koji se koriste za implementaciju baze podataka, a omogućuju opis i rukovanje podacima na jednostavan način zahvaljujući korisničkom sučelju na visokoj razini (neovisno o strukturi podataka u računalu) koje sadrži skup programskih rješenja za lako i jednostavno manipuliranje podacima kao što je primjerice generator izvještaja. [4]

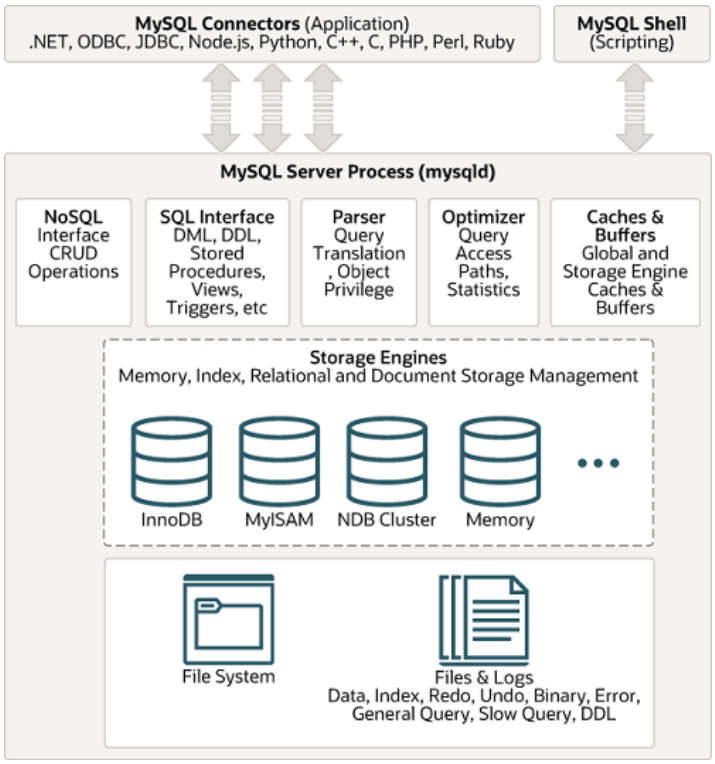
U siječnju 2022. pet najpopularnijih sustava za upravljanje bazama podataka na svijetu bili su: [11]

1. Oracle,
2. MySQL,
3. Microsoft SQL Server,
4. PostgreSQL,
5. MongoDB .

3.3.1. MySQL

U ovom će se radu koristiti sustav za upravljanje bazama podataka MySQL. MySQL je najpopularniji sustav za upravljanje bazama podataka otvorenog koda u svijetu. Postoje besplatna inačica i inačica MySQL-a koja se plaća. Pisan je u programskim jezicima C i C++ te koristi visoko optimiziranu biblioteku klasa koja mu omogućuju veliku brzinu. Zadnja verzija MySQL-a je 8.0, a i dalje je u razvoju. [12] Prednost mu je što je dobar i za male i za velike baze podataka jer se temelji na relacijskom modelu podataka.

MySQL ima klijentsko-poslužiteljsku arhitekturu koja se sastoji od tri sloja: klijentski, poslužiteljski i mehanizam za pohranu (eng. storage engine).



Slika 1: MySQL arhitektura (Izvor: https://dev.mysql.com/doc/refman/8.0/en/pluggable-storage-overview.html)

Klijentski sloj arhitekture predstavlja dio za interakciju s krajnjim korisnicima sustava. Koristeći korisničko sučelje ili naredbeni redak za slanje naredbi poslužiteljskom sloju, korisnik dobiva odgovarajući odgovor na ekranu uključujući i poruke greške u slučaju pogrešnog slanja naredbe. [13] Neki od važnih servisa klijentskog sloja su upravljanje spajanjima na bazu, autentikacija i sigurnost. [14]

Poslužiteljski sloj arhitekture predstavlja dio u kojem se odvija cijela logika sustava. Prima zahtjeve koje korisnik šalje i vraća odgovarajuće povratne informacije. Neke od komponenata poslužiteljskog sloja su upravljanje dretvama (eng. thread) koje čine poveznicu s korisnikom, prevoditelj (eng. parser) koji poslanu naredbu pretvara u oblik razumljiv računalu, optimizator (eng. optimizer) koji pronalazi optimalni način za izvršenje naredbe, međuspremnik i predmemorija (eng. buffer and cache) u kojima su spremljeni rezultati prethodnih upita ili naredbi, predmemorija metapodataka u kojoj su spremljene informacije o bazama (podaci o podacima). [14]

Treći je sloj MySQL mehanizam za pohranu kojim se određuje korištenje načina kojim se pristupa podacima na disku. Najpopularniji mehanizmi za pohranu su InnoDB, MyISAM, NDB Cluster I Memory. U MySQL-u zadani mehanizam za pohranu je InnoDB, a postavku je moguće promijeniti globalno na razini sustava kao i na pojedinim tablicama. Moguće je u istoj bazi koristiti različite mehanizme, ali se preporuča definiranje cijele baze na istom mehanizmu. U sljedećoj tablici nalazi se usporedba karakteristika najpopularnijih mehanizama za pohranu. [15]

Tablica 1: Usporedba karakteristika mehanizama za pohranu

Svojstvo	InnoDB	MyISAM	NDB	Memory
Ograničenje veličine	64 TB	256 TB	348 EB	Raspoloživa RAM memorija
Transakcije	Da	Ne	Da	Ne
Opseg zaključavanja	Slog	Tablica	Slog	Tablica
Podrška za prostorno orijentirane podatke	Da	Da	Da	Ne
Podržane vrste indeksa	B-tree, Full-text Clustered	B-tree, Full-text	T-tree Hash	B-tree, Hash
Predmemoriranje podataka	Da	Ne	Da	Sve je u RAM memoriji
Predmemoriranje indeksa	Da	Da	Da	Sve je u RAM memoriji
Kompresija podataka	Da	Da	Ne	Ne
Enkripcija podataka	Da	Da	Da	Da
Replikacija	Da	Da	Da	Da
Podrška za vanjske ključeve	Da	Ne	Da	Ne
Podrška za distribuirane baze	Ne	Ne	Da	Ne

(Izvor: Nenad Crnko, 2017)

3.4. ERA dijagram

ERA dijagram koristi se za planiranje implementacije baze podataka kako bi se osiguralo optimalno rješenje dizajna baze podataka. Omogućuje bolje shvaćanje strukture podataka i aplikacijske domene čime se postiže bolji dizajn baze podataka. ERA dijagramom predstavljaju se korisnički zahtjevi u obliku entiteta, atributa i veza koji se kasnije u sustavu za upravljanje bazama podataka implementiraju kao tablice za svaki entitet sa stupcima koje određuju njihovi atributi, a veze u dijagramu implementiraju se kao vanjski ključevi. [8]

Kao što je ranije već spomenuto, entiteti su objekti iz realnog svijeta o kojem se prikupljaju podaci, a u ERA dijagramu prikazuju se pravokutnicima. Atributi su svojstva entiteta relevantna za aplikacijsku domenu, odnosno svojstva koja je potrebno pohraniti u bazu podataka. U ERA dijagramu, atributi se prikazuju u pravokutniku entiteta ispod njegovog imena. [8]

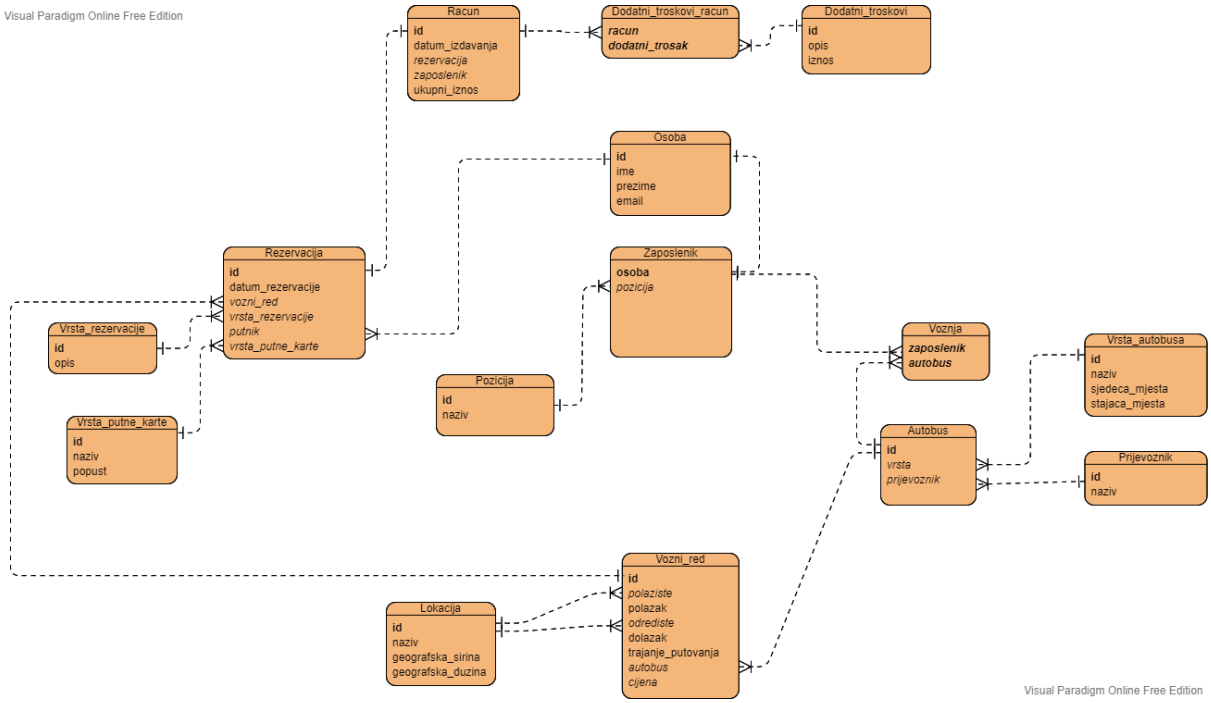
Kao i u realnom svijetu, entiteti su međusobno povezani. Da bi se taj odnos između entiteta prikazao, koriste se veze. Za prikaz veze između entiteta, potrebno je znati količinu tipova entiteta koji sudjeluju u vezi, kardinalnost i opcionalnost. [8]

Prema broju tipova entiteta koji sudjeluju u vezi, postoje unarne veze, binarne veze, ternarne veze i veze višeg reda. [8] Unarna veza je veza između istog tipa entiteta – može se reći da je tip entiteta povezan sa samim sobom. Binarna veza je veza između dva tipa entiteta te se ona najčešće koristi. Ternarna veza je veza između tri tipa entiteta. U vezi između tipova entiteta mogu sudjelovati i više od tri tipa entiteta te se u tom slučaju koriste veze višeg reda.

Za veze je još bitno odrediti minimalan i maksimalan broj sudionika u vezi. Kardinalnost predstavlja maksimalni broj sudionika u vezi, a prema kardinalnost razlikuju se veze jedan naprema jedan (1:1), jedan naprema više (1:M) i više naprema više (M:N). Opcionalnost predstavlja minimalan broj sudionika u vezi na temelju čega razlikujemo veze nula (0) ili jedan (1). Drugim riječima, opcionalnost govori o tome mora li entitet sudjelovati u vezi (jedan) ili ne mora, ali može (nula). [8]

3.4.1. ERA dijagram za autobusni kolodvor

U praktičnom dijelu ovog rada, optimizirat će se upiti nad tablicama baze podataka za autobusni kolodvor. Baza podataka napravljena je prema sljedećem ERA dijagramu.



Slika 2: ERA dijagram za autobusni kolodvor

3.5. SQL

SQL je jezik visoke razine koji se koristi za rad s bazama podataka. Puni naziv je na engleskom jeziku Structured Query Language što bi u doslovnom prijevodu značilo strukturirani upitni jezik. Uz postavljanje upita kao što mu ime govori, SQL se koristi i za kreiranje, mijenjanje, brisanje podataka, objekata u bazi kao i same baze podataka. [10]

Naredbe SQL-a mogu se kategorizirati u pet skupina:

- DDL (eng. Data Definition Language),
- DML (eng. Data Manipulation Language),
- DQL (eng. Data Query Language),
- DCL (eng. Data Control Language),
- TCL (eng. Transaction Control Language).

DDL čine naredbe za definiranje sheme baze podataka, odnosno naredbe za kreiranje, mijenjanje i brisanje objekata u bazi podataka. Najvažnije DDL naredbe jesu CREATE (za kreiranje objekata u bazi ili same baze podataka), DROP (za brisanje objekata u bazi ili same baze podataka i ALTER (promjena strukture baze podataka).

DML čine naredbe za upravljanje podacima, odnosno naredbe za dodavanja, mijenjanje, brisanje podataka. Najvažnije DML naredbe jesu INSERT (za unos podataka u

tablicu), UPDATE (za promjenu postojećih podataka u tablici) i DELETE (za brisanje redova iz tablice)

DQL čini naredba za postavljanje upita nad podacima i takva je samo jedna naredba, a to je SELECT.

DCL čine naredbe za određivanje prava za kontrolu nad bazom podataka.

TCL čine naredbe za upravljanje transakcijama, odnosno skupom zadataka koji se izvršavaju jednim pozivom. [16] [10]

4. Performanse baza podataka

U informatičkoj industriji sve se više pažnje obraća na performanse sustava te na optimizaciji faktora koji utječu na njegove performanse. Poboljšanje performansi može se shvatiti kao ekonomičnost, odnosno da se uz što manje resursa postigne što bolje učinak u sustavu, a za to je potrebno praćenje i unaprjeđenje svih segmenata sustava.

Tako je važno pratiti i poboljšati performanse baze podataka na koju utječu mnogi faktori. Osim toga što performanse ovise i o odabranom sustavu za upravljanje bazom podataka, glavni faktori koji utječu na performanse baze podataka su:

- radno opterećenje (eng. workload),
- propusnost (eng. throughput),
- resursi (eng. resources),
- optimizacija (eng. optimization),
- konflikti (eng. contention).

Prema glavnih pet faktora koji utječu na performanse baze podataka proizlazi sljedeća definicija:

> „Performanse baze podataka mogu se definirati kao optimizacija resursa na način da se poveća propusnost i pritom minimiziraju konflikti kako bi se omogućilo obrađivanje najvećeg mogućeg radnog opterećenja.“ [17]

Za praćenje metrika i poboljšanje performansi koriste se razne metode i alati, a da bi se njih razumjelo, potrebno je prvo pojasniti faktore koji utječu na performanse.

4.1. Faktori utjecaja

4.1.1. Radno opterećenje

Radno opterećenje (eng. workload) predstavlja ukupnu potražnju određenih procesa koja se zahtijeva od sustava za upravljanje bazama podataka u danom trenutku. Radno opterećenje baze podataka obuhvaća upite, transakcije, skupne procese (eng. batch jobs), analize podataka i uslužne programe te naredbe sustava upućene sustavu za upravljanje bazama podataka u bilo kojem trenutku. Ukupno radno opterećenje itekako utječe na performanse baze podataka, a varira u različitim vremenskim razdobljima. [17]

Ponekad se može radno opterećenje može predvidjeti, odnosno očekivati na temelju dotadašnjeg praćenja. [17] Na primjer, očekuje se veliko radno opterećenje nad sustavima za upravljanje bazama podataka tijekom radnog vremena radnim danima dok se vikendom i radnim danima u noćnim satima ne očekuje veće radno opterećenje. Naravno, radno opterećenje ovisi o veličini baze podataka te količini korisničkih zahtijeva. Praćenjem radnog opterećenja i prepoznavanjem vremenskog perioda kada je ono najveće, može se optimizirati upotreba resursa čime se poboljšava obrada radnog opterećenja.

4.1.2. Propusnost

Propusnost (eng. throughput) je ukupna sposobnost sustava za obradu podataka, a ovisi i o hardverskoj i o softverskoj mogućnosti obrade. Ovisi o učinkovitosti operacijskog sustava, brzini procesora, brzini I/O diska, propusnosti memorije i drugim hardverskim specifikacijama. Propusnost sustava za upravljanje podataka mjeri se kao prosječno vrijeme odgovora (eng. response time) ili u broju upita (ili transakcija) po sekundi. [17] [18]

4.1.3. Resursi

Resurse (eng. resources) predstavljaju hardverski i softverski alati koje koristi sustav za upravljanje bazom podataka. Kao i kod propusnosti, naglasak se stavlja na hardver i njegove performanse. Da bi se moglo predviđati kakve bi performanse baza podataka mogla imati, bitno je znati performanse hardverskih komponenti kao što su procesori i tvrdi diskovi, odnosno bitno je znati kako se koristi memorija na računalu. [17] [18]

4.1.4. Optimizacija

Optimizacija (eng. optimization) sustava za upravljanje bazom podataka uvelike utječe na performanse baze podataka. Optimizacijom se postiže veća propusnost poboljšanjem softverskih komponenti.

Kako bi se relacijskom optimizatoru omogućio najbolji pristup podacima, trebalo bi optimizirati konfiguracijske parametre baze podataka, parametre sustava, raspodjelu podataka te dizajn tablice i SQL upite. [18] Kada se govori o optimizaciji sustava za upravljanje bazom podataka, najveći naglasak stavlja se na optimizaciji upita o čemu će se više govoriti u sljedećem poglavlju.

Osim optimizacije sustava za upravljanje bazom podataka, na rad cijelog sustava utječu i komponente na koje relacijski optimizator ne utječe, a to su dobar dizajn aplikacije, učinkovito i kvalitetno kodiranje skripti, izbjegavanje ponavljajućeg koda... [17]

4.1.5. Konflikti

Konflikti (eng. contention) nastaju u trenutku kada je radno opterećenje nad nekim resursom veliko, odnosno kada dvije ili više komponenti pokušavaju koristiti isti resurs na konfliktan način. Primjer konflikta bi bio kada bi dva korisnika u istom trenutku pokušala izmijeniti isto polje jednog retka u tablici. Sustav u tom slučaju ne bi znao čiju promjenu primijeniti te bi pokušao primijeniti obje, ali se javlja konflikt. [18]

Sustav za upravljanje podataka koristi mehanizam zaključavanja kako bi osigurao integritet podataka, a mehanizam zaključavanja ne dopušta trenutno mijenjanje podataka ostalim korisnicima u trenutku u kojem neki korisnik mijenja te podatke dok taj korisnik ne završi s izmjenom. Nastankom i povećanjem konflikata, dolazi do zaključavanja čime se uzrokuje smanjenje propusnosti. [17]

4.2. Praćenje performansi

Kako bi se osiguralo poboljšanje performansi i korištenje baza podataka, potrebno je učinkovito i kontinuirano pratiti faktore koji utječu na performanse. Danas se za to koriste razni programi pomoću kojih se, na temelju izvještaja o performansama, lako mogu otkriti i riješiti problemi. Neke od metrika praćenja baza podataka jesu praćenje postavki i infrastrukture baze podataka, radnog opterećenja, broj spajanja na bazu kao i operacije nad tablicama, broj transakcija, zaključavanje tablica i potrošnju memorije. [19]

Programi za praćenje performansi imaju već automatski postavljene parametre kao mjerilo na temelju kojih se rade izvještaji o rezultatima praćenja. Parametri se mogu ručno mijenjati, ovisno o aplikacijskoj infrastrukturi i željenim performansama. Program kontinuirano uspoređuje parametre s trenutnim metrikama performansi, a ako trenutne metrike poprilično odstupaju od zadanih mjerila, šalje se upozorenje administratoru baze podataka kako bi pravovremeno reagirao. Podaci o usporedbama zadanog mjerila i trenutnih metrika se pohranjuju, sortiraju i analiziraju te ih program koristi za određivanje normalnih odstupanja od mjerila i anomalija u radu baze podataka. Na temelju spremljenih podataka i izvješća, mogu se ranije prepoznati temeljni problemi čijim se rješavanjem omogućuje bolji rad baze podataka u budućnosti. [19]

4.2.1. Alati za praćenje performansi

Praćenje performansi baze podataka pomoću alata omogućuje uvid i u hardverske i u softverske performanse, a neki od tih alata jesu: [19]

- SolarWinds Database Performance Monitor,

- SolarWinds Database Performance Analyzer,
- SQL Sentry,
- Paessler PRTG Network Monitor.

SolarWinds Database Performance Monitor je najpopularniji alat za praćenje performansi baza podataka. Prati učestalost implementacije, smanjenje neuspjelih implementacija, dostupnost i vrijeme potrebno za promjene u svakom trenutku svakoga dana. Također, analizira trendove arhitekture baza podataka na temelju kojih nudi preporuke za implementaciju skalabilne i optimizirane baze podataka. Osim širokog raspona korisnih značajki koje alat nudi, ima jednostavno i intuitivno korisničko sučelje. [19]

SQL Sentry prati performanse baza podataka na SQL Serveru na raznim platformama. Sa zastarjelim dizajnom korisničkog sučelja, nudi dva načina nadzorne ploče, a to su povijest i uzorak. Način povijesti prikazuje performanse tijekom određenog vremenskog razdoblja, a uzorak prikazuje trenutne performanse. [19]

Paessler PRTG Network Monitor osim praćenja performansi samih baza podataka, prati performanse cijele infrastrukture informacijske tehnologije što znači praćenje baze, aplikacije, paketa, prometa, usluga u oblaku, web usluga, sigurnosti, virtualnih i fizičkih okruženja, korištenje diska, hardvera... Alat ima različita sučelja za mobilne uređaje i osobna računala, a podržava širok raspon tehnologija. [19]

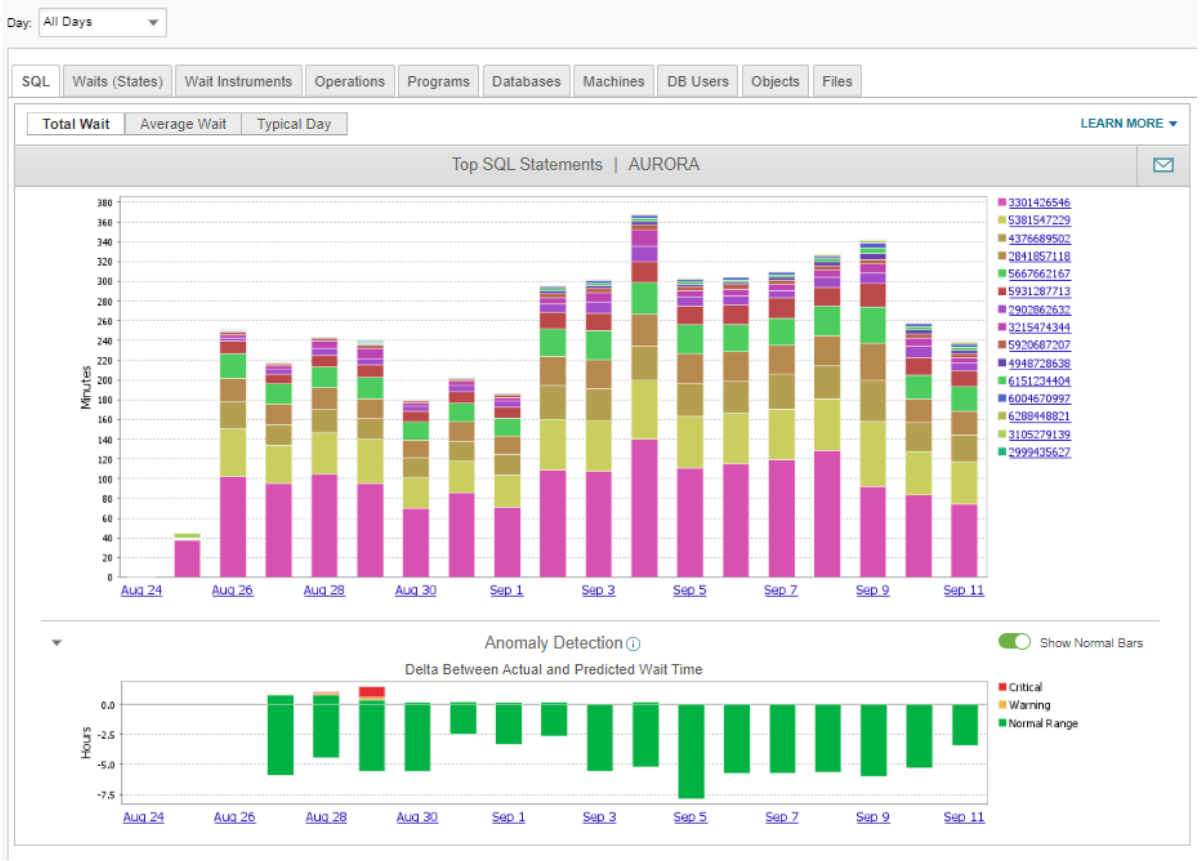
4.2.2. Alat SolarWinds Database Performance Analyzer

SolarWinds Database Performance Analyzer je alat čiji je fokus stavljen na otkrivanje anomalija u performansama pomoću strojnog učenja čime se omogućuje predviđanje prekida rada, a može se ga se konfigurirati da prati višeplatformske baze podataka. [19]

Slika 3: Početna stranica alata SolarWinds Database Performance Analyzer

Na početnoj stranici alata SolarWinds Database Performance Analyzer može se vidjeti popis baza podataka čije se performanse prate kao i sažetak njihovih performansi. Grafovi prikazuju baze podataka s najvećim vremenom odgovora na temelju mjerenja i kategoriziranja vremena potrošenog od zahtjeva za upit i odgovora na upit. Kada se upit pokrene, upit je kategoriziran u različita stanja i kontinuiranim mjerenjem, alat prepoznaje na koje se upite troši najviše vremena. Interaktivna online demonstracija ovog alata prati 34 baze podataka u različitim sustavima za upravljanje bazama podataka. [20]

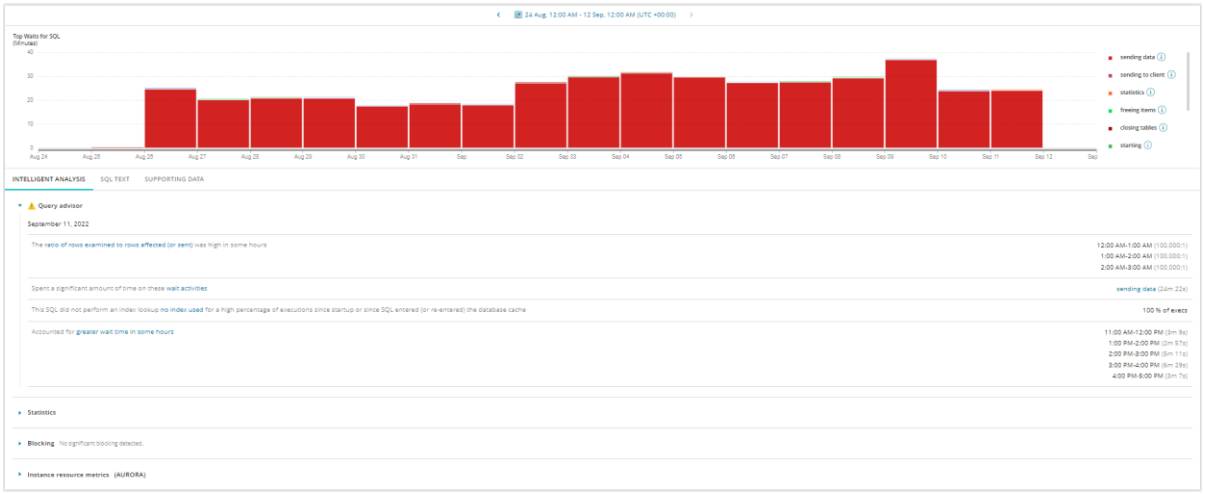
Koristi se i za analizu vremena dohvaćanja rezultata za pojedini upit, a osim vremena čekanja, prikazuje i stanje resursa što omogućuje prikaz odnosa između izvedenih upita i njihovih utjecaja na resurse poslužitelja. Također, SolarWinds Database Performance Analyzer izvještava i o stanju čekanja upita zbog blokiranja. [20]



Slika 4: Graf naredbi s najvećim vremenom čekanja (gore) i graf anomalija (dolje) u bazi podataka „AURORA“

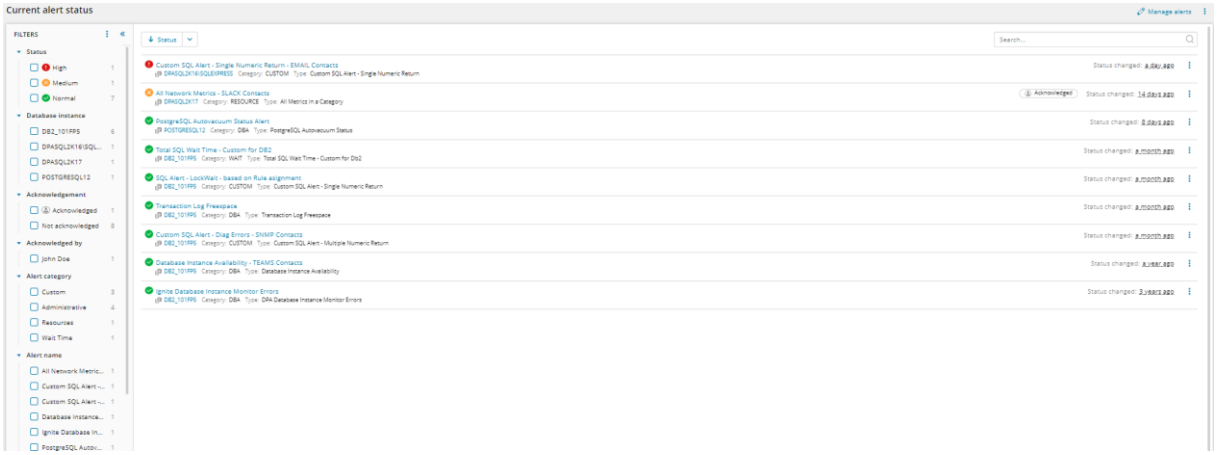
Odabirom jedne od baza podataka s popisa na početnoj stranici, mogu se vidjeti detalji o performansama baze podataka po danima. Klikom na određeni dan, prikazuje se

detaljniji izvještaj o performansama po satima. Za svaki upit koji je izveden, mogu se pronaći detaljne statistike o izvođenju tog upita uz grafičke prikaze što olakšava snalaženje.



Slika 5: Graf vremena čekanja za jedan od upita u bazi podataka „AURORA“

Alat nudi i generiranje grafičkih izvještaja, ali i kreiranje poruka upozorenja. Također se nudi i prikaz svih generiranih izvještaja i poruka upozorenja te povijest poslanih poruka upozorenja.



Slika 6: Kreirane poruke upozorenja za bazu podataka „AURORA“

Potrebno je neko vrijeme za snalaženje u alatu SolarWinds Database Performance Analyzer i pronalaženje svih mogućnosti koje nudi, ali na svakom koraku nudi se više informacija o samoj opciji koju se odabire te na koji način SolarWinds Database Performance Analyzer dolazi do tih podataka.

4.3. Usporedba performansi DBMS-ova

Za usporedbu performansi različitih sustava za upravljanje bazama podataka potrebno je odrediti metrike performansi, odabrati odgovarajuće SUBP-ove te alat kojim se mjere performanse. Kamil Kolonko [21] napravio je usporedbu performansi relacijskog i nerelacijskog sustava za upravljanje bazama podataka tako da je za različita radna opterećenja mjerio propusnost i ukupno vrijeme izvođenja svih operacija radnog opterećenja. Također se mjerila latencija (u mikrosekundama), odnosno prosječno vrijeme izvođenja operacija. Koristio je alat YCSB (Yahoo! Cloud Serving Benchmark) za dobivanje rezultata, a usporedio je performanse najpopularnijih SUBP-ova Oracle kao relacijski SUBP i MongoDB kao nerelacijski (i NoSQL) SUBP.

Da bi se usporedile performanse, baza podataka u oba sustava mora imati isti skup podataka. Karakteristike odabranog skupa podataka dane su u tablici, a na temelju tog skupa podataka implementirane su baze podataka u Oracleu i MongoDB-u.

Tablica 2: Karakteristike promatranog skupa podataka

Skup podataka	Broj tablica	Broj redaka	Broj stupaca	Ukupna veličina [B]
TSE	4039	4130774	20193	198276112

(Izvor: Kamil Kolonko, 2018)

Performanse su izmjerene za različita radna opterećenja kako bi se što preciznije odredila propusnost za različit skup operacija koje se izvode nad bazom podataka. Kolonko [21] je koristio radna opterećenja prikazana u Tablici 2, pri čemu su vrijednosti operacija čitanja, ažuriranja, skeniranja, unošenja i čitanje-ažuriranje-unošenje za svako radno opterećenje dane kao proporcije u odnosu na ukupan broj operacija (suma u svakom retku tablice mora biti 1).

Tablica 3: Radna opterećenja

Radno opterećenje	Broj operacija	Operacija čitanja	Operacija ažuriranja	Operacija skeniranja	Operacija unošenja	Operacija čitanje-ažuriranje-unošenje
A (Veliko ažuriranje)	1000	0.5	0.5	0	0	0
B (Veliko čitanje)	1000	0.95	0.05	0	0	0
C (Samo čitanje)	1000	1	0	0	0	0
D (Čitanje najnovijeg)	1000	0.95	0	0	0.05	0
E (Mali dometi)	1000	0	0	0.95	0.05	0
F (Veliko unošenje)	1000	0.5	0	0	0	0.5

(Izvor: Kamil Kolonko, 2018)

U daljnjim opisima rezultata za svako od radnih opterećenja navedenih u tablici, vidjet će se koliko NoSQL baza podataka ima bolje performanse od relacijske baze podataka, odnosno sustava Oracle. MongoDB prikazuje značajno bolje rezultate u propusnosti i ukupnom vremenu izvođenja operacija kao i latenciji za svako radno opterećenje. Prikazani su rezultati za prosječnu, minimalnu i maksimalnu latenciju pri čemu je minimalna latencija najkraće vrijeme izvođenja jedne operacije od zadanih 1000, a maksimalna latencija je najduže vrijeme izvođenja jedne operacije od zadanih 1000. Prosječna latencija je kvocijent zbroja latencija svih operacija i ukupnog broja operacija koje se izvršavaju.

4.3.1. Rezultati za radno opterećenje A

Za 1000 operacija u radnom opterećenju pri čemu 50% operacija čine operacije čitanja, a drugih 50% operacija čine operacije ažuriranja, MongoDB daje rezultate propusnosti od 1021.45 operacija po sekundi i ukupnog vremena izvođenja svih 1000 operacija od 979 ms. Za isto radno opterećenje, Oracle ima propusnost od 54.44 operacija po sekundi, a ukupno vrijeme izvođenja svih 1000 operacija jest 18369 ms.

Tablica 4: Latencija za radno opterećenje A

	MongoDB		Oracle	
	Čitanje	Ažuriranje	Čitanje	Ažuriranje
Prosječna latencija [µs]	599.85	1087.50	24226.56	9724.71
Minimalna latencija [µs]	92.0	174.0	83.0	4244.0
Maksimalna latencija [µs]	14663.0	131711.0	10870783	22159.0

(Izvor: Kamil Kolonko, 2018)

Što se tiče propusnosti i ukupnog vremena izvođenja operacija, MongoDB je za radno opterećenje A u velikoj prednosti. Isto vrijedi i za prosječnu latenciju koja je drastično viša za Oracle u usporedbi sa sustavom MongoDB. Doduše, manja razlika između minimalne i maksimalne latencije za operaciju ažuriranja prikazuje da je ažuriranje u sustavu Oracle puno stabilnije dok je čitanje stabilnija operacija u sustavu MongoDB.

4.3.2. Rezultati za radno opterećenje B

Za 1000 operacija u radnom opterećenju pri čemu 95% operacija čine operacije čitanja, a preostalih 5% operacija čine operacije ažuriranja, MongoDB daje rezultate propusnosti od 797.45 operacija po sekundi i ukupnog vremena izvođenja svih 1000

operacija od 1254 ms. Za isto radno opterećenje, Oracle ima propusnost od 211.82 operacija po sekundi, a ukupno vrijeme izvođenja svih 1000 operacija jest 4721 ms.

Tablica 5: Latencija za radno opterećenje B

	MongoDB		Oracle	
	Čitanje	Ažuriranje	Čitanje	Ažuriranje
Prosječna latencija [µs]	1095.04	1632.7	2756.91	10926.16
Minimalna latencija [µs]	73.0	485.0	98.0	6132.0
Maksimalna latencija [µs]	121279.0	15223.0	650239.0	13183.0

(Izvor: Kamil Kolonko, 2018)

MongoDB ima bolju propusnost i ukupno vrijeme izvođenja operacija kao i prosječnu latenciju od sustava Oracle i za radno opterećenje B. Oracle jedino prednjači u maksimalnoj latenciji za operaciju ažuriranja. Iako je MongoDB bolji u performansama, zanimljivo je da, u odnosu na radno opterećenje A, ima slabiji rezultat dok se za Oracle vidi poboljšanje u propusnosti kao u za ukupno vrijeme izvođenja operacija. Iako i ovdje operacija ažuriranja u sustavu Oracle ima stabilnije rezultate, to nije pridoneslo poboljšanju prosječne latencije te MongoDB i dalje ima drastično bolji rezultat.

4.3.3. Rezultati za radno opterećenje C

Za 1000 operacija u radnom opterećenju pri čemu 100% operacija čine operacije čitanja, MongoDB daje rezultate propusnosti od 1213.59 operacija po sekundi i ukupnog vremena izvođenja svih 1000 operacija od 824 ms. Za isto radno opterećenje, Oracle ima propusnost od 249.31 operacija po sekundi, a ukupno vrijeme izvođenja svih 1000 operacija jest 4011 ms.

Tablica 6: Latencija za radno opterećenje C

	MongoDB	Oracle
	Čitanje	Čitanje
Prosječna latencija [µs]	687.11	2856.57
Minimalna latencija [µs]	78.0	114.0
Maksimalna latencija [µs]	118975.0	565759.0

(Izvor: Kamil Kolonko, 2018)

Za radno opterećenje C, nakon analize prethodnih radnih opterećenja, MongoDB očekivano prikazuje bolje performanse.

4.3.4. Rezultati za radno opterećenje D

Za 1000 operacija u radnom opterećenju pri čemu 95% operacija čine operacije čitanja, a preostalih 5% operacija čine operacije unošenja, MongoDB daje rezultate propusnosti od 1612.9 operacija po sekundi i ukupnog vremena izvođenja svih 1000 operacija od 620 ms. Za isto radno opterećenje, Oracle ima propusnost od 267.02 operacija po sekundi, a ukupno vrijeme izvođenja svih 1000 operacija jest 3745 ms.

Tablica 7: Latencija za radno opterećenje D

	MongoDB		Oracle	
	Čitanje	Unošenje	Čitanje	Unošenje
Prosječna latencija [µs]	487.1	631.73	2135.34	8526.81
Minimalna latencija [µs]	63.0	191.0	64.0	4768.0
Maksimalna latencija [µs]	105471.0	12279.0	1017343.0	13311.0

(Izvor: Kamil Kolonko, 2018)

Rezultati za radno opterećenje D također prikazuju bolje performanse MongoDB sustava za upravljanje bazama podataka s propusnošću više od pet puta većom od propusnosti sustava Oracle.

4.3.5. Rezultati za radno opterećenje E

Za 1000 operacija u radnom opterećenju pri čemu 95% operacija čine operacije skeniranja, a preostalih 5% operacija čine operacije unošenja, MongoDB daje rezultate propusnosti od 855.43 operacija po sekundi i ukupnog vremena izvođenja svih 1000 operacija od 1169 ms. Za isto radno opterećenje, Oracle ima propusnost od 118.4 operacija po sekundi, a ukupno vrijeme izvođenja svih 1000 operacija jest 4889 ms.

Tablica 8: Latencija za radno opterećenje E

	MongoDB		Oracle	
	Skeniranje	Unošenje	Skeniranje	Unošenje
Prosječna latencija [µs]	988.57	970.32	3357.46	7604.23
Minimalna latencija [µs]	198.0	413.0	207.0	4576.0
Maksimalna latencija [µs]	152831.0	15743.0	620543.0	11591.0

(Izvor: Kamil Kolonko, 2018)

Za razliku od prethodnih radnih opterećenja, u radnom opterećenju E 95% operacija čini skeniranje koje je slično čitanju, ali za razliku od čitanja, ne koristi indekse za dohvaćanje rezultata za upite već analizira cijeli skup podataka. MongoDB i ovdje ima daleko bolje performanse od sustava Oracle, ali upola slabije performanse u odnosu na prethodno radno opterećenje. Oracle također ima slabije rezultate u odnosu na prethodno radno opterećenje.

4.3.6. Rezultati za radno opterećenje F

Za 1000 operacija u radnom opterećenju pri čemu 50% operacija čine operacije čitanja, a preostalih 50% operacija čine operacije čitanje-ažuriranje-unošenje, MongoDB daje rezultate propusnosti od 1191.9 operacija po sekundi i ukupnog vremena izvođenja svih 1000 operacija od 839 ms. Za isto radno opterećenje, Oracle ima propusnost od 118.4 operacija po sekundi, a ukupno vrijeme izvođenja svih 1000 operacija jest 8446 ms.

Tablica 9: Latencija za radno opterećenje F

	MongoDB			Oracle		
	Čitanje	Ažuriranje	Čitanje-ažuriranje-unošenje	Čitanje	Ažuriranje	Čitanje-ažuriranje-unošenje
Prosječna latencija [µs]	434.29	445.24	1073.04	2967.66	8633.52	11010.91
Minimalna latencija [µs]	82.0	147.0	234.0	83.0	4112.0	4312.0
Maksimalna latencija [µs]	155755.0	14495.0	186239.0	565247.0	19663.0	24111.0

(Izvor: Kamil Kolonko, 2018)

U radnom opterećenju F analizira se operacija čitanje-ažuriranje-unošenje koja se koristi za čitanje entiteta u bazi, ažuriranje njegovih podataka te spremanja rezultata u transakciji ažuriranja. Zbog toga svih 1000 operacija vrši čitanje dok 500 operacija uz čitanje, vrši ažuriranje i unošenje. Od svih analiziranih radnih opterećenja, u radnom opterećenju F su najveće razlike u rezultatima. MongoDB za ovo radno opterećenje ima deset puta bolje performanse od sustava Oracle.

5. Optimizacija upita

Kada se govori o bazama podataka i sustavima za upravljanje bazama podataka, poboljšanje upitnih performansi može se postići kupnjom nove opreme s jakim procesorom i velikom količinom memorije kao i dobrim podešavanjem baze podataka koje uključuje dobar dizajn baze podataka i pravilno korištenje indeksa, ali najveći naglasak stavlja se na optimizaciju upita. Pridržavanjem pravila i preporuka kod zadavanja upita, performanse baze podataka mogu se poboljšati do 80%. [8] Optimizacija upita jest postupak pronalaženja i izvođenja najboljeg i najbržeg načina za izvođenje SQL upita (za dohvaćanje podataka), odnosno naredbi (ažuriranje ili brisanje podataka). [4]

5.1. Koraci izvršenja upita

Nakon što klijent pokrene izvođenje upita, prvo se provjerava predmemorija upita. Ako je pronađen odgovarajući upit, rezultati upita dohvaćaju se iz predmemorije. U slučaju da upit nije pronađen u predmemoriji, sljedeći je korak prevođenje, predprocesiranje i optimizacija upita u plan upita. Zatim mehanizam za izvođenje upita izvodi plan upita pozivanjem aplikacijskog programskog sučelja mehanizma za pohranu. Nakon toga, poslužitelj šalje rezultate upita klijentu. [22]

U predmemoriji sprema se tekst upita zajedno s njegovim rezultatom. Predmemorija osjetljiva je na velika i mala slova, stoga za identičan upit poslužitelj vraća rezultate iz predmemorije umjesto izvođenja ostalih koraka izvršenja upita ispočetka. [12]

MySQL-ov prevoditelj (eng. parser) interpretira i provjerava valjanost upita na način da ga rastavlja u tokene od kojih sastavlja stablo raščlambe (eng. parse tree). Osigurava da su tokeni u upitu sintaktički ispravni te u pravilnom redoslijedu. [22]

Predprocesor nastavlja provjeru valjanosti stabla raščlambe koju prevoditelj ne može obaviti kao što su provjera postoje li tablice i stupci u tablicama. Također provjerava aliase zadane u upitu kako bi se osigurala nedvosmislenost referenciranja stupaca. Uz to, predprocesor provjerava i privilegije, odnosno ima li određeni korisnik prava na čitanje ili mijenjanje podataka. [22]

Nakon svih izvršenih provjera, stablo raščlambe je spremno da ga optimizator pretvori u plan izvršenja upita. MySQL-ov optimizator temelji se na predviđanju troškova planova izvedbe upita i odabirom najjeftinijeg. Da bi optimizator napravio optimalan plan upita, vrši optimizacije nad klauzulama koje se mogu podijeliti na statičke i dinamičke optimizacije. Statičke optimizacije su transformacije klauzula u ekvivalentan oblik primjenom algebarskih

pravila (asocijativnost, komutativnost, de Morganove formule, distributivnost). Mogu se izvesti samo jednom, a uvijek će biti valjane. Dinamičke optimizacije su transformacije koje ovise o mnogim čimbenicima kao što su kontekst tablica i broj redaka u tablici. Moraju se izvesti prilikom svakog izvršenja upita. [22]

Odabirom optimalnog plana izvođenja upita, MySQL-ov mehanizam za izvođenje upita koristi odabrani plan upita za izvođenje upita. Da bi izvršio upit, poslužitelj prati naredbe dobivene u planu izvođenja upita na način da poziva metode implementirane u mehanizmu za pohranu. [22]

Zadnji korak jest slanje upita klijentu. Ovaj se korak uvijek izvršava – i kada je upit pronađen u predmemoriji i kada nije, a u slučaju da upit nije valjan klijentu se vraća poruka greške kao rezultat. [22]

5.2. Planiranje upita

Upit nad velikom tablicom može se izvršiti bez čitanja svih redaka te se spajanje više tablica može izvršiti bez kartezijevog produkta (usporedbe svake kombinacija redaka). [12] SQL sustavi za upravljanje podataka imaju integriran optimizator pomoću kojeg se izvodi planiranje upita. [23] Ovisno o strukturi baze podataka, odnosno detaljima tablica i stupaca kao i indeksa, te uvjetima postavljenim u uvjetu upita ili naredbe, optimizator razmatra različite načine izvođenja upita ili naredbe. Plan izvršenja upita je skup operacija koje optimizator odabire za izvođenje najefikasnijeg upita. [12] Plan izvršenja upita poznat je i pod imenom plan objašnjenja (eng. explain plan) i upravo se korištenjem klauzule „EXPLAIN“ dobiju detalji o procijenjenom trošku upita te o vremenu izvršenja.

„EXPLAIN“ klauzula prikazuje plan upita, odnosno način na koji MySQL izvršava naredbe. Kada ju se koristi u kombinaciji s SQL upitom, upit se ne izvršava već se vraćaju koraci potrebni za izvršenje da bi se dobio željeni rezultat. [23] „EXPLAIN“ se može koristiti s naredbama „SELECT“, „DELETE“, „INSERT“, „REPLACE“ i „UPDATE“.

U rezultatu koji kombinacija klauzuli vraća, uključene su i informacije o spajanju tablica te redoslijedu spajanja tablica. Ako se ne koristi redoslijed spajanja zadan u upitu, može se upit započeti klauzulom „SELECT STRAIGHT_JOIN“ koji optimizatoru savjetuje spajanje tablica redoslijedom zadanim u upitu. Ipak, to se ne preporučuje jer „STRAIGHT_JOIN“ može spriječiti korištenje indeksa. Korištenjem „EXPLAIN“ može se vidjeti gdje bi trebalo dodati indekse u tablice da bi se naredba izvršavala korištenjem indeksa što rezultira bržim izvođenjem. [12]

Interpretacija plana upita je složena i zahtijeva se određena praksa kako bi se shvatili dobiveni planovi upita. Kako plan upita sadrži puno podataka, vrlo je važno razumjeti što ti podaci znače, odnosno znače li dobivene vrijednosti da je upit efikasan ili ga je potrebno optimizirati.

5.2.1. Interpretacija planera upita

Da bi se uz procijenjene troškove upita, koje vraća naredba „EXPLAIN“, dobile informacije i o stvarnim troškovima izvođenja individualnih iteratora, koristit će se naredba „EXPLAIN ANALYZE“. [24] Interpretacija rezultata ove naredbe objasnit će se na primjeru upita koji dohvaća ime, prezime i naziv pozicije zaposlenika iz baze podataka za autobusni kolodvor.

```
SELECT ime, prezime, naziv FROM `zaposlenik`
      JOIN osoba ON zaposlenik.osoba=osoba.id
      JOIN pozicija ON zaposlenik.pozicija=pozicija.id
```

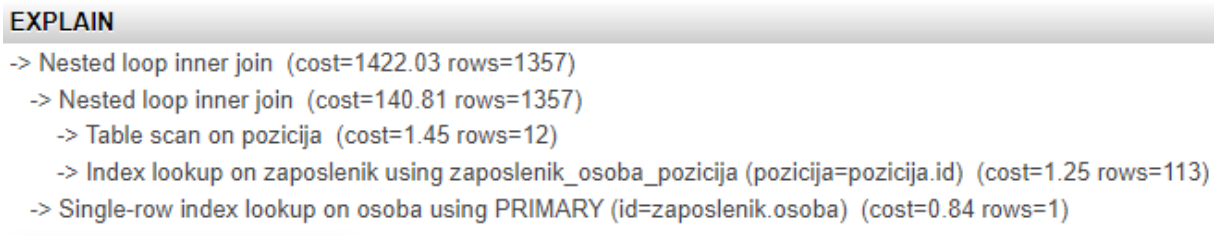
Ovaj upit vraća rezultat od 6 redaka prikazan na sljedećoj slici, a bilo mu je potrebno 0.0062 sekunde da se izvrši.

ime	prezime	naziv
Stjepko	Stjepić	Prodavač/ica karata
3ab08a0841a5760b6cd50fa052d4e3c8	32d3680de02ae27f64bbcc916cfaf8ea	Prodavač/ica karata
d635a4696551ccaf4167263417c355aa	a71feb15e115cfad145016dd6159da76	Prodavač/ica karata
bc2e173802eeb4d2a70e81d34933125b	998ed69809735ede69cd27dfd347b399	Prodavač/ica karata
fe96a31437d2eb71c0d544bb47fe4dc3	a3d7b93ebcae6c992e56343f85209dd2	Prodavač/ica karata
365304f93b42debd9e01539e21a535fc	59a7bab0fb9c783761aa21c43c3abcd4	Prodavač/ica karata

Slika 7: Prvih 6 redaka rezultata upita o zaposlenicima

Korištenjem naredbe „EXPLAIN“ uz parametar „FORMAT“ s vrijednošću „TREE“ dobit će se u rezultatu plan upita zajedno s procijenjenim troškovima i procijenjenim brojem redaka koje upit vraća. Trošak se definira kao vrijeme utrošeno po jedinici. [23] Jednostavni upiti imaju trošak niži od 1000, a srednje „skupi“ upiti od 1000 do 100000 dok skupi upiti imaju trošak veći od 100000.

```
EXPLAIN FORMAT=TREE
SELECT ime, prezime, naziv FROM `zaposlenik`
      JOIN osoba ON zaposlenik.osoba=osoba.id
      JOIN pozicija ON zaposlenik.pozicija=pozicija.id
```



Slika 8: Plan upita o zaposlenicima

Plan upita je vraćen i prvi korak izvršavanja je spajanje tablica, a nakon toga u trećoj je liniji skeniranje indeksa po primarnom ključu iz tablice *zaposlenika*. U zagradi se vidi trošak tog koraka i broj pronađenih redaka. Treći je korak u četvrtoj liniji, a to je traženje primarnih ključeva u tablici *pozicija* prema vanjskim ključevima iz tablice *zaposlenik*. Ta dva koraka pripadaju „JOIN“ operaciji u drugoj liniji koja ima ukupni trošak 2.95 i vraća 6 redaka. Sljedeći je korak traženje primarnih ključeva u tablici *osoba* po vanjskom ključu iz tablice *zaposlenik*.

Ipak, da bi se uz procijenjeni trošak dobio i stvarni trošak, potrebno je koristiti naredbu „EXPLAIN ANALYZE“.

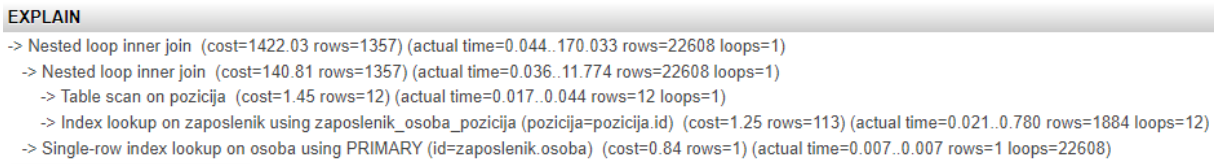
```
EXPLAIN ANALYZE

SELECT ime, prezime, naziv FROM `zaposlenik`

      JOIN osoba ON zaposlenik.osoba=osoba.id

      JOIN pozicija ON zaposlenik.pozicija=pozicija.id
```

Rezultat u svakoj liniji uz zagradu u kojoj je procijenjeni trošak, vraća na kraju linije i zagradu u kojoj je vrijednost stvarno utrošenog vremena za dohvaćanje prvog retka i stvarno utrošenog vremena za dohvaćanje svih redaka uz broj pročitanih redaka i broj petlji izvršenja te linije. Za razliku od troška, stvarno vrijeme se mjeri u milisekundama.



Slika 9: Plan upita o zaposlenicima sa stvarnim utroškom

Može se primijetiti da je fizičko spajanje tablica izvedeno pomoću algoritma ugniježđene petlje spajanja (eng. nested loop join). Uz ugniježđenu petlju spajanja, ostale dvije vrste fizičkog spajanja su hash spajanje (eng. hash join) i spajanje pridruživanjem (eng. merge join). Najčešće se izvodi spajanje pomoću ugniježđene petlje spajanja pri čemu se

čitaju reci iz prve tablice (odabire se ona s manjim brojem redaka) jedan po jedan prosljeđujući pojedini redak ugniježđenoj petlji koja obrađuje drugu (moguće istu) tablicu u spajanju. Hash spajanjem kreira se hash tablica i onda se u njoj pronalaze odgovarajući reci, a koristi se samo kada je ispunjen uvjet jednakog spajanja. Spajanje pridruživanjem dohvaća retke s više simultano skeniranih raspona u tablici i spaja njihove rezultate u jedan. [12]

5.3. SQL izrazi

SQL naredbama izvodi se glavna logika baza podataka, a sljedeće smjernice pomažu ubrzati sve vrste MySQL aplikacija. Dok MySQL automatski radi neke optimizacije kod obrade, bitno je i pridržavati se smjernica kako bi se upite optimiziralo na najbolji mogući način. [12]

5.3.1. Optimizacije koje provodi optimizator

Kao što je već spomenuto, optimizator provodi transformacije upita u drugi upit koji vraća identičan rezultat. Kada mu stigne upit, optimizator izbacuje suvišne uvjete i pretvara ga u čitljiviji oblik. Suvišni uvjeti koji se izbacuju su oni koji su uvijen istiniti (`WHERE 5 = 5`). [12]

Kod jednostavnih operatora usporedbe, optimizator na lijevu stranu operatora stavlja ime stupca, a na desnu vrijednost (u slučaju da se ne uspoređuju dva stupca nego se uspoređuje stupac s lako dostupnom vrijednošću) bez obzira na to kako je u upitu postavljeno pa tako uvjetni izraz `WHERE 30 = sjedeca_mjesta` pretvara u uvjetni izraz `WHERE sjedeca_mjesta = 30`. Jedna od transformacija je i rješavanje algebarskih izrazan pa tako optimizator uvjet `WHERE cijena = 300 + 50` transformira u uvjet `WHERE cijena = 350`. [12]

Osim toga, optimizator koristi tranzitivne transformacije kako bi pojednostavio uvjete tako da se umjesto usporedbe dva stupca i dodatnog uvjeta uspoređivanja jednog od njih s nekom vrijednošću, uspoređuje svaki od ta dva stupca s lako dostupnom vrijednošću. Primjer bi bio izraz `WHERE sjedeca_mjesta = stajaca_mjesta AND stajaca_mjesta = 30` kojem je, nakon tranzitivne transformacije, ekvivalentan izraz `WHERE sjedeca_mjesta = 30 AND stajaca_mjesta = 30`. [12]

Za uvjete raspona, neke izraze optimizator smatra ekvivalentnima i čita ih jednako, iako se koriste različite vrste operatora. Tako optimizator jednako čita izraz `WHERE sjedeca_mjesta IN (40,60,10)` i izraz `WHERE sjedeca_mjesta=40 OR sjedeca_mjesta=60 OR sjedeca_mjesta=10`. Još jedan primjer su izrazi `WHERE`

`sjedeca_mjesta BETWEEN 10 AND 40` i `WHERE sjedeca_mjesta >= 10 AND sjedeca_mjesta <= 40` koje MySQL smatra ekvivalentnima. Optimizator primjenjuje zakone asocijativnosti i komutativnosti kod nizanja uvjeta operatorima „AND“ ili „OR“. Također primjenjuje zakon distributivnosti nad uvjetima. [12]

Ako je moguće napisati uvjet koristeći operatore uspoređivanja umjesto operatora „NOT“, optimizator tada i izvede transformaciju jer se operatori uspoređivanja brže izvode. Primjer bi bio izraz `WHERE NOT cijena > 200` kojeg se pretvara u oblik bez operatora „NOT“: `WHERE cijena <= 200`.

Kod nizanja uvjeta korištenjem operatora „AND“, optimizator prvo provjerava uvjet nad stupcem koji je indeksiran. Ako to nije slučaj, koristi sekvencijalno skeniranje. U slučaju da je u uvjetima više stupaca indeksirano, prvo provjerava onaj stupac za koji je indeks najranije kreiran. [12]

Dok se korištenje vanjskog spajanja tablica klauzulom „OUTER JOIN“ ne preporuča, u nekim slučajevima MySQL optimizator vrši transformacije vanjskog spajanja u unutrašnje spajanje. [22]

5.3.2. Klauzula „WHERE“ i operatori

Za zadavanje uvjeta u upitu koristi se klauzula „WHERE“. Za optimalno postavljanje uvjeta, bitno je izbjegavati suvišne izraze kod nizanja uvjeta i postavljanje lažnih uvjeta. [8] Primjer suvišnih izraza bi bio upit `SELECT * FROM `osoba` WHERE 2+2=4 AND ime='Nola'` u kojem je uvjet `2+2=4` suvišan jer je uvijek istinit pa se umjesto njega može pokrenuti upit `SELECT * FROM `osoba` WHERE ime='Nola'`. Optimizator MySQL sam izbacuje suvišne izraze pa se u planu upita vidi provjera samo uvjetnog izraza `ime='Nola'`.

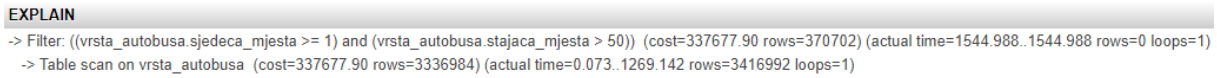
EXPLAIN

```
-> Filter: (osoba.ime = 'Nola')  (cost=329215.82 rows=304905) (actual time=0.108..1887.287 rows=1 loops=1)
  -> Table scan on osoba  (cost=329215.82 rows=3049045) (actual time=0.102..1522.004 rows=3000246 loops=1)
```

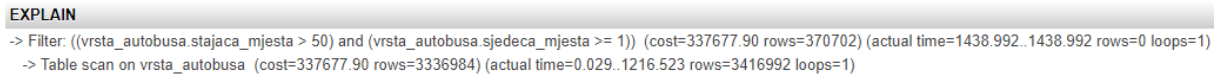
Slika 10: Plan upita s izbačenim suvišnim uvjetom

Također, kako vrijedi zakon komutativnosti, ako je više uvjeta kod kojih se koristi operator „AND“ (svi uvjeti moraju biti istiniti za redak da bi bio vraćen u rezultatu), optimalno je odmah nakon klauzule „WHERE“ navesti uvjet za koji je vjerojatnost da je uvjet istinit najmanja. [8] Dakle, uvjetima se mogu mijenjati mjesta što neće utjecati na rezultat, ali će utjecati na brzinu dohvaćanja rezultata. Primjer bi bio upit `SELECT * FROM vrsta_autobusa`

`WHERE sjedeca_mjesta >= 1 AND stajaca_mjesta > 50` u kojem je uvjet `sjedeca_mjesta >= 1` uvijek istinit, a uvjet `stajaca_mjesta > 50` uvijek neistinit. Ako im se zamijene mjesta (`SELECT * FROM vrsta_autobusa WHERE stajaca_mjesta > 50 AND sjedeca_mjesta >= 1`), upit će se brže izvršiti jer se odmah eliminiraju svi redovi.



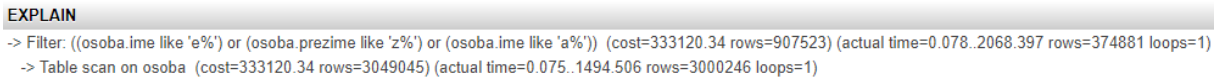
Slika 11: Plan ne-optimiziranog upita s više uvjeta (operator „AND“)



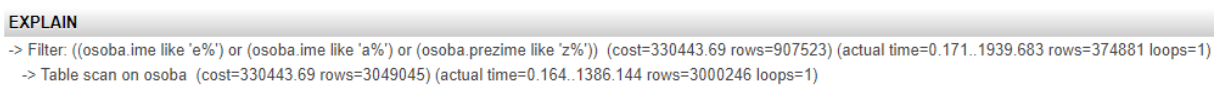
Slika 12: Plan optimiziranog upita s više uvjeta (operator „AND“)

Po istom zakonu komutativnosti, ako se koristi operator „OR“ (bar jedan uvjet mora biti istinit za redak da bi bio vraćen u rezultatu), optimalno je odmah nakon klauzule „WHERE“ navesti uvjet za koji je vjerojatnost da je istinit najveća. [8] U slučaju kada je prvi uvjet istinit, ostali uvjeti se ne moraju provjeravati.

Po zakonu asocijativnosti, uvjeti se mogu i grupirati. Prema tome je bolje grupirati uvjete nad istim stupcima. [8] Može ih se grupirati zagradama radi bolje čitljivosti ili na način da se samo navode jedan iza drugog. Primjer bi bio upit `SELECT * FROM `osoba` WHERE ime LIKE 'e%' OR prezime LIKE 'z%' OR ime LIKE 'a%'` gdje se mogu grupirati uvjeti `ime LIKE 'e%'` i `ime LIKE 'a%'`. Time se dobiva nešto brži upit `SELECT * FROM `osoba` WHERE ime LIKE 'e%' OR ime LIKE 'a%' OR prezime LIKE 'z%'`.



Slika 13: Plan ne-optimiziranog upita s više negrupiranih uvjeta (operator „OR“)



Slika 14: Plan optimiziranog upita s više grupiranih uvjeta (operator „OR“)

Ako se postavlja više uvjeta pomoću logičkog operatora „OR“, njega se može zamijeniti logičkim operatorom „IN“. [8] Tako bi se za upit `SELECT * FROM `vrsta_autobusa` WHERE sjedeca_mjesta=40 OR sjedeca_mjesta=60 OR sjedeca_mjesta=10`, moglo napisati `SELECT * FROM `vrsta_autobusa` WHERE sjedeca_mjesta IN (40,60,10)`. Time se dobivaju sljedeći planovi upita pri čemu je, iako optimizator ove upite čita jednako, upit u kojem se koristi operator „IN“ brže dohvatio rezultate.

```
EXPLAIN
-> Filter: ((vrsta_autobusa.sjedeca_mjesta = 40) or (vrsta_autobusa.sjedeca_mjesta = 60) or (vrsta_autobusa.sjedeca_mjesta = 10))  (cost=339530.25 rows=904323) (actual time=0.190..2740.432 rows=116831 loops=1)
  -> Table scan on vrsta_autobusa  (cost=339530.25 rows=3336984) (actual time=0.187..2194.415 rows=3416992 loops=1)
```

Slika 15: Plan ne-optimiziranog upita s više uvjeta (operator „OR“)

```
EXPLAIN
-> Filter: (vrsta_autobusa.sjedeca_mjesta in (40,60,10))  (cost=337677.90 rows=1001095) (actual time=0.031..1548.778 rows=116831 loops=1)
  -> Table scan on vrsta_autobusa  (cost=337677.90 rows=3336984) (actual time=0.030..1243.460 rows=3416992 loops=1)
```

Slika 16: Plan optimiziranog upita s više uvjeta (operator „IN“)

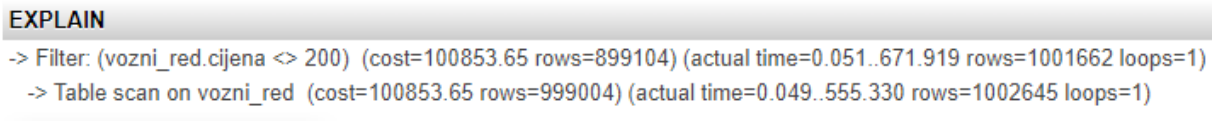
Također, ako se traži vrsta autobusa s brojem sjedećih mjesta između 10 i 40, ne bi trebalo pisati svaku brojku u listu izraza u zagradi, nego iskoristiti logički operator „BETWEEN“, pa bi takav upit glasio `SELECT * FROM `vrsta_autobusa` WHERE sjedeca_mjesta BETWEEN 10 AND 40`.

Osim zakona komutativnosti i asocijativnosti, poželjno je primjenjivati i zakon distributivnosti čime se pojednostavljuju uvjeti pa je lakše izvršiti upit. Primjer bi bio `SELECT * FROM `rezervacija` WHERE vrsta_rezervacije=1 AND (vrsta_putne_karte=6 OR vrsta_putne_karte=1)` kojem je ekvivalentan upit `SELECT * FROM `rezervacija` WHERE (vrsta_rezervacije=1 AND vrsta_putne_karte=6) OR (vrsta_rezervacije=1 AND vrsta_putne_karte=1)`. Ovakve pretvorbe radi i sam optimizator pa je za oba upita vratio plan u kojem provodi prvi navedeni upit.

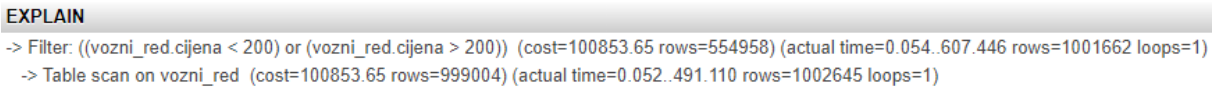
```
EXPLAIN
-> Filter: (rezervacija.vrsta_rezervacije = 1)  (cost=91.41 rows=101) (actual time=0.538..0.946 rows=96 loops=1)
  -> Index range scan on rezervacija using fk_rezervacija_vrsta_karte, with index condition: ((rezervacija.vrsta_putne_karte = 6) or (rezervacija.vrsta_putne_karte = 1))  (cost=91.41 rows=202) (actual time=0.537..0.935 rows=202 loops=1)
```

Slika 17: Plan upita za ekvivalente upite

Bitno je spomenuti i operator različitosti „<>“. Ako je moguće napisati uvjet koristeći druge operatore uspoređivanja umjesto operatora „<>“, onda tako treba i učiniti jer se ostali operatori uspoređivanja brže izvode. Primjer bi bio upit `SELECT * FROM `vozni_red` WHERE cijena <> 200` koji se može zapisati u sljedećem obliku `SELECT * FROM `vozni_red` WHERE cijena <= 200`. Preporuča se pratiti ovu smjernicu jer optimizator još uvijek ne vrši transformacije za ovakve slučajeve. [12]



Slika 18: Plan ne-optimiziranog upita koristeći operator „<>“



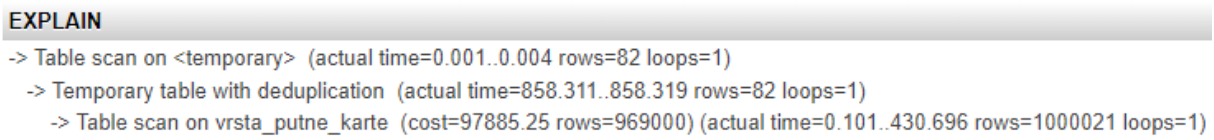
Slika 19: Plan optimiziranog upita izbacujući operator „<>“

Za logički operator „LIKE“, treba izbjegavati korištenje zamjenskog znaka „%“ na početku vrijednosti koju se traži [25], kao što je upit `SELECT * FROM `vrsta_autobusa` WHERE naziv LIKE '%bus'`. Za tablicu koja ima puno podataka te ako su u recima veliki tekstualni podaci, ovakav upit će znatno usporavati performanse.

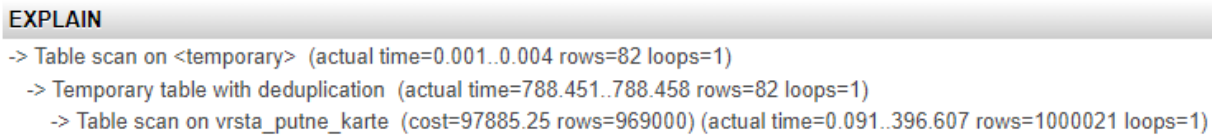
Što se tiče operatora uspoređivanja, najbrže se izvode respektivno „=“, „<“ i „>“, „LIKE“ i „<>“. Bolje performanse imaju upiti u kojem se uvjeti uspoređivanja vrše nad numeričkim podacima od onih koji se vrše nad nizom znakova koji pak imaju bolje performanse od uvjeta uspoređivanja nad dva stupca. [8]

5.3.3. Klauzula „DISTINCT“

Klauzula „DISTINCT“ koristi se za dohvaćanje različitih vrijednosti u stupcu. Za to se može koristiti i klauzula „GROUP BY“, ali ako je moguće vrijednosti dobiti pomoću „DISTINCT“ klauzule, onda nju treba i iskoristiti jer se time postižu bolje performanse. [8] Primjer bi bio upit `SELECT popust FROM `vrsta_putne_karte` GROUP BY popust` u kojem se dohvaćaju sve vrijednosti popusta grupirane po popustu kako ne bi bilo duplikata u rezultatu. Taj se upit može brže izvesti ako se dohvaćaju sve različite vrijednosti popusta i tada upit glasi `SELECT DISTINCT popust FROM `vrsta_putne_karte``. Iako je procijenjeno vrijeme jednako, stvarno vrijeme izvođenja bolje je s korištenjem klauzule „DISTINCT“.



Slika 20: Plan upita koristeći „GROUP BY“

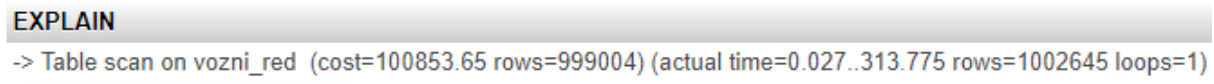


Slika 21: Plan upita koristeći „DISTINCT“

5.3.4. Klauzula „SELECT“

Kod korištenja klauzule „SELECT“, a koristit će se za svaki upit, bitno je navesti nakon nje samo stupce čiji su podaci relevantni. U svakom slučaju, važno je izbjeći dohvaćanje nepotrebnih podataka, a najgori slučaj koji treba izbjeći je dohvaćanje svih podataka, odnosno korištenje „SELECT *“. Dohvaćanjem svih stupaca, stavlja se dodatni teret na bazu podataka što rezultira slabijim performansama. Primjer bi bio dohvaćanje svih podataka, a treba dohvatiti samo cijene putovanja `SELECT * FROM `vozni_red``. U ovom slučaju treba samo dohvatiti stupac cijena jer su ostali stupci redundantni `SELECT cijena FROM `vozni_red``.

Slika 22: Plan ne-optimiziranog upita korištenjem „SELECT *“

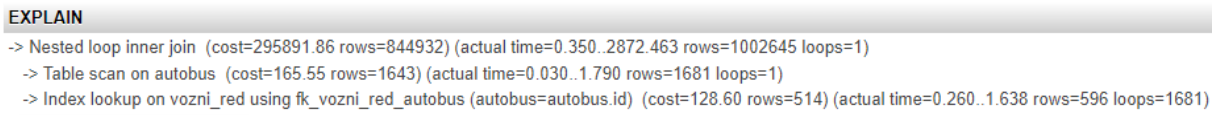


Slika 23: Plan optimiziranog upita korištenjem „SELECT cijena“

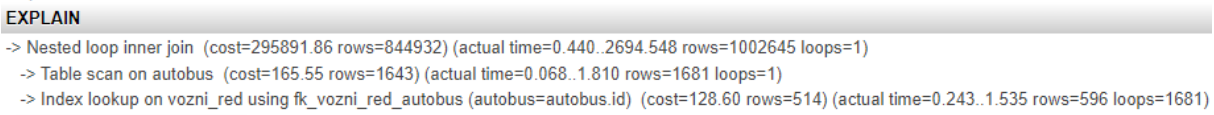
Iako se dohvaća isti broj redova i isti su koraci u planu upita, vidi se da dohvaćanje jednog stupca u „SELECT“ klauzuli puno kraće traje od dohvaćanja svih stupaca.

5.3.5. Klauzula „JOIN“

Kod spajanja tablica može se koristiti stara sintaksa u kojoj se u klauzuli „FROM“ zadaju nazivi tablica koje treba spojiti, a u klauzuli „WHERE“ uvjeti na temelju kojih se spajaju. Primjer stare sintakse jest upit `SELECT * FROM vozni_red, autobus WHERE vozni_red.autobus=autobus.id`. Nova sintaksa uključuje korištenje klauzule „JOIN“ s kojom, osim što pruža više mogućnosti spajanja, utječe na poboljšanje performansi baze podataka. Isti upit, pisan novom sintaksom, bi bio `SELECT * FROM vozni_red JOIN autobus ON vozni_red.autobus=autobus.id`. Ako se pogledaju planovi navedenih upita koji vraćaju iste podatke, a razlika je u korištenju stare i nove sintakse, lako je zaključiti da se korištenjem nove sintakse znatno brže izvode upiti.



Slika 24: Plan upita korištenjem stare sintakse za spajanje tablica



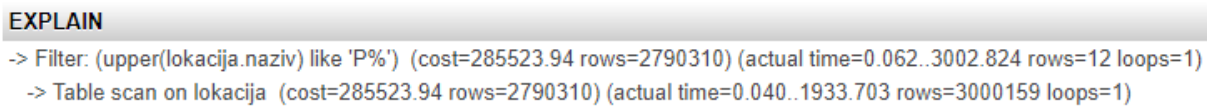
Slika 25: Plan upita korištenjem nove sintakse za spajanje tablica

Osim boljih performansi, korištenje klauzule „JOIN“ pruža više opcija spajanja. To su unutarnje i vanjsko spajanje. Preporuka je koristiti unutarnje spajanje kad god je to moguće i izbjegavanje vanjskog spajanja. Prekomjernim korištenjem vanjskog spajanja znatno se utječe na performanse baze podataka te se sporije izvršava upit.

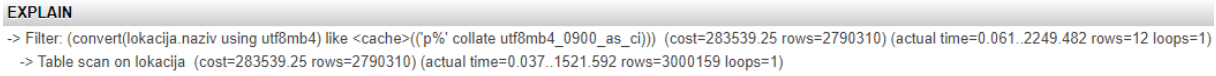
5.3.6. Funkcije nad stupcima

Kod provjere sadrži li stupac određeni tekstualni znak ili znakove, često se koriste funkcije nad stupcima u upitu. Primjer bi bio `SELECT * FROM` `` `lokacija` `` `WHERE UPPER(naziv) LIKE 'P%'`. Korištenje funkcija nad stupcima onemogućava korištenje indeksa nad tim stupcima što usporava performanse, pod uvjetom da indeks nije kreiran nad rezultatom funkcije. U tom slučaju bi trebalo kreirati novi indeks ili novi stupac kako bi se poboljšale performanse. [25] Postoji i drugi način za postavljanje danog upita kako bi se izbjegle funkcije nad stupcima, ali i riješila osjetljivost na velika i mala slova u upitu. Osjetljivost na velika i mala slova se mogu promijeniti u postavkama skupa znakova (eng.

charachter set) za taj stupac ili dodati u upitu za potrebe dohvaćanja rezultata. [12] Takav bi primjer bio `SELECT * FROM `lokacija` WHERE naziv LIKE 'p%' COLLATE utf8mb4_0900_as_ci`.



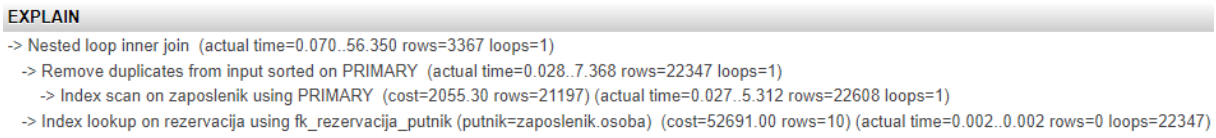
Slika 26: Plan ne-optimiziranog upita s funkcijom nad stupcem



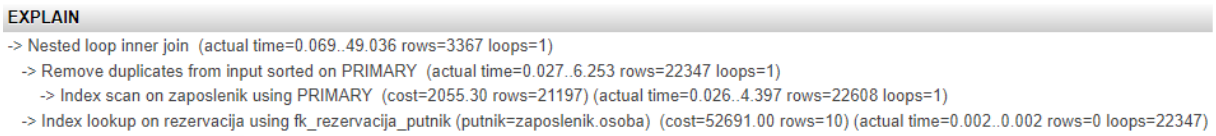
Slika 27: Plan optimiziranog upita s promjenom charset-a

5.3.7. Korelirani podupiti

Korelirani podupit jest podupit koji sadrži referencu na tablicu koja se pojavljuje u vanjskom upitu. [12] MySQL prvo dohvaća rezultate podupita, a zatim vanjskog upita. To se kod koreliranih upita zakomplicira te su zbog toga jako spori. Preporuča se izbjegavanje koreliranih upita. Primjer koreliranog upita jest `SELECT * FROM rezervacija WHERE EXISTS (SELECT * FROM zaposlenik WHERE rezervacija.putnik=zaposlenik.osoba)`. Međutim, i korelirani upiti se mogu optimizirati. Prema tome, bolje rješenje jest `SELECT * FROM rezervacija WHERE putnik IN (SELECT osoba FROM zaposlenik)`.

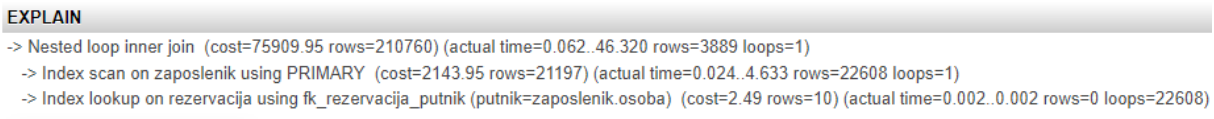


Slika 28: Plan ne-optimiziranog upita s koreliranim podupitom



Slika 29: Plan optimiziranog upita s koreliranim podupitom

Osim što su spori, teško ih je i osmisliti, ali i teško ih je i čitati. Ukoliko je moguće, trebalo bi umjesto koreliranih upita koristiti spajanje tablica klauzulom „JOIN“. Tako bi se isti rezultati kao za prethodni upit dobili upotrebom klauzule „JOIN“, a taj upit jest `SELECT * FROM rezervacija JOIN zaposlenik ON rezervacija.putnik=zaposlenik.osoba`.



Slika 30: Plan optimiziranog upita izbjegavanjem koreliranog upita

6. Zaključak

U ovom je radu dan uvid u faktore koji generalno utječu na performanse baza podataka. Performanse baza podataka širok pojam i ovise o mnogim faktorima – hardverskim mogućnostima, sustavu za upravljanje baza podataka, konfiguraciji baze podataka, dizajnu i arhitekturi baze podataka te o načinu postavljanja upita. I dalje se radi na poboljšanju performansi pojedinog sustava za upravljanje bazama podataka, ali i na alatima kojima se te performanse mogu pratiti. Vrlo je važno kontinuirano pratiti performanse, pogotovo za velike baze podataka jer se na taj način brzo mogu riješiti problemi, ali i otkriti razlog nastanka problema što onda otvara mogućnog otklanjanja uzroka.

Za optimizaciju upita bitno je razumjeti na koji način optimizator sustava za upravljanje bazama podataka određuje plan upita koji će izvesti jer se na taj način može saznati na koji način treba optimalno postaviti upit kojeg će optimizator dodatno optimizirati da bi se upit izveo brže. Budući da se optimizacijom upita znatno mogu poboljšati performanse baza podataka, preporuka je držati se smjernica za optimalno postavljanje upita.

Performanse baza podataka razlikuju se u različitim sustavima za upravljanje bazama podataka. Kako su NoSQL sustavi predviđeni za velike baze s velikom količinom podataka, pokazalo se da imaju i znatno bolje performanse. SQL sustavi za upravljanje bazama podataka nešto su sporiji, ali su stabilni i lako je rukovati podacima u njima. MySQL kao jedan od najpopularnijih sustava za upravljanje bazama podataka ima vrlo dobre performanse.

Popis literature

[1] M. Maleković i K. Rabuzin, *Uvod u baze podataka*. Varaždin: Fakultet organizacije i informatike, Sveučilište u Zagrebu, 2015.
[2] „BAZA PODATAKA: ŠTO JE, VRSTE I PRIMJERI - TEHNOLOGIJA I INOVACIJE - 2022“, *Encyclopedia Titanica*. https://hr.encyclopedia-titanica.com/base-de-datos (pristupljeno 23. lipanj 2022.).
[3] „database“. https://dictionary.cambridge.org/dictionary/english/database (pristupljeno 23. lipanj 2022.).
[4] R. Vujnović, *SQL i relacijski model podataka*. Zagreb: ZNAK, 1995.
[5] „Types of Databases“, *GeeksforGeeks*, 08. svibanj 2020. https://www.geeksforgeeks.org/types-of-databases/ (pristupljeno 23. lipanj 2022.).
[6] „DBMS | Types of Databases - javatpoint“, *www.javatpoint.com*. https://www.javatpoint.com/types-of-databases (pristupljeno 24. lipanj 2022.).
[7] „What Are the Different Types of Databases? | Indeed.com“, *Indeed Career Guide*. https://www.indeed.com/career-advice/career-development/types-of-databases (pristupljeno 23. lipanj 2022.).
[8] K. Rabuzin, *SQL - napredne teme*. Varaždin: Fakultet organizacije i informatike, Sveučilište u Zagrebu, 2014.
[9] „What is a Primary Key? - Definition from Techopedia“, *Techopedia.com*. http://www.techopedia.com/definition/5547/primary-key (pristupljeno 25. lipanj 2022.).
[10] K. Rabuzin, *Uvod u SQL*. Varaždin: Fakultet organizacije i informatike, Sveučilište u Zagrebu, 2011.
[11] „Most popular database management systems 2022“, *Statista*, 23. svibanj 2022. https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/ (pristupljeno 16. lipanj 2022.).
[12] „MySQL“. https://www.mysql.com/ (pristupljeno 25. lipanj 2022.).
[13] „What is MYSQL architecture?“, *Educative: Interactive Courses for Software Developers*. https://www.educative.io/answers/what-is-mysql-architecture (pristupljeno 11. rujan 2022.).
[14] „Architecture of MySQL“, *GeeksforGeeks*, 05. veljača 2021. https://www.geeksforgeeks.org/architecture-of-mysql/ (pristupljeno 11. rujan 2022.).
[15] „MySQL Storage Engine | sistemac.srce.hr“. https://sistemac.srce.hr/node/118 (pristupljeno 11. rujan 2022.).
[16] „SQL | DDL, DQL, DML, DCL and TCL Commands“, *GeeksforGeeks*, 06. studeni 2017. https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/ (pristupljeno 26. lipanj 2022.).
[17] „Defining Database Performance“, *Database Trends and Applications*, 12. listopad 2010. https://www.dbta.com/Columns/DBA-Corner/Defining-Database-Performance-70236.aspx (pristupljeno 30. lipanj 2022.).
[18] „Defining Database Performance | Tanzu Greenplum Docs“. https://gpdb.docs.pivotal.io/6-15/admin_guide/perf_intro.html (pristupljeno 01. srpanj 2022.).
[19] „Database Monitoring: Tips, Metrics and Top 5 Monitoring Tools“, *TekTools*, 06. prosinac 2021. https://www.tek-tools.com/database/database-monitoring-tools-and-tips (pristupljeno 24. srpanj 2022.).
[20] „Database Performance Analyzer (DPA) | SolarWinds“. https://www.solarwinds.com/database-performance-analyzer (pristupljeno 12. rujan 2022.).

[21] K. Kolonko, „Performance comparison of the most popular relational and non-relational database management systems“, Faculty of Computing Blekinge Institute of Technology, 2018.
[22] B. Schwartz, P. Zaitsev, i V. Tkachenko, *High Performance MySQL*, 3. izd. O’Reilly Media, 2012.
[23] U. Malik, M. Goldwasser, i B. Johnston, *SQL za analizu podataka*. Beograd, Srbija: Kompjuter biblioteka, 2019.
[24] „MySQL :: MySQL EXPLAIN ANALYZE“. https://dev.mysql.com/blog-archive/mysql-explain-analyze/ (pristupljeno 28. kolovoz 2022.).
[25] „MySQL Performance Tuning Tips To Optimize Database“, *The Official Cloudways Blog*, 18. ožujak 2019. https://www.cloudways.com/blog/mysql-performance-tuning/ (pristupljeno 31. kolovoz 2022.).

Popis slika

Popis tablica

Prilozi (1)

Prilog 1: suprina_lidija_zavrsni_rad.sql