

**INTERNA STRUKTURA I ORGANIZACIJA INDEKSA KOD MySQL BAZE PODATAKA**

Seminarski rad

Studijski program: Računarstvo i Informatika

Modul: Bezbednost računarskih sistema

Predmet: Sistemi za upravljanje bazama podataka

Student:

Milan Stanković, br. ind. 1407

Niš, april 2022. godina

Mentor:

Prof. Doc. Aleksandar Stanimirović

UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

Univerzitet u Nišu

Elektronski Fakultet

**INTERNA STRUKTURA I ORGANIZACIJA INDEKSA KOD MySQL BAZE PODATAKA**

Master akademske studije

**SEMINARSKI RAD:**

**Studijski program:** Računarstvo i Informatika

**Modul:** Bezbednost Računarskih Sistema

**Predmet:** Sistemi za upravljanje bazama podataka

**Student:** Milan Stanković, br. ind. 1407

**Emali:** milan.mixy.stankovic@elfak.rs

**Mentor:** Prof. Doc. Aleksandar Stanimirović

Niš, april 2022. godina

**SADRŽAJ**

[**1.** **Uvod** 3](#_Toc101564996)

[**2.** **Relacione baze podataka** 4](#_Toc101564997)

[2.1. Organizacija baze podataka 4](#_Toc101564998)

[2.2. DBMS – menadžer baze podataka 5](#_Toc101564999)

[2.2.1. Funkcije upravljanja bazom podataka 6](#_Toc101565000)

[2.2.2. Struktura sistema 6](#_Toc101565001)

[**3.** **MySQL i njegova interna struktura i organizacija indeksa** 7](#_Toc101565002)

[3.1. MySQL 7](#_Toc101565003)

[3.2. Indeksi 7](#_Toc101565004)

[3.2.1. Kako indeksi povećavaju performanse čitanja 8](#_Toc101565005)

[3.2.2. Kako indeksi smanjuju performanse upisa 9](#_Toc101565006)

[3.3. Kako MySQL koristi indekse 9](#_Toc101565007)

[3.4. Tipovi indeksa i indeksiranje kod MySQL-a 10](#_Toc101565008)

[3.4.1. Parcijalni indeksi 10](#_Toc101565009)

[3.4.2. Multicolumn indeksi 11](#_Toc101565010)

[3.4.3. Redosled indeksa 11](#_Toc101565011)

[3.4.4. Indeksi kao ograničenja 12](#_Toc101565012)

[3.5. Struktura indeksa 12](#_Toc101565013)

[3.5.1. B-Tree indeksi (B-Stablo) 12](#_Toc101565014)

[3.5.2. Hash indeksi 13](#_Toc101565015)

[3.5.3. R-Tree indeksi (R-Stablo) 14](#_Toc101565016)

[3.6. Indeksi i tipovi tabela 15](#_Toc101565017)

[3.6.1. MyISAM tabele 16](#_Toc101565018)

[3.6.2. Heap tabele 16](#_Toc101565019)

[3.6.3. BDB tabele 17](#_Toc101565020)

[3.6.4. InnoDB tabele 17](#_Toc101565021)

[3.7. Ograničenja i nedostaci indeksa kod MySQL-a 17](#_Toc101565022)

[**4.** **Zaključak** 20](#_Toc101565023)

[**5.** **Literatura** 20](#_Toc101565024)

# **Uvod**

Relacione baze podataka predstavljaju širok deo upotrebe u svetu informacionih tehnologija. Veliki sistemi koriste upravo relacione baze podataka kako bi čuvale ogromne količine svojih podataka, pa se odatle i javlja potreba za konstantnim razvojem i unapređenjem, zarad poboljšanja samog poslovanja kompanija koje ih koriste. Zbog toga su se razvijale razne tehnike koje su se odnosile na samu unutrašnju organizaciju baza podataka i koje su uticale na način kako će se vršiti interakcija menadžera baze podataka i same baze.

Primarna prednost pristupa relacione baze podataka je mogućnost stvaranja smislenih informacija spajanjem tabela. Spajanje tabela kompanijama omogućava da predstave odnose između podataka ili kako se tabele povezuju. SQL uključuje mogućnost brojanja, dodavanja, grupisanja i kombinovanja upita i sve neophodne operacije koje se koriste za pregled i čitanje podataka. SQL može da obavlja osnovne matematičke i međuzbirne funkcije i logičke transformacije. Analitičari mogu da poređaju rezultate po datumu, imenu ili bilo kojoj koloni. Sve ove funkcionalnosti koriste kompanijama kako bi bolje i efikasnije obrađivale svoje podatke koje aktivno upotrebljavaju u poslovanju.

Glavni fokus kada je upitanju sam razvoj i kod koga se javljaju varijacije kod različitih baza podataka je softver koji je zadužen za upravljanje bazom podataka (DBMS) na kome će se većinom zasnivati ovaj rad. Jedna od najpopularnijih korišćenih baza podataka je MySQL, čija se upotreba prostire od jednostavnih korisničkih projekata pa do velikih industrija (*tzv.* Enterprajs, *eng.* Enterprise), a koja se koristi u raznim delatnostima.

U ovom radu će glavni fokus biti na unutrašnju organizaciju ovog tipa baze podataka. Biće opisani svi koncepti unutrašnje organizacije i arhitekture vezane za održavanje indeksa, kako ih baza upotrebljava i koje su prednosti i nedostaci ovakvog načina organizacije. Sve ove karakteristike su na prvi pogleda sakrivene od korisnika i administratora baze podataka i time se stiče utisak da abstrakcije cele baze podataka ali ipak oni predstavljaju ključnu ulogu kod ovih relacionih sistema.

[1]

# **Relacione baze podataka**

Relaciona baza podataka je kolekcija stavki podataka sa unapred definisanim odnosima između njih. Ove stavke su organizovane kao skup tabela sa kolonama i redovima. Tabele se koriste za čuvanje informacija o objektima koji će biti predstavljeni u bazi podataka. Svaka kolona u tabeli sadrži određenu vrstu podataka, a polje čuva stvarnu vrednost atributa. Redovi u tabeli predstavljaju kolekciju povezanih vrednosti jednog objekta ili entiteta. Svaki red u tabeli može biti označen jedinstvenim identifikatorom koji se zove primarni ključ, a redovi među više tabela mogu biti povezani korišćenjem stranih ključeva. Ovim podacima se može pristupiti na mnogo različitih načina bez reorganizacije samih tabela baze podataka. [2]

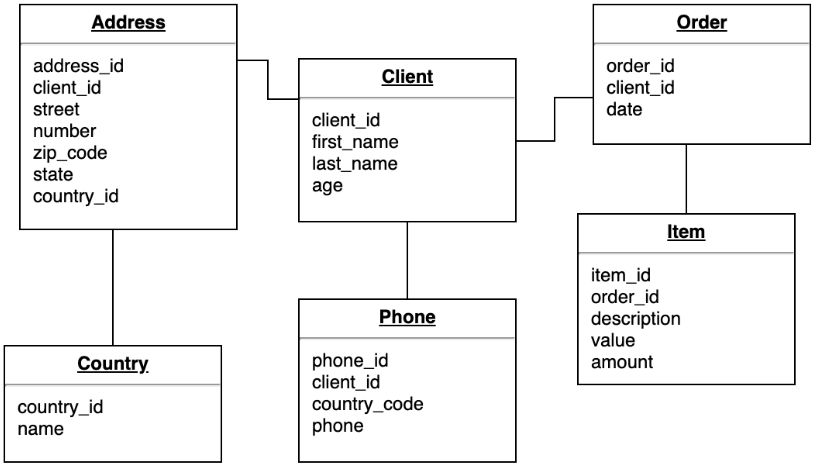
## **Organizacija baze podataka**

Ova organizacija omogućava da se preuzme potpuno novu tabela iz podataka u jednoj ili više tabela sa jednim upitom. Takođe omogućava kompanijama da bolje razume odnose između svih dostupnih podataka i stekne nove uvide za donošenje boljih odluka ili identifikaciju novih mogućnosti. Na primer, zamislite da vaša kompanija održava tabelu klijenata koja sadrži podatke kompanije o svakom nalogu klijenta u jednu ili više tabela transakcija koje sadrže podatke koji opisuju pojedinačne transakcije.

Kolone (ili polja) za tabelu kupaca mogu biti ID klijenta, naziv kompanije, adresa kompanije itd.; kolone za tabelu transakcija mogu biti Datum transakcije, ID klijenta, Iznos transakcije, Način plaćanja, itd. Tabele se mogu povezati na osnovu zajedničkog polja ID klijenta. Prema tome, možete postaviti upit u tabeli da biste proizveli vredne izveštaje, kao što je konsolidovana izjava korisnika.

Generatori izveštaja uzimaju ove upite i pokreću ih na zahtev da bi kreirali formalne izveštaje. Mnogi dokumenti koje kompanije vode da bi pratile zalihe, prodaju, finansije ili čak izvele finansijske projekcije potiču iz relacione baze podataka koja radi iza scene.

[3]



Slika 1. – Prikaz dijagrama šeme baze podataka.

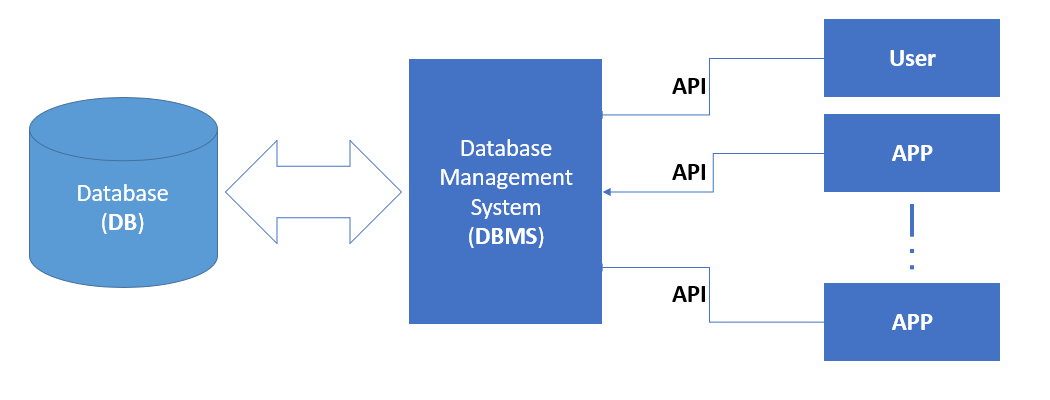
## **DBMS – menadžer baze podataka**

DBMS (DataBase Management System, RDBMS) odnosno sistemi za upravljanje bazama podataka su softverski sistemi koji se koriste za skladištenje, preuzimanje i pokretanje upita o podacima. DBMS služi kao interfejs između krajnjeg korisnika i baze podataka, omogućavajući korisnicima da kreiraju, čitaju, ažuriraju i brišu podatke u bazi podataka. DBMS upravlja podacima, izvršnom jedinicom baze podataka i šemom baze podataka, omogućavajući korisnicima i drugim programima da manipulišu podacima ili ih ekstrahuju. Ovo pomaže u obezbeđivanju bezbednosti podataka, integriteta podataka, istovremenosti i jedinstvenih procedura administracije podataka.

DBMS optimizuje organizaciju podataka prateći tehniku dizajna šeme baze podataka koja se zove normalizacija, koja deli veliku tabelu na manje tabele kada bilo koji od njenih atributa ima redundantnost vrednosti. DBMS nudi mnoge prednosti u odnosu na tradicionalne sisteme datoteka, uključujući fleksibilnost i složeniji sistem rezervnih kopija.

Sistemi za upravljanje bazama podataka mogu se klasifikovati na osnovu različitih kriterijuma kao što su model podataka, distribucija baze podataka ili brojevi korisnika. Najrasprostranjeniji tipovi DBMS softvera su relacioni, distribuirani, hijerarhijski, objektno orijentisani i mrežni.

[4]



Slika 2. – Prikaz uloge DBMS-a i njegova komunikacija.

Izvan sveta profesionalne informacione tehnologije, termin baza podataka se često koristi da se odnosi na bilo koju kolekciju povezanih podataka (kao što je tabela ili indeks kartica) jer zahtevi za veličinu i upotrebu obično zahtevaju korišćenje sistema za upravljanje bazom podataka. I baza podataka i njen DBMS su usklađeni u skladu sa principima određenog modela baze podataka. „Sistem baze podataka“ se odnosi zajedno na model baze podataka, sistem za upravljanje bazom podataka i bazu podataka. [5]

**RDBMS** radi sa operativnim sistemom za implementaciju relacione baze podataka u sistem za skladištenje računara, upravlja korisnicima, omogućava pristup mreži i olakšava testiranje integriteta baze podataka i pravljenje rezervnih kopija.

### **Funkcije upravljanja bazom podataka**

Postojeći DBMS-ovi pružaju različite funkcije koje omogućavaju upravljanje bazom podataka i njenim podacima koji se mogu klasifikovati u četiri glavne funkcionalne grupe:

* **Definicija podataka** – Kreiranje, modifikacija i uklanjanje definicija koje definišu organizaciju podataka.
* **Ažuriranje** – Umetanje, modifikacija i brisanje stvarnih podataka.
* **Preuzimanje** – Pružanje informacija u obliku koji se može direktno koristiti ili za dalju obradu od strane drugih aplikacija. Preuzeti podaci mogu biti dostupni u formi koja je u osnovi ista kao što se čuva u bazi podataka ili u novom obliku dobijenom izmenom ili kombinovanjem postojećih podataka iz baze podataka.
* **Administracija** – Registrovanje i nadgledanje korisnika, sprovođenje bezbednosti podataka, praćenje performansi, održavanje integriteta podataka, bavljenje kontrolom istovremenosti i oporavak informacija koje su oštećene nekim događajem kao što je neočekivani sistemski kvar.

[5]

### **Struktura sistema**

Fizički, serveri baza podataka su namenski računari koji drže stvarne baze podataka i pokreću samo DBMS i srodni softver. DBMS se nalaze u srcu većine aplikacija za baze podataka. DBMS-ovi mogu biti izgrađeni oko prilagođenog jezgra za više zadataka sa ugrađenom podrškom za umrežavanje, ali moderni DBMS-ovi se obično oslanjaju na standardni operativni sistem da bi obezbedio ove funkcije.

Baze podataka i DBMS-ovi se mogu kategorizovati pre modelima baze podataka koje podržavaju, tipovima računara na kojima rade (od klasteri servera do mobilnih telefona), jeziku upita koji se koristi za pristup bazi podataka i njihova interna implementacija koja se odnosi na performanse, skalabilnost, otpornost i bezbednost.

Postoji veliki broj različitih relacionih baza podataka, i pri tome, iako konceptualno rade isto, svaka od njih ima određene prednosti i nedostatke kao i razlike u unitrašnjoj organizaciji koja je sakrivena od strane kranjeg korisnika. Neke od najpoznatijih relacionih baza podataka su:

* Oracle
* Microsoft SQL Server
* PostgreSQL
* Amazon Aurora
* MariaDB
* MySQL

# **MySQL i njegova interna struktura i organizacija indeksa**

U ovom delu biće obrađena struktura indeksa kod MySQL DBMS-a kao i njihova organizacija. Biće opisani različiti tipovi indeksa, načini njihove upotrebe kao i prikaz upotrebe i definicije indeksa na samim primerima.

## **MySQL**

MySQL je sistem upravljanja relacionim bazama podataka otvorenog koda (RDBMS). Njegovo ime je kombinacija reči „Mi“, imena ćerke suosnivača Majkla Videnijusa, i „SQL“.

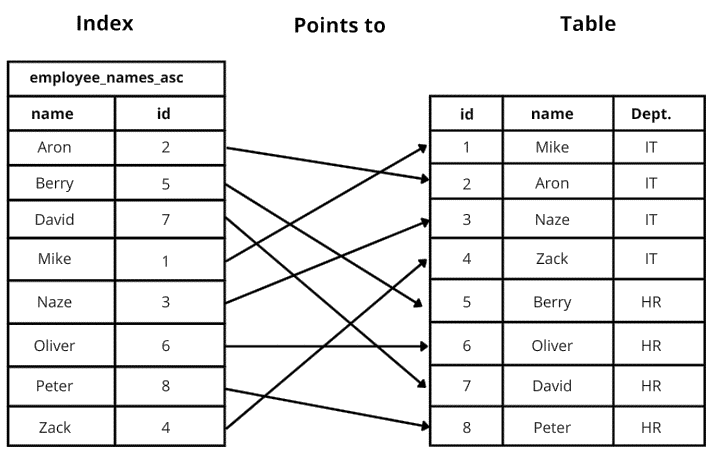
MySQL je besplatan softver otvorenog koda pod upravljanjem i kontrolom GNU Opšte javne licence, a dostupan je i pod raznim vlasničkim licencama. MySQL je bio u vlasništvu i sponzorisan od strane švedske kompanije MySQL AB, koju je kupio Sun Microsystems (sada Oracle Corporation). Kasnije 2010 godine, kada je Oracle kupio Sun, Widenius je razdvojio MySQL projekat otvorenog koda da bi napravio MariaDB.

MySQL ima samostalne klijente koji omogućavaju korisnicima direktnu interakciju sa MySQL bazom podataka koristeći SQL, ali se MySQL češće koristi sa drugim programima za implementaciju aplikacija kojima je potrebna sposobnost relacione baze podataka. MySQL je komponenta softverskog paketa LAMP web aplikacija (i drugih), što je akronim za Linux, Apache, MySQL, Perl/PHP/Pithon. MySQL koriste mnoge veb aplikacije vođene bazama podataka, uključujući Drupal, Joomla, phpBB i WordPress. MySQL takođe koriste mnoge popularne web stranice, uključujući Facebook, Flickr, MediaWiki, Twitter, i YouTube.

[6]

## **Indeksi**

Svrha kreiranja indeksa na određenoj tabeli u bazi podataka je da ubrza pretragu kroz tabelu i pronađete red ili redove koi su zahrtevani. Loša strana je što indeksi usporavaju dodavanje redova ili ažuriranje postojećih redova za tu tabelu. Dakle, dodavanje indeksa može povećati performanse čitanja i smanjiti performanse pisanja. Indeksi se takođe koriste za sprovođenje ograničenja jedinstvenosti.



Slika 3. – Osnovni koncept indeksa kod baze podataka. [8]

### **Kako indeksi povećavaju performanse čitanja**

Kada se radi identifikovanje podatka obično je to kolona id, i obično je cifra koja se povećava sa svakim novim redom u tabeli. Dakle, kada se preuzima određeni red iz baze podataka koristeći njen **ID**, baza podataka ne mora da pretražuje svaki red da bi pronašla onaj koji tražite; podaci su već sortirani i mogu se pretraživati korišćenjem efikasnih algoritama. Čak i ako tabela nije sortirana po indeksima, kolona sa primarnim ključem tabela se automatski indeksira, što znači da postoji kopija te kolone sa sortiranim podacima koje će baza podataka koristiti za pretragu. I taj indeks sadrži pokazivače na stvarne redove u tabeli, tako da kada pronađe tačan **ID** iz tih kopiranih podataka, tačno zna gde da pronađe ostatak informacija za taj red. Indeksi na kolonama koje nisu kolona primarnog ključa rade na isti način. Na primer, recimo da postoji tabela sa člancima u bazi podataka koja sadrži kolone za naslov, telo i public\_at, kao i ID primarnog ključa. U zavisnosti od toga kako je neophodno pretraživati ove podatke u svojoj aplikaciji, možda je optimalno da se dodaju indeksi u neke od ovih kolona kako bi se poboljšale performanse čitanja. Ako aplikacija ima funkciju da pretražujete članke prema njihovom naslovu, možda bi bilo pametno staviti indeks u kolonu naslova. Ovo će stvoriti kopiju te kolone u kojoj su poređani naslovi svih članaka. Možda aplikacija takođe omogućava korisnicima da pregledaju članke iz određenog vremenskog perioda koji oni definišu. Umesto da baza podataka mora da proverava kolonu public\_at svakog reda, može se staviti indeks na tu kolonu i baza podataka će moći lako da pronađe članke koji su objavljeni u tom vremenskom okviru jer će svi biti tik jedan pored drugog. Međutim, čini se nepotrebnim (i pomalo smešnim) stavljati indeks na kolonu tela jer je malo verovatno da će aplikacija ispitivati tabelu članaka po punom sadržaju svakog članka jer za to postoje druge bolje tehnologije.

Indeksi takođe mogu biti korisni za kolone stranog ključa kada se radi o asocijacijama. Recimo da tabela **članaka** takođe sadrži kolonu **author\_id** koja odgovara koloni **id** u tabeli **korisnika**. Ako je indeks na kolonu **author\_id**, kada se pretražuju svi članci određenog autora, rezultati se mogu naći mnogo brže jer će svi članci tog autora biti grupisani zajedno.

### **Kako indeksi smanjuju performanse upisa**

Cena poboljšanja vremena čitanja baze podataka korišćenjem indeksa je što vreme pisanja trpi. Dodavanje novog reda u tabelu bez indeksa je jednostavno. Baza podataka pronalazi sledeći raspoloživi prostor u tabeli za dodavanje novog unosa i dodaje ga, to je to. Međutim, kada dodaje novi red u tabelu sa jednim ili više indeksa, baza podataka dodaje novi unos u tabelu, a zatim mora da doda novi unos u svaki indeks u toj tabeli, pazeći da ubaci unos u ispravno mesto u indeksu da bi se osiguralo da su podaci pravilno sortirani. A ovo smanjenje performansi se odnosi na kreiranje, ažuriranje i brisanje tabele. Iz tog razloga treba izbegavati dodavanje nepotrebnih indeksa u tabele, a indekse koji se više ne koriste treba ukloniti.

Dodavanje indeksa se odnosi na poboljšanje performansi upita za pretragu. Možda je cilj vaše baze podataka da obezbedi skladište podataka u koje se često piše i retko čita. Ako je to slučaj, smanjenje performansi uobičajene operacije, pisanja, verovatno nije vredno povećanja performansi koje dobijate čitanjem. [7]

## **Kako MySQL koristi indekse**

Većina MySQL indeksa (PRIMARY KEY, UNIQUE, INDEX, and FULLTEXT) se čuva u B-stablima. Izuzeci: Indeksi prostornih podataka koriste R-stabla. MEMORY tabele takođe podržavaju heš indekse, InnoDB koristi obrnute liste za FULLTEXT indekse.

MySQL koristi indekse da bi obavio sledeće operacije:

* Da pronađe redove koji se poklapaju sa WHERE delom upita.
* Da eliminiše redove iz razmatranja. Ako postoji izbor između više indeksa, MySQL obično koristi indeks koji pronalazi najmanji broj redova.
* Ako tabela ima indeks sa više kolona, optimizator može koristiti bilo koji krajnji levi prefiks indeksa za traženje redova. Na primer, ako postoji indeks sa tri kolone (kol1, kol2, kol3), moguće kombinacije indeksiranja su (kol1), (kol1, kol2) i (kol1, kol2, kol3).
* Za preuzimanje redova iz drugih tabela prilikom izvođenja spajanja. MySQL može efikasnije da koristi indekse na kolonama ako su istog tipa i veličine. U ovom kontekstu, VARCHAR i CHAR se smatraju istim ako su deklarisani kao iste veličine. Na primer, VARCHAR(10) i CHAR(10) su iste veličine, ali VARCHAR(10) i CHAR(15) nisu.
* Za poređenja između kolona nebinarnih stringova, obe kolone treba da koriste isti skup znakova. Na primer, poređenje kolone utf8 sa kolonom latin1 isključuje upotrebu indeksa.
* Poređenje različitih kolona (poređenje kolone stringa sa vremenskom ili numeričkom kolonom, na primer) može sprečiti upotrebu indeksa ako se vrednosti ne mogu direktno porediti bez konverzije. Za datu vrednost kao što je 1 u numeričkoj koloni, može da se uporedi sa bilo kojim brojem vrednosti u koloni stringa kao što su '1', '1', '00001' ili '01.e1'. Ovo isključuje upotrebu bilo kakvih indeksa za kolonu stringova.
* Da biste pronašli MIN() ili MAX() vrednost za određenu indeksiranu kolonu key\_col. Ovo optimizuje pretprocesor koji proverava da li koristite WHERE key\_part\_N = konstantu na svim ključnim delovima koji se javljaju pre key\_col u indeksu. U ovom slučaju, MySQL traži jedan ključ za svaki izraz MIN() ili MAX() i zamenjuje ga konstantom. Ako su svi izrazi zamenjeni konstantama, upit se vraća odjednom.

**Primer:**



* Za sortiranje ili grupisanje tabele ako se sortiranje ili grupisanje vrši na krajnjem levom prefiksu upotrebljivog indeksa (na primer, ORDER By key\_part1, key\_part2). Ako iza svih ključnih delova stoji DESC, ključ se čita obrnutim redosledom. (ili, ako je indeks opadajući indeks, ključ se čita unapred.)
* U nekim slučajevima, upit se može optimizovati za preuzimanje vrednosti bez konsultovanja redova podataka. (Indeks koji obezbeđuje sve neophodne rezultate za upit naziva se indeks pokrivanja.) ​​Ako upit iz tabele koristi samo kolone koje su uključene u neki indeks, izabrane vrednosti se mogu preuzeti iz stabla indeksa radi veće brzine.

**Primer:**



Indeksi su manje važni za upite u malim tabelama ili velikim tabelama gde upiti izveštaja obrađuju većinu ili sve redove. Kada upit treba da pristupi većini redova, uzastopno čitanje je brže od rada kroz indeks. Sekvencijalno čitanje minimizira pristupe disku / memoriji, čak i ako nisu svi redovi potrebni za upit.

[10]

## **Tipovi indeksa i indeksiranje kod MySQL-a**

### **Parcijalni indeksi**

Indeksi kompenzuju prostor za performanse. Ali ponekad ne želimo da zamenimo previše prostora za performanse koje tražimo. Srećom, MySQL pruža veliku kontrolu nad tim koliko prostora koriste indeksi. Recimo da postoji tabela phone\_book sa 2 milijarde redova u njoj. Dodavanje indeksa na prezime\_name zahtevaće mnogo prostora. Ako je prosečno prezime dugo 8 bajtova, to se svodi na otprilike 16 GB prostora za deo podataka indeksa, pokazivači redova su tu bez obzira šta radite, i dodaju još 4-8 bajtova po zapisu.

Umesto da indeksiramo celo prezimo, možemo indeksirati samo prvih 4 bajta:



Ovim smo smanjili zahtev za prostor indeksa za polovinu, naravno kompromis je što MySQL ne može eliminisati toliko redova koristeći ovaj indeks. primer



Ovo će pročitati sve redove koji imaju prezime **Jones** ali pored toga i **Jonesy** i druge sinonime i slična prezimena, pa će nakon toga izbaciti nepotrebne kolone.

### **Multicolumn indeksi**

MySQL omogućava kreiranje indeksa i po više kolona. Ovo može doneti benefita kada se vrši upit za sve kolone u WHERE klauzuli ili ako jedna kolona nema dovoljno raznovrsnosti. Zajedno sa ovim se mogu koristiti i parcijalni indeksi kako bi se smanjio memorijsko zauzeće.





Ovo nam daje benefit kada želimo detaljniju pretragu, jer je mnogo više redova koji imaju prezime koje počinje sa Wood nego redova kojima prezime počinje sa Wood i ime sa Josh. Ovo značajno povećava broj polja koja će biti eliminisana i time ubrzai upit čitanja.

Kada su upitanju indeksi nad više kolona, oni imaju delove za svaku kolkonu koji se nazivaju deo ključa. Takođe ovi indeksi se nazivaju kompozitni ili složeni. Ovo je dosta bolje od korišćenja posebnog indeksa za obe kolone jer će MySQL uvek koristiti samo jedan od njih.

### **Redosled indeksa**

Za razliku od nekoh drugih RDBMS-a gde postoje različiti načini sortiranja indeksa (rastući, opadajući ili neki drugi redosled), MySQL ne pruža tu vrstu kontrole, već on koristi svoju internu organizaciju i sortiranje indeksa. Od verzije 4.0 pruža jako dobru optimizaciju.

Na primer neke organizacije baze podataka mogu vrlo brzo da izvrše upit koji je označen sa DESC (descending), do će jako sporo izvršavati upit koji označen sa ASC (Ascending). Ovo se dešava zato što neke bazepodataka čuvaju indekse u opadajućem redosledu i to je mnogo pogodnije prilikom čitanja u opadajućem redosledu. Dok u slučaju kada se koristi čitanje u rastućem reosledu, koristi se dva prolaza što je dosta sporije jer se prvo pronalaze svi odgovarajući rezultati i nakon toga se prolazi još jednom kako bi se rezultati sortirali. MySQL je dovoljno inteligentan da odluči kako bi obilazio indekse u obrnutom redosledu kako bi bio brz i kod rastućeg i opadajućeg redosleda čitanja.

### **Indeksi kao ograničenja**

Indeksi se ne koriste uvek za lociranje odgovarajućih redova za upit. Jedinstveni indeks navodi da se određena vrednost može pojaviti samo jednom u datoj koloni. U primeru telefonskog imenika, moguće je kreirati jedinstveni indeks na phone\_number da bi se osiguralo da se svaki telefonski broj pojavljuje samo jednom.



Jedinstveni indeks u ovom slučaju ima dve namene. Prva namena je da funkcioniše kao bilo koji drugi indeks prilikom čitanja na osnovu telefonskog broja. Međutim, takođe proverava svaku vrednost kada pokušava da ubaci ili izmeni zapis kako bi se potvrdilo da vrednost već ne postoji. Na ovaj način indeks funkcioniše kao ograničenje.

Jedinstveni indeksi koriste isto toliko prostora koliko i nejedinstveni indeksi. Vrednost svake kolone kao i lokacija zapisa se čuvaju. Ovo može biti gubitak ako se koriste jedinstveni indeks kao ograničenje, a nikada kao indeks. U ovom slučaju, nema potrebe da MySQL čuva lokacije svakog zapisa u indeksu jer ih nikada neće koristiti.

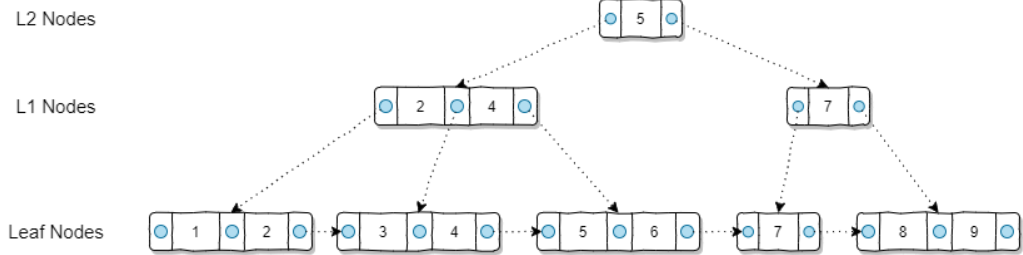
## **Struktura indeksa**

Iako postoje različiti tipovi indeksa, oni nisu konkretno vezani samo za MySQL, već se slični indeksi mogu sresti i kod drugih DBMS-ovima.

### **B-Tree indeksi (B-Stablo)**

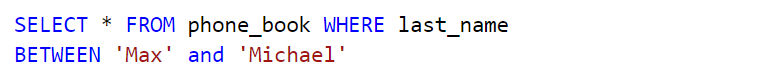
B-stablo, ili uravnoteženo stablo, je najčešći tip indeksa. Praktično svi serveri baza podataka i ugrađene biblioteke baza podataka nude indekse B-stabla, često kao podrazumevani tip indeksa. Obično su podrazumevani zbog svoje jedinstvene kombinacije fleksibilnosti, veličine i opšte dobrih performansi.

Kao što naziv implicira, B-stablo je struktura drveta. Čvorovi su raspoređeni u sortiranom redosledu na osnovu ključnih vrednosti. Za B-stablo se kaže da je uravnoteženo jer nikada neće postati iskrivljeno kako se dodaju i uklanjaju novi čvorovi. Glavna prednost ove ravnoteže je da je učinak B-stabla u najgorem slučaju uvek prilično dobar. B-stabla nude O(log n) performanse za traženje jednog zapisa. Za razliku od binarnih stabala, u kojima svaki čvor ima najviše dva deteta, B-stabla imaju mnogo ključeva po čvoru i ne rastu „visoko“ ili „duboko“ tako brzo kao binarno drvo.



Slika 4. – Prikaz strukture B-stabla. [12]

Indeksi B-stabla nude veliku fleksibilnost kada je potrebno izvršiti upite. Upiti zasnovani na opsegu, kao što su sledeći, mogu se vrlo brzo rešiti.



Server jednostavno pronađe prvi zapis „**Max**“ i poslednji „**Michael**“ zapis. Tada zna da su sve između takođe pogodci. Isto važi i za skoro svaki upit koji uključuje razumevanje opsega vrednosti, uključujući MIN() i MAX(), pa čak i upit otvorenog opsega kao što je sledeći:

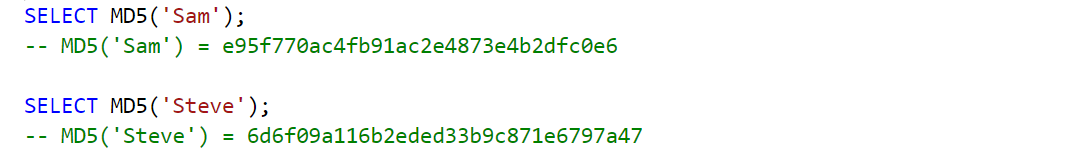


MySQL će jednostavno pronaći poslednji podataka sa vrednošću „**Zinn**“ i prebrojati sve zapise izvan njega u stablu indeksa.

### **Hash indeksi**

Drugi najpopularniji indeksi su zasnovani na heš. Ovi heš indeksi više liče na heš tabelu nego na stablo. Struktura je veoma ravna u poređenju sa drvetom. Umesto da zahtevaju zapise indeksa na osnovu poređenja vrednosti ključa sa sličnim vrednostima ključa, heš indeksi se zasnivaju na rezultatu propustanja svakog ključa kroz heš funkciju. Posao heš funkcije je da generiše polujedinstvenu heš vrednost (obično numeričku) za bilo koji dati ključ. Ta vrednost se zatim koristi za određivanje u kom slotu treba staviti ključ.

Na primer, može se videti MD5 algoritam:



Kada smo kod MD5 algoritma, on generise 128-bitne vrednosti što znači da imamo ogroman broj kombinacija. S obzirom da računari nemaju ni probližno prostora u memoriji za sve slotove, hež adrese tabele su uvek zavisne od raspoložive memorije. Uobičajena tehnika koja smanjuje mogući prostor ključeva heš tabele je dodeljivanje fiksnog broja slotova, često relativno velikog prostog broja kao što je 35,149. Zatim podelite rezultat heš funkcije sa prostim brojem i koristite ostatak da odredite u koju kantu vrednost spada.

Krajnji rezultat je da heš indeks pruža veoma brze pretrage, uglavnom O(1), osim ako se ne bavite heš funkcijom koja ne proizvodi dobar raspored vrednosti za vaše konkretne podatke. Dok indeksi zasnovani na heš-u generalno pružaju neke od najbržih traženja ključeva, oni su takođe manje fleksibilni i manje predvidljivi od drugih indeksa. Oni su manje fleksibilni jer upiti zasnovani na opsegu ne mogu da koriste indeks. Dobre heš funkcije generišu veoma različite heš vrednosti za slične vrednosti, tako da server ne može da pravi nikakve pretpostavke o redosledu podataka unutar strukture indeksa. Pogrešna kombinacija podataka i heš funkcije može dobesti do heš tabele, koja će prouzrokovati slučaj da performanse trpe.

Heš indeksi funkcionišu relativno dobro za većinu tekstualnih i numeričkih tipova podataka. Pošto heš funkcije efikasno smanjuju ključeve proizvoljne veličine na malu heš vrednost, one obično ne koriste toliko prostora kao mnogi indeksi zasnovani na stablu.

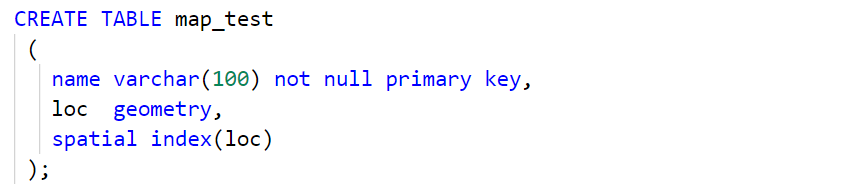
### **R-Tree indeksi (R-Stablo)**

Indeksi R-stabla se koriste za prostorne ili N-dimenzionalne podatke. Oni su prilično popularni u aplikacijama za mapiranje i geonauku, ali podjednako dobro funkcionišu i u drugim situacijama u kojima se zapisi često ispituju na osnovu dve ose ili dimenzije: dužine i širine, visine i težine, itd.

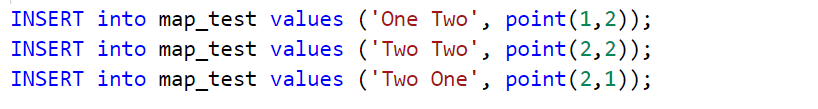
S’ obzirom da su dodati u verziji 4.1, indeksi R-stabla su relativno novi u MySQL-u. Implementacija MySQL-a je zasnovana na OpenGIS specifikacijama. Podrška za prostorne podatke u drugim popularnim serverima baza podataka je često zasnovana na OpenGIS specifikacijama, tako da bi sintaksa trebala biti poznata za sve slične proizvode. Prostorni indeksi možda nisu poznati mnogim dugogodišnjim korisnicima MySQL-a, tako da postoji dosta mogućnosti primene istih.

Napravićemo tabelu koja sadrži prostorne podatke, dodati nekoliko tačaka koristeći koordinate X, Y i pitati MySQL koje tačke spadaju u granice nekih poligona.

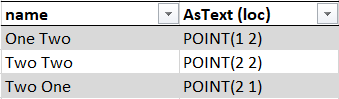
**Kreiranje tabele**:



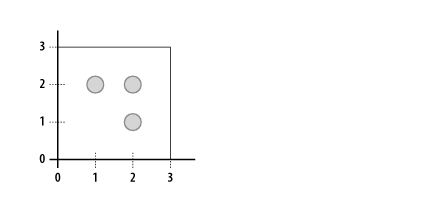
**Popunjavanje tabele:**



**Prikaz podataka:**



**Prikaz tačaka na grafu:**



MySQL indeksira različite oblike koji mogu biti prikazani (tačke, linije, poligoni) koristeći minimalni granični trougao oblika. Da bi se to postiglo, izračunava se najmanji pravougaonik koji je moguće iscrtati a da sadrži oblik. MySQL čuva koordinate tog pravougaonika i koristi ih kada treba da pronnađe oblik zadatog prostora.

## **Indeksi i tipovi tabela (Storage Engine)**

Kada je upitanju indeksiranje kod MySQL-a, postoje različite tabele koje se koriste kako bi se vodila evidencija o indeksima i kako bi se sama tabela uredila. Indeksiranje predstavlja način da se neuređena tabela dovede u red koji će maksimizirati efikasnost upita tokom pretraživanja. Kada je tabela neindeksirana, upit verovatno neće prepoznati redosled redova kao optimizovan na bilo koji način, pa će stoga vaš upit morati da pretražuje redove linearno. Zbog toga postoje različiti izvršni softveri koji implementiraju različite tehnike i mogućnosti indeksiranja.

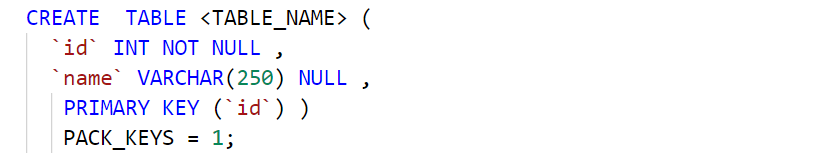
### **MyISAM tabele**

Podrazumevani tip tabele koju obezbeđuje MySQL primenjuje indekse B-stabla, a od verzije 4.1.0 obezbeđuje i indekse R-stabla za prostorne podatke. Pored standardnih prednosti koje dolaze sa dobrom implementacijom B-stabla, MyISAM dodaje još dve važne, ali relativno nepoznate funkcionalnosti kompresije prefiksa i upakovanih ključeva.

**Kompresija prefiksa** se koristi za izdvajanje uobičajenih prefiksa u string ključevima. Na primer, u tabeli koja čuva URL adrese, bilo bi gubljenje prostora za MySQL da uskladišti „http://“ ili „https://“ u svakom čvoru B-stabla. Pošto je konkretan prefiks čest kod velikog broja ključeva, vrši se komprimovanje tog zajedničkog prediksa kako bi zauzimalo manje prostora.

Upakovane ključeve je najbolje smatrati kompresijom prefiksa za celobrojne ključeve. Pošto se celobrojni ključevi prvo skladište sa višim delom bajtova, uobičajeno je da velika grupa ključeva deli zajednički prefiks jer se najviši bitovi broja menjaju mnogo ređe. Omogućavanje upakovanih ključeva se vrši na sledeći način:

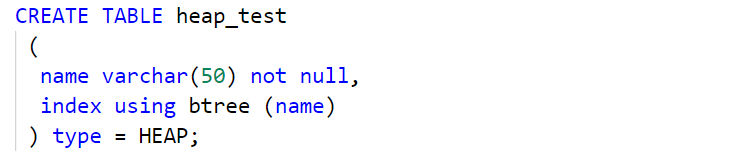




MySQL čuva indekse tabale u „**\*.MYI**“ fajlu tabele. [13]

### **Heap tabele**

Heap jedini MySQL tip tabele u memoriji prvobitno je napravljen sa podrškom samo za heš indekse. Od verzije 4.1.0, međutim, može se birati između B-stabla i heš indeksa u Heap tabelama. Podrazumevano je i dalje da se koristi heš indeks, ali navođenje B-stabla je jednostavno:



Kombinovanjem fleksibilnosti indeksa B-stabla i sirove brzine tabele u memoriji, performanse upita privremenih tabela je teško nadmašiti, ali, ako je potrebno brzo traženje sa jednim ključem, podrazumevani heš indeksi u Heap tabelama će dati dobre rezultate jer su veoma efikasni u prostoru. Podaci indeksa za Heap tabele se uvek čuvaju u memoriji kao i podaci.

### **BDB tabele**

MySQL Berkeley DB (BDB) tabele pružaju samo indekse B-stabla. Ovo može biti iznenađenje za dugogodišnje BDB korisnike koji su možda upoznati sa njegovim osnovnim indeksima zasnovanim na hešovima. Indeksi se čuvaju u istoj datoteci kao i sami podaci.

BDB indeksi, poput onih u MyISAM, takođe obezbeđuju kompresiju prefiksa. Kao i InnoDB, BDB takođe koristi klasterizovane indekse, a BDB tabele zahtevaju primarni ključ. Ako se ne obezbedi, MySKL kreira skriveni podrazumevani primarni ključ koji koristi interno za lociranje redova. Zahtev postoji jer BDB uvek koristi primarni ključ za lociranje redova. Unosi indeksa se uvek odnose na redove koji koriste primarni ključ, a ne na fizičku lokaciju zapisa. To znači da su pretraživanja zapisa na sekundarnim indeksima nešto sporija od pretraživanja primarnog ključa.

### **InnoDB tabele**

InnoDB tabele pružaju indekse B-stabla. Indeksi ne obezbeđuju pakovanje ili kompresiju prefiksa. Pored toga, InnoDB takođe zahteva primarni ključ za svaku tabelu. Međutim, kao i kod BDB-a, ako se primarni ključ ne definiše, MySQL će obezbediti 64-bitnu vrednost.

Indeksi se čuvaju u prostoru tabele InnoDB, baš kao i mapa podataka i podaci (definicije tabela, itd.). Štaviše, InnoDB koristi klasterizovane indekse. Vrednost primarnog ključa direktno utiče na fizičku lokaciju reda kao i na njegov odgovarajući indeksni čvor. Zbog toga su traženja zasnovana na primarnom ključu kod InnoDB veoma brza. Kada se indeksni čvor pronađe, relevantni zapisi će verovatno već biti keširani u InnoDB baferu.

## **Full-Text indeksi**

Full-Text indeks je poseban tip indeksa koji može brzo da pronađe lokaciju svake različite reči u polju. MySQL pruža podršku za indeksiranje celog teksta u MyISAM tabelama. Indeksi punog teksta se grade na osnovu jednog ili više tekstualnih polja (VARCHAR, TEKST, itd.) u tabeli. Full-Text indeks se takođe čuva u *.MYI* datoteci same tabele. Implementira se kreiranjem normalnog dvodelnog MyISAM B-stablo indeksa u kome je prvo polje VARCHAR, a drugo FLOAT. Prvo polje sadrži indeksiranu reč, a FLOAT je njena lokalna težina u redu.

Pošto generalno sadrže jedan zapis za svaku reč u svakom indeksiranom polju, indeksi celog teksta mogu prilično brzo da postanu veliki. Srećom, MySQL B-stablo indeksi su prilično efikasni, tako da je prostor koji zauzima Full-Text indeks veoma isplativ gledano na povećanje performansi.

U ovom slučaju imamo standardan SQL upit koji može biti dosta neoptimalan i spor:



U slučaju kada se koristi Full-Text indeks, može se desiti da se upit izvršava i hiljadu puta brže:



## **Ograničenja i nedostaci indeksa kod MySQL-a**

Mnogo je slučajeva kada MySQL jednostavno ne može da koristi indekse da bi zadovoljio određeni upit. Iz tog razloga neophodno je unapred odkriti sve slučajeve u kojima se može javiti ovaj problem sa nemogućnošću iskorišćavanja indeksa.

**Džoker pretraga (wildcard matching)**

Kada imamo upit koji koristi džoker (*eng.* wildcard) pretragu “bound”:



Ovaj upit mora biti spor, jer MySQL zahteva da skenira svaki red u tabeli, i neće čak ni pronaći sve pojave, jer „bound“ može biti praćeno nekim oblikom interpunkcije, sufiksa ili prefiksa. Rešenje je, naravno, da se napravi Full-Text indeks na polju page\_tekt.

Kada se vrši pretraga sa delimičnim rečima, stvari se brzo komplikuju. Recimo da je potrebno da se pronađe broj telefona za svakoga čije prezime sadrži niz „son“, kao što su Džonson, Ansona ili Bronson. Taj upit bi izgledao ovako:



**Regularni izrazi**

Korišćenje regularnog izraza ima slične probleme. Na primer ukoliko je potrebno pronaći sva prezimena koja se završavaju sa „ith“, kao što je Smith, ili „son“ kao Džonson, prva ideja koja došla u opticaju za rešavanje je regularan izraz. Međutim, taj način će raditi sporo, i to iz istih razloga zbog kojih su pretrage džoker znakova spore. Jednostavno ne postoji uopšten i efikasan način da se izgradi indeks koji olakšava izvršenje upita proizvoljnih džoker znakova ili pretraga regularnih izraza.

Postoje načini za zaobilaženje ovog problema ali nisu najpraktičniji. Ovo se postiže okretanjem prezimena a zatim i okretanjem samog upita, pa bi se tu poboljšanje dobilo samo zato što se vrši pretraga sa početka reči.

**Loša statistika i evidencija indeksa**

Ako statistika internog indeksa MySQL-a postane oštećena ili na neki drugi način netačna (kao rezultat pada ili slučajnog gašenja servera), MySQL može početi da pokazuje veoma čudno ponašanje. Ako je statistika jednostavno pogrešna, može se desiti da više ne koristi indeks za vaš upit. Ili može koristiti indeks samo povremeno. Rezultat ovoga je da MySQL veruje da je broj redova koji odgovaraju upitu toliko visok da bi zapravo bilo efikasnije izvršiti kompletno skeniranje tabele. Pošto su skeniranja tabele prvenstveno sekvencijalna čitanja, ona su brža od čitanja velikog procenta zapisa pomoću indeksa.

**Previše podudarnih redova**

Ako tabela zaista ima previše redova koji se podudaraju sa upitom, performanse mogu biti prilično spore. Koliko je redova previše za MySQL? To zavisi. Ali dobro pravilo je da kada MySQL veruje da se više od oko 30% redova verovatno podudara, izvršiće skeniranje tabele umesto da koristi indeks.

# **Zaključak**

Indeksiranje je moćna struktura kod MySQL-a koja se može iskoristiti da bi se dobila najbrža vremena odgovora na uobičajene upite. MySQL upiti postižu efikasnost generisanjem manje tabele, koja se zove indeks, iz određene kolone ili skupa kolona. Ove kolone, koje se nazivaju ključ, mogu se koristiti za postizanje jedinstvenosti individualnih podataka / unosa. [15]

Korišćenje indeksa za pomoć pri velikim blokovima tabela, podaci mogu imati značajan uticaj na smanjenje izvršavanja MySQL upita i, samim tim, ukupni troškovi resursa sistema. Neindeksirane tabele nisu ništa drugo do neuređene liste; stoga, MySQL izvršni softver ili dbms mora da ih pretražuje od početka do kraja. Ovo može imati mali uticaj kada radite sa malim tabelama, ali može dramatično uticati na vreme pretrage za veće tabele. [16]

U današnjoj primeni relacionih baza podataka, ne samo kod MySQL-a već i svih ostalih veoma je važno voditi računa o performansama, jer same relacione baze kao visoko uređen i organizovan sistem dosta kasne po performansama u odnosu na No-SQL baze podataka, stoga je neophodno koristiti sve moguće načine kako bi se ubrzao celokupan relacioni sistem. Ovde je primena indeksa veoma bitna jer oni nude veoma velika unapređenja u odnosu na standardno postvaljene baze podataka.

Upotreba Indeksa se mora unapred dobro isplanirati kako bi se postigla što veća optimizacija i kako bi same performanse bile dobre, jer ukoliko se oni ne upotrebe na pravi način, umesto ubrzanja i optimizovanja sistema mogu dovesti do suprotnog efektra gde će siste biti dosta sporiji i gde će se uvesti samo dodatno kašnjenje prilikom izvršenja upita.

Krajnja stavka kod uvođenja indeksa je, da jednom kada se ceo sistem postavi i kada se svi indeksi unesu (kada se postigne određena količina podataka koja opravdava korišćenje indeksa) veoma važan korak je i održavanje tih indeksa. U modernim sistemima uključujući i MySQL, sami sistemi vrše održavanje indeksa i njihovo ažuriranje, međutim potrebno je i povremeno vršiti pregled informacija ono indeksiranju u samom MySQL-u kako bi se imao uvid u sistem.

# **Literatura**

[1] [Relational database - Wikipedia](https://en.wikipedia.org/wiki/Relational_database) - <https://en.wikipedia.org/wiki/Relational_database>

[2] [What is a Relational Database? – Amazon Web Services (AWS)](https://aws.amazon.com/relational-database/) - <https://aws.amazon.com/relational-database/>

[3] [Relational Databases Explained | IBM](https://www.ibm.com/cloud/learn/relational-databases) - <https://www.ibm.com/cloud/learn/relational-databases>

[4] [What is Database Management Systems (DBMS)? | AppDynamics](https://www.appdynamics.com/topics/database-management-systems#~1-what-is-dbms) - <https://www.appdynamics.com/topics/database-management-systems#~1-what-is-dbms>

[5] [Database - Wikipedia](https://en.wikipedia.org/wiki/Database) - <https://en.wikipedia.org/wiki/Database>

[6] [MySQL - Wikipedia](https://en.wikipedia.org/wiki/MySQL) - <https://en.wikipedia.org/wiki/MySQL>

[7] [The Basics of Database Indexes For Relational Databases | by Jimmy Farillo | Medium](https://medium.com/@jimmyfarillo/the-basics-of-database-indexes-for-relational-databases-bfc634d6bb37) - <https://medium.com/@jimmyfarillo/the-basics-of-database-indexes-for-relational-databases-bfc634d6bb37>

[8] [Database Index: An Introduction for Beginners (makeuseof.com)](https://www.makeuseof.com/database-index-beginners/) - <https://www.makeuseof.com/database-index-beginners/>

[9] [MySQL :: MySQL 8.0 Reference Manual :: 8.3 Optimization and Indexes](https://dev.mysql.com/doc/refman/8.0/en/optimization-indexes.html) - <https://dev.mysql.com/doc/refman/8.0/en/optimization-indexes.html>

[10] [MySQL :: MySQL 8.0 Reference Manual :: 8.3.1 How MySQL Uses Indexes](https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html) - <https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html>

[11] [4. Indexes - High Performance MySQL [Book] (oreilly.com)](https://www.oreilly.com/library/view/high-performance-mysql/0596003064/ch04.html) - <https://www.oreilly.com/library/view/high-performance-mysql/0596003064/ch04.html>

[12] [1\*a-i\_EluPLupSju9uxCg3GQ.png (820×205) (medium.com)](https://miro.medium.com/max/1400/1*a-i_EluPLupSju9uxCg3GQ.png) - [https://miro.medium.com/max/1400/1\*a-i\_EluPLupSju9uxCg3GQ.png](https://miro.medium.com/max/1400/1*a-i_EluPLupSju9uxCg3GQ.png)

[13] [To learn MySQL PACK\_KEYS with example | CreativeDev (thecreativedev.com)](https://www.thecreativedev.com/to-learn-mysql-pack-keys-with-example/) - <https://www.thecreativedev.com/to-learn-mysql-pack-keys-with-example/>

[14] [Relational Database Indexes and When to Use Them | by Kyle Farmer | Level Up Coding (gitconnected.com)](https://levelup.gitconnected.com/relational-database-indexes-and-when-to-use-them-fb2104cf7af4) - <https://levelup.gitconnected.com/relational-database-indexes-and-when-to-use-them-fb2104cf7af4>

[15] [MySQL Performance: How To Leverage MySQL Database Indexing - Liquid Web](https://www.liquidweb.com/kb/mysql-optimization-how-to-leverage-mysql-database-indexing/) - <https://www.liquidweb.com/kb/mysql-optimization-how-to-leverage-mysql-database-indexing/>

[16] [The Benefits of Indexing Large MySQL Tables | Guidelines for SQL | Drupal Wiki guide on Drupal.org](https://www.drupal.org/docs/7/guidelines-for-sql/the-benefits-of-indexing-large-mysql-tables) - <https://www.drupal.org/docs/7/guidelines-for-sql/the-benefits-of-indexing-large-mysql-tables>