

Univerzitet u Nišu  
Elektronski fakultet Niš



*Seminarski rad*

Sistemi za upravljanje bazama podataka

**Interna struktura i organizacija indeksa kod MySQL baze  
podataka**

Mentor:

Prof. dr Aleksandar Stanimirović

Student:

Svetlana Mančić 1423

Niš, april 2024.

# Sadržaj

1. Uvod	3
2. Interna organizacija i struktura indeksa	5
2.1. Strategije organizacije podataka na disku	5
2.2. Indeksi - funkcija i arhitektura	6
2.3. Klasifikacija indeksa	7
2.4. Struktura indeksa	9
2.5. Poređenje indeksnih struktura u odnosu na uslov selekcije	13
3. Interna organizacija i struktura indeksa kod MySQL	14
3.1. Klasterizovani indeks	17
3.2. Neklasterizovani indeksi	18
3.2.1. Sekundarni indeksi	21
3.2.2. Unique indeks	25
3.3. Fulltext indeks	26
3.4. Prostorni indeks	29
3.5. Adaptive Hash Index	31
4. Zaključak	33
Literatura	34

# 1. Uvod

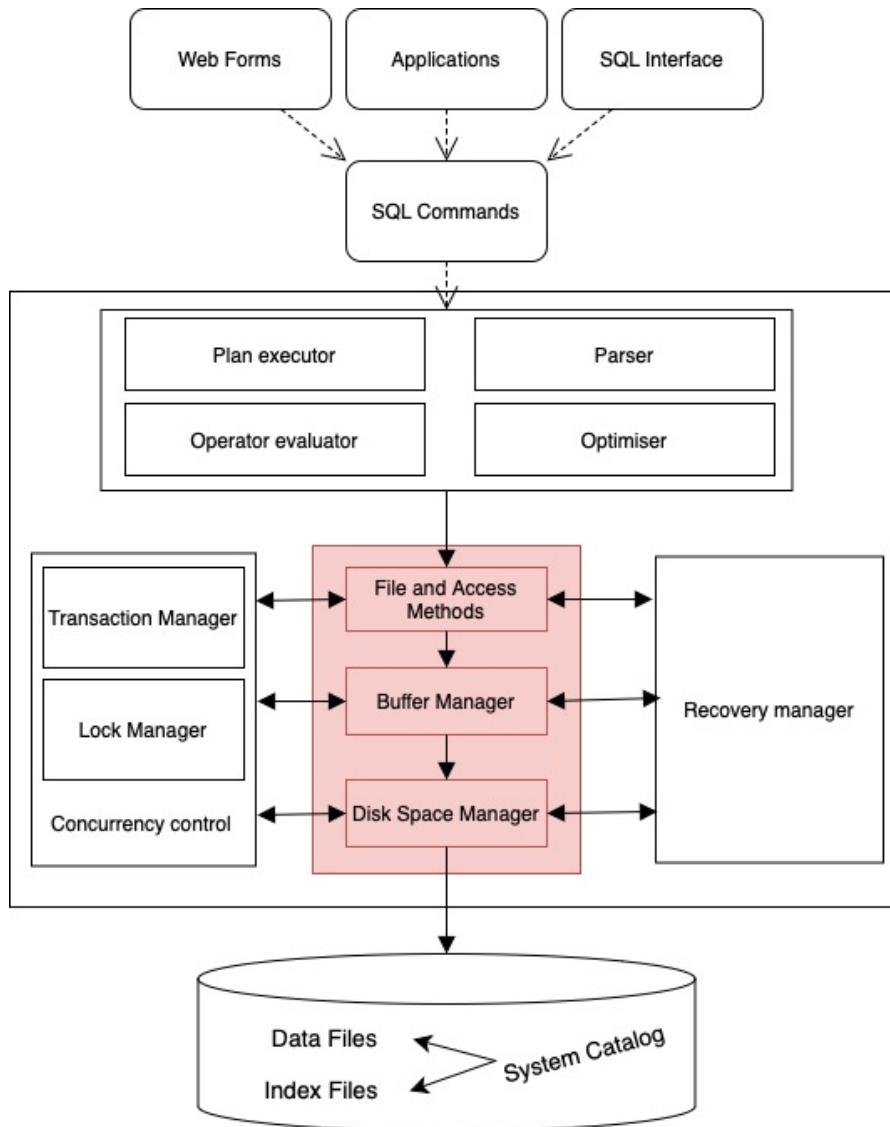
Sistemi za upravljanje bazama podataka (eng. *Database Management Systems - DBMS*) predstavljaju softver za skladištenje ogromne količine podataka i upravljanje istim. Pored toga što pružaju jednostavan način za skladištenje i upravljanje podacima, obezbeđuju njihov integritet i bezbednost, konkurentan pristup podacima, zaštitu u slučaju da dođe do pada sistema i smanjuju vreme potrebno za razvoj aplikacija, koje koriste skladištene podatke [1].

Primarna uloga DBMS je skladištenje ogromne količine podataka, koji moraju da persistiraju nakon završetka programa. Stoga, podaci se čuvaju na spoljnim uređajima i pribavljaju u glavnu memoriju kada je potrebna obrada. Jedinica informacija koja se čita sa spoljnih uređaja i upisuje na spoljne uređaje je **stranica** (eng. *Page*) [1].

Kada govorimo o čuvanju podataka na spoljnim uređajima, izdvajaju se dva tipa uređaja na kojima se podaci mogu skladištiti. Diskovi su eksterni uređaji za skladištenje podataka kod kojih je moguć pristup nasumičnoj stranici po fiksnoj ceni. Jedan od načina da se smanji cena čitanja podataka sa diska jeste njihovo sekvencijalno čitanje, umesto nasumičnog. Drugi tip eksternih uređaja za čuvanje podataka su trake, koje nasuprot diskovima, omogućavaju samo sekvencijalno čitanje stranica. Trake jesu jeftinije od diskova, ali zbog sekvencijalnog čitanja stranica nisu pogodne za aplikacije koje se izvršavaju u realnom vremenu, već se koriste za arhiviranje podataka [1].

Tipičan sistem za upravljanje bazom podataka ima nivovsku arhitekturu. Kako je tema rada interna struktura i organizacija indeksa, u nastavku će pojasniti slojeve arhitekture, koji su zaduženi za organizaciju skladišta podataka i optimizaciju pristupa podacima. Na slici 1, crvenom bojom su obeleženi slojevi, koji imaju pomenutu funkciju. Najniži nivo arhitekture, ujedno i najbliži eksternim uređajima na kojima se skladište podaci je menadžer prostora na disku (eng. *Disk Space Manager*), čija je funkcija rezervisanja prostora, oslobađanje prostora, čitanje sa diska i upis stranica na disk po nalogu viših slojeva arhitekture. Iznad menadžera prostora nalazi se bafer menadžer (eng. *Buffer Manager*), koji predstavlja interfejs između menadžera diska i sloja datoteka, čija je funkcija prebacivanje stranica iz eksternog skladišta u bafer glavne memorije. Troškovi operacija učitavanja podataka u glavnu memoriju i upisivanja iz memorije na disk su veći u odnosu na tipične operacije baze podataka. Stoga, DBMS pažljivo koristi optimizaciju u cilju smanjenja cene ovih operacija. Za ovo je zadužen sloj datoteka i metoda pristupa (eng. *File and Access Methods Layer*), čija je funkcija organizovanje podataka da bi se omogućio brz pristup i

prosleđivanje instrukcija bafer menadžeru kada je neophodno čitanje stranica sa diska ili upis stranica na disk [1].



*Slika 1. Arhitektura sistema za upravljanje bazama podataka*

U drugom poglavlju biće obrazložen način organizacije podataka, koji se skladište na eksternim uređajima. Pored toga, biće reči o načinima za optimizaciju brzine pristupa skladištenim podacima. Biće predložena klasifikacija indeksa. Detaljno će biti opisana struktura indeksa kod sistema za upravljanje bazama podataka.

U trećem poglavlju biće opisana organizacija i interna struktura indeksa kod MySQL servera. Biće predložena klasifikacija indeksa. Indeksi će biti demonstrirani primerima.

Zaključak će biti u četvrtom poglavlju.

## 2. Interna organizacija i struktura indeksa

### 2.1. Strategije organizacije podataka na disku

Podaci kod DBMS-a su organizovani u vidu kolekcije slogova, ili datoteka, gde svaka datoteka ima jednu ili više stranica. Svaki slog identifikovan je jedinstvenim identifikatorom, za koji se koristi termin *record ID* ili kraće *rid*. Ako je poznat *record ID*, DBMS koristi tu informaciju za pronalaženje stranice na disku, koja sadrži slog sa zadatim *rid* [1].

Datoteka je bitna apstrakcija u sistemu za upravljanje bazama podataka i implementira se uz pomoć sloja datoteka. Operacije nad datotekom uključuju kreiranje, uništavanje, upisivanje slogova u datoteku i brisanje slogova iz datoteke. Takođe, podržava operaciju skeniranja, koja omogućava sekvensijalni prolaz kroz sve slogove datoteke. Sloj datoteka čuva kolekciju slogova kao stranice na disku, prati stranice alocirane za svaku datoteku i, s upisivanjem i brisanjem slogova iz datoteke, takođe prati slobodan prostor na alociranim stranicama. Strategija organizacije datoteka predstavlja metod organizacije slogova u datoteci kada se skladišti na disku. Za svaku strategiju postoje situacije kada je idealna za korišćenje, kao i situacije kada nije pogodna. Neke od strategija organizacije su [1]:

- Neuređeni fajlovi (eng. *heap files*)
- Uređeni fajlovi (eng. *sorted files*)
- Indeksi

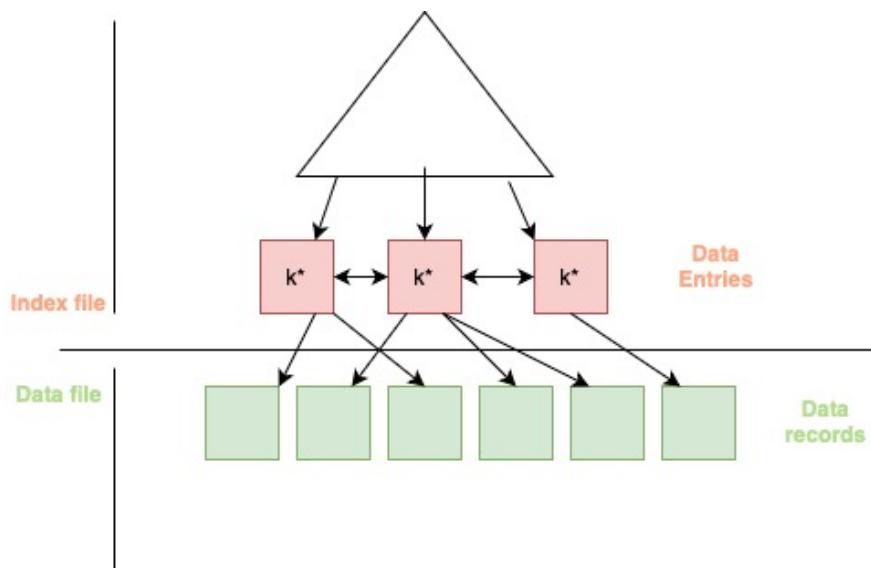
Najjednostavnija strategija organizacije slogova je neuređeni fajl, kod kog se slogovi čuvaju u nasumičnom redosledu na stranicama. Ovakav način organizacije podržava operaciju vraćanja svih slogova ili određenog sloga sa navedenim *rid*. Struktura neuređenog fajla je jednostavna, ali ne pruža način za brzo pronalaženje željenih podataka [1].

Pored neuređenih fajlova, koriste se i uređeni fajlovi, gde se slogovi čuvaju u određenom redosledu. Uređeni fajlovi su pogodni kada je neophodno vratiti slogove u nekom redosledu ili ako se vraća samo skup slogova iz nekog opsega. Jedan od nedostataka sortiranih fajlova jeste visoka cena upisa ili brisanja iz njih, jer je neophodno održati uređeni redosled slogova [1].

## 2.2. Indeksi - funkcija i arhitektura

Indeks je struktura podataka koja organizuje slogove podataka na disku kako bi određene operacije pretraživanja bile optimizovane. Indeks omogućava efikasno nalaženje svih slogova, koji zadovoljavaju određeni uslov pretrage u poljima **ključa traženja** (eng. *search key*) indeksa. Takođe, moguće je kreirati dodatne indekse za datu kolekciju slogova, svaki sa različitim ključem traženja, kako bi se ubrzale operacije traženja, koje nisu efikasno podržane od strane strategije organizacije datoteka, koja se koristi za skladištenje slogova. Ključ traženja može biti bilo koji atribut relacije, deo ili prefiks nekog atributa relacije, koji može biti ključ za pretraživanje indeksa, koji je definisan za relaciju [1].

Za zapise čuvane u indeks fajlu može se koristiti termin *data entry*. **Data entry** sa vrednošću k za ključ traženja, u oznaci  $k^*$ , sadrži dovoljno informacija za lociranje jednog ili više slogova, koji za ključ traženja imaju vrednost k (slika 2). Ovo znači da svaki *data entry* ima pokazivač na jedan ili više redova u tabeli. Svaki *data entry* ima jedinstvenu vrednost za ključ pretraživanja, koji se koristi za indeksiranje [1].



Slika 2. Tipična arhitektura indeksa

Kada se koristi indeks za pretraživanje, prvo se traži odgovarajući *data entry* iz indeksa, koji zadovoljava vrednost za ključ traženja. Kada se pronađe *data entry*, on koristi se za lociranje i čitanje traženih podataka iz tabele.

Postoje tri glavne alternative za čuvanje *data entry* u indeksu [1]:

- *Data entry k\** je zapravo zapis (sa ključem traženja k),
- *Data entry* je  $(k, rid)$  par, gde je *rid record id* zapisa sa ključem traženja k,
- *Data entry* je  $(k, rid-list)$  par, gde je *rid-list* lista *record id*-jeva zapisa sa ključem traženja k.

Ako se indeks koristi za skladištenje zapisa, kao u alternativi 1, svaki *data entry k\** je zapis sa ključem traženja k. Ovakav indeks može se smatrati posebnom strategijom organizacije fajlova, koja se može koristiti umesto uređenog ili neuređenog fajla. Za neku kolekciju podataka može postojati najviše jedan indeks koji koristi alternativu 1, inače dolazi do dupliranja slogova, redundantnosti u slogovima podataka, što potencijalno dovodi do nekonzistentnosti [1].

Alternative 2 i 3, koje sadrže *data entry* koji imaju pokazivač na zapis, su nezavisne od strategije organizacije datoteke. Alternativa 3 nudi bolju iskorišćenost prostora od alternative 2, ali *data entries* su promenljive dužine u zavisnosti od broja slogova, koji imaju vrednost k za ključ traženja [1].

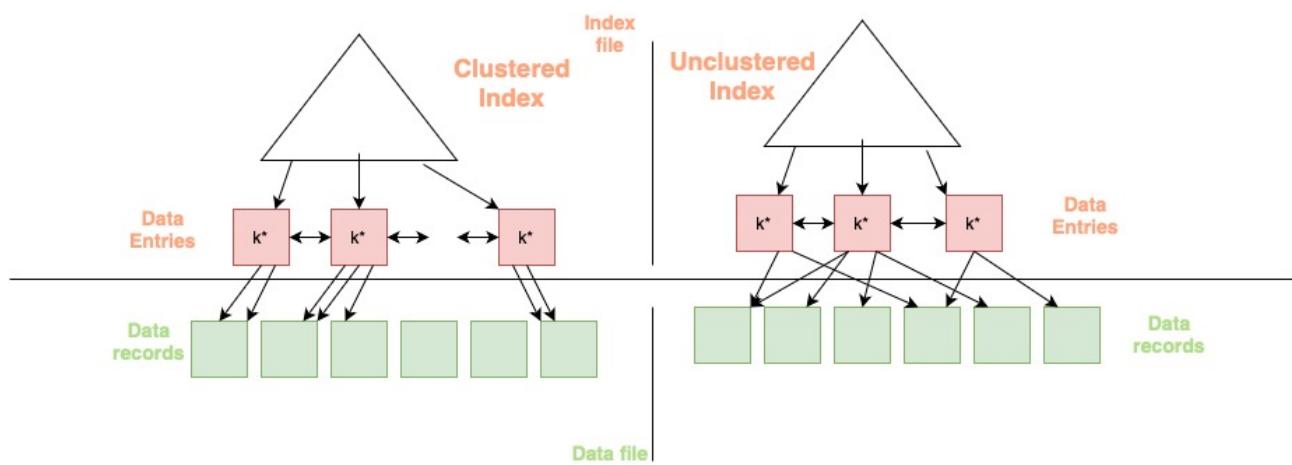
## 2.3. Klasifikacija indeksa

Kada je datoteka organizovana tako da je redosled slogova isti ili sličan kao redosled *data entries* u nekom indeksu, kažemo da je takav indeks **klasterizovan** (eng. *Clustered Index*), u suprotnom je **neklasterizovan** indeks (eng. *Unclustered Index*). Treba naglasiti da cena preuzimanja slogova značajno varira u zavisnosti od toga da li je indeks klasterizovan ili neklasterizovan. Kod klasterizovan indeksa, pošto je datoteka sortirana po ključu traženja, preuzima se samo nekoliko stranica, bez potrebe da se pretraži cela datoteka. Kod neklasterizovanog indeksa slogovi nisu sortirani, tako da može biti potreban onoliki broj I/O operacija koliko ima *data entries* koji pripadaju opsegu traženja [1].

Indeks koji koristi alternativu 1 je klasterizovan po definiciji, jer je redosled slogova u datoteci isti kao redosled *data entries* u indeksu. Indeks koji koristi alternative 2 i 3 može biti klasterizovan samo ako su slogovi sortirani na osnovu polja, koje se koristi kao ključ traženja. U suprotnom, redosled slogova je nasumičan i nema razloga da *data entries* u indeksu budu organizovani na isti način. U praksi, datoteke se retko održavaju u sortiranom redosledu, jer je to preskupo za

održavanje kada se podaci menjaju. Stoga, u praksi, klasterizovani indeks je indeks koji koristi alternativu 1, a indeksi koji koriste alternativu 2 i 3 su neklasterizovani [1].

Na slici 3 je poređenje arhitekture klasterizovanog i neklasterizovanog indeksa. Kao što je ranije opisano, na slici 3 se vidi da je redosled slogova u datoteci isti kao i redosled *data entries* u indeksu, kada je u pitanju klasterizovan indeks. Kod neklasterizovanog indeksa redosled *data entries* i slogova datoteke nije isti.



Slika 3. Arhitektura klasterizovanog i neklasterizovanog indeksa

Indeks nad setom polja koja uključuju primarni ključ naziva se **primarni indeks**, drugi indeksi nazivaju se **sekundarnim**. Ponekad se termini primarni i sekundarni indeks koriste sa drugačijim značenjem: indeks koji koristi alternativu 1 naziva se primarni indeks, a indeks koji koristi alternative 2 i 3 naziva se sekundarni [1].

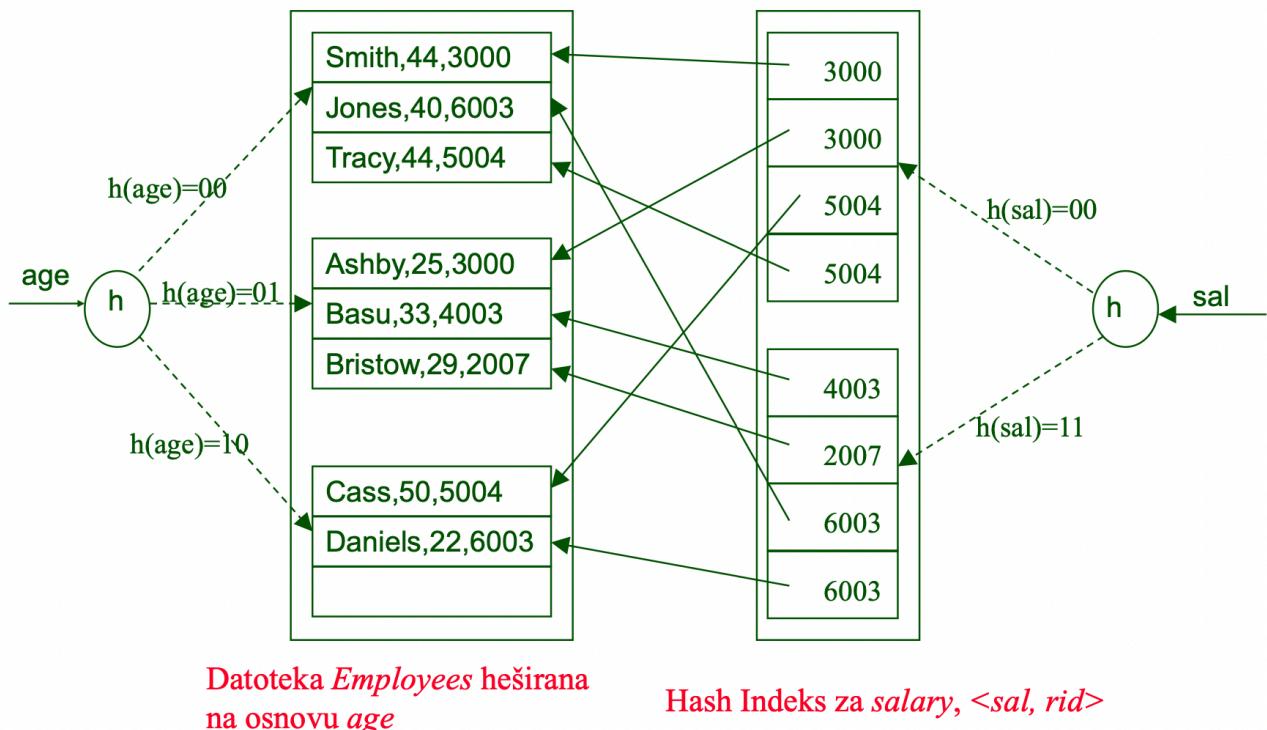
*Data entries* se smatraju duplikatima ako imaju istu vrednost u polju, koje se koristi za ključ traženja povezanih sa indeksom. Primarni indeks garantuje da ne sadrži duplike, ali indeks kreiran nad drugim poljima može sadržati duplike. Generalno, može se smatrati da sekundarni indeks sadrži duplike. Ako znamo da duplikati ne postoje, odnosno da ključ traženja sadrži neko polje koje je kandidat za ključ, takav indeks nazivamo **unique** indeks [1].

## 2.4. Struktura indeksa

Jedan način organizacije *data entries* je heširanje *data entries* ključem traženja, a drugi način organizacije *data entries* je kreiranje strukture nalik stablu, koja usmerava pretragu *data entries* [1].

Slogove možemo organizovati tehnikom heširanja za brzo pronalaženje slogova sa zadatom vrednošću za ključ traženja. Kod ovog pristupa, slogovi se grupišu u *buckets*, gde se *bucket* sastoji od primarne stranice (eng. *primary page*) i, po potrebi, dodatnih stranica (eng. *overflow pages*) povezanih u lanac. Kom *bucket*-u pripada slog odlučuje se primenom heš funkcije. Zadavanjem broja *bucket*-a, ovakva struktura omogućava vraćanje primarne stranice za *bucket* jednom ili dve operacije čitanja [1].

Pri dodavanju, slog se upisuje u odgovarajući *bucket*, uz alociranje dodatnih stranica ukoliko je to potrebno. Za pretragu slogova sa zadatom vrednošću za ključ traženja, primenjuje se heš funkcija za identifikaciju *bucket*-a kom slog pripada i pretražuju se sve stranice u tom *bucket*-u. Ako nemamo vrednost ključa traženja, odnosno ako je indeks baziran na drugom polju, moramo pretražiti sve stranice datoteke [1].

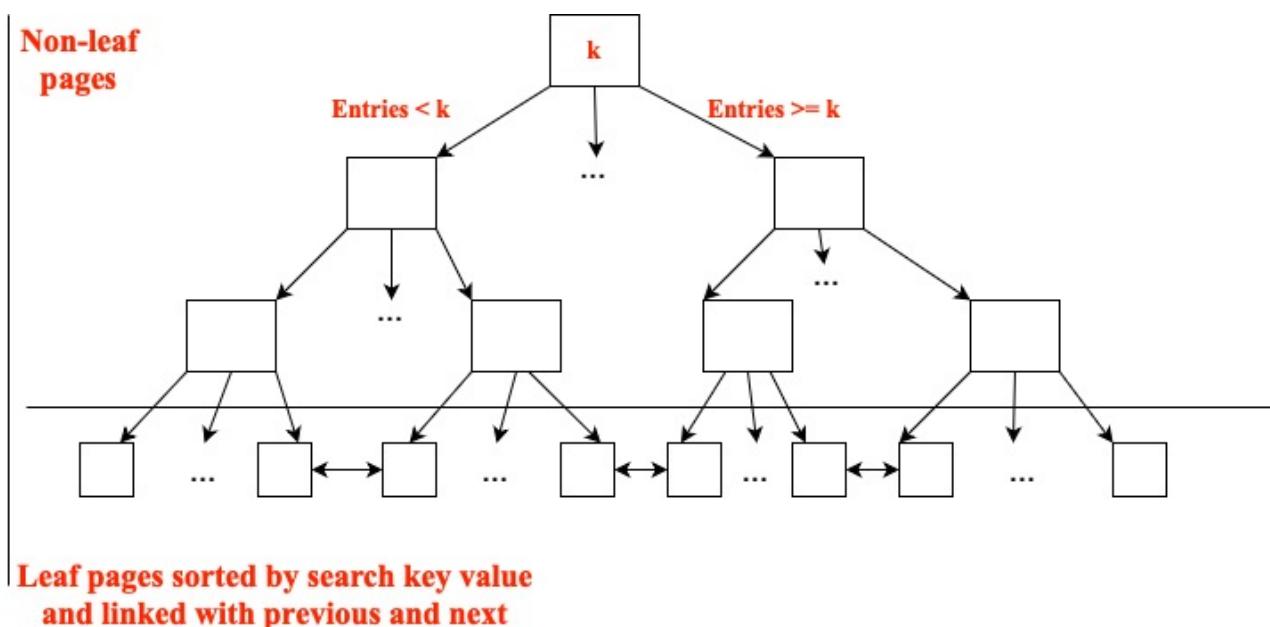


*Slika 4. Struktura hash-based indeksa*

Na slici 4 je primer *hash-based* indeksa. U ovom primeru su podaci čuvani u datoteci heširani na osnovu polja *age*. *Data entries* kod prvog indeksa su slogovi, koji se čuvaju u datoteci. Primenom

heš funkcije na polje *age* identificuje se stranica kojoj slog pripada. Heš funkcija *h* je jednostavna, konvertuje vrednost polja *age* u binarnu reprezentaciju i koristi dva najniža bita za identifikaciju *bucket-a*. Drugi indeks za ključ traženja koristi polje *sal*, a *data entries* su oblika  $\langle sal, rid \rangle$ . Polje *rid* sadrži pokazivač na slogove, koji za ključ traženja imaju vrednost *sal* [1].

Kod strukture stabla, *data entries* su sortirani po vrednosti ključa traženja, i održava se hijerarhijska struktura, koje usmerava pretragu ka odgovarajućim stranicama (slika 5). Pretraga počinje od najvišeg čvora, koji se naziva koren (eng. *root*), i prati se putanja niz stablo dok se ne dođe do traženog sloga. Čvorovi, koji nisu listovi, sadrže pokazivače koji pretragu usmeravaju ka odgovarajućim listovima. Levi pokazivač čvora, koji za ključ traženja ima vrednost *k*, ukazuje na levo podstablo u kom se nalaze *data entries*, koji za ključ traženja imaju vrednost manju od *k*. Desni pokazivač čvora, koji za ključ traženja ima vrednost *k*, ukazuje na desno podstablo u kom se nalaze *data entries*, koji za ključ traženja imaju vrednost veću ili jednaku *k*. Da bi se olakšalo pronalaženje svih listova koji ispunjavaju uslov traženja, stranice su dvostruko ulančane i sadrže pokazivače na prethodnu i sledeću stranicu. Hijerarhijska struktura se održava umetanjem novih *data entries* u stablo kada se slogovi dodaju u datoteku i brisanjem *data entries* iz stabla kada se slogovi brišu iz datoteke [1].



Slika 5. Struktura tree-based indeksa

Najčešći tip indeksne strukture koja se zasniva na stablu je B-stablu. B+ stabla podrazumevaju samobalansirajuća stabla, koja su optimizovana za pristup disku. Dizajnirana su tako da umanjuju broj I/O operacija diska time što su sve putanje od korena do lista jednake. Broj I/O operacija

jednak je visini stabla, odnosno broj operacija jednak je dužini puta od korena do lista plus broj listova koji ispunjavaju uslov pretrage [1].

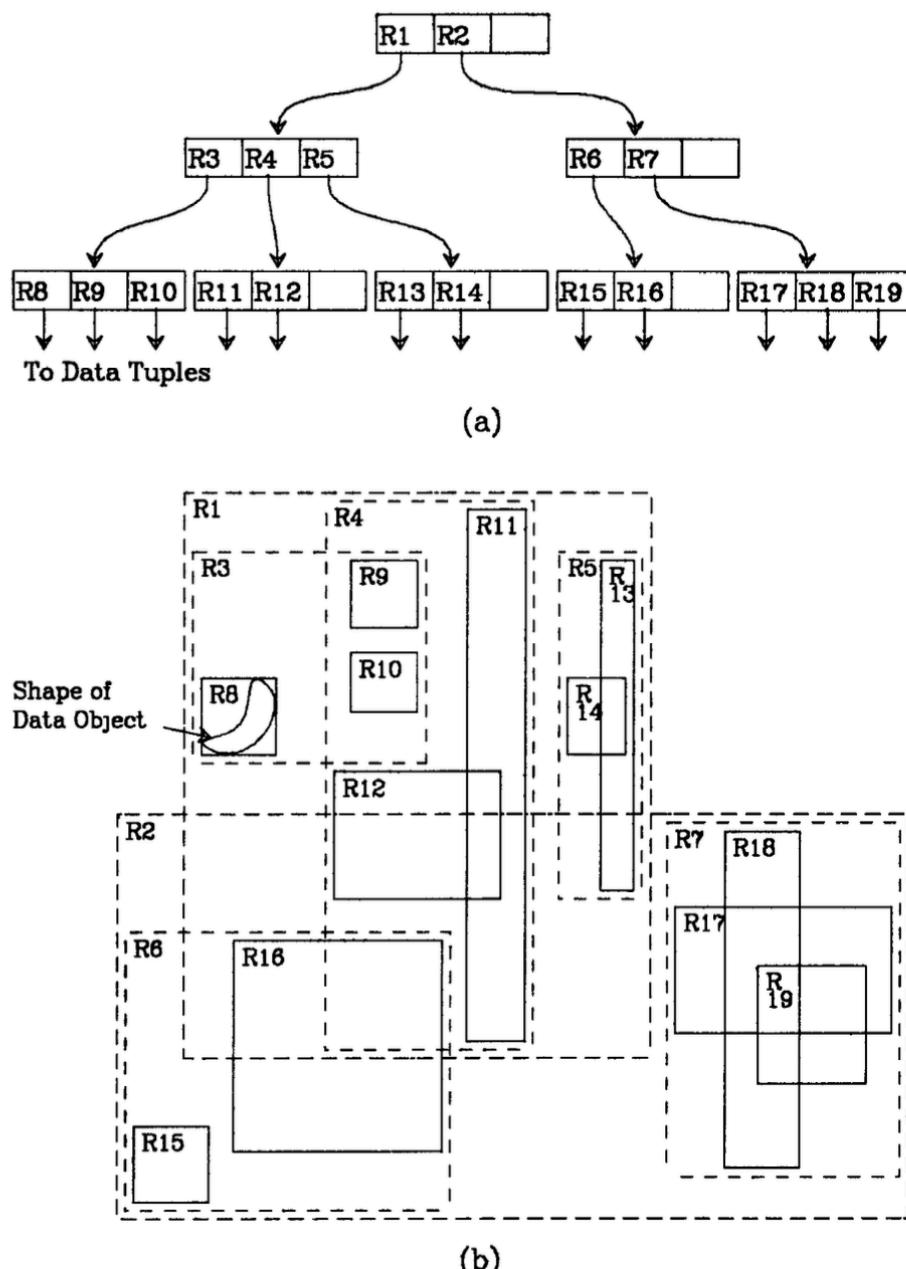
Pored B i B+ stabla, koriste se i R stabla kod prostornih indeksa. R stablo predstavlja hijerarhijsku strukturu podataka baziranu na B+ stablu. Koriste se za dinamičku organizaciju grupe n-dimenzionalnih geometrijskih objekata predstavljajući ih pomoću n-dimenzionalnih pravougaonika minimalnih dimenzija (eng. *minimum bounding n-dimensional rectangles - MBRs*). Svaki čvor R stabla odgovara MBR, koji povezuje njegove potomke. Listovi R stabla sadrže pokazivače na podatke. Čvorovi odgovaraju stranicama na disku. R stablo je struktura dizajnirana tako da prostorna pretraga zahteva posećivanje malog broja čvorova. Indeks, koji ima strukturu R stabla, je potpuno dinamički. Dodavanje i brisanje može biti pomešano sa pretragama, nije potrebna periodična reorganizacija stabla [2].

Listovi R stabla sadrže slogove indeksa u formi ( $I$ , *tuple-identifier*), gde *tuple-identifier* označava prostorne podatke, a  $I=(I_0, I_1, \dots, I_{n-1})$  je n-dimenzionalni pravougaonik, koji sadrži indeksirane prostorne podatke. Čvorovi stabla, koji nisu listovi, sadrže slogove u formi ( $I$ , *child-pointer*), gde *child-pointer* predstavlja adresu čvora R stabla, a  $I$  pokriva sve pravougaonike u sloganima čvora potomka [2].

R stablo reda (m, M) ima sledeće karakteristike [2]:

- Svaki list (osim ako je koren) može imati najviše M zapisa, dok je minimalan broj zapisa  $m \leq M/2$ . Svaki zapis je u formi (*mbr*, *oid*), gde se *mbr* odnosi na oznaku pravougaonika, koji sadrži objekat, a *oid* na identifikator objekta.
- Broj zapisa u čvorovima, koji nisu listovi, je takođe između m i M. Svaki zapis je u formi (*mbr*, *p*), gde je *p* pokazivač na čvor potomak, a *mbr* se odnosi na oznaku pravougaonika, koji sadrži pravougaonike potomka.
- Najmanji broj zapisa u korenu je 2, osim ako nije list. U tom slučaju može imati 0 ili 1 zapis.
- Svi listovi R stabla su na istom nivou.

Na slici 6 pod b je geometrijska reprezentacija objekata. Pravougaonici R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18 i R19 se skladište u listovima R stabla. Na istoj figuri nalaze se i pravougaonici R1, R2, R3, R4, R5, R6 i R7, koji organizuju prethodno nabrojane pravougaonike u čvorovima stabla. Na slici 6 pod a je struktura R stabla, koje predstavlja objekte pod b, ako se uzme da je  $M=3$ , a  $m=2$ . Očigledno je da nekoliko R stabla može predstaviti istu grupu objekata. Rezultujuće R stablo odlučuje se na osnovu redosleda dodavanja ili brisanja zapisa [2].



Slika 6. Struktura R-stabla i geometrijska reprezentacija podataka [2]

Algoritam traženja počinje od korena stabla, slično kao kod B stabla. Međutim, možda više od jednog podstabla trenutnog čvora mora biti ispitano, zbog čega performanse nije moguće tačno predvideti. Ipak, algoritam traženja može eliminisati nerelevantne regije indeksa i ispitati samo podatke koji su blizu traženog područja [2].

## 2.5. Poređenje indeksnih struktura u odnosu na uslov selekcije

Prva stvar koju treba razmotriti pri kreiranju indeksa je očekivano opterećenje i česte operacije. Različite strategije organizacije datoteka i indeksi podržavaju efikasno izvršavanje različitih operacija [1].

Generalno, indeks podržava efikasno vraćanje podataka, koji zadovoljavaju zadati uslov selekcije. Kada govorimo o uslovima selekcije, oni mogu biti jednakost ili opseg. *Hash-based* indeksi su optimizovani samo kada je uslov selekcije jednakost i loše prolaze kod selekcije opsega, gde su obično gori nego skeniranje kompletnog sadržaja datoteke. *Tree-based* indeksi podržavaju oba uslova selekcije efikasno, što objašnjava njihovu široku primenu [1].

I *tree-based* i *hash-based* indeksi podržavaju operacije umetanja, brisanja, ažuriranja efikasno. *Tree-based* indeksi nude superiornu alternativu za održavanje potpuno sortiranih datoteka u vidu B+ stabla. U odnosu na sortirane datoteke *tree-based* indeksi imaju dve glavne prednosti [1]:

- Efikasno podnose umetanje i brisanje *data entries*.
- Pronalaženje odgovarajuće stranice lista indeksa pri pretraživanju slogova po vrednosti ključa traženja je mnogo brže od binarne pretrage stranica u sortiranoj datoteci.

Jedna manja je što stranice sortirane datoteke mogu biti alocirane u fizičkom redosledu na disku, što čini mnogo brže čitanje nekoliko stranica u sekvencijalnom redosledu. Naravno, umetanje i brisanje u sortiranu datoteku su veoma skupe operacije. Varijanta B+ stabla, koja se naziva *Indexed Sequential Access Method (ISAM)*, nudi benefite sekvencijalne alokacije stranica listova, kao i benefite brze pretrage. Umetanje i brisanje nisu efikasni kao kod B+ stabla, ali su mnogo bolji nego kod sortirane datoteke [1].

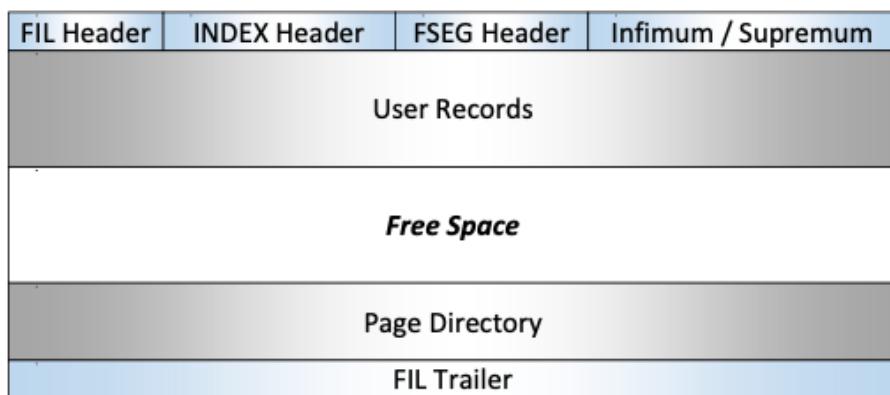
### 3. Interna organizacija i struktura indeksa kod MySQL

MySQL je sistem za upravljanje bazam podataka otvorenog koda, razvijen, distribuiran i podržan od strane Oracle korporacije. Koristi se za relacione baze podataka. Jedan je od najranije razvijenih RDBMS-a otvorenog koda. Koristi SQL jezik za upite. Podrazumevano skladište kod MySQL servera verzije 8.3 je InnoDB, koje balansira između visoke pouzdanosti i visokih performansi [3].

InnoDB koristi dva različita prostora tabele (eng. *tablespace*): ibdataX, koji se koristi za sistemske tabele, i \*.ibd datoteke, koje se koriste za virtuelne tabele i kreira se za svaku tabelu, ako je *file-per-table* aktivran. Ovi tipovi datoteka u osnovi imaju istu strukturu, s tim da sistemske datoteke sadrže dodatne stranice, locirane na fiksnim pozicijama [4].

InnoDB nema posebnu strukturu skladišta podataka, dizajniran je tako da se svi podaci skladište u indeksu. Stoga, kreira B-stablo na osnovu svakog primarnog ključa i skladišti podatke u stranicama indeksa. Veličina jedne stranice je 16KB, a može se smanjiti na 8KB ili 4KB, ili povećati na 32KB ili 64KB. Stranice su grupisane u *extent*-e veličine 1MB za stranice do 16KB (64 uzastopnih stranica od 16KB, ili 128 uzastopnih stranica od 8KB). Datoteke u prostoru tabele kod InnoDB nazivaju se **segmenti**. Za segment se najpre alocira prvih 32 stranica kod InnoDB. Nakon toga, InnoDB počinje da alocira čitave *extent*-e za segment. Može dodati do 4 *extent*-a odjednom za veliki segment kako bi se obezbedio sekvensijalni upis/čitanje podataka iz istog segmenta [4].

Stranice indeksa kod InnoDB skladišta podataka imaju strukturu kao na slici 7. Plavom bojom obeleženi su slogovi na definisanim pozicijama fiksne dužine, a sivom bojom podaci koji se dinamički menjaju. Delovi stranice su *header*, *User Records*, *Page Directory* i *trailer* [5].



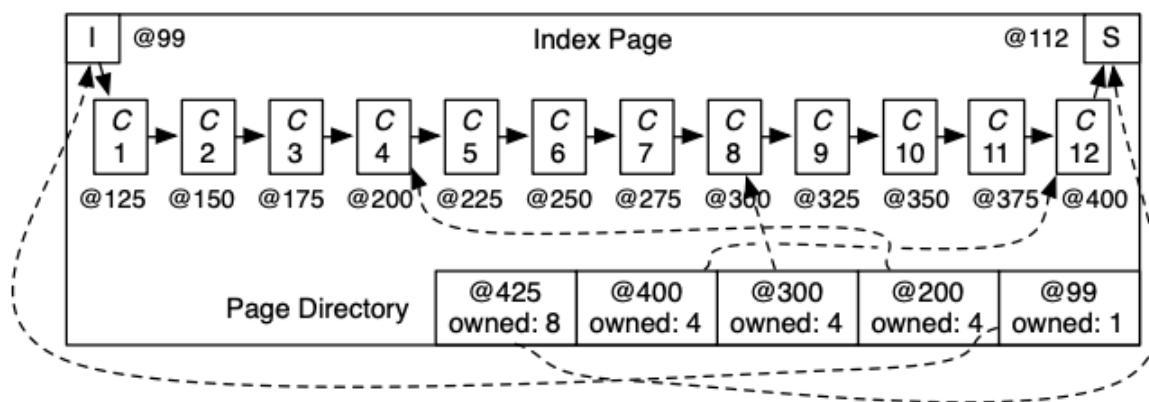
Slika 7. Struktura stranice indeksa kod InnoDB skladišta [5]

*Header* je dužine 120B i ima više delova u kojima se skladište: pokazivači na prethodnu i sledeću stranicu, koji se koriste za navigaciju, podatke za upravljanje indeksom, pokazivač na segment, koji

sadrži podatke indeksa, infimum, koji predstavlja pokazivač na prvi slog indeksa, i supremum, na koji ukazuje poslednji slog indeksa, što se u toku navigacije koristi kao signal da je algoritam traženja došao do kraja stranice [5].

Sekcija *User Records* sadrži slogove stranice. Slogovi se čuvaju kao jednostruka lančana lista, počev od infimuma završavajući se supremumom. Svi slogovi imaju pokazivač na sledeći u rastućem redosledu [5].

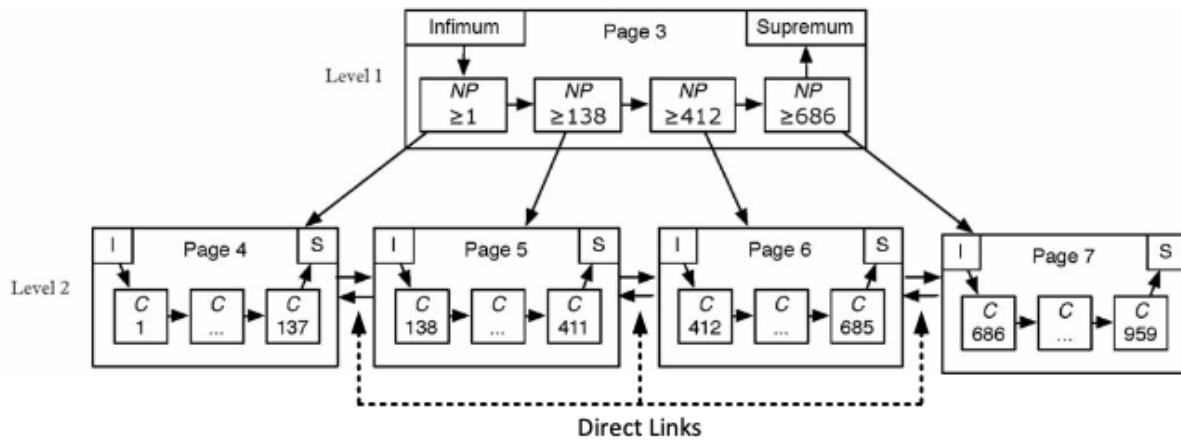
Sekcija *Page Directory* ima fiksni pomeraj od 0x3FF8 bajta, gde počinje *trailer*, i raste ka sekciji *User Records*. Između njih je slobodan prostor stranice. Sadrži ključeve slogova u rastućem redosledu, a broj elemenata definisan je u *header* sekciji. Služi za efikasnu navigaciju kroz stablo. Kako je prolaz kroz stranicu korišćenjem pokazivača na naredni slog skupa operacija, posebno ako stranica ima na stotine slogova, čuva se pokazivač na svaki 4-8 slog da bi se koristilo binarno traženje. Na slici 8 je predstavljen deo stranice, gde se mogu videti putanje kojima se dolazi do određenih slogova stranice. Slovom I označen je infimum, a slovom S supremum. Putanja polazi od infimuma, prolazi kroz slogove podataka, a poslednji slog ukazuje na supremum. Dodatno, *page directory* ima pokazivače na infimum, supremum i svaki četvrti slog na stranici [5].



Slika 8. Logička povezanost slogova na stranici indeksa kod InnoDB [5]

U *trailer* sekciji je *checksum*, koji garantuje integritet podataka [5].

Na slici 9 je delimična struktura indeksa, gde se vidi *root* čvor i listovi. Na osnovu slike može se izvršiti poređenje stranica čvorova i listova. Kod unutrašnjih čvorova stabla, slogovi sadrže pokazivače na čvorove potomke u stablu, dok slogovi listova sadrže podatke. Fizički, slogovi se skladište u redosledu kreiranja, odnosno zauzimaju naredni slobodan prostor u trenutku pisanja, a logički su ulančani uz pomoć *next* pokazivača u rastućem redosledu, što se vidi na slici. Dodatno, stranice listova imaju pokazivače na sledeću stranicu, odnosno prethodnu, preko kojih se dolazi do naredne stranice i proverava da li i ona ispunjava uslov traženja.



Slika 9. Interna struktura i organizacija stranica indeksa [5]

Za svaki indeks kod InnoDB alociraju se dva segmenta, jedan za *nonleaf* stranice B-stabla, a drugi za *leaf* stranice. Održanje neprekidnosti stranica listova na disku omogućava bolje sekvenčialne I/O operacije, jer one sadrže podatke koji se čitaju [4].

InnoDB indeksne strukture su B-stabla, sa izuzetkom prostornih indeksa, koji koriste R-stabla, koja su specijalizovana za indeksiranje višedimenzionalnih podataka. Heš indeksi se kod InnoDB koriste samo interno u određenim slučajevima (*adaptive hash index*) [6].

U osnovi, kod MySQL servera postoje dva tipa indeksa: **klasterizovani** i **neklasterizovani**. Oba koriste strukturu B-stabla. Tabela može imati samo jedan klasterizovani indeks i više neklasterizovanih indeksa. Kod InnoDB skladišta, razlika između ova dva tipa je u tome što se u listovima klasterizovanog indeksa skladište slogovi podataka, a u listovima neklasterizovanog indeksa nalazi se vrednost ključa traženja i vrednost primarnog ključa tabele, na osnovu kog je formiran klasterizvani indeks.

Pored klasterizovanih i neklasterizovanih indeksa, kod MySQL servera postoje i *full-text* indeksi i prostorni indeksi. U tabeli 1 prikazana je klasifikacija indeksa na osnovu strukture.

Struktura indeksa	Tip indeksa
<b>B-stablo</b>	Klasterizovani indeks
	Neklasterizovani indeksi
<b>Invertovani indeks</b>	Full-text indeksi
<b>R-stablo</b>	Prostorni indeksi
<b>Heš tabela</b>	Adaptive Hash Index

*Tabela 1. Klasifikacija indeksa na osnovu strukture skladišta*

U nastavku će detaljnije opisati gore navedene tipove indeksa, njihovu strukturu i upotrebu kod MySQL servera.

### 3.1. Klasterizovani indeks

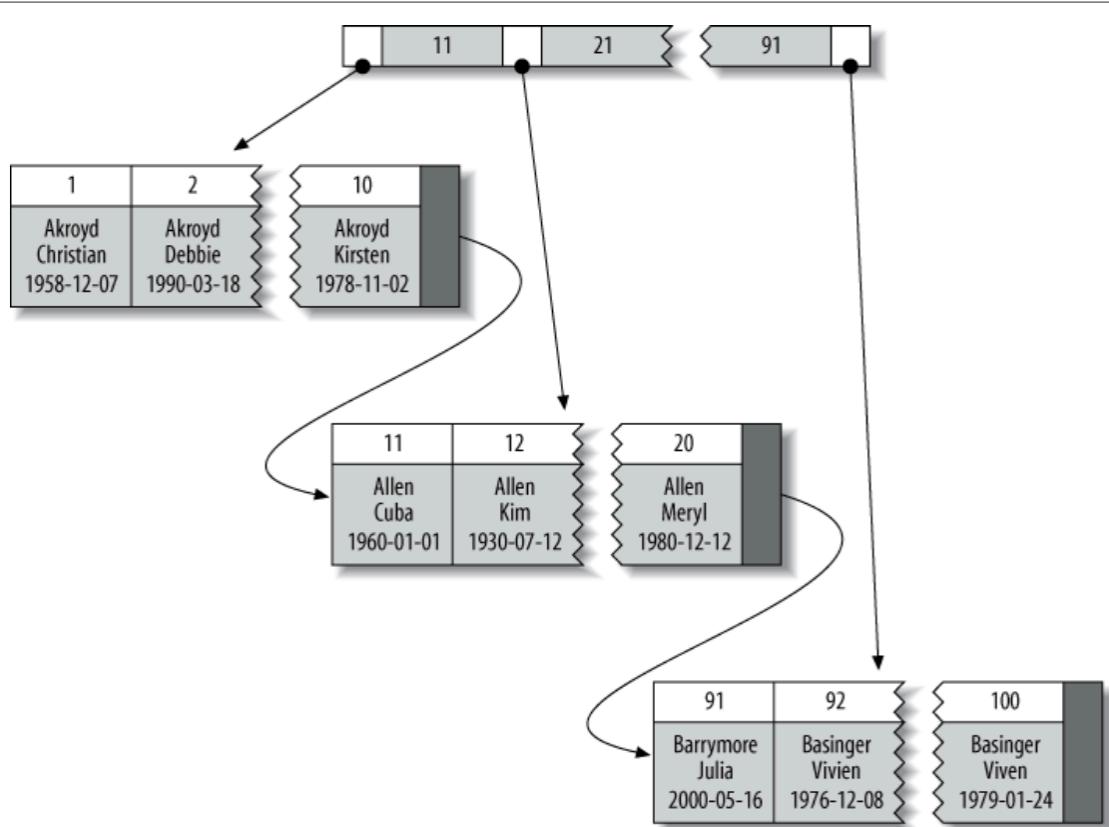
Iako se klasificuje kao indeks, klasterizovani indeks je više način skladištenja podataka. Kod InnoDB skladišta, redovi tabele se ne skladište u posebnim stranicama, već upravo u stranicama listova klasterizovanog indeksa (slika 10). Klasterizovani indeks kod InnoDB ima strukturu B-stabla. Moguće je imati samo jedan klasterizovni indeks po tabeli, jer nema smisla skladištiti podatke na dva mesta [7].

Svaka InnoDB tabela ima poseban indeks, klasterizovani indeks, koji čuva redove podataka iz tabele. On se kreira automatski, indeksiranjem podataka na osnovu primarnog ključa. Ukoliko nije definisan primarni ključ tabele, InnoDB će pokušati da koristi *unique nonnullable* indeks umesto njega [8]. Ako tabela nema primerni ključ ni pogodan *UNIQUE* indeks, InnoDB generiše skriveni klasterizovani indeks, *GEN\_CLUST\_INDEX*, na sintetičkoj koloni koja ima vrednosti *row ID*. *Row ID* je 6 bajtno polje, koje monotno raste s dodavanjem novih redova u tabelu. Stoga, redovi uređeni po redosledu dodavanja [7].

Kako se u praksi najčešće definiše PRIMARY KEY za svaku tabelu pri kreiranju, klasterizovani indeks može se smatrati sinonimom za primarni indeks [7].

Pristup redovima kroz klasterizovani indeks je brzo, jer pretraga indeksa vodi direktno do stranice, koja sadrži redove podataka. Ako je tabela velika, arhitektura klasterizovanog indeksa često zahteva manje I/O operacija diska u poređenju sa organizacijom skladišta, koja ne čuva redove podataka u stranicama indeksa, već u zasebnim stranicama [7].

Na slici 10 delom je prikazana struktura klasterizovanog indeksa kod InnoDB skladišta. Ono što se može primetiti jeste da se u stranicama listova indeksa skladište kompletni redovi tabele, dok stranice ostalih čvorova stabla sadrže samo vrednosti indeksiranih kolona. U ovom primeru indeksirane kolone imaju celobrojne vrednosti. Takođe, na slici se vidi da stranice imaju pokazivače na sledeću stranicu, tako da je broj I/O operacija jednak visini stabla, plus broj stranica, koje ispunjavaju uslov pretrage [8].

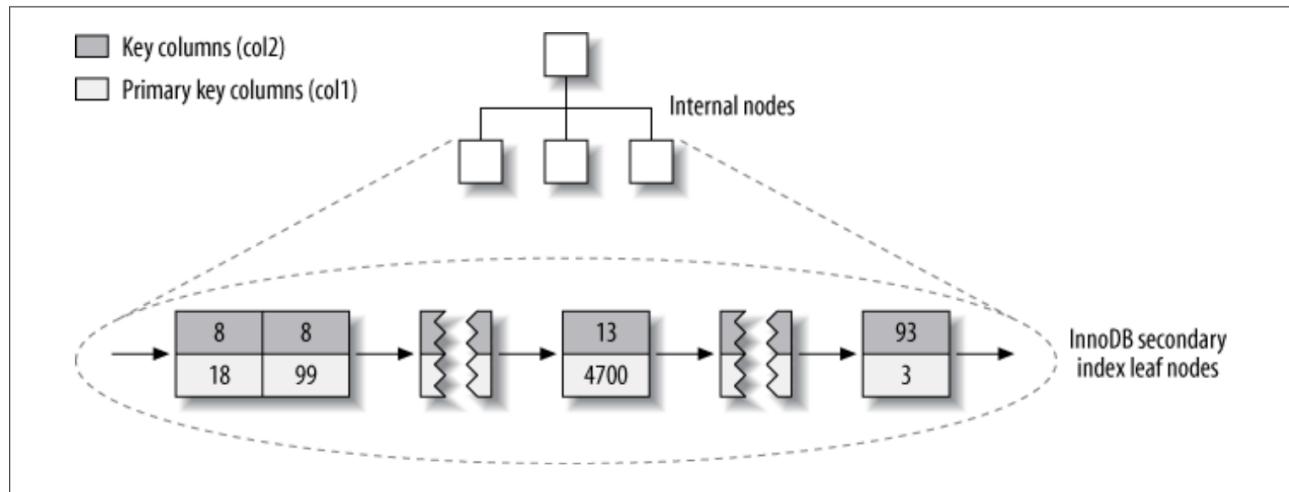


Slika 10. Struktura klasterizovanog indeksa kod InnoDB skladišta [8]

### 3.2. Neklasterizovani indeksi

Neklasterizovani indeksi su sinonim za sekundarne indekse kod InnoDB skladišta. Kod njih se kolonama definisanim za sekundarni indeks pridružuje kolona primarnog ključa. Za razliku od

klasterizovanih indeksa, gde se u stranicama listova skladište podaci iz tabele, kod sekundarnog indeksa u stranicama listova, pored vrednosti indeksirane kolone, skladišti se vrednost primarnog ključa (u primeru koji je na slici 11, za indeksiranje se koristi col2, kojoj se dodaje col1, kao primarni ključ). Ovo znači da ako se za pretragu koristi sekundarni indeks, neophodna je pretraga dva indeksa, sekundarnog i klasterizovanog. Prvo se u sekundarnom indeksu traži vrednost primarnog ključa podatka iz tabele, a potom se na osnovu vrednosti primarnog ključa iz sekundarnog indeksa pristupa podacima u klasterizovanom indeksu [8].

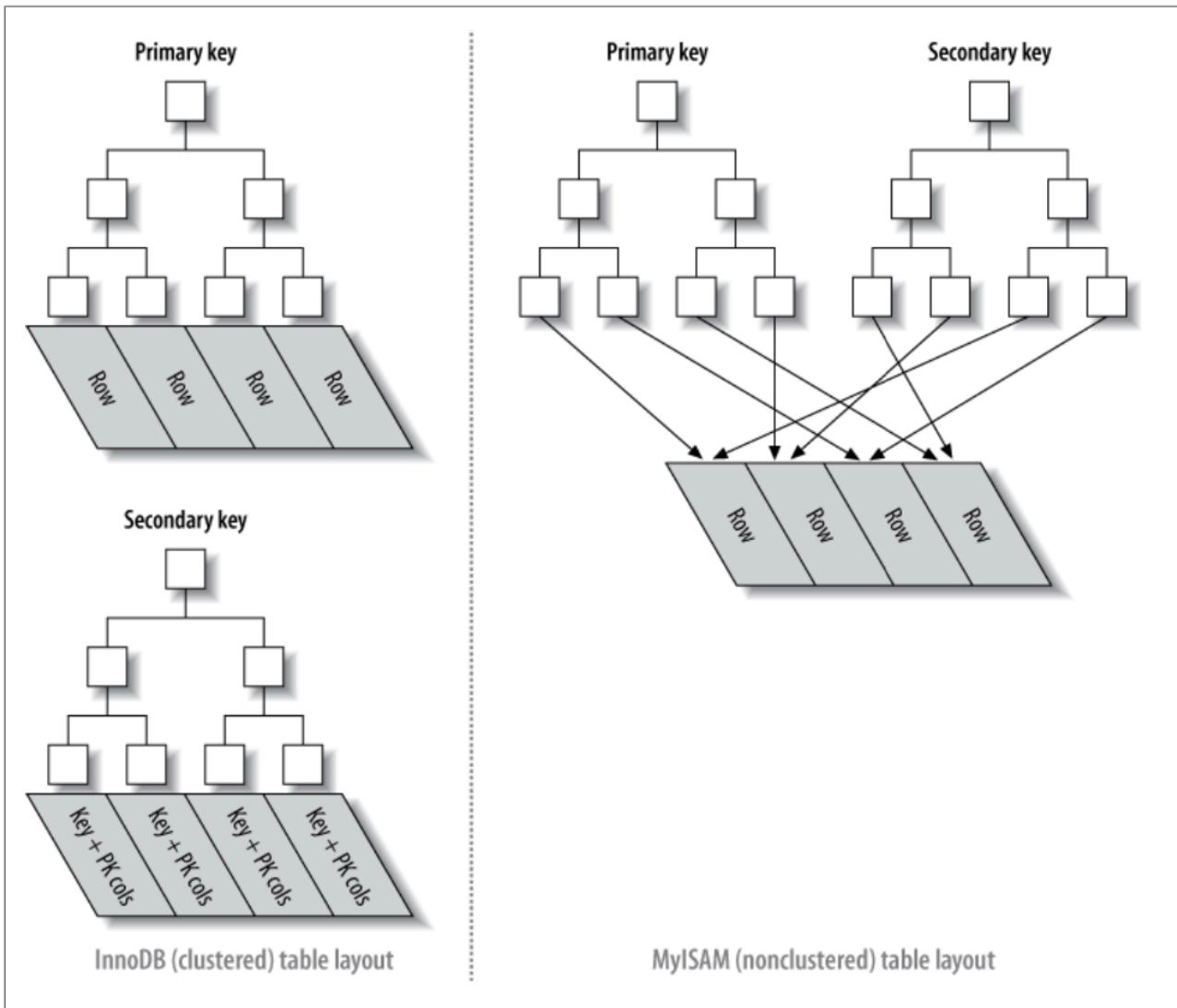


Slika 11. Struktura sekundarnog indeksa kod InnoDB skladišta [8]

Na slici 11 prikazana je struktura sekundarnog indeksa kod InnoDB skladišta. Može se primetiti da listovi sadrže vrednosti indeksiranih kolona i vrednost primarnog ključa.

Za razliku od InnoDB klasterizovanih i neklasterizovanih indeksa o kojima je do sada bilo reči, kod MyISAM skladišta, koje je bilo podrazumevano skladište do verzije 8.0 MySQL servera, primarni i sekundarni indeks se ne razlikuju po strukturi. Primarni indeks kod MyISAM je jednostavno jedinstven, *nonnullable* indeks. Na slici 12 desno je struktura primarnog indeksa kod MyISAM skladišta. Listovi indeksa sadrže vrednost ključa traženja i pokazivač na red u tabeli [8].

Dakle, kod MyISAM skladišta nema skladištenja podataka u indeksu kao kod InnoDB skladišta. Svi MyISAM indeksi imaju istu strukturu, dok se struktura klasterizovanih i neklasterizovanih kod InnoDB razlikuje (slika 12 levo) [8].



Slika 12. Razlika u strukturi indeksa kod MyISAM i InnoDB skladišta [8]

Kada su neklasterizovani indeksi u pitanju, za indeksiranje se može koristiti prefiks ili cela kolona, može se indeksirati više kolona, a indeks se može kreirati i na virtuelnim kolonama, kada se za ključ traženja ne navodi naziv kolone, već funkcija koja se primenjuje na kolonu.

### 3.2.1. Sekundarni indeksi

Sekundarni indeksi kod InnoDB imaju strukturu B-stabla, koja omogućava brzu pretragu indeksa navođenjem vrednosti, skupa vrednosti ili opsega vrednosti, uz korišćenje operatora kao što su =, <, >=, BETWEEN, IN i drugih, unutar WHERE klauzule [9].

Recimo da imamo tabelu u kojoj se skladiste podaci o korisnicima (slika 13).

```
create_people_sql = """
CREATE TABLE `people` (
  `id` int NOT NULL AUTO_INCREMENT,
  `first_name` varchar(50) COLLATE utf8_unicode_ci NOT NULL,
  `last_name` varchar(50) COLLATE utf8_unicode_ci NOT NULL,
  `email` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
  `city` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
  `country` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
  `phone_number` varchar(25) COLLATE utf8_unicode_ci NOT NULL,
  `birthdate` date NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
"""
```

Slika 13. Tabela people

```
1768 rows in set (0.08 sec)

mysql> select * from people where first_name='David';
```

Slika 14. Primer upita i vreme izvršenja

Ako u tabeli imamo oko 110000 redova i želimo da pretražimo korisnike po imenu (slika 14), bez indeksa MySQL mora da pročita sve redove tabele i poredi ih da li ispunjavaju uslov jednakosti, što nije efikasno. Na slici 15 prikazan je rezultat izvršenja EXPLAIN naredbe za upit sa slike 14, gde se vidi da je za pronalaženje redova, koji ispunjavaju uslov selekcije, moralo biti ispitano 109186 redova.

```
mysql> explain select * from people where first_name='David';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE     | people | NULL      | ALL   | NULL          | NULL | NULL    | NULL | 109186 | 10.00  | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

Slika 15. Izlaz EXPLAIN naredbe kada se ne koristi indeks

S porastom broja redova u tabeli, neophodno je naći efikasan način za pronalaženje podataka koji ispunjavaju uslov selekcije. Ovakav slučaj se rešava kreiranjem sekundarnog indeksa za polje *first\_name* u tabeli (slika 16).

```
[mysql]> create index name on people(first_name);
Query OK, 0 rows affected (0.16 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Slika 16. Kreiranje sekundarnog indeksa nad kolonom *first\_name*

Kreiranjem sekundarnog indeksa nad kolonom *first\_name* omogućena je efikasna pretraga podataka po ovoj koloni. Smanjuje se vreme potrebno za nalaženje podataka, kao i broj I/O operacija diska, jer su podaci u indeksu sortirani na osnovu kolone *first\_name*. Ako ponovimo upit nakon kreiranja indeksa videćemo da je vreme izvršenja znatno manje. U ovom slučaju, isti upit se izvršava za 0.01s nakon kreiranja indeksa, dok je prethodno izvršenje trajalo 0.08s (slika 17).

```
-----+
1768 rows in set (0.01 sec)

mysql> select * from people where first_name='David';
```

Slika 17. Primer upita i vreme izvršenja

Na slici 18 prikazan je rezultat izvršenja *EXPLAIN* naredbe za isti upit, gde se vidi da je za pronalaženje redova, koji ispunjavaju uslov selekcije, korišćen indeks *name* i ispitano je samo 1768 redova, onoliko koliko i ispunjava uslov selekcije, umesto oko 110000 redova.

```
mysql> explain select * from people where first_name='David';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE     | people | NULL      | ref  | name          | name | 152    | const | 1768 | 100.00 | NULL  |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

Slika 18. Izlaz EXPLAIN upita nakon kreiranja indeksa

Za indeksiranje kolone se ne mora koristiti cela kolona, već se može uzeti prefiks (slika 19).

```
[mysql] > create index name on people(first_name(5));
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Slika 19. Kreiranje sekundarnog indeksa nad prefiksom kolone *first\_name*

Ako se vrednost *first\_name* kolone obično razlikuje u prvih 5 karaktera, pretrage uz pomoć indeksa neće biti mnogo sporije nego kada je za indeksiranje korišćena cela kolona, ali korišćenjem prefiksa kolone umesto cele kolone značajno se smanjuje veličina indeks fajla, a može ubrzati i *INSERT* operacije. Kreiranje indeksa nad prefiksom kolone moguće je samo za kolone tekstualnog tipa (VARCHAR, CHAR, TEXT) ili binarnog string tipa (BINARY, VARBINARY, BLOB) [10].

Ako je ključ traženja duži od dužine prefiksa, indeks se koristi za eliminisanje redova tabele kod kojih nije pronađeno podudaranje, dok se ispituje podudaranje sa ostalim redovima [9].

Sekundarni indeks se može kreirati nad više kolona, a takav indeks se naziva *multiple-column* indeks ili kompozitni indeks. Kompozitni indeks može imati do 16 kolona, a za određene tipove podataka, mogu se koristiti i prefiksi kolona. MySQL može koristiti kompozitne indekse za upite koji testiraju sve kolone iz indeksa ili za upite koji testiraju samo prvu, samo prve dve, samo prve tri kolone i tako dalje. Redosled navođenja kolona kod kompozitnog indeksa je bitan, jer jedan kompozitni indeks može ubrzati više različitih upita nad istom tabelom. Recimo da imamo indeks (col1, col2, col3), on se može koristiti kod upita koji testiraju samo col1, col1 i col2 ili sve tri kolone. Ne može se koristiti kod upita koji testiraju col2, col3 ili col3 i col2, jer je col1 izostavljena iz upita [9].

Za tabelu sa slike 13, može se kreirati kompozitni indeks sa slike 20.

```
[mysql] > create index name_composite on people(first_name, last_name desc);
Query OK, 0 rows affected (0.17 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Slika 20. Kompozitni indeks nad kolonama *first\_name* i *last\_name*

Za kolonu *last\_name* navedena je opcija *DESC*, što znači da će vrednosti kolone u indeksu biti skladištene u opadajućem redosledu.

Indeks *name\_composite* iz prethodnog primera uključuje kolone *first\_name* i *last\_name*. Može se koristi kod upita koji definišu različite kombinacije vrednosti za *first\_name* i *last\_name*. Takođe, primenjuje se i kod upita, koji uključuju i samo *first\_name* kolonu, jer je ona prva navedena pri kreiranju indeksa. Kod kreiranja indeksa umesto cele kolone mogao je biti korišćen i prefiks kolone. Tako, indeks *name\_composite* koristi se svih upita na slici 21.

```
mysql> explain select * from people where first_name like 'Christine' and last_name like 'Evans';
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | people | NULL | range | name_composite | name_composite | 304 | NULL | 1 | 100.00 | Using index condition |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from people where first_name like 'Christine';
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | people | NULL | range | name_composite | name_composite | 152 | NULL | 198 | 100.00 | Using index condition |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from people where first_name like 'Chris%';
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | people | NULL | range | name_composite | name_composite | 152 | NULL | 1392 | 100.00 | Using index condition |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from people where first_name like 'Christine' and (last_name like 'Evans' or last_name like 'Walker');
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | people | NULL | range | name_composite | name_composite | 304 | NULL | 2 | 100.00 | Using index condition |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from people where first_name like 'Christine' and last_name>='E' and last_name<='Z';
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | people | NULL | range | name_composite | name_composite | 304 | NULL | 150 | 100.00 | Using index condition |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

Slika 21. Primeri upita za indeks *name\_composite*

Međutim, ako se u *WHERE* klauzuli ispituje *last\_name*, a *first\_name* ne, indeks *name\_composite* ne može se primeniti, zbog redosleda navođenja kolona pri kreiranju indeksa (slika 22).

```
mysql> explain select * from people where last_name='Evans';
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | people | NULL | ALL | NULL | NULL | NULL | NULL | 109186 | 10.00 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from people where last_name='Evans' or first_name='Christine';
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | people | NULL | ALL | name_composite | NULL | NULL | NULL | 109186 | 19.00 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

Slika 22. Upiti kod kojih se ne koristi indeks *name\_composite* za optimizaciju pretrage

MySQL podržava kreiranje sekundarnih indeksa nad virtuelnim, generisanim kolonama. Ovo je slučaj kada se umesto kolone navodi izraz funkcije, što omogućava indeksiranje vrednosti koje se

ne skladište direktno u tabeli. Na slici 23 prikazan je primer kreiranja indeksa, gde se umesto cele kolone *phone\_number* indeksira prvih 4 karaktera funkcijom *SUBSTRING()*. Da bi se ovaj indeks koristio kod izvršenja upita neophodno je da u upitu takođe bude navedena funkcija *SUBSTRING()* sa istim argumentima.

```
mysql> create index phone on people((substring(phone_number,1,4)));
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain select * from people where substring(phone_number,1,4)="+381";
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE      | people | NULL       | ref  | phone        | phone | 15    | const | 181  | 100.00 | NULL   |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

*Slika 23. Primer indeksiranja virtuelnih kolona*

### 3.2.2. Unique indeks

Pored toga što se koriste za brzo lociranje redova, koji zadovoljavaju uslov traženja, indeksi se koriste i za postavljanje ograničenja. *Unique* indeks predstavlja ograničenje, koje uslovljava da sve vrednosti u indeksu moraju biti jedinstvene, što garantuje da ne postoje dva reda u indeksiranoj tabeli, koja imaju iste vrednosti za indeksirane kolone. *Unique* indeks može se kreirati na jednoj ili više kolona, ili na prefiks kolone. Ako se navede prefiks vrednosti kolone u *unique* indeksu, vrednosti kolone moraju biti jedinstvene unutar dužine prefiksa. *Unique* indeks dozvoljava više NULL vrednosti za kolone koje mogu biti NULL [9].

*Unique* indeksi su korisni za postavljanje ograničenja, kojim se zahteva jedinstvenost podataka. Na primer, kada su u pitanju identifikatori redova koji moraju da budu jedinstveni. Kreiranjem *unique* indeksa garantovan je integritet podataka, odnosno baza će primeniti indeks kao ograničenje i sprečiti upisivanje duplikata u tabelu [8].

Tabela *people* o kojoj je do sada bilo reči ima kolonu *phone\_number*, koja bi trebala da ima jedinstvene vrednosti. Ovo ograničenje nije definisano u samom početku, ali se može dodati komandom *ALTER TABLE* (slika 24). Na slici 24 prikazano je izvršenje komande *ALTER TABLE*, kojom se dodaje ograničenje *unique* za kolonu *phone\_number*. Na slici je, takođe, izvršenje komande *EXPLAIN*, za upit koji vraća podatke o osobama čiji broj telefona počinje sa +381. Vidi se da je selekcija koristila prethodno kreirani indeks *phone\_number*.

```

mysql> alter table people add unique(phone_number);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain select * from people where phone_number like '+381%';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE      | people | NULL       | range | phone_number | phone_number | 77    | NULL | 181 | 100.00 | Using index condition |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

```

*Slika 24. Kreiranje unique indeksa za kolonu phone\_number*

Kod MyISAM skladišta, razlika u strukturi između primarnog i *unique* indeksa ne postoji. Međutim, kod InnoDB skladišta, kao što je opisano u poglavlju 3.2. postoji. *Unique* indeks ima strukturu neklasterizovanog indeksa i u listovima B-stabla indeksa čuva vrednost ključa traženja i vrednost primarnog ključa, a primarni indeks je zapravo klasterizovani indeks [8].

### 3.3. Fulltext indeks

*Fulltext* indeksi koriste se za *full-text* pretrage. InnoDB i MyISAM skladišta podržavaju kreiranje *fulltext* indeksa nad kolonama tekstualnog tipa (CHAR, VARCHAR ili TEXT). Za indeksiranje se uvek koristi cela kolona, prefiks kolone kod *fulltext* indeksa ne može se koristiti. *Fulltext* pretraga koristi se kod *MATCH()* ... *AGAINST* sintakse [11].

InnoDB *fulltext* indeksi imaju strukturu invertovanog indeksa. Invertovani indeks skladišti listu reči, i za svaku od njih, listu dokumenata u kojima se ona javlja i poziciju u dokumentu u vidu *byte offset-a* [11].

Na slici 25 je upit kojim se kreira tabela *paragraphs*, koja ima kolone *id* i *paragraph*. Kolona *paragraph* je VARCHAR tipa i pogodna je za kreiranje *fulltext* indeksa. InnoDB koristi jedinstveni identifikator dokumenta - *DOC\_ID* da mapira reči u *fulltext* indeksu na slogove dokumenta u kojima se reč pojavljuje. Za mapiranje je neophodno da postoji kolona *FTS\_DOC\_ID* u indeksiranoj tabeli. Ako se ne definiše kolona *FTS\_DOC\_ID*, InnoDB automatski dodaje skrivenu *FTS\_DOC\_ID* kolonu pri kreiranju *fulltext* indeksa. Kod tabele *paragraphs* nije definisana, pa je InnoDB automatski dodaje.

```

create_paragraphs_table = """
CREATE TABLE `paragraphs` (
    `id` int NOT NULL AUTO_INCREMENT,
    `paragraph` varchar(4000) COLLATE utf8_unicode_ci NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
"""

```

Slika 25. Upit za kreiranje tabele *paragraphs* za koju se kreira fulltext indeks

Na slici 26 je upit kojim se kreira *fulltext* indeks nad kolonom *paragraph*, tabele *paragraphs*. Kada se kreira *fulltext* indeks, uz njega se kreira i skup indeksnih tabela, takođe prikazane na slici 26.

```

mysql> create fulltext index words on paragraphs(paragraph);
Query OK, 0 rows affected, 1 warning (0.38 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> select table_id, name, space from information_schema.innodb_tables where name like 'BazePodataka/%';
+-----+-----+-----+
| table_id | name | space |
+-----+-----+-----+
| 1112 | bazepodataka/people | 51 |
| 1152 | bazepodataka/locations | 91 |
| 1173 | bazepodataka/fts_0000000000048e_being_deleted | 112 |
| 1174 | bazepodataka/fts_0000000000048e_being_deleted_cache | 113 |
| 1175 | bazepodataka/fts_0000000000048e_config | 114 |
| 1176 | bazepodataka/fts_0000000000048e_deleted | 115 |
| 1177 | bazepodataka/fts_0000000000048e_deleted_cache | 116 |
| 1167 | bazepodataka/fts_0000000000048e_0000000000000139_index_1 | 106 |
| 1168 | bazepodataka/fts_0000000000048e_0000000000000139_index_2 | 107 |
| 1169 | bazepodataka/fts_0000000000048e_0000000000000139_index_3 | 108 |
| 1170 | bazepodataka/fts_0000000000048e_0000000000000139_index_4 | 109 |
| 1171 | bazepodataka/fts_0000000000048e_0000000000000139_index_5 | 110 |
| 1172 | bazepodataka/fts_0000000000048e_0000000000000139_index_6 | 111 |
| 1166 | bazepodataka/paragraphs | 105 |
+-----+-----+-----+
14 rows in set (0.00 sec)

```

Slika 26. Kreiranje fulltext indeksa

Imena pomoćnih tabela počinju sa *fts\_* i završavaju se sa *index\_#* (slika 26). Svaka od njih povezana je sa indeksnom tabelom uz pomoć heksadecimalne vrednosti, koja se nalazi u nazivu pomoćne tabele i označava *table\_id* indeksne tabele. Konkretno, na slici 26, *table\_id* indeksne tabele je 1166, što je u heksadecimalnom sistemu 048e, kao što se može videti u nazivu datoteka pomoćnih tabela invertovanog indeksa [12].

Kada se ulazni dokumenti tokenizuju, svaki token se upisuje u indeks tabele zajedno za informacijama o poziciji i *DOC\_ID*. Reči su potpuno sortirane i particionisane unutar šest indeks tabela na osnovu težine prvog karaktera u reči [12].

Invertovani indeks je particionisan u šest pomoćnih indeks tabela da bi podržao paralelno kreiranje indeksa. Koriste se dve niti za tokenizaciju, sortiranje, umetanje reči i povezanih podataka u indeks tabele. Broj radnih niti može se konfigurisati `innodb_ft_sort_pll_degree` promenljivom [12].

Ostale tabele su zajedničke i koriste se za rukovanje brisanjem i skladištenjem internog stanja *fulltext* indeksa. Za razliku od tabela invertovanog indeksa, koje se kreiraju za svaki *fulltext* indeks, ove tabele su zajedničke za sve *fulltext* indekse, koji se kreiraju nad određenom tabelom. Zajedničke indeksne tabele ostaju u memoriji čak iako dođe do brisanja *fulltext* indeksa. Kada se *fulltext* indeks obriše, kolona *FTS\_DOC\_ID* koja je kreirana za indeks ostaje, jer njeno brisanje zahteva obnavljanje prethodno indeksiranih tabela. Zajedničke indeksne tabele su neophodne za upravljanje *FTS\_DOC\_ID* kolonom [12].

Tabele *fts\_\*\_being\_deleted* i *fts\_\*\_being\_deleted\_cache* sadrže *DOC\_ID* dokumenata koji su obrisani i čiji podaci su u procesu brisanja iz *fulltext* indeksa. Tabele *fts\_\*\_deleted* i *fts\_\*\_deleted\_cache* sadrže *DOC\_ID* dokumenata koji su obrisani, ali njihovi podaci još nisu obrisani iz indeksa. Tabela *fts\_\*\_config* čuva informacije o internom stanju *fulltext* indeksa [12].

Na slici 27 je rezultat izvršenja upita, koji vraća sve paragafe koji sadrže reč *letter* i reč *prevent*.  
Takođe, prikazan je i rezultat EXPLAIN naredbe, gde se vidi da se koristi *fulltext* indeks *words*, kreiran na slici 26.

Slika 27. Primer upita koji koristi fulltext index

### 3.4. Prostorni indeks

MySQL podržava indeksiranje kolona višedimenzionalnih tipova, kao što su POINT i GEOMETRY. Indeksi nad kolonama tog tipa nazivaju se **prostorni indeksi** (eng. *Spatial Index*).

Prostorni indeksi imaju strukturu R-stabla. Kod InnoDB skladišta, prostorni indeksi nad kolonama prostornog tipa imaju sledeće karakteristike [9]:

- Prostorni indeks se može kreirati samo za kolone prostornog tipa, ne može se kreirati nad više kolona prostornog tipa.
- Indeksirane kolone ne smeju sadržati NULL vrednosti.
- Ne može se koristiti prefiks kolone, već se indeksira kompletna kolona.

Da bi optimizator koristio prostorni indeks u procesu optimizacije, kolona nad kojom se kreira prostorni indeks mora da ima definisan SRID (eng. *Spatial Reference System Identifier*). SRID atribut ukazuje da vrednosti skladištene u toj koloni pripadaju određenom prostornom referentnom sistemu (eng. *Spatial Reference System - SRS*). Kod InnoDB dozvoljne su SRID vrednosti za geometrijski i geografski SRS, dok je kod MyISAM dozvoljen samo geometrijski SRS [13].

Kod MyISAM i InnoDB tabele, operacije traženja za kolone sa prostornim podacima mogu se optimizovati prostornim indeksom. Najčešće operacije su [14]:

- Traženje svih objekata koji sadrže datu tačku,
- Traženje svih objekata koji se preklapaju sa datim regionom.

Optimizator proverava SRID atribut indeksiranih kolona, na osnovu čega odlučuje koji SRS se koristi za poređenje, i izvršava odgovarajuće proračune. Optimizator razmatra prostorne indekse samo za SRID-ograničene kolone [13]:

- Indeksi na kolonama ograničeni na Dekartov SRID omogućavaju izračunavanje Dekartovog graničnog okvira
- Indeksi na kolonama ograničeni na geografski SRID omogućavaju izračunavanje geografskih graničnih okvira.

Optimizator ignoriše prostorne indekse na kolonama, koje nemaju SRID atribut, jer nisu SRID-ograničene [13].

Na slici 28 se nalazi upit za kreiranje tabele *locations*. Ono što je bitno primetiti kod tabele *locations* jeste *coords* kolona, koja je tipa *geometry* i za koju je definisan SRID atribut. Pored kolone *coords*, tabela ima i druge kolone, koje će se koristi u izvršavanju upita. Pored definicije kolona, definisan je i primarni indeks tabele, kao i prostorni indeks *g* nad kolonom *coords*.

```
create_geo_table = """
CREATE TABLE `locations` (
    `id` int NOT NULL AUTO_INCREMENT,
    `coords` geometry NOT NULL SRID 4326,
    `city` varchar(255) UNIQUE NOT NULL,
    `country` varchar(5) NOT NULL,
    `continent` varchar(15) NOT NULL,
    `capital` varchar(50) NOT NULL,
    PRIMARY KEY (`id`),
    SPATIAL KEY `g` (`coords`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
"""
```

Slika 28. Tabela sa kolonom prostornog tipa i prostornim indeksom

Za podatke prostornog tipa, kod MySQL postoje posebne funkcije koje se mogu primeniti. U primeru, koji se nalazi na slici 29, korišćena je funkcija *ST\_Within*, koja određuje da li se određeni prostorni podatak unutar drugog prostornog podatka. U ovom slučaju, vrši se ispitivanje da li su koordinate grada unutar zadatog područja. Na slici 29 je rezultat izvršenja upita koji vraća naziv grada, koordinate, zemlju i kontinent svih torki koje zadovoljavaju uslov.

```

mysql> select city,st_astext(coords) as coordinates, country, continent from locations where st_within(coords,st_srid(st_geomfromtext('polygon((-10 -10, -10 10, 10 10, 10 -10, -10 -10))'),4326));
+-----+-----+-----+-----+
| city | coordinates | country | continent |
+-----+-----+-----+-----+
| Kafanchan | POINT(9.58126 8.2926) | NG | Africa |
| Efon-Alaaye | POINT(7.65649 4.92235) | NG | Africa |
| Ilesa | POINT(7.62789 4.74161) | NG | Africa |
| Olupona | POINT(7.6 4.18333) | NG | Africa |
| Shagamu | POINT(6.8485 3.64633) | NG | Africa |
| Tchaourou | POINT(8.88649 2.59753) | BJ | Africa |
| Mampong | POINT(7.06273 -1.4001) | GH | Africa |
| New Yekepa | POINT(7.57944 -8.53778) | LR | Africa |
| Idenao | POINT(4.2475 9.00472) | CM | Africa |
| Nkpor | POINT(6.15038 6.83042) | NG | Africa |
| Yenagoa | POINT(4.92675 6.26764) | NG | Africa |
| Tarkwa | POINT(5.30383 -1.98956) | GH | Africa |
| Bibiani | POINT(6.46346 -2.31938) | GH | Africa |
| Bonoua | POINT(5.27247 -3.59625) | CI | Africa |
| Lakota | POINT(5.84752 -5.682) | CI | Africa |
+-----+-----+-----+-----+
15 rows in set (0.00 sec)

```

Slika 29. Primer upita koji koristi prostorni indeks za selekciju

Na slici 30 je rezultat *EXPLAIN* naredbe upita sa slike 29, gde se može videti da je za izvršenje korišćen prostorni indeks *g*.

```

mysql> explain select city,st_astext(coords) as coordinates, country, continent from locations where st_within(coords,st_srid(st_geomfromtext('polygon((-10 -10, -10 10, 10 10, 10 -10, -10 -10))'),4326));
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | locations | NULL | range | g | g | 34 | NULL | 15 | 100.00 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

```

Slika 30. Rezultat izvršenja EXPLAIN naredbe za upit sa slike 29

Pored *ST\_Within*, postoje i druge funkcije za rad sa prostornim podacima, koje se mogu naći na lokaciji navedenoj u literaturi [15].

### 3.5. Adaptive Hash Index

Kod InnoDB skladišta korisniku nije omogućeno kreiranje heš indeksa, međutim, samo skladište koristi interno koristi heš indekse. *Adaptive hash index* (skraćeno *API*) InnoDB skladištu omogućava da se ponaša kao *in-memory* baza podataka s odgovarajućom kombinacijom opterećenja i dovoljno memorije za skup bafera, bez žrtvovanja svojstava kao što su transakcije i pouzdanost [16].

Na osnovu zapaženih šabloni pretraga, kreira se heš indeks korišćenjem prefiksa ključa traženja. Prefiks može biti proizvoljne dužine i ne mora sadržati sve vrednosti iz B-stabla. Kreiraju se na zahtev stranica, kojima se često pristupa [16].

Ako gotovo cela tabela može da stane u glavnu memoriju, heš indeks ubrzava izvršavanje upita omogućavajući direktni pristup bilo kom elementu, pretvarajući indeksiranu vrednost u pokazivač na podatak. InnoDB ima mehanizam koji prati pretrage indeksa i, ukoliko primeti da bi upiti mogli imati koristi od kreiranja heš indeksa, kreira ga automatski [16].

U određenim slučajevima, ubrzanje koje se dobije heš indeksom daleko je manje u odnosu na uložen trud za praćenje pregleda indeksa i održavanje strukture heš indeksa. Upiti sa *LIKE* operatorima i *wildcard* karakterima obično nemaju korist od heš indeksa. Za pretrage koje nemaju korist od *API*, njegovo isključivanje smanjuje nepotrebno trošenje resursa sistema i uticaj na performanse [16].

*API* se može isključiti setovanjem *innodb\_adaptive\_hash\_index* promenljive na OFF (slika 31).

```
[mysql] > set global innodb_adaptive_hash_index=OFF;
Query OK, 0 rows affected (0.00 sec)
```

Slika 31. Isključivanje adaptive hash index-a

## 4. Zaključak

Cilj ovog rada bio je da opiše internu strukturu i organizaciju indeksa kod sistema za upravljanje bazama podataka. Struktura i organizacija indeksa opisana je na primeru MySQL baze podatka. Data je kategorizacija indeksa na osnovu strukture i tipa. Dalje, opisana je razlika u strukturi indeksa kod MyISAM, koje je ranije bilo podrazumevano skladište, i InnoDB skladišta, koje je *default*-no kod novijih verzija MySQL servera. Za svaki tip indeksa, koji je podržan kod InnoDB skladišta, priložen je primer tabele i način kreiranja indeksa, kao i upiti kod kojih se indeks koristi.

Korišćenje indeksa može značajno poboljšati performanse upita, ali, takođe, moraju se napraviti kompromisi pri kreiranju indeksa. Iako deluje primamljivo kreiranje indeksa za svaku moguću kolonu, koja se koristi u upitu, nepotrebni indeksi predstavljaju tračenje memoriskog prostora i vremena za MySQL da odluči koji indeks da koristi. Indeksi takođe povećavaju cenu *insert*, *update* i *delete* operacija, jer svaki indeks mora biti ažuriran. Neophodno je pronaći balans za postizanje željene brzine izvršavanja upita uz optimalan set indeksa.

U drugom poglavlju obrazložen je način organizacije podataka, koji se skladište na eksternim uređajima. Pored toga, bilo je reči o načinima optimizacije brzine pristupa skladištenim podacima. Predložena je klasifikacija indeksa. Detaljno je opisana struktura indeksa kod sistema za upravljanje bazama podataka.

U trećem poglavlju opisana je organizacija i interna struktura indeksa kod MySQL servera. Predložena je klasifikacija indeksa kod MySQL baze podataka. Indeksi su demonstrirani primerima.

# Literatura

- [1] R. Ramakrishnan, J. Gehrke, *Database Management Systems Third Edition*, McGraw-Hill Higher Education, 2003
- [2] A. Guttman, *R-trees. A Dynamic Index Structure For Spatial Searching*, University of California, Berkley, 1984
- [3] What is MySQL? <https://dev.mysql.com/doc/refman/8.3/en/what-is-mysql.html> (pristup: 14.04.2024.)
- [4] Pages, Extents, Segmends and Tablespaces, <https://dev.mysql.com/doc/refman/8.3/en/innodb-file-space.html>, (pristup: 14.04.2024.)
- [5] P. Kieseberg, S. Schrittwieser, P. Fruhwirt, E. Weippl, *Analysis of the Internals of MySQL/InnoDB B+ Tree Index Navigation from a Forensic Perspective*, International Conference on Software Security and Assurance (ICSSA), 2019
- [6] The Physical Structure of an InnoDB Index, <https://dev.mysql.com/doc/refman/8.3/en/innodb-physical-structure.html>, (pristup: 14.04.2024.)
- [7] Clustered and Secondary Indexes, <https://dev.mysql.com/doc/refman/8.3/en/innodb-index-types.html>, (pristup: 14.04.2024.)
- [8] B. Schwartz, P. Zaitsev, V. Tkachenko, *High Performance MySQL Third Edition*, O'Reilly, 2012
- [9] CREATE INDEX Statement, <https://dev.mysql.com/doc/refman/8.3/en/create-index.html>, (pristup: 15.04.2024.)
- [10] Column Indexes, <https://dev.mysql.com/doc/refman/8.3/en/column-indexes.html>, (pristup 15.04.2024.)
- [11] InnoDB Full-Text Indexes, <https://dev.mysql.com/doc/refman/8.3/en/innodb-fulltext-index.html#innodb-fulltext-index-design>, (pristup: 15.04.2024.)
- [12] InnoDB Full-Text Index Design, <https://dev.mysql.com/doc/refman/8.0/en/innodb-fulltext-index.html#innodb-fulltext-index-design>, (pristup: 16.04.2024.)
- [13] SPATIAL Index Optimization, <https://dev.mysql.com/doc/refman/8.3/en/spatial-index-optimization.html>, (pristup: 18.04.2024.)

- [14] Optimizing Spatial Analysis, <https://dev.mysql.com/doc/refman/8.4/en/optimizing-spatial-analysis.html> (pristup: 18.04.2024.)
- [15] Spatial Function Reference, <https://dev.mysql.com/doc/refman/8.4/en/spatial-function-reference.html> (pristup: 18.04.2024.)
- [16] Adaptive Hash Index, <https://dev.mysql.com/doc/refman/8.3/en/innodb-adaptive-hash.html>, (pristup: 14.04.2024.)