Fizičko projektovanje BP

Projektovanje baza podataka 2023/24

Mogućnosti za optimizaciju

- Na hardverskom nivou
- Na nivou BP

Optimizacije na nivou BP

- Da li su pogodno odabrane kolone i njihovi tipovi podataka?
- Da li su dobro odabrani indeksi?
- Da li se koristi odgovarajuća mašina za skladištenje podataka (engl. *storage engine*)?
- Napredniji koncepti (kompresija tabela zarad manjeg broja I/O operacija; konkurentnost; konfiguracija keširanja)

Optimizacije na nivou BP

Odabir kolona i njihovih tipova podataka

- Odabir kolona zavisi od prirode problema koji BP rešava. Jedno od svojstava koje je potrebno razmotriti je učestalost ažuriranja.
  Na osnovu veće učestalosti se može doneti odluka o većem broju tabela (sa manje kolona). Suprotnu odluku (o manjem broju tabela sa većim brojem kolona) ima smisla razmotriti u slučaju obrade velike količine podataka.

Optimizacije na nivou BP

Odabir kolona i njihovih tipova podataka

- O odabiru pogodnih tipova je već bilo reči na uvodnom času o DDL-u. Precenjivanjem širine tipova potencijalno se povećava i broj I/O operacija jer se time ograničava broj redova tabele po stranici.

Optimizacije na nivou BP

Odabir indeksa

- O odabiru i isprobavanju različitih tipova indeksa je bilo reči na prethodnom tročasu. U nastavku će biti razmotreni još neke aspekti indeksa kao sredstva za optimizaciju BP.
- Uopšteno govoreći, treba pažljivo odlučiti koje kolone ima smisla indeksirati. Indeksiranje svih kolona može imati negativan uticaj na performanse. Naime, SUBP treba da odluči koji indeks odabira, a sa svakim dodatim indeksom, nakon operacija dodavanja, ažuriranja i brisanja (koje se odnose na novoindeksirane kolone), potrebno je i dodatno vreme da se ažuriraju odgovarajući indeksi.

Optimizacije na nivou BP

Mašine za skladištenje podataka

- Trenutne verzije MySQL-a podrazumevano koriste *InnoDB* mašinu za skladištenje koju karakteriše visoka pouzdanost i dobre performanse.
- Kao deo naredbe, prilikom kreiranja tabele, moguće je navesti i mašinu `'ENGINE=...'` koja će biti korišćenja za skladištenje te tabele.
- Tabelarni prikaz svih podržanih mašina na MySQL serveru se može dobiti `‘SHOW ENGINES;‘` komandom.

Optimizacije na nivou BP

Mašine za skladištenje podataka

Neke od mašina podržanih u MySQL-u:

- InnoDB
- MyISAM
- Memory
- CSV
- Archive
- NDB
- Merge
- Federated

Optimizacije na nivou BP

Mašine za skladištenje podataka

Odabir mašine treba izvršiti uzevši u obzir koje funkcionalnosti mašina podržava - tj. razmatranjem osobina koje su poželjne za dobro funkcionisanje BP koja se projektuje. U nastavku ćemo uporediti par funkcionalnosti InnoDB i Memory mašine:

- Indeksiranje b-stablima i enkripcija podataka podržani su od strane obe mašine.
- Klasterovani indeksi i kompresovani podaci nisu podržani kod Memory mašine.
- Integritet stranog ključa nije moguće obezbediti kod Memory mašine. Same SQL naredbe kojima se kreira ključ će biti parsirane, ali će se uslov ignorisati, tj. neće biti provera.
- Indeksi za pretragu teksta (engl. *full-text indexes*) nisu podržani kod Memory mašine. Kod InnoDB mogu se koristiti nad kolonama koje su `CHAR`, `VARCHAR` ili `TEXT` tipa.

Optimizacije na nivou BP

Mašine za skladištenje podataka

- Podrška za geoprostorne (engl. *geospatial*) tipove podataka i indekse nije prisutna kod Memory mašine.
- Heš indeksi su podržani kod Memory mašine i moguće je eksplicitno ih navesti kao tip prilikom kreiranja indeksa. Sa druge strane, kod InnoDB je takođe moguće navesti ih, ali će se ipak kreirati (podrazumevano) b-stablo. Izvesna podrška kod InnoDB mašine ipak postoji! U pitanju je mehanizam takozvanih *adaptivnih heš indeksa* koji podrazumeva interno korišćenje heš indeksa kada se za to stvore uslovi. To se dešava u pozadini i o tome će više reči biti u praktičnom primeru u nastavku.

Optimizacije na nivou BP

Mašine za skladištenje podataka

- Nivo na kojem se vrši zaključavanje (engl. *locking granularity*) je kod InnoDB pojedinačni red, a kod Memory tabela.
- Replikacija je u velikoj meri podržana samo u slučaju InnoDB.
- Ograničenje pri skladištenju je 64TB kod InnoDB, a kapacitet radne memorije u slučaju Memory mašine.
- Rad sa transakcijama je podržan samo u slučaju InnoDB.

Optimizacije na nivou BP

Format reda (InnoDB i MySQL)

Prilikom kreiranja tabela, moguće je navesti način na koji će se red skladištiti na disku `'ROW_FORMAT=...'`. InnoDB podržava 4 moguće vrednosti:

1. REDUNDANT - stariji format koji se čuva zbog kompatibilnosti sa ranijim verzijama MySQL-a
2. COMPACT - zauzima oko 20% manje prostora u odnosu na prethodni
3. DYNAMIC - slično kao COMPACT, uz poboljšanje skladištenja (potencijalno dugačkih) vrednosti u kolonama tipa promenljive dužine
4. COMPRESSED - slično kao DYNAMIC, uz dodatnu uštedu prostora usled korišćenja kompresije podataka iz tabela i indeksa

Optimizacije na nivou BP

Prostor tabela (InnoDB i MySQL)

- Više informacija o pojedinačnim formatima je dostupno u priloženim linkovima, a da bi razlike u detaljima prethodnih formata redova bilo moguće razmatrati potrebno je znati i da se redovi skladište u stranicama (blokovima) fiksne dužine, a stranice se skladište u prostorima tabela.
- MySQL razilkuje nekoliko različitih prostora tabela, od kojih su za razumevanje fizičke organizacije važni: sistemski, opšti i pojedinačni fajl po tabeli.
- Komandom `'show variables like 'inno%';'` prikazuju se trenutna podešavanja InnoDB mašine. Na podešavanja je moguće uticati prilikom startovanja servera, komandom iz MySQL-a ili pomoću konfiguracionog fajla (npr. globalnog, na putanji `/etc/mysql/my.cnf`).

Optimizacije na nivou BP

Prostor tabela (InnoDB i MySQL)

- Ukoliko je podešavanje ‘`innodb_file_per_table`‘ postavljeno na vrednost `ON`, svaka BP će imati sopstveni direktorijum, u kojem se nalazi određeni broj (`*.ibd` ili `*.frm`) datoteka koje odgovaraju tabelama.
- Vrednost promenljive ’`innodb_data_file_path`’ može biti nalik na ’`ibdata1:12M:autoextend`’. U tom slučaju, na većini Linuks sistema, podrazumevano se čuva fajl `/var/lib/mysql/ibdata1` u kojem se nalaze metainformacije.
- Na osnovu prethodnog nije teško locirati fajlove koji odgovaraju tebelama. Za tabelu `Employees` baze `office` sa časova prethodnih nedelja, odgovarajuća `*.ibd` datoteka je na putanji `/var/lib/mysql/office/Employees.ibd`.
- U `*.ibd` datotekama se nalazi jedno (ili više) b+stablo koje je uređeno na osnovu primarnog ključa, dok se u `*.frm` datotekama nalazi shema tabele.

Optimizacije na nivou BP

Prostor tabela (InnoDB i MySQL)

- Iz razloga razbacanosti prilikom čuvanja (meta)informacija o tabelama, a u nameri da se drugačije rasporede podaci u prostoru, nije dovoljno samo pomeriti odgovarajući direktorijum. Takođe, sve prethodne podrazumevane vrednosti i putanje se odnose na ovu kombinaciju (i verziju!) SUBP i mašine za skladištenje.
- Ako zanemarimo konačnost kapaciteta hardvera na kojem se nalazi server, softverski uzrokovana (ili zadata) memorijska ograničenja takođe postoje.

Optimizacije na nivou BP

Prostor tabela (InnoDB i MySQL)

- Kada je u pitanju MySQL, ne postoji unapred zadata gornja granica broja tabela ili broja BP. Međutim, fajl sistem na kojem se server izvršava može nametati ograničenje u smislu broja direktorijuma ili datoteka koji predstavljaju jednu tabelu.
- Mašina za skladištenje takođe može imati postavljeno podešavanje kojim se ograničava broj tabela. U slučaju InnoDB, maksimum je 4 milijardi tabela.
- MySQL ograničava broj kolona na ne više od 4096 kolona po tabeli. Što se tiče veličine vrsta, ograničenje je 65535 bajtova, pri čemu veći tipovi podataka poput BLOB i TEXT doprinose sa svega 9-12B, s obzirom da se ne skladište na istoj fizičkoj lokaciji kao i ostatak reda. Više o tim ograničenjima ovde.

Optimizacije na nivou BP

Postupak selekcije vrsta iz tabele (InnoDB i MySQL)

Da bi se pronašla vrsta čija je vrednost primarnog ključa (zarad jednostavnosti jedan atribut) jednaka nekom N, ugrubi postupak je:

- Proverava se da li je tabela već otvorena, a neke (meta)informacije se prebacuju u keš, a ujedno se locira putanja ka `*.ibd` datoteci.
- Identifikuje se primarni ključ i počevši od korena b+stabla (koje odgovara toj tabeli) pretraga se kreće naniže, do lista u kojem se čuva tražena stranica. Uopšteno gledano, stranica neće imati veliki broj redova, a može se desiti i da se neka već nalazi u *buffer pool*[1]-u.

[1] Deo radne memorije za keširanja podataka o tabelama i indeksima. Više ovde.

Optimizacije na nivou BP

Postupak selekcije vrsta iz tabele (InnoDB i MySQL)

- Sem prethodnih koraka, neophodno je razrešiti i komplikacije u kontekstu transakcija i tipova podataka koji se ne skladište na istoj fizičkoj lokaciji kao i ostatak reda.
- Da bi se pronašla vrsta gde se selekcija vrši po atributu prema kojem ne postoji fizičko uređenje u memoriji (tj. klasterovani indeks) postupak i dalje odgovara algoritmu b+stabla, ali je manje efikasan.
- Komandom `'SHOW GLOBAL STATUS LIKE 'Opened_tables';'` se može pročitati broj operacija otvaranja tabela od početka rada servera.
- Detaljnije informacije o načinu otvaranja i zatvaranja tabela u MySQL-u.

Tehnike optimizacije upita

- Pored korišćenja indeksa, postoji niz drugih tehnika za optimizaciju upita u MySQL bazi podataka. Ove tehnike su ključne za poboljšanje performansi baza podataka, posebno u situacijama kada indeksi nisu dovoljni ili nisu optimalni za određene vrste upita.
- Neke od tehnika su:
  - Transformisanje SELECT naredbi
  - Denormalizacija
  - Particionisanje tabela
  - Keširanje rezultata upita
  - Korišćenje materijalizovanih pogleda

Tehnike optimizacije upita

Transformisanje SELECT naredbi

- Formiranje efikasnih SELECT naredbi igra ključnu ulogu u optimizaciji performansi upita u bazi podataka. Neki od saveta prilikom pisanja upita su:
- **Izbegavati korišćenje naredbe SELECT ***:
  Prilikom pisanja upita, ključno je identifikovati kolone je potrebno prikazati u rezultatu. Iako je koristi kao prečica za ispis vrednosti svih kolona, ova naredba može biti dosta skupa ukoliko tabela ima veliki broj kolona, od kojih većina nije bitna za konkretni upit. Sem loših performansi, upit napisan na ovaj način može proizvesti greške unutar aplikacije. Ukoliko se tabeli doda nova kolona, SELECT * će obuhvatiti i tu novu kolonu što može dovesti do neželjenih promena u podacima koje aplikacija prima, što može poremetiti funkcionalnost. Kako bi se ovakvi scenariji izbegli, sigurnije je u SELECT delu upita *navesti listu kolona* koji su od interesa za taj upit.

- **Dohvatati onoliko redova koliko je potrebno**:
  Razmotrimo primer veb stranice koja prikazuje artikle odredjene onlajn prodavnice. Na svakoj stranici se prikazuje po 10 proizvoda. Ukoliko bi se u bazi skladistio ogroman broj artikala, učitavanje svih artikala a zatim izdvajanje prvih 10 bi zahtevalo dosta vremena, tako da bi korisnik proveo neko vreme čekajući učitavanje željene stranice. Optimizacija ovakvog upita moguća je korišćenjem **LIMIT klauzule**. Na kraju upita je samo potrebno postaviti LIMIT koliko redova je potrebno da se učita, umesto cele tabele. Ukoliko bi bilo potrebno učitati drugih 10 redova, **LIMIT 10 OFFSET 10** bi bila naredba koji je potrebno navesti na kraju upita. **OFFSET** označava počev od kog reda je potrebno čitati podatke.

- **Izbegavati DISTINCT**:
  Ovom naredbom se ukljanjaju duplikati iz rezultata upita i može se smatrati neefikasnom po performanse čitavog upita. Naime, svi redovi prvo bivaju sortirani a zatim se duplikati eliminišu iz tako dobijenog rezultata. Mogući efikasniji način da se izdvoje jedinstveni redovi u rezultatu jeste pomoću klauzule **GROUP BY**. Razlog za korišćenje ove klauzule leži u tome da GROUP BY u toku grupisanja podataka može koristiti postojeće indekse nad tim kolonama, za razliku od DISTINCT. Drugi način za dobijanje jedinstvenih redova u rezultatu jeste **dodavanje nove kolone** u SELECT delu koja čini da nikoja dva reda nisu ista.

- Koristiti INNER JOIN umesto Dekartovog spajanja sa WHERE uslovom
- Ne koristiti LEFT, RIGHT i OUTTER JOIN kada to nije potrebno, već INNER JOIN.
- Prilikom spajanja tabela, osigurati da su kolone po kojima se vrši spajanje istog tipa:
  - Ukoliko se dve tabele spajaju po kolonama, tako da je jedna kolona numeričkog tipa, a druga tekstualnog, potrebno je prilikom spajanja u svakom koraku vrednost kolone numeričkog tipa konvertovati u tekstualni tip, pa tek onda izvršiti poredjenje da li se vrednosti poklapaju. Ovaj problem očigledno leži u dizajnu same baze podataka, i konverzijom tipova unutar upita je kratkoročno rešen problem spajanja tabela po kolonama različitog tipa. Ukoliko je potrebno izvršiti dosta upita koji spajaju ovakve tabele, performanse sistema postaju sve lošije. Najbolje rešenje je promeniti tipove kolona u samim tabelama ukoliko je to moguće.

- Izbegavati korišćenje funkcija nad kolonama po kojima se vrši spajanje tabela:

```
SELECT * FROM users JOIN orders
ON UPPER(users.username) = orders.username
```

I u ovom slučaju, problem leži u samom dizajnu baze podataka.

- Izbegavati korišćenje funkcija nad kolonama koje se nalaze u WHERE klauzuli:
  =>*Nemoguće korišćenje postojećih indeksa nad kolonama koje se nalaze u WHERE klauzuli.*

```
SELECT count(*)
FROM orders
WHERE CAST(order_timestamp AS DATE) > '2024-02-01';
```

=>*izvršiti transformaciju nad konstantom:*

```
SELECT count(*)
FROM orders
WHERE order_timestamp > '2024-02-01 00:00:00';
```

- *Ne može se korišćenje svake funkcije izbeći (npr. SUBSTRING), ali kada god se u WHERE klauzuli nađe funkcija, uvek treba razmišljati o mogućim alternativama.*
- *Drugi način za rešavanje ovakvog problema u MySQL jeste pravljenje indeksa nad odredjenim izrazom -* **'funkcijski indeks'**
- Kreiranje funkcijskog indeksa:

```
ALTER TABLE table_name
ADD COLUMN virtual_column INT AS (some_expression) VIRTUAL,
ADD INDEX index_name (virtual_column);
```

- Koristiti EXISTS umesto IN u radu sa podupitima:

  SELECT * FROM table1
  WHERE table1.column **IN** (SELECT column FROM table2);

  Koristiti:

  SELECT * FROM table1
  WHERE **EXISTS** (SELECT table2.column FROM table2
  WHERE table1.column = table2.column);

  Korišćenje EXISTS je često efikasnije od IN, iz razloga što EXISTS prestaje sa pretragom čim naiđe na prvi red koji zadovoljava uslov, dok IN vrši pretrago do kraja. Takođe, EXISTS ume da rukuje sa NULL vrednostima, za razliku od IN, koji može prouzrokovati neočekivano ponašanje koje može proizvesti grešku a samim tim i sporije izvršavanje upita.

- Umesto COUNT(*) koristiti EXISTS za proveru da li je tabela prazna: U slučaju velikog broja redova, EXISTS će se zaustaviti na prvom redu i vratiti True, dok će COUNT proći kroz celu tabelu i prebrojati sve redove.
- Uslove pisati u WHERE umesto u HAVING klauzuli kad god je to moguće:

```
SELECT session_date, count(user_id) nr_sessions
FROM sessions
GROUP BY session_date
HAVING count(user_id) > 0;
```

```
SELECT session_date, count(user_id) nr_sessions
FROM sessions
WHERE user_id is not null
GROUP BY session_date
```

- Korititi UNION/UNION ALL umesto OR u WHERE klauzulama:

  SELECT id FROM products
  WHERE category = 'Electronics' OR price > 100

  =>

  SELECT * FROM products
  WHERE category = 'Electronics'

  UNION

  SELECT * FROM products
  WHERE price > 100;

- Ukoliko je moguće koristiti UNION ALL umesto UNION

Zadatak 1

- Napisati upit kojim se izdvajaju jedinstvena imena i prezimena studenata koji su rodjeni između 2000 i 2002 godine, i slušaju predmet sa sifrom R273. Ispisati podatke o najviše 10 studenata.

  Relacije na raspolaganju:

  Student(id,ime,prezime,datumRodjenja)
  Slusa(idStudenta,idPredmeta)

  Upit napisati tako da se izvršava što efikasnije. Po potrebi definisati odgovarajuće indekse.

Zadatak 2

- Tranformisati dati upit tako da je njegovo izvršavanje što efikasnije:

```
SELECT *
FROM Orders LEFT JOIN Users
                ON Orders.UserID=Users.ID
WHERE Users.ID is not NULL AND Orders.Price*0.9 <1200;
```

Relacije: Orders(Id,Price,UserID)

Users(ID,Adress)

Pretpostaviti da nad relacijama nije definisan nijedan indeks. Po potrebi definisati odgovarajuće indekse.