Fizičko projektovanje baza podataka

prema knjizi: Database Management Systems, Ramakrishnan Raghu, Gehrke Johannes

Ivana Tanasijević, ivana@matf.bg.ac.rs
Matematički fakultet, Beograd

Najbitnije merilo pri projektovanju neke baze podataka su performanse pri izvođenju upita koji se najčešće izvršavaju i uobičajenih operacija ažuriranja. Prvi korak pri postizanju dobrih performansi je dobar dizajn, kojim ćemo se mi ovde baviti.

Nakon projektovanja konceptualne sheme, što obuhvata kreiranje skupova relacija, pogleda, ograničenja, bazu podataka treba i fizički projektovati.

1. Uvod

Fizički dizajn treba biti vođen prirodom podataka i načinom na koji će se oni koristiti. Važno je razumeti radno opterećenje (eng. workload) koji baza podataka mora podržati, a koji se sastoji od kombinacije upita i operacija ažuriranja. Treba razmisliti o korisničkim zahtevima, koliko želimo da se neki upiti brzo izvršavaju ili koliko mislimo da ćemo imati transakcija u sekundi.

Opis radnog opterećenja treba da obuhvati sledeće:

1. Listu upita i njihove učestalosti u odnosu na sve upite i ažuriranja
2. Listu ažuriranja i njihove učestalosti
3. Cilj performansi za svaki tip upita ili ažuriranja.

Za svaki upit se mora odrediti:

- Kojim se relacijama pristupa
- Koji se atributi traže u SELECT iskazu
- Nad kojim atributima se vrši spajanje ili selekcija u WHERE iskazu i koliko su ti uslovi selektivni.

Za svako ažuriranje se mora odrediti:

- Nad kojim atributima se vrši spajanje ili selekcija u WHERE iskazu i koliko su ti uslovi selektivni
- Tip ažuriranja (INSERT, DELETE, UPDATE) i relacija koja se ažurira
- Za UPDATE komandu, polje koje treba ažurirati.

Efikasnost ažuriranja se može poboljšati dobrim izborom indeksa ali se mora razmišljati i o ažuriranju indeksa pri promeni podataka.

Važne odluke koje treba napraviti prilikom fizičkog projektovanja uključuju sledeće:

1. Koje indekse napraviti:
   - Koje relacije indeksirati i koje polje ili koja polja izabrati za ključ pretrage indeksa
   - Za svaki indeks odrediti da li treba biti klasterovan ili ne, da li treba biti redak ili gust.

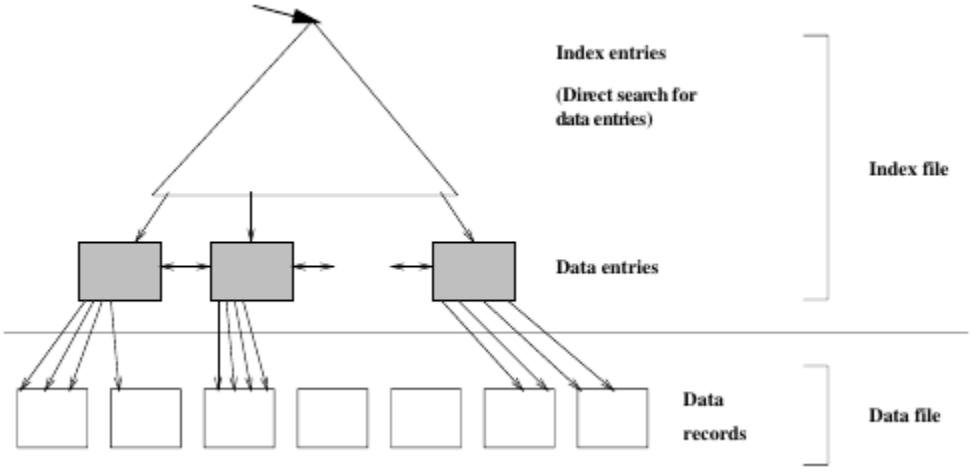
2. Da li treba praviti sledeće promene u konceptualnoj shemi u cilju poboljšanja performansi:

- Alternativne normalizovane sheme
- Denormalizacija
- Vertikalno particionisanje
- Pogledi.

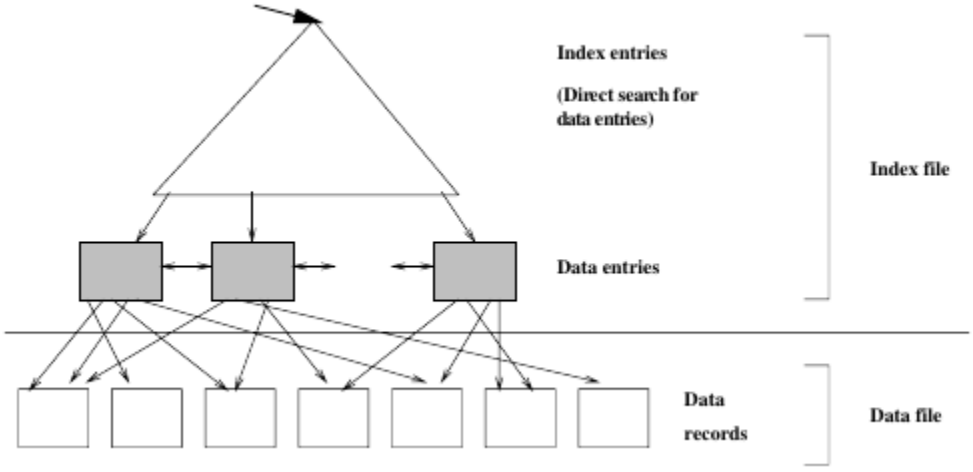
3. Da li treba preformulisati upite koji se često izvršavaju tako da budu efikasniji?

Detaljnije informacije o radnom opterećenju je teško postići samo na osnovu početnog dizajna. Kasnije podešavanje sistema na osnovu konkretne upotrebe je takođe veoma od značaja.

Kada je fajl sa podacima organizovan tako da redosled zapisa odgovara redosledu zapisa u indeksu, tada se za indeks kaže da je *klasterovan*. Sa druge strane, ukoliko je redosled zapisa u fajlu proizvoljan i ne postoji razuman način da se sličan rakav redosled napravi i u indeksu, tada je indeks *neklasterovan*.



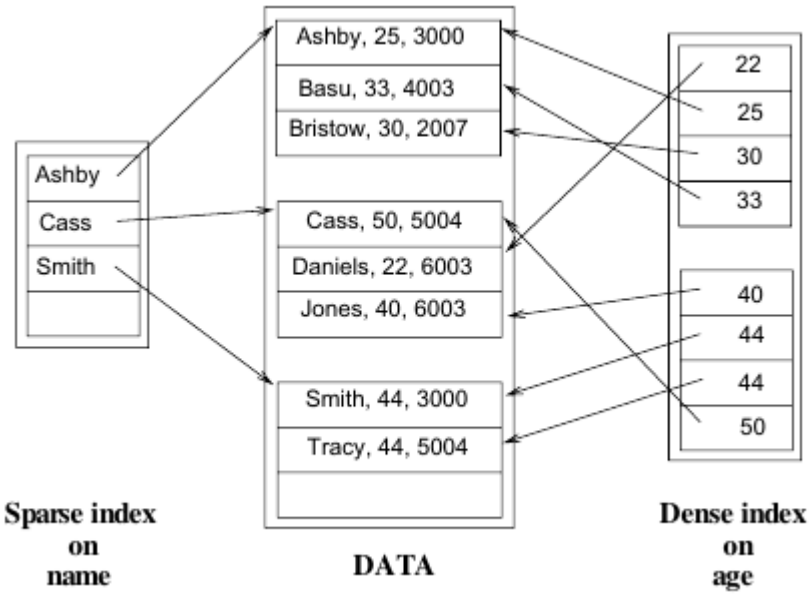
Slika 1 Klasterovan indeks



Slika 2 Neklasterovan indeks

Fajl sa podacima može biti klasterovan samo po jednom ključu pretrage, odnosno može imati jedan klasterovani indeks, dok može imati i više neklasterovanih indeksa.

Za indeks se kaže da je *gust* ukoliko sadrži najmanje jedan zapis za svaku ključnu vrednost pretrage koja se pojavljuje u zapisu indeksnog fajla. Indeks je *redak* ako sadrži po jedan zapis za svaku stranicu fajla sa podacima. Ne može se napraviti redak indeks koji nije klasterovan, stoga može postojati samo jedan redak indeks.



Slika 2 Redak na suprot gustom indeksu

2. Uputstva pri kreiranju indeksa

Prilikom razmatranja koje indekse treba napraviti, kreće se sa listom upita, uključujući i upite koji su delovi operacija ažuriranja. Uopšteno, za dobavljanje intervala podataka se preporučuje B+ stablo, za dobavljanje konkretnih vrednosti se uobičajeno koristi heš indeks. Klasterovanje će pomoći kod intervala podataka, i kod upisa sa konkretnom vrednosti ukoliko po nekoliko zapisa ima istu vrednost za polje po kome se vrši pretraga.

Pre nego što se uvede indeks, treba razmisliti kakav uticaj on ima prilikom ažuriranja i u skladu sa tim odlučiti šta je najbolje rešenje.

Uputstva za kreiranje indeksa bi bila sedeća:

Uputstvo 1 (da li napraviti indeks): Ne treba praviti indeks ukoliko on neće doprineti ubrzanju izvršavanja nekog upita, uključujući i one koji su deo ažuriranja. Kada god je moguće, izabrati indeks koji poboljšava više od jedan upit.

Uputstvo 2 (izbor ključa pretrage): Atributi koji se pominju u WHERE iskazu su kandidati za indeksiranje.

- Izdvajanje po tačnoj vrednosti sugeriše da treba razmotriti indeks nad izabranim atributima i to heš indeks.
- Intervalno izdvajanje sugeriše da treba razmotriti B+ stablo nad izdvojenim atributima.

Uputstvo 3 (ključ pretrage od više atributa): Treba ga razmotriti u sledećim situacijama:

- WHERE iskaz sadrži uslov nad više od jednog atributa.
- Ako omogućava index-only strategije (tj. tamo gde se pristup relacijama može izbeći) u važnim upitima. Ovo može voditi ka atributima koji bi bili deo ključa iako se ne pominju u WHERE iskazu.

Pri kreiranju indeksa nad ključevima pretrage sa više atributa, ukoliko se prihvataju i intervalni upiti, treba biti pažljiv oko redosleda atributa u ključu pretrage.

Uputstvo 4 (klasterovanje): Najviše jedan indeks nad datom relacijom može biti klasterovan, što veoma povećava performanse. Stoga treba pažljivo odabrati klasterovan indeks.

- Upiti intervala mogu veoma mnogo dobiti klasterovanim indeksom. Ukoliko se upotrebljava nekoliko intervalnih upita nad relacijom, koji uključuju drugačije skupove atributa, pri odabiru koji indeks treba biti klasterovan, treba razmotriti selektivnost upita i njihove relativne učestalosti u radnom opterećenju.
- Ukoliko indeks omogućava index-only strategiju za upit koji teba da ubrza, tada nije potrebno da bude klasterovan. Klasterovanje ima uticaja samo kod dobavljanja podataka.

Uputstvo 5 (heš indeks naspram stabla): Uglavnom se preporučuje B+ stablo, zato je dobro i za upite intervala i za upite sa tačnom vrednosti. Heš indeks je bolji u sledećim situacijama:

- Ovaj indeks bi trebalo da se koristi kod ugnježdene petlje spajanja, indeksirane relacije unutrašnjih relacija, gde ključ pretrage uključuje kolone po kojima se vrši spajanje. Razlog tome je taj što se izdvanje po jednakosti vrši za svaku n-torku u spoljnoj petlji.
- Kada postoje veoma važni upiti jednakosti, a pri tome ne postoje upiti intervala, uključujući atribute koji su ključ pretrage.

Uputstvo 6 (balansiranje cene ažuriranja indeksa): Nakon sastavljanja liste željenih indeksa treba razmotriti njihov uticaj na ažuriranja u radnom opterećenju.

- Ukoliko održavanje indeksa usporava operaciju ažuriranja, razmotriti njegovo odbacivanje.
- Treba imati na umu da dodavanje indeksa može ubrzati i data ažuriranja.

3. Primeri odabira indeksa

Krenimo sa jednostavnim upitom:

```
SELECT    E.ename, D.mgr
FROM      Employees E, Departments D
WHERE     D.dname='Toy' AND E.dno=D.dno
```

Relacije koje se pominju su Employees i Departments, i oba uslova u WHERE iskazu uključuju jednakosti. Prema uputstvima bi trebalo napraviti heš indekse nad uključenim atributima. Za početak jasno je da treba napraviti indeks nad atributom dname relacije Departments. Kada se izdvoje torke koje zadovoljavaju taj uslov, potrebno je imati heš indeks nad atributom dno relacije Employees.

Pretpostavimo da je WHERE iskaz izmenjen na

D.dname='Toy' AND E.dno=D.dno AND E.age=25.

Jedan dobar plan je da dobijemo torke iz relacije Departments koje zadovoljavaju uslov nad dname, da dobijemo torke iz Employees koje njima odgovaraju, izdvajanje na osnovu age se tada vrši u hodu. Ne

bi imalo mnogo smisla da imamo indeks i nad dno i nad age. Ukoliko izdvojimo torke po dname, a zatim po age, tada će biti dovoljno malo broj torki koje mogu sve stati u memoriju, pa nije potrebno praviti i indeks nad dno. Tako da ako već imamo indeks nad age, koji se na primer koristi u nekom drugom upitu, tada nećemo praviti indeks i nad dno.

Sledeći upit uključuje izdvajanje intervala:

```
SELECT    E.ename, D.dname
FROM      Employees E, Departments D
WHERE     E.sal BETWEEN 10000 AND 20000
          AND E.hobby='Stamps' AND E.dno=D.dno
```

Postoje dva izdvajanja u relaciji Employees i jedno u relaciji Departments. Stoga je jasno da relacija Employees treba da bude spoljašnja, a Departments unutrašnja, pa nad atributom dno relacije Departments treba napraviti heš indeks. B+ drvo bi pomoglo za atribut sal pogotovo ako je klasterovan. Heš indeks bi pomogao kod atributa hobby. Ukoliko jedan od ovih indeksa već postoji, treba primeniti taj indeks, zatim indeks nad dno relacije Departments, a zatim u hodu uraditi preostala izdvajanja i projekcije. Ukoliko su oba indeksa dostupna optimizator će izabrati jedan od ta dva na osnovu toga koji dobavlja manje torki, a to zavisi od konkretnih podataka. Ukoliko ima malo ljudi kojima je plata u ovom intervalu, ali ima mnogo onih sa ovim hobijem, tada je najbolji B+ indeks nad sal. U suprotnom, heš indeks nad atributom hobby je najbolji.

Ukoliko su konstante u upitu poznate, moć izdvajanja se može odrediti na osnovu statistike nad podacima. U suprotnom, čini se da bi izdvajanje jednakostima trebalo da bude selektivnije od intervalnog izdvajanja, pa bi razumna odluka bila da se kreira heš indeks nad hobby. Nekada, konstante u upitu nisu poznate, kao na primer kod nekih dinamičkih upita. Tada, ukoliko je upit veoma važan, mogli bismo izabrati B+ stablo nad sal i heš indeks nad hobby i ostaviti optimizatoru da izabere u trenutku izvršavanja.

4. Klasterovanje i indeksiranje

Upiti koji izdvajaju na osnovu intervala su dobri kandidati za klasterovani indeks:

```
SELECT    E.dno
FROM      Employees E
WHERE     E.age > 40
```

Ukoliko imamo B+ stablo nad age, to možemo iskoristiti kako bismo izdvojili torke koje zadovoljavaju uslov. Da li je ovaj indeks koristan zavisi od toga koliko je uslov selektivan. Ukoliko su na primer svi zaposleni stariji od 40, neće biti nikakve koristi od ovog indeksa. Sa druge strane, ukoliko je takvih 10% zaposlenih, korisnost indeksa zavisi od toga da li je klasterovan ili ne. Ukoliko indeks nije klasterovan, I/O operacije bi bile skuplje nego kod sekvencijalne pretrage. Sa druge strane, ukoliko se koristi B+ i klasterovanje, bilo bi potrebno približno 10% od I/O operacija sekvencijalne pretrage.

Razmotrimo sledeći primer

```
SELECT    E.dno, COUNT(*)
FROM      Employees E
WHERE     E.age > 10
```

```
GROUP BY    E.dno
```

Ukoliko je dostupno B+ stablo nad age, tada bi njega trebalo iskoristiti, zatim sortirati torke i odgovoriti na upit. Ovo nije dobro rešenje ukoliko su svi zaposleni stariji od 10 godina, a posebno je loše rešenje ukoliko indeks nije klasterovan.

Razmotrimo da li bi indeks nad dno bolje zadovoljio naše zahteve. Možemo koristiti indeks (heš ili B+ stablo) da dobijemo sve torke grupisane po dno, a zatim za svako dno izbrojati torke gde je age>10. Efikasnost zavisi od toga da li je indeks klasterovan. Ako jeste, plan je najbolji ukoliko uslov nad age nije veoma selektivan. Ukoliko indeks nije klasterovan, mogli bismo da izvršavamo jedan I/O po torci u relaciji Employees, što je jako loše. Optimizator bi izabrao plan baziran na sortiranju dno. Stoga, trebalo bi napraviti klasterovani indeks nad dno ako uslov nad age nije veoma selektivan. Ukoliko jeste selektivan, trebalo bi umesto njega kreirati indeks nad age, ne obavezno klasterovan.

Klasterovanje je takodje važno za indeks nad ključem pretrage koji ne sadrži kandidat za ključ, odnosno kada više zapisa može imati istu vrednost. Razmotrimo sledeći primer:

```
SELECT      E.dno
FROM        Employees E
WHERE       E.hobby=‘Stamps’
```

Ukoliko mnogo ljudi sakuplja markice, tada bi neklasterovan indeks nad hobby bio veoma neefikasan. Jeftinije bi bilo pretražiti sve torke i u hodu primeniti selekciju. Stoga, ukoliko je ovaj upit važan, treba razmotriti pravljenje klasterovanog indeksa. Sa druge strane, ukoliko je eid ključ relacije Employees i uslov je

```
E.hobby=‘Stamps’ and E.eid=552
```

znamo da najviše jedna torka zadovoljava uslov, pa ne bi bilo koristi od klasterovanog indeksa.

Klasterovani indeks je posebno važan kod ugnježdenih petlji. Razmotrimo primer sa početka

```
SELECT      E.ename, D.mgr
FROM        Employees E, Departments D
WHERE       D.dname=‘Toy’ AND E.dno=D.dno
```

Ukoliko je broj torki koje zadovoljavaju uslov dname mali, tada je bolje napraviti neklasterovani indeks nad dname. Sa druge strane, Employees je relacija unutrašnje petlje, a dno nije kandidat za ključ, što je jaki argument da indeks nad dno bude klasterovan, pošto se za svaku vrednost iz spoljašnje petlje vrše izdvajanja po jednakosti u unutrašnjoj petlji. Tako da je čak klasterovani indeks u unutrašnjoj petlji veći prioritet od klasterovanog indeksa u spoljašnjoj petlji. Naravno, treba uzeti u obzir i ostale parametre, kao što su moć indvajanja i važnost upita.

Sledeći primer ilustruje kako se klasterovani indeks može koristiti kod sort-merge spajanja:

```
SELECT      E.ename, D.mgr
FROM        Employees E, Departments D
WHERE       E.hobby=‘Stamps’ AND E.dno=D.dno
```

Predloženi su prethodni indeksi pod pretpostavkom da postoji samo nekoliko osoba na odeljenju za igračke. Pretpostavimo da ima dosta više osoba koje skupljaju markice. Tada bi dobro rešenje bio klasterovani indeks B+ stablo nad dno relacije Departments. Neklasterovani indeks bi u ovom slučaju bio jako skup. Ukoliko ne postoji indeks nad dno relacije Employees, možemo zatražiti torke (na primer koristeći klasterovani indeks nad hobby), primeniti uslov hobby, a zatim u hodu sortirati prema dno.

Značaj klasterovanja zavisi od broja torki koje se dobavljaju. Neklasterovani indeks je dobar koliko i klasterovani kod izdvajanja jedne torke (na primer ako je atribut kandidat za ključ). Kako broj torki raste, neklasterovani indeks postaje skuplji od sekvencijalnog dobavljanja cele relacije. To je zato što se svaka stranica dobavlja tačno jednom, dok bi se kod postojanja indeksa dobavljala onoliko puta koliko ima torki.

Uglavnom se zapisi jedne relacije smeštaju u jedan fajl. Nekada jedan fajl može sadržati i zapise iz više relacija. Ovakva postavka se naziva ko-klasterovanje dve relacije. Razmotrićemo kada možemo imati koristi od ko-klasterovanja.

Pogledajmo sledeće relacije:

Parts(pid: integer, pname: string, cost: integer, supplierid: integer)
Assembly(partid: integer, componentid: integer, quantity: integer)

Neka je svaki deo sastavljen od jednog ili više delova. Pretpostavimo da je čest upit naći sve poddelove svih delova koje dobavlja dati dobavljač:

```
SELECT    P.pid, A.componentid
FROM      Parts P, Assembly A
WHERE     P.pid = A.partid AND P.supplierid = 'Acme'
```

Dobar plan bi bio da izdvojimo torke po uslovu supplierid, a zatim da primenimo klasterovani indeks nad partid. Ukoliko je ovakvo izdvajanje uobičajeno, možemo dalje optimizovati ko-klasterovanjem dve tabele. Tada bismo mogli da smestimo dve tabele zajedno tako što bismo posle svakog dela smestili sve zapise iz relacije Assembly koji predstavljaju njegove poddelove. Ovo jeste poboljšanje, posto nije potreban indeks za petlju kojom bismo tražili poddelove za svaki deo koji je izdvojen uslovom po supplierid.

Ukoliko želimo da nađemo poddelove svih delova, dobar izbor bi bio pravljenje klasterovanog indeksa nad partid. Ili, još bolje, pravljenje klasterovanog indeksa nad pid i nad partid, a zatim izvođenje merge-sort spajanja, koristeći indekse za dobijanje sortiranih torki. Ova strategija je uporediva sa ko-klasterovanom organizacijom, koja predstavlja samo jedno čitanje skupa torki.

Pravo unapređenje ko-klasterovanjem se može videti na sledećem primeru:

```
SELECT    P.pid, A.componentid
FROM      Parts P, Assembly A
WHERE     P.pid = A.partid AND P.cost=10
```

Pretpostavimo da mnogo delova ima cenu 10. Tada bismo napravili indeks nad cost i izdvojili delove koji zadovoljavaju uslov. Ko-klasterovanjem izbegavamo pravljenje indeksa nad poljem pid. Ovakva

organizacija je veoma bitna u slučajevima kada imamo više nivoa hijerarhije i kada treba nad time da uradimo obilazak.

Sumarizovano ko-klasterovanje bi bilo:

- Može da ubrza spajanje, posebno spajanje strani ključ-ključ koje odgovara relacijama koje imaju kardinalnost 1:N
- Sekvencijalna pretraga bilo koje od relacija postaje sporija
- Ažuriranje je sporije

5. Indeksi nad ključevima pretrage koji se sastoje od više atributa

Pravljenje indeksa nad više atributa je ponekad bolje od pravljenja više indeksa nad jednim atributom. Na primer, ukoliko želimo torke prema uslovu sal=4000 and age=30, bolji je indeks <sal, age> od indeksa <sal> i <age>. Indekse nad više ključeva pretrage ćemo zvati kompozitni indeks. Oni mogu da podrže višedimenzionalne upite intervala.

Razmotrimo sledeći primer:

```
SELECT    E.eid
FROM      Employees E
WHERE     E.age BETWEEN 20 AND 30
          AND E.sal BETWEEN 3000 AND 5000
```

Kompozitni indeks <age, sal> će pomoći ukoliko je uslov selektivan. Heš indeks neće pomoći, stoga je B+ stablo preporučljivo. Takođe, klasterovani indeks ima prednost nad neklasterovanim. Ukoliko su oba uslova podjednako selektivna nije važan redosled <age, sal> ili <sal, age>. Međutim, redosled nekada može praviti veliku razliku, kao u sledećem primeru:

```
SELECT    E.eid
FROM      Employees E
WHERE     E.age = 25
          AND E.sal BETWEEN 3000 AND 5000
```

Ovde bi bilo preporučlljivo napraviti kompozitni klasterovani indeks B+ stablo <age, sal>, pošto se torke prvo sortraju po age, pa unutar toga po sal.

6. Indeksi koji omogućavaju index-only plan

Indeks koji se koristi samo za index-only čitanje ne treba da bude klasterovan pošto se pomoću njega ne dobavljaju torke. Samo gusti indeksi mogu da se koriste u ove svrhe.

Sledeći primer traži sve menadžere odeljaka koji imaju bar jednog zaposenog:

```
SELECT    D.mgr
FROM      Departments D, Employees E
WHERE     D.dno=E.dno
```

Primetiti da se ne traže torke relacije Employees. Tada bi najbolji bio gust neklasterovani indeks nad poljem dno relacije Employees. Takođe primetiti da nije bitno da li je indeks klasterovan, pošto se ne

dobavljaju torke.

Pogledajmo sledeći primer:

```
SELECT    D.mgr, E.eid
FROM      Departments D, Employees E
WHERE     D.dno=E.dno
```

Ukoliko imamo indeks nad dno relacije Employees, možemo ga iskoristiti da dobijemo torke tokom spajanja, ali ukoliko indeks nije klasterovan, ovaj pristup ne bi bio efikasan. Sa druge strane, pretpostavimo da imamo gusto B+ stablo nad <dno, eid>, tada su sve informacije koje nam trebaju sadržane u indeksu. Možemo koristi indeks kako bismo našli prvo pojavljivanje datog dno, sva ostala pojavljivanja se nalaze odmah potom u indeksu. Tada možemo da izračunamo upit koristeći umetnute petlje spajanja nad indeksima i to nad Departments kao spoljašoj relaciji i index-only pretragu kao unutrašnjoj relaciji.

Sledeći primer pokazuje kako agregirane operacije mogu uticati na izbor indeksa:

```
SELECT    E.dno, COUNT(*)
FROM      Employees E
GROUP BY  E.dno
```

Ukoliko je dostupan gusti indeks, heš ili B+ stablo, možemo odgovoriti na upit samo pretregom indeksa i prebrojavanjem broja zapisa u samom indeksu. Dakle, nije bitno da li je indeks klasterovan ili ne.

Malo izmenjen primer:

```
SELECT    E.dno, COUNT(*)
FROM      Employees E
WHERE     E.sal=10,000
GROUP BY  E.dno
```

Možemo iskoristit index-only plan ukoliko imamo kompozitno B+ stablo nad <sal, dno> ili <dno, sal>. Ukoliko koristimo prvi indeks, on se ne može koristiti ako se izmeni uslov sal > 10,000. Drugi indeks se može koristiti i u tom slučaju, s tim što je manje efikasan jer se moraju pregledati sve torke. U svim ovim slučajevima se i dalje radi o zapisima koji su samo u indeksu.

Pretpostavimo da treba da nađemo sledeće:

```
SELECT    E.dno, MIN(E.sal)
FROM      Employees E
GROUP BY  E.dno
```

Ukoliko želimo iskoristiti index-only plan, moramo uključiti oba atributa u indeks i napraviti kompozitno B+ stablo nad <dno, sal>.

Razmotrimo sledeći upit:

```
SELECT    AVG (E.sal)
FROM      Employees E
WHERE     E.age = 25
          AND E.sal BETWEEN 3000 AND 5000
```

Gusto kompozitno B+ stablo nad <age, sal> omogućava index-only plan. Takođe, to isto omogućava i gusto kompozitno B+ stablo nad <sal, age>, ali se tada više zapisa pregledava nego u prvom slučaju.

7. Pregled podešavanja baze podataka

Nakon početnog projektovanja baze podataka, sama upotreba može biti izvor veoma vrednih informacija o tome šta treba izmeniti u početnom dizajnu. Neki od uslova u početnom radnom opterećenju se mogu pokazati kao netačni, dok se drugi mogu pokazati ispravnim. Takođe, veličina podataka se ne mora poklapati sa onom koja je predviđena na početku. Neprekidno podešavanje baze podataka je važno sa aspekta performansi. Ovde ćemo pričati o podešavanju indeksa, podešavanju konceptualne sheme i podešavanju upita.

7.1 Podešavanje indeksa

Početni izbor indeksa se može poboljšati iz više razloga. Jedan od razloga je da je posmatrano radno opterećenje drugačiji od onog koji smo pretpostavljali. Takodje, mogu se identifikovati neki drugi upiti kao važni, a koje nismo uvrstili u početnom dizajnu. Takođe, može se ispostaviti da optimizator ne pravi odluke onako kako smo pretpostavljali.

7.2 Podešavanje konceptualne sheme

Ponekad je potrebno izmeniti početnu shemu na osnovu iskustva pri upotrebi baze. Važna napomena koju treba razumeti u procesu izgradnje konceptualne sheme je da, pored brige o redudantnosti koja se rešava normalizacijom, treba je napraviti i tako da bude u skladu sa upitima i ažuriranjima koja su deo radnog opterećenja. Treba uzeti u obzir sledeće:

- Možda je pogodnija 3NF umesto BCNF
- Ukoliko postoje dva načina da se shema dekomponuje treba izabrati onaj koji je bolji za konkretno radno opterećenje.
- Ponekad je dobro dalje dekomponovati relaciju koja je već u BCNF
- Ukoliko to poboljšava performanse, treba razmisliti o denormalizaciji, odnosno zameni više relacija koje su dobijene normalizacijom od jedne veće, tom većom relacijom.
- Razmatranje normalizacije se odnosi na dekompoziciju, što predstavlja vertikalno particionisanje. Druga tehnika koju treba razmotriti je horizontalna dekompozicija, kojom bi se dobile dve relacije sa istim shemama, ali sa na primer različitim ograničenjima ili indeksima.

Prilikom menjanja prvobitne sheme treba razmisliti da li je dobro napraviti poglede kojima bi se maskirale ove promene ukoliko to pojednostavljuje korišćenje baze od strane korisnika kojima je prvobitna shema prirodnija ili su se navikli na njeno korišćenje.

7.3 Podešavanje upita i pogleda

Ukoliko primetimo da se upit izvršava sporije nego što smo očekivali, trebalo bi istražiti zašto se to dešava. Često preformulacija upita uz podešavanje indeksa rešava problem. Prvo treba proveriti da li sistem koristi plan koji mi očekujemo da koristi. Neke od situacija koje optimizator ne rešava efikasno

su:

- Izdvajanje koje uključuje null vrednosti
- Izdvajanje koje sadrži aritmetičke izraze ili izraze sa stringovima ili uslovi u kojima se koristi OR. Na primer, E.age = 2 * D.age u WHERE izrazu optimizator će lepo primeniti indeks nad E.age, ali neće nad D.age.
- Optimizator ne može da prepozna sofisticirani plan kao što je index-only plan nad agregiranim upitom koji uključuje GROUP BY. Što ste više svesni prednosti i mana datog sistema, to bolje po vas.

Neki sistemi dozvoljavaju korisnicima da vode plan kojim se biraju strategije tako što mogu da postave uputstva optimizatoru oko izabira indeksa.

8. Izbori prilikom podešavanja konceptualne sheme

Ilustrovaćemo izbore u podešavanju konceptualne sheme kroz nekoliko primera koristeći sledeće sheme:

Contracts(cid: integer, supplierid: integer, projectid: integer,
deptid: integer, partid: integer, qty: integer, value: real)
Departments(did: integer, budget: real, annualreport: varchar)
Parts(pid: integer, cost: integer)
Projects(jid: integer, mgr: char(20))
Suppliers(sid: integer, address: char(50))

Koristićemo se sledećom notacijom: svaki atribut čemo obeležavati jednim slovom, dok čemo sheme obeležavati nizom slova koji odgovaraju njihovim atributima. Prema tome, relacija Contracts se obeležava sa CSJDPQV.

Postoje dva uslova ograničenja nad relacijom Contracts. Ne postoje dva različita Ugovora kojima isti projekat naručuje neki deo, što se može zapisati sa JP → C. Takođe, odeljenje naručuje najviše jedan deo od bilo kod dobavljača, što se zapisuje sa SD → P.

8.1. Postavljanje slabije normalne forme

Ova relacija nije u BCNF, ali jeste u 3NF. Da li treba da raščlanimo relaciju Contracts? Kandidat ključa su C i JP. Relacija se može zameniti relacijama CJP, SDP, CSJDQV. Ukoliko je jedan od važnih upita naći broj kopija Q dela P naručenog u ugovoru C, tada ćemo imati spajanje razdvojenih relacija, što je lošije nego da smo imali samo jednu relaciju.

8.2 Denormalizacija

Možemo ići i dalje, tako da dozvolimo neka ponavljanja. Neka je upit proveriti da je vrednost ugovora manja od budžeta odeljenja koje je obuhvaćeno ugovorom. Možda možemo odlučiti da dodamo polje budžet u Contracts. Relacija tada neće biti u 3NF.

8.3 Izbori pri dekompoziciji

- Možemo ostaviti Contracts takvu kakva jeste i prihvatiti postojeće ponavljanje
- Možemo je dovesti u BCNF sa očuvanjem svih zavisnosti ili bez nekih zavisnosti, koje na primer

možemo modelovati uslovom ograničenja, koji je pak skup.

8.4 Vertikalna dekompozicija

Pretpostavimo sa smo odlučili da raščlanimo Contracts na SDP i CSJDQV koje su u BCNF. Pretpostavimo da su sledeći upiti česti:

- Naći ugovore dobavljača S
- Naći ugovore odeljenja D.

Relacije tada možemo raščlaniti na CS, CD, CJQV.

8.5 Horizontalna dekompozicija

Neka postoje različita pravila kod obrade ugovora čija je vrednost veća od 10,000 i onih čija je vrednost manja. Jedno rešenje je da napravimo B+ stablo kojim čeo izdvajati interval. Drugo rešenje je da napravimo dve relacije LargeContracts i SmallContracts. Ovo je pogotovo primamljivo ako bi drugi upiti mogli da dobiju od uvođenja klasterovanog indeksa.

Još jedan primer bio bio sledeći: ukoliko bi relacija Contracts imala polje godina i ako bi se obrađivali ugovori samo iz neke godine, tada bi imalo smisla razdvojiti Contracts relacije po godini.

9. Izbori prilikom podešavanja upita i pogleda

Razmotrimo sledeći primer:

```
SELECT      E.dno
FROM        Employees E
WHERE       E.hobby='Stamps' OR E.age=10
```

Ukoliko imamo indekse nad hobby i nad age, optimizator to možda neće prepoznati i vršiće sekvencijalnu pretragu. Treba razmisliti o preformulacije upita tako da predstavlja uniju dva upita.

Takođe treba izbegavati skupe operacije, kao što je DISTINCT kada god je to moguće.

Ponekad se upit sa GROUP BY i HEAVING izrazima može preformulisati tako da ih ne koristi.

Često se komplikovani upiti pišu u koracima, tako da se koriste privremene relacije. Kada je moguće, to treba izbegavati. Na primer, sledeći upit:

```
SELECT      *
INTO        Temp
FROM        Employees E, Departments D
WHERE       E.dno=D.dno AND D.mgrname='Robinson'

SELECT      T.dno, AVG (T.sal)
FROM        Temp T
GROUP BY    T.dno
```

se može preformulisati na sledeći način:

```
SELECT    E.dno, AVG (E.sal)
FROM      Employees E, Departments D
WHERE     E.dno=D.dno AND D.mgrname='Robinson'
GROUP BY  E.dno
```

U poslednjem primeru se može koristiti index-only plan, tako da se i ne traže torke iz relacije, što nije slučaj u prethodnom primeru.

Sa druge strane, ukoliko optimizator nije u stanju da nađe dobar plan za komplikovane upite (uglavnom su to ugnježđeni upiti), treba razmisliti da li bi se mogli preformulisati tako da koriste privremene relacije. Kada god je moguće treba preformulisati ugnježđeni upit kako bi se toga oslobodili.

Pitanja i zadaci iz navedene knjige

Exercise 16.1 Consider the following relations:

Emp(eid: integer, ename: varchar, sal: integer, age: integer, did: integer)
Dept(did: integer, budget: integer, floor: integer, mgr eid: integer)

Salaries range from $10,000 to $100,000, ages vary from 20 to 80, each department has about five employees on average, there are 10 floors, and budgets vary from $10,000 to $1,000,000. You can assume uniform distributions of values.

For each of the following queries, which of the listed index choices would you choose to speed up the query? If your database system does not consider index-only plans (i.e., data records are always retrieved even if enough information is available in the index entry), how would your answer change? Explain briefly.

1. Query: Print ename, age, and sal for all employees.
   (a) Clustered, dense hash index on ename, age, sal fields of Emp.
   (b) Unclustered hash index on ename, age, sal fields of Emp.
   (c) Clustered, sparse B+ tree index on ename, age, sal fields of Emp.
   (d) Unclustered hash index on eid, did fields of Emp.
   (e) No index.

2. Query: Find the dids of departments that are on the 10th floor and that have a budget of less than $15,000.
   (a) Clustered, dense hash index on the floor field of Dept.
   (b) Unclustered hash index on the floor field of Dept.
   (c) Clustered, dense B+ tree index on floor, budget fields of Dept.
   (d) Clustered, sparse B+ tree index on the budget field of Dept.
   (e) No index.

3. Query: Find the names of employees who manage some department and have a salary greater than $12,000.
   (a) Clustered, sparse B+ tree index on the sal field of Emp.
   (b) Clustered hash index on the did field of Dept.
   (c) Unclustered hash index on the did field of Dept.

(d) Unclustered hash index on the did field of Emp.
(e) Clustered B+ tree index on sal field of Emp and clustered hash index on the did field of Dept.

4. Query: Print the average salary for each department.
(a) Clustered, sparse B+ tree index on the did field of Emp.
(b) Clustered, dense B+ tree index on the did field of Emp.
(c) Clustered, dense B+ tree index on did, sal fields of Emp.
(d) Unclustered hash index on did, sal fields of Emp.
(e) Clustered, dense B+ tree index on the did field of Dept.

Exercise 16.2 Consider the following relation:

Emp(eid: integer, sal: integer, age: real, did: integer)

There is a clustered index on eid and an unclustered index on age.

1. Which factors would you consider in deciding whether to make an index on a relation a clustered index? Would you always create at least one clustered index on every relation?
2. How would you use the indexes to enforce the constraint that eid is a key?
3. Give an example of an update that is definitely speeded up because of the available indexes.
4. Give an example of an update that is definitely slowed down because of the indexes.
5. Can you give an example of an update that is neither speeded up nor slowed down by the indexes?

Exercise 16.3 Consider the following BCNF schema for a portion of a simple corporate database (type information is not relevant to this question and is omitted):

Emp (eid, ename, addr, sal, age, yrs, deptid)
Dept (did, dname, floor, budget)

Suppose you know that the following queries are the six most common queries in the workload for this corporation and that all six are roughly equivalent in frequency and importance:

List the id, name, and address of employees in a user-specified age range.
List the id, name, and address of employees who work in the department with a user-specified department name.
List the id and address of employees with a user-specified employee name.
List the overall average salary for employees.
List the average salary for employees of each age; that is, for each age in the database,
list the age and the corresponding average salary.
List all the department information, ordered by department floor numbers.

1. Given this information, and assuming that these queries are more important than any updates, design a physical schema for the corporate database that will give good performance for the expected workload. In particular, decide which attributes will be indexed and whether each index will be a clustered index or an unclustered index. Assume that B+ tree indexes are the only index type supported by the DBMS and that both single- and multiple-attribute keys are permitted. Specify your physical design by identifying the attributes that you recommend indexing on via clustered or unclustered B+ trees.

2. Redesign the physical schema assuming that the set of important queries is changed to be the following:

List the id and address of employees with a user-specified employee name.
List the overall maximum salary for employees.
List the average salary for employees by department; that is, for each deptid value, list the deptid value and the average salary of employees in that department.
List the sum of the budgets of all departments by floor; that is, for each floor, list the floor and the sum.

Exercise 16.4 Consider the following BCNF relational schema for a portion of a university database (type information is not relevant to this question and is omitted):

Prof(ssno, pname, office, age, sex, specialty, dept did)
Dept(did, dname, budget, num majors, chair ssno)

Suppose you know that the following queries are the five most common queries in the workload for this university and that all five are roughly equivalent in frequency and importance:

List the names, ages, and offices of professors of a user-specified sex (male or female) who have a user-specified research specialty (e.g., recursive query processing). Assume that the university has a diverse set of faculty members, making it very uncommon for more than a few professors to have the same research specialty.
List all the department information for departments with professors in a user-specified age range.
List the department id, department name, and chairperson name for departments with a user-specified number of majors.
List the lowest budget for a department in the university.
List all the information about professors who are department chairpersons.

These queries occur much more frequently than updates, so you should build whatever indexes you need to speed up these queries. However, you should not build any unnecessary indexes, as updates will occur (and would be slowed down by unnecessary indexes). Given this information, design a physical schema for the university database that will give good performance for the expected workload. In particular, decide which attributes should be indexed and whether each index should be a clustered index or an unclustered index. Assume that both B+ trees and hashed indexes are supported by the DBMS and that both single- and multiple-attribute index search keys are permitted.

1. Specify your physical design by identifying the attributes that you recommend indexing on, indicating whether each index should be clustered or unclustered and whether it should be a B+ tree or a hashed index.

2. Redesign the physical schema assuming that the set of important queries is changed to be the following:

List the number of different specialties covered by professors in each department, by department.
Find the department with the fewest majors.
Find the youngest professor who is a department chairperson.

Exercise 16.5 Consider the following BCNF relational schema for a portion of a company database (type information is not relevant to this question and is omitted):

Project(pno, proj name, proj base dept, proj mgr, topic, budget)
Manager(mid, mgr name, mgr dept, salary, age, sex)

Note that each project is based in some department, each manager is employed in some department, and the manager of a project need not be employed in the same department (in which the project is based). Suppose you know that the following queries are the five most common queries in the workload for this university and that all five are roughly equivalent in frequency and importance:

List the names, ages, and salaries of managers of a user-specified sex (male or female) working in a given department. You can assume that while there are many departments, each department contains very few project managers.
List the names of all projects with managers whose ages are in a user-specified range (e.g., younger than 30).
List the names of all departments such that a manager in this department manages a project based in this department.
List the name of the project with the lowest budget.
List the names of all managers in the same department as a given project.

These queries occur much more frequently than updates, so you should build whatever indexes you need to speed up these queries. However, you should not build any unnecessary indexes, as updates will occur (and would be slowed down by unnecessary indexes). Given this information, design a physical schema for the company database that will give good performance for the expected workload. In particular, decide which attributes should be indexed and whether each index should be a clustered index or an unclustered index. Assume that both B+ trees and hashed indexes are supported by the DBMS, and that both single- and multiple-attribute index keys are permitted.

1. Specify your physical design by identifying the attributes that you recommend indexing on, indicating whether each index should be clustered or unclustered and whether it should be a B+ tree or a hashed index.

2. Redesign the physical schema assuming that the set of important queries is changed to be the following:

Find the total of the budgets for projects managed by each manager; that is, list proj mgr and the total of the budgets of projects managed by that manager, for all values of proj mgr.
Find the total of the budgets for projects managed by each manager but only for managers who are in a user-specified age range.
Find the number of male managers.
Find the average age of managers.

Exercise 16.6 The Globetrotters Club is organized into chapters. The president of a chapter can never serve as the president of any other chapter, and each chapter gives its president some salary. Chapters keep moving to new locations, and a new president is elected when (and only when) a chapter moves. The above data is stored in a relation G(C,S,L,P), where the attributes are chapters (C), salaries (S), locations (L), and presidents (P). Queries of the following form are frequently asked, and you must be able to answer them without computing a join: “Who was the president of chapter X when it was in

location Y?”

1. List the FDs that are given to hold over G.
2. What are the candidate keys for relation G?
3. What normal form is the schema G in?
4. Design a good database schema for the club. (Remember that your design must satisfy the query requirement stated above!)
5. What normal form is your good schema in? Give an example of a query that is likely to run slower on this schema than on the relation G.
6. Is there a lossless-join, dependency-preserving decomposition of G into BCNF?
7. Is there ever a good reason to accept something less than 3NF when designing a schema for a relational database? Use this example, if necessary adding further constraints, to illustrate your answer.

Exercise 16.7 Consider the following BCNF relation, which lists the ids, types (e.g., nuts or bolts), and costs of various parts, along with the number that are available or in stock:

Parts (pid, pname, cost, num avail)

You are told that the following two queries are extremely important:

Find the total number available by part type, for all types. (That is, the sum of the num avail value of all nuts, the sum of the num avail value of all bolts, etc.)
List the pids of parts with the highest cost.

1. Describe the physical design that you would choose for this relation. That is, what kind of a file structure would you choose for the set of Parts records, and what indexes would you create?
2. Suppose that your customers subsequently complain that performance is still not satisfactory (given the indexes and file organization that you chose for the Parts relation in response to the previous question). Since you cannot afford to buy new hardware or software, you have to consider a schema redesign. Explain how you would try to obtain better performance by describing the schema for the relation(s) that you would use and your choice of file organizations and indexes on these relations.
3. How would your answers to the above two questions change, if at all, if your system did not support indexes with multiple-attribute search keys?

Exercise 16.8 Consider the following BCNF relations, which describe employees and departments that they work in:

Emp (eid, sal, did)
Dept (did, location, budget)

You are told that the following queries are extremely important:

Find the location where a user-specified employee works.
Check whether the budget of a department is greater than the salary of each employee in that department.

1. Describe the physical design that you would choose for this relation. That is, what kind of a file

structure would you choose for these relations, and what indexes would you create?

2. Suppose that your customers subsequently complain that performance is still not satisfactory (given the indexes and file organization that you chose for the relations in response to the previous question). Since you cannot afford to buy new hardware or software, you have to consider a schema redesign. Explain how you would try to obtain better performance by describing the schema for the relation(s) that you would use and your choice of file organizations and indexes on these relations.

3. Suppose that your database system has very inefficient implementations of index structures. What kind of a design would you try in this case?

Exercise 16.9 Consider the following BCNF relations, which describe departments in a company and employees:

Dept(did, dname, location, managerid)
Emp(eid, sal)

You are told that the following queries are extremely important:

List the names and ids of managers for each department in a user-specified location, in alphabetical order by department name.
Find the average salary of employees who manage departments in a user-specified location. You can assume that no one manages more than one department.

1. Describe the file structures and indexes that you would choose.
2. You subsequently realize that updates to these relations are frequent. Because indexes incur a high overhead, can you think of a way to improve performance on these queries without using indexes?

Exercise 16.10 For each of the following queries, identify one possible reason why an optimizer might not find a good plan. Rewrite the query so that a good plan is likely to be found. Any available indexes or known constraints are listed before each query; assume that the relation schemas are consistent with the attributes referred to in the query.

1. An index is available on the age attribute.

```
SELECT E.dno
FROM Employee E
WHERE E.age=20 OR E.age=10
```

2. A B+ tree index is available on the age attribute.

```
SELECT E.dno
FROM Employee E
WHERE E.age<20 AND E.age>10
```

3. An index is available on the age attribute.

```
SELECT E.dno
FROM Employee E
```

```
WHERE 2*E.age<20
```

4. No indexes are available.

```
SELECT DISTINCT *
FROM Employee E
```

5. No indexes are available.

```
SELECT AVG (E.sal)
FROM Employee E
GROUP BY E.dno
HAVING E.dno=22
```

6. sid in Reserves is a foreign key that refers to Sailors.

```
SELECT S.sid
FROM Sailors S, Reserves R
WHERE S.sid=R.sid
```

Exercise 16.11 Consider the following two ways of computing the names of employees who earn more than $100,000 and whose age is equal to their manager's age. First, a nested query:

```
SELECT E1.ename
FROM Emp E1
WHERE E1.sal > 100 AND E1.age = ( SELECT E2.age
                                  FROM Emp E2, Dept D2
                                  WHERE E1.dname = D2.dname
                                        AND D2.mgr = E2.ename )
```

Second, a query that uses a view definition:

```
SELECT E1.ename
FROM Emp E1, MgrAge A
WHERE E1.dname = A.dname AND E1.sal > 100 AND E1.age = A.age

CREATE VIEW MgrAge (dname, age)
        AS SELECT D.dname, E.age
           FROM Emp E, Dept D
           WHERE D.mgr = E.ename
```

1. Describe a situation in which the first query is likely to outperform the second query.
2. Describe a situation in which the second query is likely to outperform the first query.
3. Can you construct an equivalent query that is likely to beat both these queries when every employee who earns more than $100,000 is either 35 or 40 years old? Explain briefly.