Eksterno sortiranja

Sortiranje se koristi za implementaciju mnogih relacionih operacija. Pri tome je osnovni problem to što su relacije obično jako velike i ne mogu se smestiti u operativnu memoriju, tako da se ne mogu koristiti tradicionalni algoritmi sortiranja koji podrazumevaju da se svi podaci nalaze u operativnoj memoriji.

Kod eksternog sortiranja se kombinuje sortiranje u memoriji sa određenim tehnikama koje imaju za cilj da minimiziraju broj ulazno/izlaznih, tj. IO operacija. S obzirom na to da vreme izvršavanja sortiranja dominantno zavisi od broja IO operacija praksa je da se „cena koštanja“, odnosno vreme izvršavanja meri brojem stranica koje je potrebno preneti sa diska u memoriji, i obrnuto iz memorije na disk.

Eksterno sortiranje ima dve glavne komponente: 1. Izvršavanje kojim se sortiraju zapisi koristeći memorijske bafere u operativnoj memoriji; 2. Neophodne IO operacije za prenos zapisa između diska i operativne memorije.

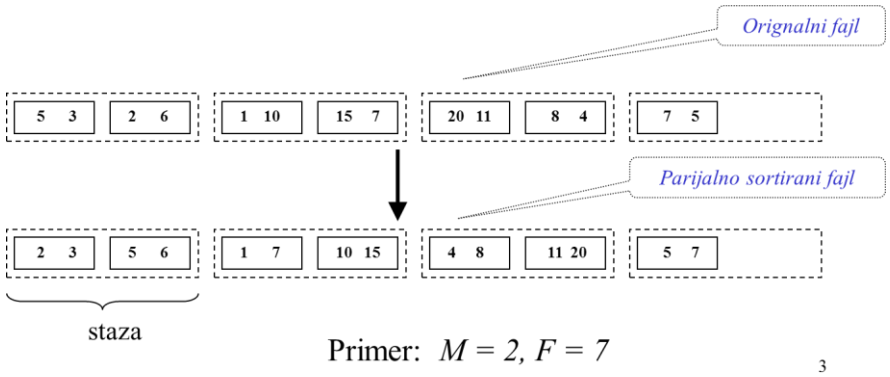
Jednostavan algoritam eksternog sortiranja

M = broj stranica (odnosno veličina bafera) na raspolaganju u operativnoj memoriji

F = broj stranica koje u relaciji koju je potrebno sortirati

Algoritam ima dve faze:

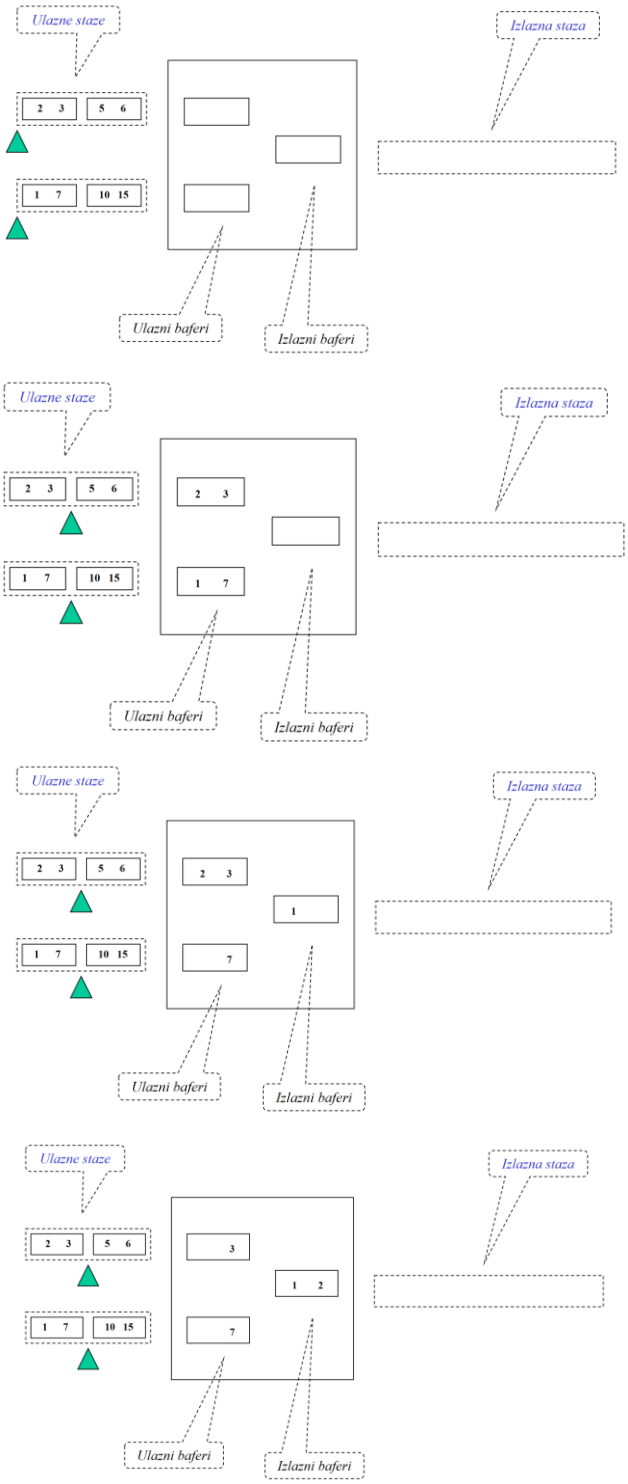
1. Fazu parcijalnog sortiranja: sortira se M stranica u baferu istovremeno; kreira se F/M sortiranih staza (eng. run) na disku; cena izvršavanja je 2F.

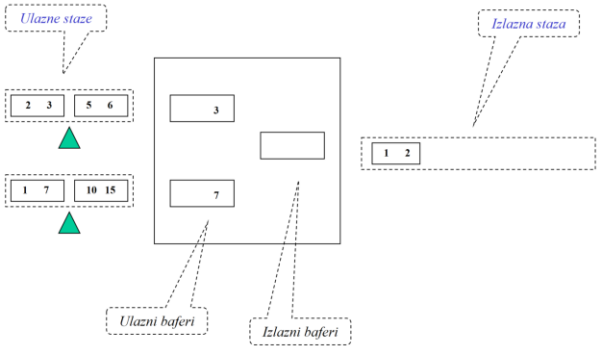


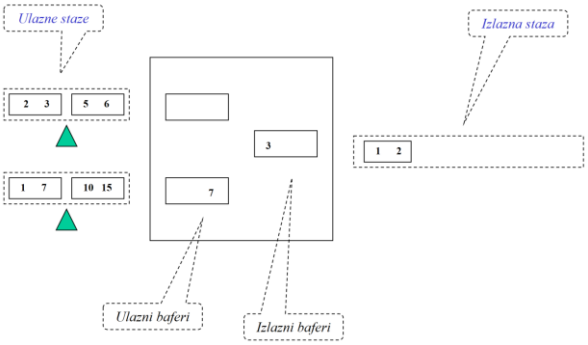
2. Faza spajanja: spajaju se sve staze u jednu koristeći M-1 stranica bafera za ulaz i 1 stranice bafera za izlaz.

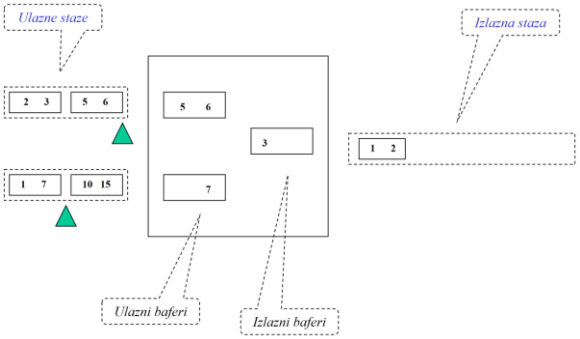
   U fazi spajanja najpre se sve staze podele u grupe od po M-1 staza, tako da se u svakom koraku faze spajanja sve staze jedne grupe spoje u jednu stazu; cena izvršavanja je 2F.

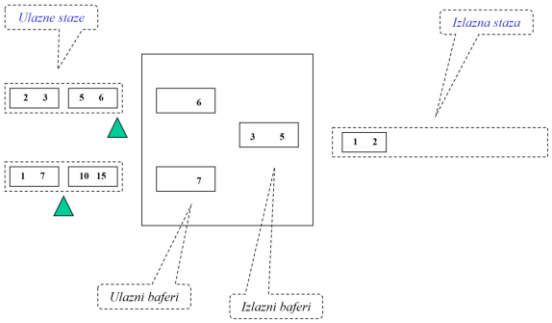
   Svaki korak faze spajanja smanjuje broj staza za faktor M-1.


Ulazne staze
Izlazna staza
2 3
5 6
1 7
10 15
Ulazni baferi
Izlazni baferi
Ulazne staze
Izlazna staza
2 3
5 6
1 7
10 15
2 3
1 7
Ulazni baferi
Izlazni baferi
Ulazne staze
Izlazna staza
2 3
5 6
1 7
10 15
2 3
1
7
Ulazni baferi
Izlazni baferi
Ulazne staze
Izlazna staza
2 3
5 6
1 7
10 15
3
1 2
7
Ulazni baferi
Izlazni baferi









Cena izvršavanja cele faze spajanja:

$(F/M)/(M-1)^K$ je broj staza nakon K koraka faze spajanja

$\lceil \text{Log}_{M-1}(F/M) \rceil$ je broj koraka spajanja potrebnih da se spoji početni skup od F/M sortiranih staza

Dakle, cena izvršavanje čitave faze je: $\lceil 2F \text{ Log}_{M-1}(F/M) \rceil \approx 2F \text{ Log}_{M-1}(F-1)$

Ukupna cena izvršavanje celog algoritma:

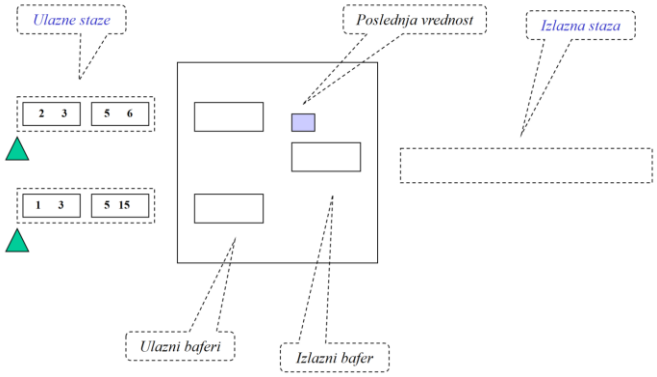
cena izvršavanja faze parcijalnog sortiranja + cena izvršavanja faze spajanja je $\approx 2F \text{ Log}_{M-1}(F)$

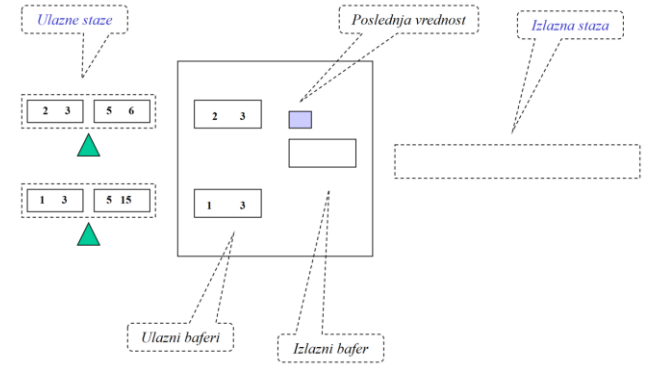
Eliminacija duplikata

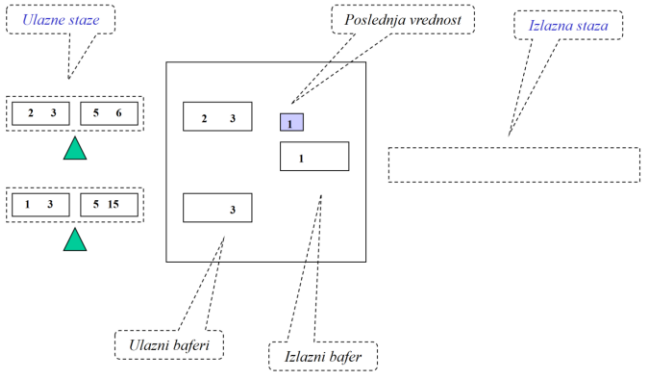
Jako važna operacija prilikom izračunavanja relacionih operatora projekcije, unije i razlike jeste eliminacija duplikata.

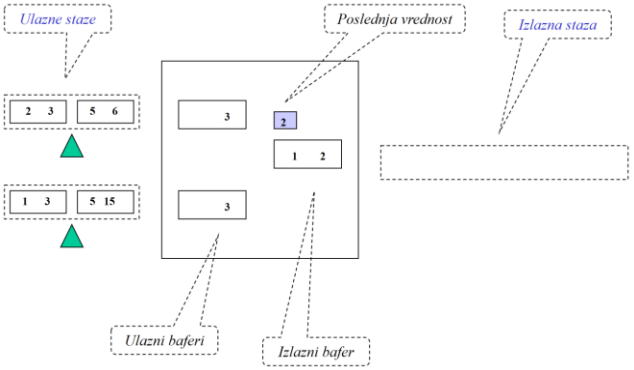
Algoritam za eliminaciju duplikata može biti: 1. Parcijalno sortiranje; 2. U poslednjem stadijumu koraka spajanja eliminisati duplikate

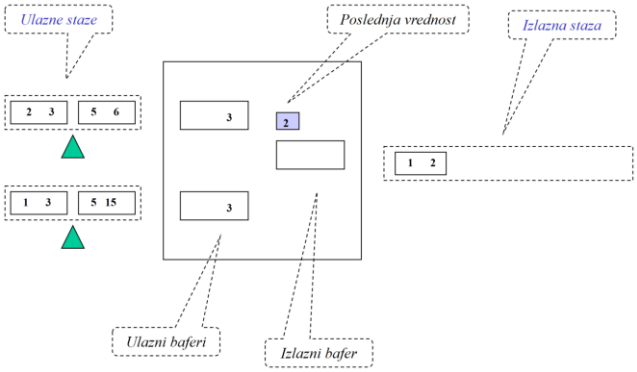
Cena izvršavanja: nema dodatnog troška u odnosu na sortiranje

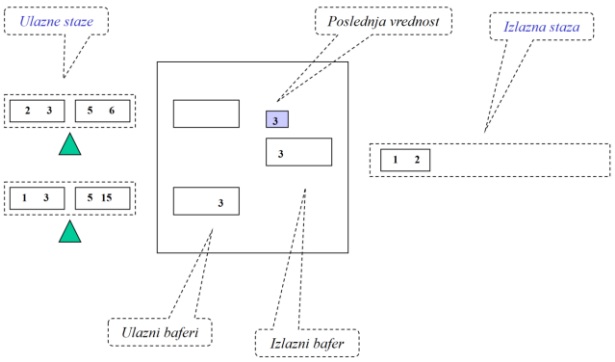


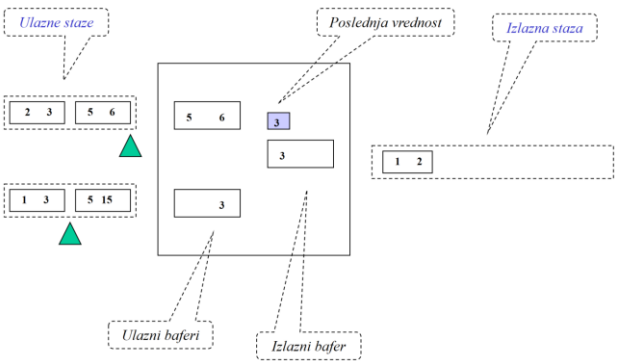


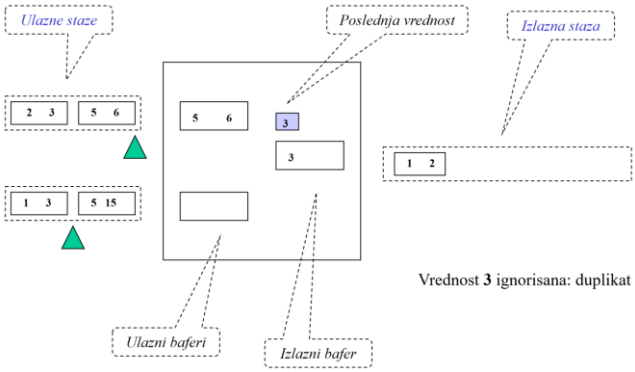



Ulazne staze
Poslednja vrednost
Izlazna staza
2 3
5 6
1 3
5 15
3
2
1 2
3
Ulazni baferi
Izlazni bafer

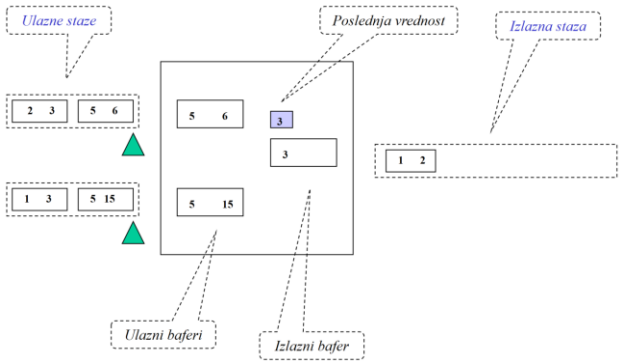

Ulazne staze
Poslednja vrednost
Izlazna staza
2 3
5 6
1 3
5 15
3
2
1 2
3
Ulazni baferi
Izlazni bafer

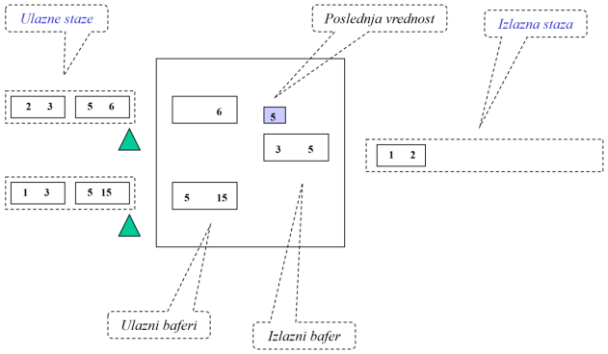

Ulazne staze
Poslednja vrednost
Izlazna staza
2 3
5 6
1 3
5 15
3
3
1 2
3
Ulazni baferi
Izlazni bafer

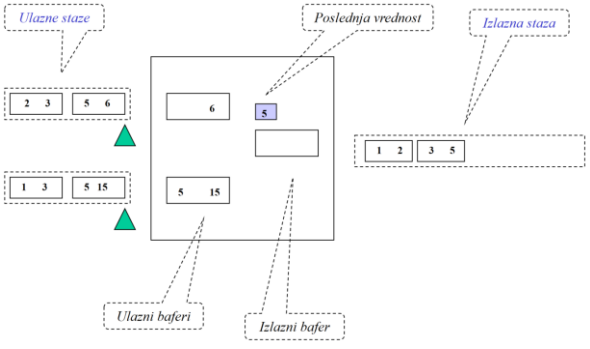

Ulazne staze
Poslednja vrednost
Izlazna staza
2 3
5 6
1 3
5 15
5 6
3
3
1 2
3
Ulazni baferi
Izlazni bafer



Vrednost **3** ignorisana: duplikat







Izračunavanje projekcije na bazi sortiranja

Algoritam: 1. Sortiranje redova relacije po ceni od 2F Log_{M-1} (F); 2. Eliminisanje neželjenih kolona za vreme faze parcijalnog sortiranja (nema dodatnog troška); 3. Eliminisanje duplikata u poslednjem koraku spajanja (nema dodatnog troška).

Cena izvršavanja: nema dodatnog troška u odnosu na sortiranje

Izračunavanje projekcije na bazi heširanja

Algoritam:

1. Faza 1: Unos redova; Uklanjanje neželjenih kolona (projekcija); Heširanje preostalih kolona koristeći heš funkciju sa opsegom 1 … M-1 tako da se kreira M-1 baketa na disku.
   Cena izvršavanja: 2F
2. Faza 2: Sortiranje svakog baketa kako bi se eliminisali duplikati;
   Cena izvršavanja (pretpostavka da svaki baket može da stane u bafer od M-1 stranice): 2F

Cena izvršavanja celog algoritma: 4F

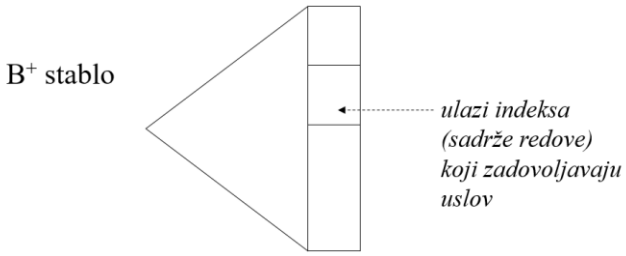
Izračunavanje restrikcije σ

Algoritmi ukoliko je restrikcija tipa $\sigma_{(attr\ op\ vrednost)}$:

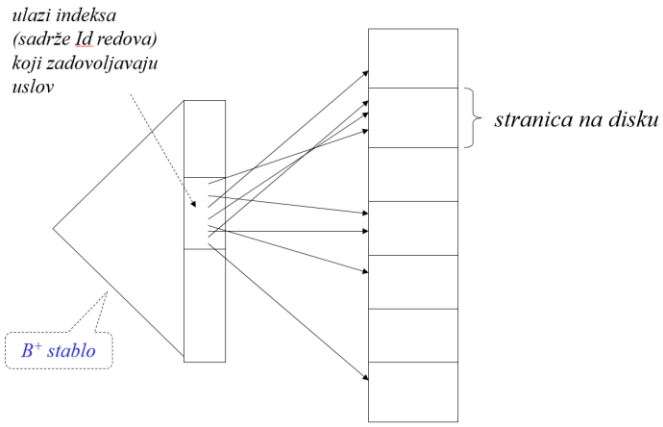
1. Ukoliko nema indeksa nad attr i redovi nisu sortirani po attr:
   Skeniranje svih stranica na disku kako bi bili određeni svi redovi koji zadovoljavaju uslov restrikcije;
   Cena izvršavanja: F

2. Ukoliko nema indeksa nad attr, ali redovi jesu sortirani po attr i ukoliko je op jedna od operacija =, >, < onda:
   Koristeći binarnu pretragu (po ceni od $Log_2(F)$) locirati prvu stranicu koja sadrži red za koji važi uslov (attr=vrednost);
   Nastaviti skeniranje dalje da se dohvate svi redovi koji zadovoljavaju uslov (attr op vrednost)
   Cena izvršavanja: $Log_2(F)$ + (broj stranica u kojima se nalaze redovi koji zadovoljavaju uslov restrikcije)

3. Ukoliko postoji klasterovani indeks na bazi B^+ stabla podignut nad attr (za slučaj da je op = ili pretraga opsega):
   Locirati prvi ulaz u indeks koji odgovara redu za koji važi (attr=vrednost);
   Svi redovi koji zadovoljavaju uslov se nalaze u sukcesivnim stranicama na disku, potrebno je dohvati ih;
   Cena izvršavanja: dubina B^+ stabla+ (broj stranica u kojima se nalaze redovi koji zadovoljavaju uslov restrikcije)



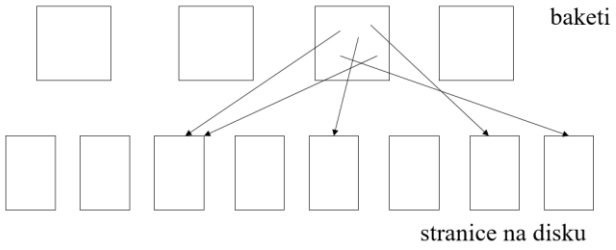
4. Ukoliko postoji neklasterovani indeks na bazi B^+ stabla podignut nad attr (za slučaj da je op = ili pretraga opsega):
   Locirati prvi ulaz u indeks koji odgovara redu za koji važi (attr=vrednost);
   Ulazi u indeks koji predstavljaju pokazivače na redove koji zadovoljavaju uslov se nalaze u sekvenci u sukcesivnim stranicama indeksa pa je zbog toga potrebno:
   a. Skenirati ulaze indeksa i sortirati id-jeve redova kako bi bile određene stranice koje sadrže te redove koji zadovljavaju uslov; b. Dohvatiti svaku stranicu koja sadrži bar jedan red koji zadovoljava uslov.
   Cena izvršavanja: dubina B^+ stabla+ (broj redovi koji zadovoljavaju uslov restrikcije)



Algoritmi ukoliko je restrikcija tipa $\sigma_{(attr = vrednost)}$:
Samo za pretrage po jednakosti, se može koristiti heš indeks. Njegova cena izvršavanja je 1,2 zbog toga što obično postoji potreba za pretragom baketa (nastalih usled overflow lanaca). Nalazi se jedinstven baket koji sadrži sve ulaze koji zadovoljavaju uslov. Postoje dva slučaja:

1. Ukoliko postoji klasterovani heš indeks podignut nad attr:
   Svi redovi koji zadovoljavaju uslov nalaze se u jednom baketu (obično veličine par stranica);
   Cena izvršavanja: 1,2 + (broj stranica koje zauzima jedan baket)

2. Ukoliko postoji neklasterovani heš indeks podignut nad attr:
   Potrebno je sortirati Id-jeve koji su pronađeni u baketu kako bi se identifikovale sve stranice koje sadrže redove koji zadovoljavaju uslov; Svaka stranica koja sadrži bar jedan takav red mora biti dohvaćena;
   Cena izvršavanja: 1,2 + min(broj redova u baketu, broj stranica fajla)



Izračunavanje spajanja koristeći blok ugnježdene petlje (block-nested loops)

Za izračunavanja spajanja relacija r i s, odnosno r $\infty_{A=B}$ s algoritam bi bio:

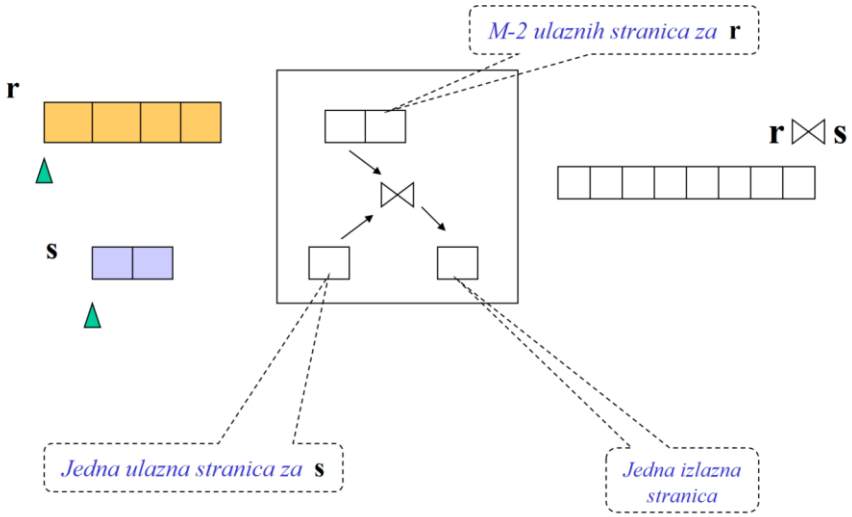
foreach page $\mathbf{p_r}$ in **r do**
  foreach page $\mathbf{p_s}$ in **s do**
    output $\mathbf{p_r}$ $\mathbf{\infty_{A=B}}$ $\mathbf{p_s}$

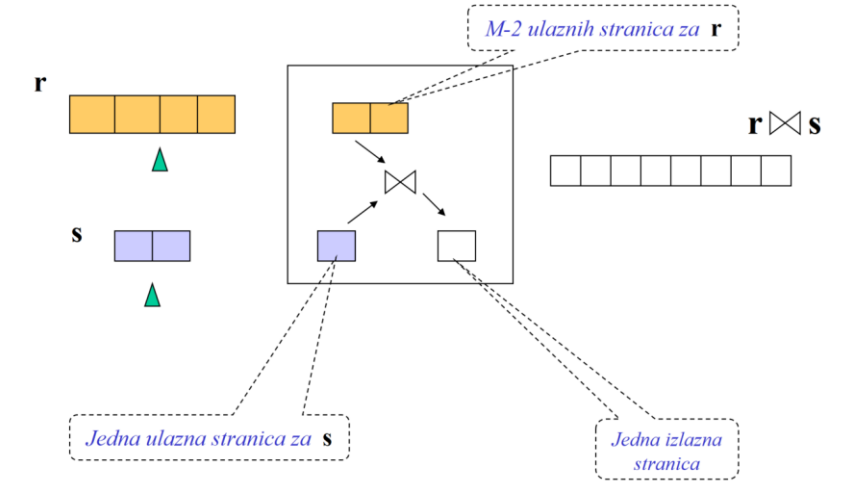
Cena izvršavanja: $ß_r + ß_r * ß_s$ + cena ispisa rezultata
Gde $ß_r$ i $ß_s$ predstavljaju broj stranica u relacijama r i s. Dakle, relacija s će biti skenirana $ß_r$ puta.

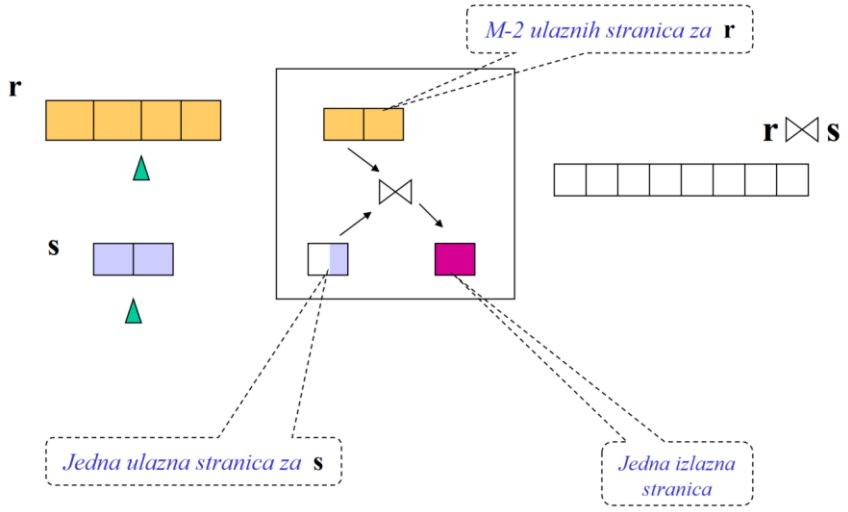
Drugim rečima treba birati manju relaciju za spoljnu petlju, jer ukoliko je $ß_r < ß_s$ onda je $ß_r + ß_r * ß_s < ß_s + ß_r * ß_s$

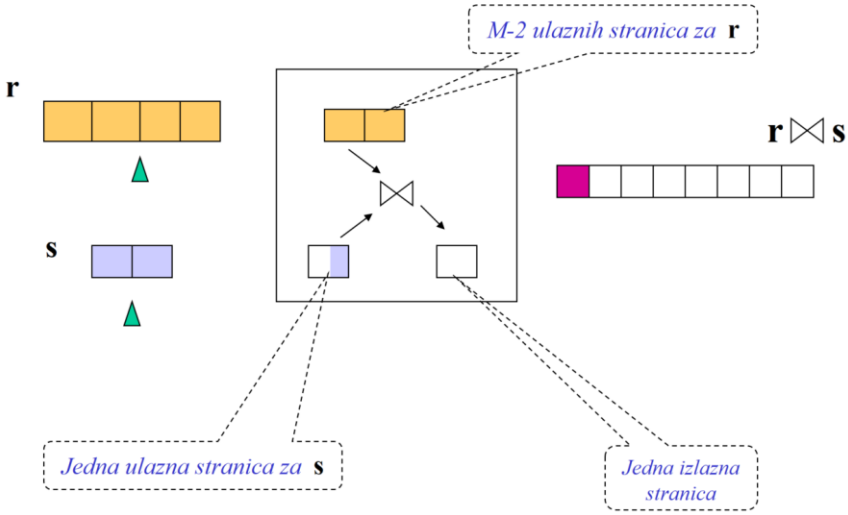
Ukoliko bi se iskoristile M stranica baferu, tako da se u M-2 stranice učita deo stranica relacija r, a potom 1 stranica iskoristi za učitavanje jedne po jedne stranice relacije s, onda se cena izvršavanja može smanjiti jer je broj skeniranja relacije s jednak $ß_r/(M-2)$.

Cena izvršavanja se može smanjiti na $ß_r + ß_r/(M-2) * ß_s$ + cena ispisa rezultata




M-2 ulaznih stranica za r
r
r ⋈ s
s
Jedna ulazna stranica za s
Jedna izlazna stranica


M-2 ulaznih stranica za r
r
r ⋈ s
s
Jedna ulazna stranica za s
Jedna izlazna stranica


M-2 ulaznih stranica za r
r
r ⋈ s
s
Jedna ulazna stranica za s
Jedna izlazna stranica

Izračunavanje spajanja koristeći indeks ugnježdene petlju (index-nested loop)

Za izračunavanja spajanja umesto skeniranja relacije s koristiti indeks nad relacijom s.

Algoritam bi bio:

foreach tuple $\mathbf{t_r}$ in **r** **do** **{**
use index to find all tuples $\mathbf{t_s}$ in **s** satisfying $\mathbf{t_r.A=t_s.B}$;
output ($\mathbf{t_r}$, $\mathbf{t_s}$)
}

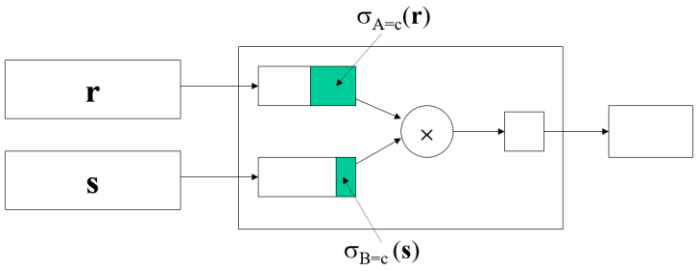
Cena izvršavanja: $ß_r + \tau_r * \omega$ + cena ispisa rezultata
Gde je τ_r broj redova u relaciji r, dok je ω prosečna cena dohvatanja svih redova u relaciji s koje odgovaraju t_r.

Drugim rečima, ovaj algoritam postaje efikasan ukoliko postoji mali broj redova u relaciji s koji odgovaraju redovima u relaciji r. Odnosno, ukoliko je ω mala vrednost i ukoliko je indeks nad s klasterovan.

Izračunavanje spajanja koristeći spajanje tokom sortiranja (sort-merge join)

Algoritam bi bio:

sort r on A;
sort s on B;
while !*eof*(r) and !*eof*(s) do **{**
scan **r** and **s** concurrently until $\mathbf{t_r.A=t_s.B=c}$;
output $\mathbf{\sigma_{A=c}(r) \times \sigma_{B=c}(s)}$
}



Cena izvršavanja: cena sortiranja + cena spajanja + cena ispisa rezultata
cena sortiranja, uz pretpostavku da je na raspolaganju M stranica bafera: $2ß_r \, Log_{M-1}(ß_r) + 2ß_s \, Log_{M-1}(ß_s)$
cena spajanja:
skeniranja $\sigma_{A=c}(r)$ i $\sigma_{B=c}(s)$ mogu biti kombinovana sa poslednjim korakom sortiranja relacija r i s, tako da nema dodatnog troška.
cena Dekartovog spajanja $\sigma_{A=c}(r)$ x $\sigma_{B=c}(s)$ zavisi od toga da li $\sigma_{A=c}(r)$ može da stane u bafer; ukoliko može onda nema

dodatnog troška; ukoliko ne može onda za svako takvo A=c treba $\sigma_{A=c}(r)$ x $\sigma_{B=c}(s)$ računati kao blok-ugnježdene petlje, tako da je cena ustvari cena spajanja.

Izračunavanje spajanja koristeći heš spajanje (hash join)

Algoritam bi bio:

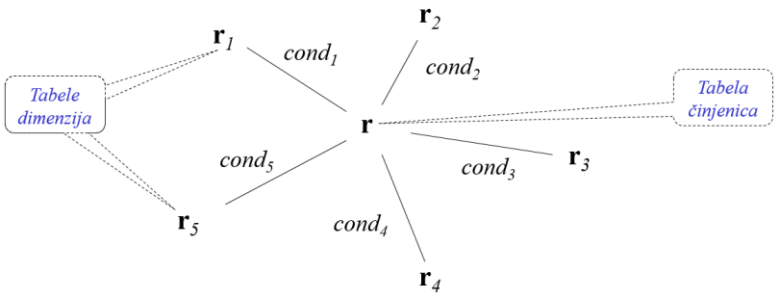
Step1: *hash* **r** on **A** and *hash* **s** on **B** into the same set of buckets;
Step2: *since matching tuples must be in same bucket, read each bucket in turn and output the result of join;*

Cena izvršavanja: 3 ($ß_r + ß_s$) + cena ispisa rezultata (pod pretpostavkom da svaki baket može da stane u memoriju)

Izračunavanje spajanja koristeći zvezda spajanje (star join)

Ukoliko je relaciona šema oblika zvezde, onda je na primer potrebno spajanje tipa:

$\mathbf{r} \infty_{cond1} \mathbf{r}_1 \infty_{cond2} \ldots \infty_{condn} \mathbf{r}_n$ pri tome svaki od uslova $cond_i$ uključuje samo atribute relacija $\mathbf{r}_i$ i **r**



Algoritam bi bio zasnovana na upotrebi tzv. bitmap indeksa (koji predstavlja skup redova sa Id-jevima redova):

Step1: *scan* **r** *and the join index* {<**r**, $\mathbf{r_1}$,…,$\mathbf{r_n}$>} in one scan;
Step2: *retrieve matching tuples in* $\mathbf{r_1}$,…$\mathbf{r_n}$ *and output the result of join;*

Po jedan bitmap indeks J_i bi se koristio za svako parcijalno spajanje $\mathbf{r} \infty_{condi} \mathbf{r}_i$
Pri tome svaki J_i predstavlja skup redova tip <v, bitmap>, gde v predstavlja Id reda u relaciji r_i, a bitmap ima vrednost 1 na k-toj poziciji ako i samo ako se k-ti red relacije r spaja sa redom na koji ukazuje v.

Dakle, najpre bi se skenirao indeks J_i i primenila logička operacija OR nad svim bitmapa. Time se dobijaju svi Id-jevi relacije r koji se spajaju sa r_i. Potom je potrebno primeniti logičku operaciju AND nad rezultujućim bitmapa dobijenih upotrebom ostalih indeksa $J_1,\ldots,J_n$. Rezultat bi bio podskup relacije r takav da sadrži sve one redove koji treba da učestvuju u zvezda spajanju. Taj podskup će imati relativno mali broj redova tako da se može koristiti indeks ugnježdena petlja.

Redosled pristupa (Access path)

Redosled pristupa je termin koji označava algoritam i strukture podataka koje se koriste za lociranje redova koji zadovoljavaju određeni uslov.

Na primer:

- Skeniranje (file scan): može da se koristi za bilo koju pretragu.
- Heš: može da se koristi za pretragu po jednakosti; svi atributi po kojima se obavlja pretraga moraju predstavljati ključ za heš indeks, kako bi isti mogao biti upotrebljen.
- B^+ stablo: može se koristiti sa pretrage po jednakosti ili pretrage opsega; atributi po kojima se obavlja pretraga moraju predstavljati prefiks ključa po kome je stablo kreirano.

  Na primer, ukoliko je stablo kreirano nad relacijom R po sekvenci atributa a2, a1, a3, a4
  a. Redosled pristupa za pretragu $\sigma_{a1>5 \text{ and } a2=3 \text{ and } a3='x'}(R)$ bi bio: pronaći prvi ulaz kod koga je a2=3 i a1>5 i a3='x' i skenirati listove stabla sve dok se ne naiđe na ulaz kod koga je a2>3 ili a3≠'x'
  b. Redosled pristupa za pretragu $\sigma_{a2=3 \text{ and } a3='x'}(R)$ bi bio: pronaći prvi ulaz kod koga je a2=3 i skenirati listove stabla sve dok se ne naiđe na ulaz kod koga je a2>3.
  c. Redosled pristupa za pretragu $\sigma_{a1>5 \text{ and } a3='x'}(R)$ bi bio: skeniranje čitave relacije R.

- Binarna pretraga: pretraga se mora obavljati po sekvenci atributa ili po nekom od prefiksa sekvence atributa po kojima je relacija sortirana.

Selektivnost nekog redosleda pristupa predstavlja broj stranica koje moraju biti dohvaćene tim redosledom pristupa. Ukoliko je za neki upit moguće koristiti više različitih redosleda pristupa, DBMS će odabrati onaj sa najmanjom selektivnošću.

Veličina domena nekog atributa je indikator selektivnosti ukoliko se kao se taj atribut koristi u uslovu pretrage. Na primer, kod pretrage $\sigma_{SifP='bp1' \text{ and } Ocena=9}$(Polaganja), redosled pristupa koristeći B^+ stablo po ključu SifP ima manju selektivnost nego redosled pristupa koristeći B^+ stablo po ključu Ocena.

Generalno, kao neka osnovna pravila bi bila: ukoliko često izvršavani upit uključuje selektovanje ili spajanje sa velikim rezultatom onda bi trebalo koristiti klasterovano B^+ stablo; ukoliko je često izvršavani upit neka pretraga po jednakosti i ima mali rezultat onda je najbolje koristiti neklasterovani hash indeks (s obzirom da samo jedan klasterovani indeks može postojati nad tabelom, onda izbor neklasterovanog indeksa ostavlja prostora da se za potrebe nekog drugog upita kreira klasterovani indeks).

Evaluacija upita

Problem SQL upita je u tome što su oni deklarativnog tipa, odnosno ne specificiraju tačan plan izvršavanja upita. Sa druge strane, relaciona algebra je proceduralni upitni jezik koji tačno definiše redosled operacija koje treba primeniti kako bi se došlo do rezultata.

Dakle, SQL upit je neophodno konvertovati u ekvivalentni iskaz relacione algebre, a potom izvršiti evaluaciju koristeći odgovarajući plan izvršavanja tog iskaza. Koji ekvivalentni iskaz je najbolje odbrati?

Na primer, naivna konverzija bi za upit:

```
SELECT DISTINCT TargetList
FROM R1, R2, …, RN
WHERE Condition
```

dala rezultujući iskaz relacione algebre:

$\pi_{TargetList}(\sigma_{Condition}(R1 \times R2 \times ... \times RN))$

Međutim, to bi moglo da dovede do vrlo neefikasnog plana izvršavanja.

Na primer, kod iskaza

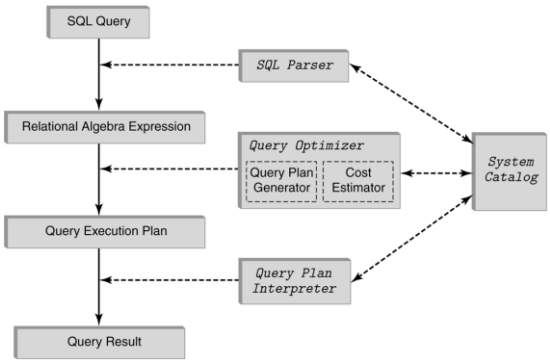
$\pi_{Name}(\sigma_{Id=ProfId \wedge CrsCode='CS532'}(Professor \times Teaching))$

Rezultat može biti manji od 100 bajtova, ali ukoliko je svaka od relacija veličine 50KB, računanje međurezultata, odnosno Dekartovog proizvoda tabela Professor i Teaching bi bio zahtevalo prostor od oko 500MB pre nego bi se taj rezultat smanjio na svega 100 bajtova.

Dakle, problem je pronaći ekvivalentni iskaz relacione algebre koji se može efikasno izvršiti.

Optimizator upita

Jedna od osnovnih komponenti arhitekture za obradu upita jeste optimizator upita.



Optimizator upita koristi algoritme zasnovane na heuristikama kako bi izvršio evaluaciju iskaza relacione algebre. Ovo uključuje: 1. procenu cene koštanja iskaza relacione algebre; 2. transformaciju jednog iskaza relacione algebre u neki drugi njemu ekvivalentan iskaz; 3. odabir redosleda pristupa (access path) za evaluaciju podiskaza.

Optimizator upita ne radi „optimizaciju“ već zapravo pokušava da nađe „razumno dobru“ strategiju za evaluaciju.

Transformacije iskaza koje čuvaju ekvivalenciju

Kako bi se iskaz relacione algebre transformisao u neki drugi njemu ekvivalentan iskaz potrebne su transformacije koje čuvaju ekvivalenciju. Svako pravilo transformacije treba: 1. da bude dokazivo ispravno; 2. da ima odgovarajuću heuristiku.

Pravila za restrikciju i projekciju:

1. Podela kompleksne restrikcije u jednostavnije
   $\sigma_{Cond1 \wedge Cond2}(R) \equiv \sigma_{Cond1}(\sigma_{Cond2}(R))$

2. Podela projekcije po fazama
   $\pi_{attr}(R) \equiv \pi_{attr}(\pi_{attr'}(R))$, samo ako je $attr \subseteq attr'$

3. Komutativnost projekcije i restrikcije
   $\pi_{attr}(\sigma_{Cond}(R)) \equiv \sigma_{Cond}(\pi_{attr}(R))$, samo ako je $attr \supseteq$ svih atributa u uslovu $Cond$

Pravila za komutativnost i asocijativnost spajanja (i Dekartovog proizvoda kao specijalnog slučaja spajanja):

1. Komutativnost spajanja
   $R \infty S \equiv S \infty R$
   Ovo se koristi u cilju smanjivanja cene izvršavanja strategija zasnovanih na ugnježdenim petljama (manja relacija treba da bude u spoljnoj petlji)

2. Asocijativnost spajanja
   $R \infty (S \infty T) \equiv (R \infty S) \infty T$
   Ovo se koristi kako bi prlikom spajanja većeg broja relacija međurezultat bio što manji. Odnosno, najpre se računaju spajanja koja generišu manji međurezultat.

3. Spajanje N relacija ima T(N)xN! različitih planova evaluacije. Pri tome je T(N) broj iskaza pod zagradama, a N! Broj mogućih permutacija. Optimizator upita ne može razmatrati sve moguće planove (određivanje

optimalnog plana bi trajao duže nego izračunavanje koristeći metod grube sile). Upravo zbog toga, optimizator upita ne generiše neophodno optimalni plan.

Pravila potiskivanja restrikcije i projekcije:

1. $\sigma_{Cond}(R \times S) \equiv R \infty_{Cond} S$
   Pri tome, Cond sadrži attribute obeju realacija R i S. Ovim se smanjuje veličina međurezultata s obzirom na to da se određeni redovi mogu odbaciti ranije.

2. $\sigma_{Cond}(R \times S) \equiv \sigma_{Cond}(R) \times S$
   Pri tome, Cond sadrži samo atribute relacije R. Ovim se smanjuje veličina međurezultata s obzirom na to da se redovi relacije R mogu odbaciti ranije.

3. $\pi_{attr}(R \times S) \equiv \pi_{attr}(\pi_{attr'}(R) \times S)$, samo ako $attributes(R) \supseteq attr' \supseteq attr \cap attributes(R)$
   Ovim se smanjuje veličina jednog od operanada proizvoda

Primer ekvivalencije iskaza:

$$\sigma_{C1 \wedge C2 \wedge C3}(R \times S) \equiv \sigma_{C1}(\sigma_{C2}(\sigma_{C3}(R \times S))) \equiv \sigma_{C1}(\sigma_{C2}(R) \times \sigma_{C3}(S)) \equiv \sigma_{C2}(R) \infty_{C1} \sigma_{C3}(S)$$

Pod pretpostavkom da C2 uključuje samo atribute iz R, da C3 uključuje samo atribute iz S, kao i da C1 uključuje atribute iz obe realcije R i S.

Primer određivanja cene upita

Neka je data relaciona šema:

```
Katedra (SifD, Naziv)
Predmet (SifP, Naziv)
Profesor (SifP, Ime, SifD)
Predaje (SifPro, SifPre, Semestar)
```

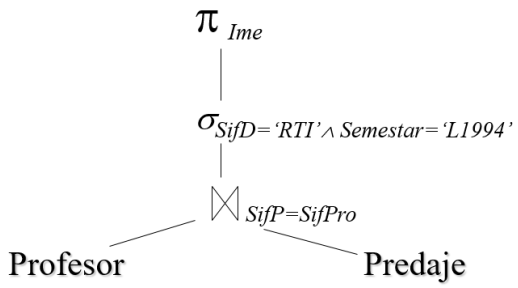
Neka je dat upit:

```
SELECT P.Ime
FROM Profesor P, Predaje T
WHERE P.SifP = T.SifPro
     AND P.SifD='RTI' AND T.Semestar='L1994';
```

Jedan ekvivalentan iskaz relacione algebre glasi (bez potiskivanja bilo koje od operacija):

$\pi_{Ime}(\sigma_{SifD='RTI' \wedge Semestar='L1994'}(\text{Profesor} \infty_{SifP=SifPro} \text{Predaje}))$

Odnosno stablo izvršavanja izgleda:



Uvidom u metapodatke sistemskog kataloga dolazi se do sledećih podataka:

Tabela Profesor(SifP, Ime, SifD) ima veličinu od 200 stranica, ima 1000 redova. Da postoji 50 katedri. Da je nad atributom SifD podignut klasterovani indeks na bazi B+ stabla sa dva nivoa. Da je nad atributom SifP podignut heš indeks.

Tabela Predaje(SifPro, SifPre, Semestar) ima veličinu od 1000 stranica, ima 10000 redova Da postoji svega 4 različita semestra. Da je nad atributom Semestar podignut klasterovani indeks na bazi B+ stabla sa dva nivoa. Da je nad atributom SifPro podignut heš indeks.

Definicija: Težina atributa (Weight) predstavlja prosečan broj redova koji poseduje određenu vrednost

Npr. težina atributa SifP je 1 (jer je primarni ključ); težina atributa SifPro je 10 (jer 10000 redova u Predaje se raspoređuje na 1000 profesora).

Ukoliko bi se za spajanje koristile blok ugnježdene petlje sa 52 stranice bafera (50 stranica za ulazne stranice tabele Profesor, 1 stranica kao ulazna stranica tabele Predaje, i 1 stranica za izlaz):

- Skeniranje tabele Profesor (spoljna petlja): transfer 200 stranica (4 iteracija od po 50 stranica)
- Određivanje odgovarajućih redova u tabeli Predaje (unutrašnja petlja): transfer 1000 stranica za svaku od iteracija spoljne petlje
- Ukupna cena: 200 + 4*1000 = 4200 stranica

Operacije restrikcije i projekcije: skeniranje redova dobijenih u međurezultatu nakon spajanja, odbacivanje redova koji nezadovoljavaju uslov restrikcije, odbacivanje nepotrebnih kolona (projekcija), upis rezultat kada je izlazni bafer pun.

Dakle, redosled pristupa bi mogao da bude: spajanje, upis međurezultata na disk, čitanje međurezultata, izračunavanje restrikcije i projekcije, upis finalnog rezultata.

Problem su nepotrebne, odnosno suvišne IO operacije.

Rešenje je upotreba protočne obrade (pipelining): raditi spajanje i restrikciju/projekciju kao korutine, po principu producer/consumer koji dele bafer u memoriji. Odnosno, kada spajanje napuni bafer, operacije restrikcije/projekcije mu pristupe i proizvedu finalni rezultat. Time se operacije restrikcije/projekcije sprovode bez dodatnih troškova.

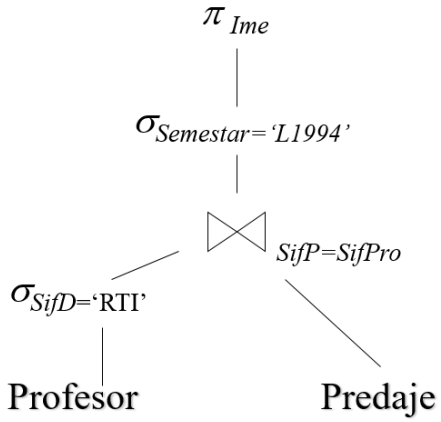
Dakle, konačna cena izvršavanja je: 4200 + (cena ispisa finalnog rezultata)

Cena ispisa finalnog rezultata se može odbaciti, s obzirom na to da će sve strategije evaluacije upita imati istu veličinu rezultata.

Drugi ekvivalentan iskaz relacione algebre glasi (parcijalno potisnuta operacija restrikcije):

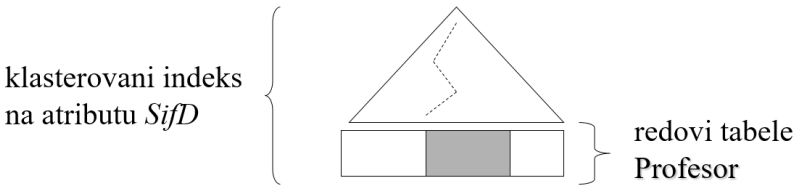
$\pi_{Ime}(\sigma_{Semestar='L1994'}(\sigma_{SifD='RTI'}(\text{Profesor}) \infty_{SifP=SifPro} \text{Predaje}))$

Odnosno stablo izvršavanja izgleda:



Izračunavanje restrikcije $\sigma_{SifD='RTI'}$ (Profesor) dovodi do smanjenja jednog od operanada u operaciji spajanja. Ova restrikcija se može izračunati koristeći klasterovan indeks na bazi B+ stabla sa dva nivoa koji postoji nad atributom SifD.

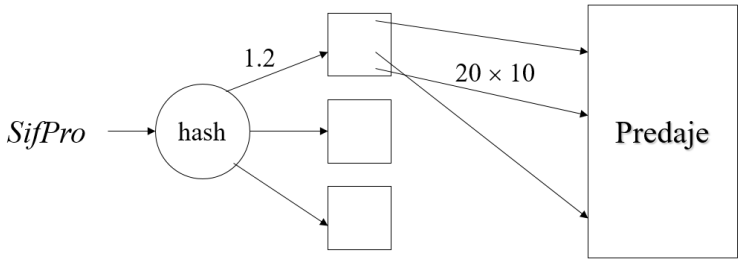
S obzirom da postoji 50 katedri i 1000 profesora, znači da je težina atributa SifD jednaka 20. Dakle, procena je da RTI katedra ima oko 20 profesora. Ovi redovi su u raspoređeni sukcesivno u približno 4 stranica tabele Profesor.



Dakle, cena restrikcije je: 4 (za dohvatanje redova) + 2 (za pretragu indeksa u obliku stabla) = 6

Ukoliko bi se za spajanje koristila indeks ugnježdena petlja koristeći heš indeks na atributu SifPro u tabeli Predaje iterirajući nad selektovanim profesorima (rezultat prethodno izračunate operacije restrikcije):

- Restrikcija nad atributom Semestar nije potisnuta, to znači da se heš indeks nad atributom SifPro u tabeli Predaje može iskoristiti. Ovo je primer prednosti kada se koristi nepotpuno potisnut plan izvršavanja upita, jer da je restrikcija nad atributom Semestar bila potisnuta, onda bi indeks nad SiPro bio izgubljen.
- Svakom profesoru odgovara oko 10 redova tabele Predaje. S obzirom da ima oko 20 profesora RTI katedre, znači treba očekivati oko 200 odgovarajućih redova u tabeli Predaje.
- Svi ulazi indeksa za određenog profesora (SifPro) se nalaze u istom baketu. Može se pretpostaviti po 1,2 operacije za dohvaanje baketa.



Cena spajanja je: 1,2*20 (za dohvatanje ulaza indeksa za 20 profesora sa RTI) + 200 (da se dohvate odgovarajući redovi u tebeli Predaje, s obzirom da je heš indeks neklasterovan) = 224

Operacije restrikcije nad atributom Semestar i projekcije nad atributom Ime se mogu izvesti bez dodatnog troška ukoliko se primeni protočna obrada.

Dakle, konačna cena izvršavanja je: 6 (restrikcija nad Profesor) + 224 (spajanje) + (cena ispisa finalnog rezultata)

Odnosno, 230 + (cena ispisa finalnog rezultata)

Ukoliko se uporede dva predložena ekvivalentan iskaza relacione algebre, dobija se 4200 na prama 230.

Procena veličine rezultata

Veoma je važno proceniti veličinu rezultata nekog relacionog iskaza zbog toga što se to koristi kao ulazni podatak za naredne faze izračunavanja i utiče na izbor kako te naredne faze treba da budu evaluirane.

Procena veličine se zasniva koristeći sedeće mere nad instancom relacije R:

Tuples(R): broj redova

Blocks(R): broj blokova

Values(R.A): broj različitih vrednosti atributa A

MaxVal(R.A): maksimalna vrednost atributa A

MinVal(R.A): minimalna vrednost atributa A

Za upit tipa:

```
SELECT TargetList
FROM R1, R2, …, RN
WHERE Condition
```

Definiše se faktor redukcije (Reduction factor) kao:

$$\frac{Blocks\ (\text{result set})}{Blocks(\mathrm{R}_1) \times ... \times Blocks(\mathrm{R}_n)}$$

Gde je „result set“ ustvari rezultat upita. Dakle, ovaj faktor služi za procenu koliko je rezultat upita manji od ulaza.

Može se pretpostaviti da su faktori redukcije koji potiču od projekcije i restrikcije međusobno nezavisni.
To znači da je:

$$reduction(Query) = reduction(TargetList) \times reduction(Condition)$$

Redukcija koja potiče od jednostavnih uslova restrikcije.

Jednakost sa konstantom:

$$reduction\ (\mathrm{R}_i.A{=}val) \ = \ \frac{1}{Values(\mathrm{R}.A)}$$

Jednakost dva atributa dveju relacija:

$$reduction\ (\mathrm{R_i}.A=\mathrm{R_j}.B) \quad = \quad \frac{1}{max(Values(\mathrm{R_i}.A),\ Values(\mathrm{R_j}.B))}$$

Ako se pretpostavi uniformna distribucija vrednosti. Kao i:

$$Tuples(\mathrm{R_i}) < Tuples(\mathrm{R_j})$$

Takođe da i svakom redu iz relacije R_i odgovara red u relaciji Rj.

Onda je broj redova koji zadovoljavaju uslov Condition jednak:

$$Values(\mathrm{R_i}.A) \times (Tuples(\mathrm{R_i})/Values(\mathrm{R_i}.A)) \times (Tuples(\mathrm{R_j})/Values(\mathrm{R_j}.\mathrm{B}))$$

Nejednakost atributa i konstante:

$$reduction\ (\mathrm{R_i}.A > val) \quad = \quad \frac{MaxVal(\mathrm{R_i}.A) - val}{MaxVal(\mathrm{R_i}.A) - MinVal(R_i.A)}$$

Redukcija koja potiče od kompleksnih uslova restrikcije:

$$reduction(Cond_1\ \mathrm{AND}\ Cond_2) = reduction(Cond_1) \times reduction(Cond_2)$$

$$reduction(Cond_1\ \mathrm{OR}\ Cond_2) = \min(1,\ reduction(Cond_1) + reduction(Cond_2))$$

Redukcija koja potiče od projekcije.

$$reduction(TargetList) \quad = \quad \frac{number\text{-}of\text{-}attributes\ (TargetList)}{\Sigma_i\ number\text{-}of\text{-}attributes\ (\mathrm{R_i})}$$

Procena težine atributa.

$$\mathrm{weight}(\mathrm{R}.A) = Tuples(\mathrm{R}) \times reduction(\mathrm{R}.A{=}value)$$

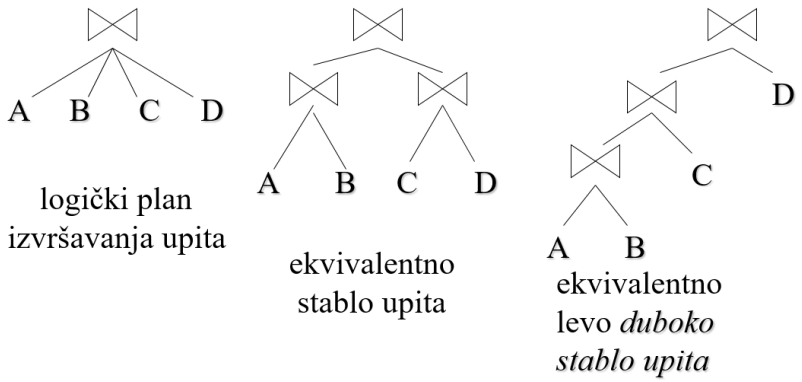
Odabir plana izvršavanja upita

Odabir se vrši kroz tri koraka:

1. Odabir logičkog plana
2. Redukcija prostora pretrage
3. Upotreba heuristika pretrage za dalju redukciju kompleksnosti

1. Odabir logičkog plana uključuje odabir stabla upita koje definiše redosled sprovođenja operacija relacione algebre. Pri tom se treba voditi heuristikom da je potpuno potisnuto stablo dobro, ali da su ponekad „skoro potpuno potisnuta stabla“ bolji izbor zbog mogućnosti upotrebe indeksa.

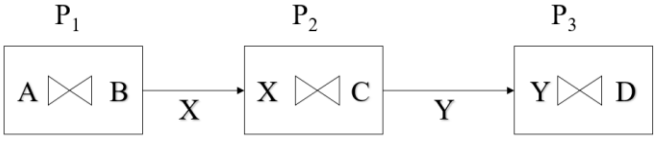
2. Redukcija prostora pretrage treba da se izbori asocijativnošću binarnih operacija (spajanje, unija, …)



Prilikom redukcije prostora pretrage postoje dva izazova:

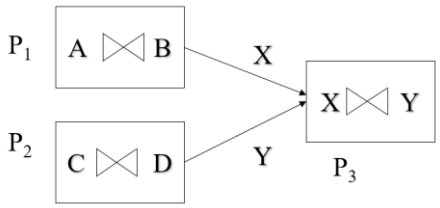
1. Kako odabrati određeni oblik stabla (tj. redosled spajanja više relacija, odnosno na koliko načina se mogu postavite zagrade kod spajanja N relacija – prostor pretrage raste rapidno)
2. Kako odabrati određenu permutaciju listova (npr. 4! permutacija za listove A, B, C, D)

S obzirom na to da postoji suviše mnogo stabala koje bi trebalo evaluirati: prihvata se određeni oblik unapred – levo duboko stablo (left-deep tree) koje omogućava upotrebu protočne obrade:



- Kada se jednom red X proizvede u okviru P1, on ne treba nikada više da bude ponovo proizveden (ali će C možda morati da bude obrađivano više puta u okviru P2 za sukcesivne delove X)
- Prednost je ta da nijedan relacija nastala kao međurezultat (ni X, ni Y) ne treba da bude u potpunosti materijalizovana i sačuvana na disk, što je naročito važno u slučajevima kada je neka od tih relacija jako velika, ali krajnji rezultat mali.

Ukoliko bi se razmotrila alternativa poput:



Može se primetiti da svaka red iz X treba da bude obrađen u kombinaciji sa svim redovima iz Y. Time svi redovi iz Y (kojih može biti mnogo) moraju biti materijalizovani u okviru P3, ili P2 mora da se izračunava više puta.

3. Upotreba heuristika pretrage je i dalje potrebna je izbor levog-dubokog stabla i dalje ostavlja veliki broj opcija (N! permutacija).

$$(((A \bowtie B) \bowtie C) \bowtie D)$$
$$(((C \bowtie A) \bowtie D) \bowtie B)$$
$$\ldots..$$

Algoritam zasnovan na heuristici (često neka vrsta dinamičkog programiranja) se koristi za dobijanje „dovoljno dobrog“ plana.

Na primer, da bi se izračunalo spajanje relacija E_1, E_2, …, E_N u nekom levom-dubokom stablu, najpre se startuje sa izrazima sa po jednom relacijom (može da uključuje restrikciju i projekciju), potom se odaberu najbolji i „približno najbolji“ plan za svaki izraz, nakon toga se ovi iskazi sa po jednom relacijom kombinuju kako bi se dobili iskazi sa po dve relacije. Zadržavaju se samo najbolji i „približno najbolji“ planovi sa po dve relacije, a nakon toga se radi kombinovanje radi dobijanje planova sa po tri relacije.

Plan se smatra „približno najboljim“ ukoliko ima neku interesantnu formu rezultata, npr. sortiranost.

Upit samo nad indeksom (index-only query)

Indeks zasnovan na B+ stablu sa atributima A_1, A_2, …, A_N ima sačuvane vrednosti svih ovih atributa za svaki red tabele. To znači da upit koji uključuje prefiks ovih atributa može biti izračunat samo koristeći indeks, bez potrebe da se pristupa podacima u tabeli. Treba imati u vidu da dodavanje posebnih indeksa za ovu svrhu opterećuje bazu zbog toga što je svaki indeks neophodno održavati prilikom svake promene podataka.