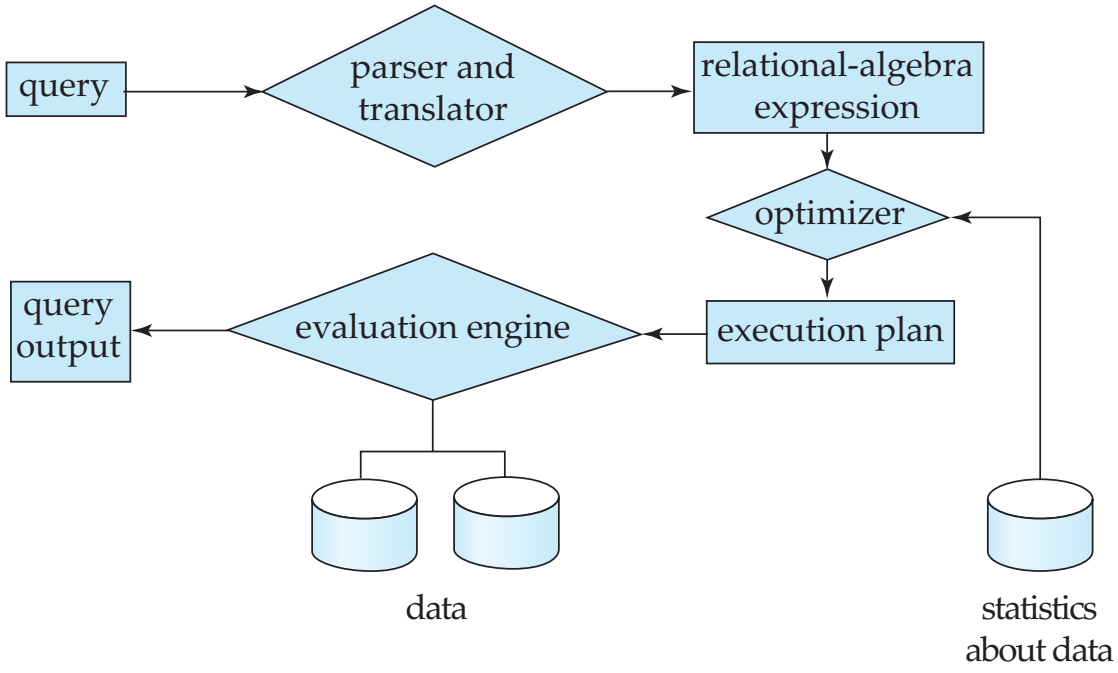
Obrada upita
Baze podataka 2

dr Miloš CVETANOVIĆ

Osnovni koraci pri obradi upita

- Parsiranje upita i prevođenje
  - Parsiranje proverava sintaksu, verifikuje postojanje relacija
  - Prevođenje u internu formu, a potom u iskaz relacione algebre
- Optimizacija
- Evaluacija
  - Plan evaluacije (**query-evaluation plan**) se izvršava, a rezultat vraća upitu



Osnovni koraci pri obradi upita - optimizacija

- Iskaz relacione algebra može biti ekvivalentnan sa većim brojem drugih iskaza
- Svaka operacija relacione algrebre moće biti evaluirana većim broj različitih algoritama
- Anotirani iskaz relacione algebra koji specificira strategiju evaluacije se naziva plan evaluacija (**evaluation-plan**)
  - Npr. Da li koristiti indeks ili skenirati čitavu relaciju
- Optimizacija: među svim ekvivalentnim planovima, odabrati onaj sa najnižom cenom
  - Cena se procenjuje na osnovu statističkih informacija iz kataloga baze podataka (broj redova u tabeli, veličina redova itd.)
  - Kako meriti cenu upita (broj transfera blokova, broj pristupa)
  - Algoritmi za evaluaciju pojedinih operacija relacione algebra
  - Kako kombinovati algoritme pojedinih operacija kako bi se evaluirao čitav iskaz

Procena cene upita

- Mnogi faktori doprinose vremenu izvršavanja
  - Pristup disku, CPU vreme, mrežna komunikacija (kod distribuiranih baza)
- Cena može biti procenjena
  - Vremenu odgovora (**response time**) – ukupno vreme potrebno da se vrati rezultata
  - Ukupan utrošak resursa (**resource consumption**)
- Vreme odgovora je teško proceniti, a minimizacija utroška resursa je dobra ideja naročito u uslovima gde se izvršava više baza nad jednim DBMS
- Ignorisanje CPU vremena zbog pojednostavljenja
  - Realni sistemi uzimaju u obzir CPU vreme (npr. predefisane konstante i faktori)
  - Uticaj mreže naročito važan kod distribuiranih i paralelnih baza

Operacija selekcije - jednakost

- Skeniranje datoteke (**file scan**)
- A1 – Linerna pretraga (**linear search**)
  Skenirati svaki blok datoteke i proveravati svaki zapise da li zadovoljava uslove selekcije
  - Cena = b_r blok transfera + 1 pretraga
    b_r označava broj blokova koji čuvaju zapise relacije r
  - Ako je selekcija po ključnom atributu, pretraga može da stane
    Cena = (b_r/2) blok transfera + 1 pretraga
  - Linearna pretraga može biti primenjenja bez obzira na
    uslov selecije, uređenosti zapisa u datoteci, dostupnosti indeksa
- Primetiti da binarna pretraga nema smisla, jer zapisi nisu sekvencijalno raspoređeni,
  a takođe, binarna pretraga zahteva više pretraga neko upotreba indeksa

Operacija selekcije - jednakost

- Skeniranje indeksa (**index scan**) – algoritam pretrage koji koristi indeks
  - Uslov pretrage mora biti po ključu pretrage nad kojim je indeks kreiran
- A2 – grupišući indeks, jednakost po ključnom atributu (**clustering index, equality on key**)
  Dohvatiti jedan zapis koji zadovoljava uslov jednakosti
  - Cena = $(h_i + 1) \cdot (t_T + t_S)$
    h_i je visina b^+-stabla indeksa (vrednost je 1 ako su unutrašnji čvorovi u memoriji)
- A3 – grupišući indeks, jednakost po ne-ključnom atributu (**clustering index, equality on nonkey**)
  Dohvatiti više zapisa koji zadovoljavaju uslov jednakosti
  - Cena = $h_i \cdot (t_T + t_S) + t_S + t_T \cdot b$
    b je broj blokova koji sadrže zapise koji zadovoljavaju uslov jednakosti
- A4 – sekundarni indeks, jednakost po ključnom ili ne-ključnom atributu (**secondary index, equality on key/non-key**)
  Dohvatiti jedan zapis ukoliko je ključ pretrage kandidat ključ
  - Cena = $(h_i + 1) \cdot (t_T + t_S)$

  Dohvatiti više zapisa ukoliko ključ pretrage nije kandidat ključ
  - Cena = $(h_i + n) \cdot (t_T + t_S)$
    n je broj zapisa koji zadovoljavaju uslov pretrage (mogu biti u različitim blokovima!)

Operacija selekcije - poređenja

- Može se implementirati selekcija poput A ≥ V ili A ≤ V koristeći
  - Linearno skeniranje ili indekse na dva načina (A5, A6)
- A5 – grupišući indeks, poređenje (**clustering index, comparison**) (relacija sortirana po A)
  - Za A ≥ V koristiti indeks sa se pronađe prvi red ≥ V a potom odatle sekvencijalno skenirati relaciju
  - Za A ≤ V samo sekvencijalno skenirati relaciju do prvog reda koji > V, bez upotrebe indeksa
- A6 – sekundarni indeks, poređenje (**secundary index, comparison**)
  - Za A ≥ V koristiti indeks sa se pronađe prvi zapis indeksa ≥ V a potom odatle sekvencijalno skenirati indeks, da se pronađu pokazivači na zapise
  - Za A ≤ V samo sekvencijalno skenirati listove indeksa da se pronađu pokazivači, do prvog zapisa indeksa koji > V
  - U oba slučaja, dohvatiti zapise u relaciji na osnovu pokazivača
  - Zahteva IO po zapisu, te je u određenim slučajevima skeniranje cele relacije “jeftinije”

Operacija selekcije – složene selekcije

- Konjukcija više uslova (AND)
- A7 – konjukcija selekcija upotrebom indeksa (**conjuctive selection using on index**)
  - Odabrati uslove i algoritme od A1 do A7 koji imaju najmanju cenu za svaki uslov
  - Testirate ostale uslove nad dohvaćenim redovima
- A8 – konjukcija selekcija upotrebom kompozitnog indeksa (**conjuctive selection using composite index**)
  - Upotrebiti kompozitni indeks ukoliko je dostupan
- A9 – konjukcija selekcija pomoću preseka identifikatora (**conjuctive selection by intersection of identifiers**)
  - Zahteva indekse sa pokazivačima na zapise relacije
  - Upotrebiti odgovarajući indeks za svaki od uslova, a potom napraviti presek svih dohvaćenih skupova pokazivača na zapise relacije
  - Potom dohvatiti zapise relacije
  - Ukoliko neki od uslova nemaju odgovarajuće indekse, obaviti proveru u memoriji

Operacija selekcije – složene selekcije

- Disjunkcija više uslova (OR)
- A10 – disjunkcija selekcija upotrebom unije identifikatora (**disjunctive selection by union of identifiers**)
  - Primenljivo samo ako svi uslovi imaju odgovarajuće indekse, u suprotnom koristiti linearno skeniranje
  - Upotrebiti odgovarajući indeks za svaki od uslova, a potom napraviti uniju svih dohvaćenih skupova pokazivača na zapise relacije
  - Potom dohvatiti zapise relacije
- Negacija (**negation**)
  - Upotrebiti linearno skeniranje
  - Ukoliko samo mali broj zapisa relacije zadovoljava negaciju uslova i ukoliko postoji indeks uslov, onda iskoristiti indeks za dohvatanje zapisa relacije

Skeniranje bitmap indeksom – složene selekcije

- Algoritam za skeniranje bitmap indeksom (**bitmap index scan**) kod PostgreSQL
  - Kompromis između sekundarnih indeksa i linearnog skeniranja
    kada broj zapisa traženih zapisa nije poznat unapred
  - Bitmapa sa 1 bitom po strani relacije
  - Koraci:
    1. Skeniranje indeksa da se pronađu identifikatori zapisa
       i postave bitovi za odgovarajuće strane u bitmapi
    2. Linearno skeniranje dohvata samo one strane za koje je bit postavljen
  - Performanse:
    Slično skeniranju indeksa kada je samo par bitova postavljeno
    Slično lineranom skeniranju kada je većina bitova postavljeno
    Nikada se ne ponaša suviše loše u poređenju sa najboljim alternativama

Operacija sortiranja

- Grupišući indeks može biti iskorišćen za čitanje relacije u sortiranom redosledu. Za slučajni, tj. random, pristup može dovesti do potrebe za jednim IO po zapisu.
- Za relacije koje mogu stati u memoriju, uobičajne tehnike sortiranja (npr. quicksort)
- Za relacije koje ne mogu stati u memoriju, dobar izbor je spoljno **sort-merge** sortiranje (tzv. spoljno sortiranje objedinjavanja)

Operacija sortiranja – spoljno sort-merge sortiranje

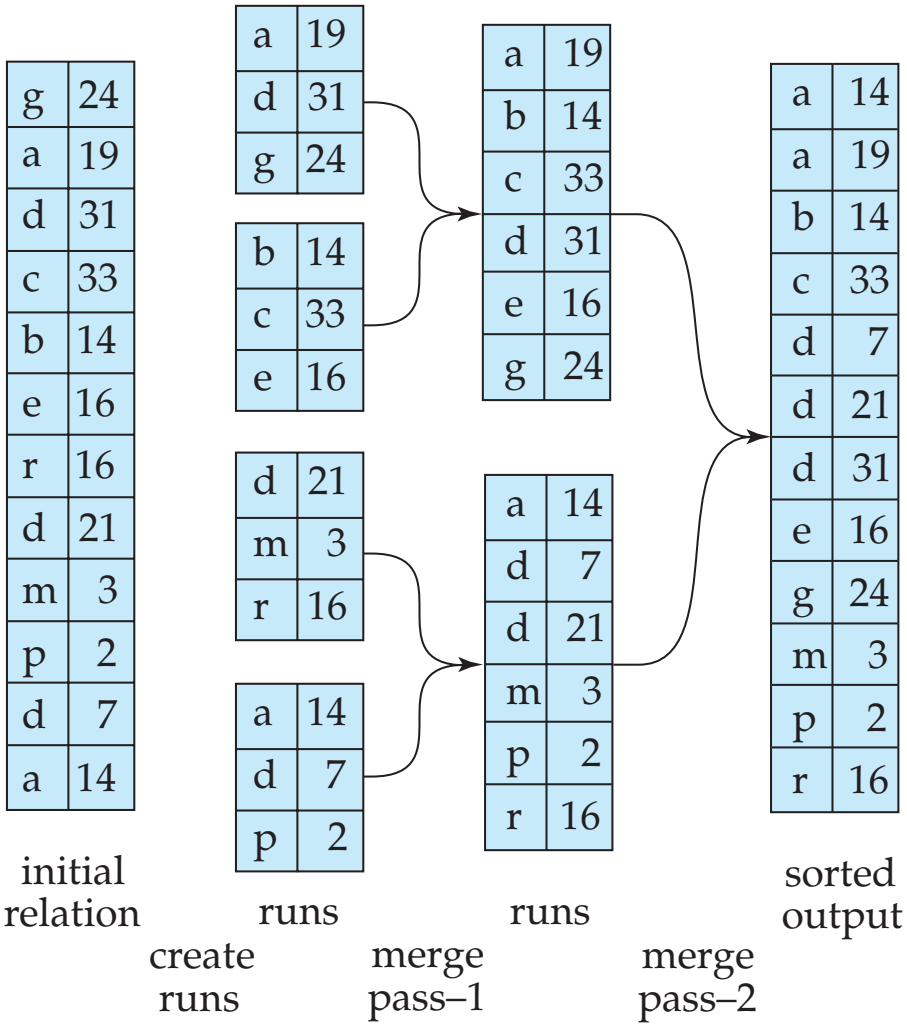
- Neka je *M* veličina dostupne memorije (izražena u broju strana)

  1. Kreirati sortirane delove (sorted runs). Neka je *i* inicijano 0.
     Ponavljati do kraja relacije:
     a. Pročitati *M* blokova relacije u memoriju
     b. Sortirati blokove koji su u memoriji
     c. Upisati sortirane podatke u R_i deo, potom inkrementirati *i*

  Neka je *N* konačna vrednost za *i*

  2. Objediniti delove (N-way merge). Pretpostavka da je $N < M$.
     a. Koristiti *N* blokova kao ulazni bafer za čuvanje delova, a 1 blok za izlazni bafer.
        Pročitati prvi blok svakog od delova u odgovarajući blok ulaznog bafera
     b. Ponavljati
        1. Odabrati prvi zapis (u sortiranom redosledu) od svih strana u ulaznom baferu
        2. Upisati zapis u izlazni bafer. Ako je izlazni bafer pun, upisati ga na disk.
        3. Obrisati zapis iz ulaznog bafera. Ako taj blok ulaznog bafera postane prazan
           onda učitati naredni blok (ako postoji) iz odgovarajućeg dela u taj blok.
     c. Dok svi ulazni baferi ne postanu prazni

Operacija sortiranja – spoljno sort-merge sortiranje



Operacija sortiranja – spoljno sort-merge sortiranje

- Ako je *N* > *M*, onda je potrebno nekoliko prolaza objedinjavanja delova.
  - U svakom prolazu, kontinualna grupa od *M-1* delova se objedinjuje.
  - Svaki prolaz smanjuje broj delova za faktor od *M-1*, i ujedno kreira delove koji su duži za isti taj faktor.
    Npr. Ako je M=11, i ako ima 90 delova, onda jedan prolaz smanjuje broj delona na 9, od kojih je svaki 10 puta veći od inicijalnog dela
  - Ponavljati prolaze dok se svi delovi ne objedine u jedan

Operacija sortiranja – spoljno sort-merge sortiranje

- Analiza cene (broj blok transfera):
  - 1 blok po delu dovodi do mnogo pretrage prilikom objedinjavanja
    pa je zbog toga bolje koristiti b_b bafer blokova za svaki od delova
    tako da se čita/upisuje b_b blokova odjednom
    Time se objedinjuje $\lfloor M/b_b \rfloor - 1$ delova u jednom prolazu
  - Ukupan broj potrebnih prolaza objedinjavanja je $\lceil \log_{\lfloor M/b_b \rfloor - 1} (b_r/M) \rceil$
    gde je b_r broj blokova relacije r.
  - Blok transferi za inicijalno kreiranje delova kao i u svakom prolazu je $2b_r$
    za finalni prolaz, se ne računa cena tj. broj upisa
    (finalni upisi se ignorišu za sve operacije, jer se izlaz operacije može proslediti kao ulaz sledećoj operaciji u nizu bez prethodnog upisivanja na disk)
    Tako da je **ukupan broj blok transfera** za spoljno sortiranje:

$$b_r(2\lceil \log_{\lfloor M/b_b \rfloor - 1} (b_r/M) \rceil + 1)$$

Operacija sortiranja – spoljno sort-merge sortiranje

- Analiza cene (broj pretraga):
  - Tokom kreiranja delova: 1 pretraga za čitanje i jedna pretraga za upis svakog dela dakle $2\lceil b_r/M \rceil$
  - Tokom objedinjavanja delova:
    Potrebno je $2\lceil b_r/b_b \rceil$ pretraga za svaki prolaz objedinjavanja
    (izuzev za poslednji prolaz za koji se upisi ne računaju)
  - Tako da je **ukupan broj pretraga** za spoljno sortiranje:

$$2\lceil b_r/M \rceil + \lceil b_r/b_b \rceil (2\lceil \log_{\lfloor M/b_b \rfloor - 1} (b_r/M) \rceil - 1)$$

Operacija spajanja

- Nekoliko različitih algoritama za implementaciju operacije spajanja:
  - Spajanje ugnježdenom petljom (**nested-loop join**)
  - Spajanje blok ugnježdenom petljom (**block nested-loop join**)
  - Spajanje indeksiranom ugnježdenom petljom (**indexed nested-loop join**)
  - Objedinjeno spajanje (**merge-join**)
  - Heš spajanje (**hash-join**)
- Izbor algoritma zavisi od procene cene
- Primer:
  - Relacija *student* ima n=5000 zapisa sačuvanih u b=100 blokova
  - Relacija *takes* ima n=10000 zapisa sačuvanih u b=400 blokova

Operacija spajanja – spajanje ugnježdenom petljom

- Da bi se izračunalo teta spajanja $r \bowtie_{\theta} s$:

  for each tuple t_r **in** r **do begin**
  for each tuple t_s **in** s **do begin**
  test pair (t_r,t_s) to see if they satisfy the join condition θ
  if they do, add $t_r \bullet t_s$ to the result.
  end
  end

- Pri tome se r zove spoljna relacija dok je s unutrašnja relacija spajanja
- Ne zahteva postojanja indeksa i može biti upotrebljena za bilo koji tip spajanja
- U najgorem slučaju, ukoliko ima dovoljno memorije da prihvati samo jedan blok svake od relacija, onda je cena $n_r \cdot b_s + b_r$ blok transfera i $n_r + b_r$ pretraga
- Ako manja relacija može u potpunosti da stane u memoriji, nju koristiti kao unutrašnju dok se time cena smanjuje na $b_s + b_r$ blok transfera i 2 pretrage
- Ako se pretpostavi najgori slučaj dobija se
  ako je *student* spoljna relacija: 5000*400+100=2 000 100 blok transfera i 5100 pretraga
  ako je *takes* spoljna relacija: 10000*100+400=1 000 400 blok transfera i 10400 pretraga
- Ako se pretpostavi da manja relacija (*student*) staje u memoriju: 500 blok transfera
- Algoritam spajanja blok ugnježdenom petljom je bolji izbor

Operacija spajanja – spajanje blok ugnježđenom petljom

- Varijanta ugnježđene petlje kod koje se svaki blok unutrašnje petlje uparuje sa svakim blokom spoljašnje petlje

for each block B_r **of** r **do begin**
  for each block B_s **of** s **do begin**
    for each tuple t_r **in** B_r **do begin**
      for each tuple t_s **in** B_s **do begin**
        Check if (t_r, t_s) satisfy the join condition
        if they do, add $t_r \bullet t_s$ to the result.
      end
    end
  end
end

Operacija spajanja – spajanje blok ugnježđenom petljom

- U najgorem slučaju cena je: $b_r \cdot b_s + b_r$ blok transfera i $2 \cdot b_r$ pretraga
  Svaki blok u unutrašnjoj relaciji *s* čita se jednom za svaki blok spoljašne relacije *r*
- U najboljem slučaju: $b_s + b_r$ blok transfera i 2 pretrage
- Poboljšanja za algoritme ugnježđene petlje i blok ugnježđene petlje:
  - Kod blok ugnježđenih petlji, se koriste M-2 bloka za smeštanje
    blokova spoljašnje relacije, jedan blok za unutrašnju petlju, jedan blok za izlaz
    Cena: $\lceil b_r / (M\text{-}2) \rceil \cdot b_s + b_r$ blok transfera i $2 \lceil b_r / (M\text{-}2) \rceil$ pretraga
  - Ako kod ekvi-spajanja atributi formiraju ključ za unutrašnju relaciju,
    onda se unutrašnja petlja može završiti čim se pronađe prvo uparivanje
  - Skenirati unutrašnju petlju unapred i unazad naizmenično,
    kako bi se iskoristili blokovi koji su trenutno u baferu (ako je LRU algoritam zamene)
  - Koristiti indeks za unutrašnju petlju ako je dostupan

Operacija spajanja – spajanje indeksiranom ugnježdenom petljom

- Indeks pretraga može zameniti skeniranje relacije ako je
  - Spajanje ekvi-spajanje ili prirodno spajanje
  - Indeks dostupan za unutrašnju relaciju po atributu spajanja (moguće je konstruisati indeks samo zbog spajanja)
- Za svaki zapis t_r u spoljašnjoj relaciji *r*, upotrebiti indeks da bi se pronašli odgovarajući zapisi u unutrašnjoj relaciji *s*
- Najgori slučaj je ako bafer ima prostora samo jedan blok relacije r
  cena je: $b_r(t_T+t_S) + n_r \cdot c$
  Gde je *c* cena pretrage indeksa i dohvatanja svih odgovarajućih zapisa relacije *s*
  Pri tome *c* može biti procenjen kao cena jedne selekcije nad relacijom *s* koristeći uslov spajanja
- Ukoliko su indeksi dostupni po atributima spajanja u obe relacije, *r* i *s*, onda koristiti relaciju sa manje zapisa kao spoljnu relaciju

Operacija spajanja – spajanje indeksiranom ugnježđenom petljom

- Određivanje *student* ⋈ *takes*, pri čemu je *student* spoljašna relacija.
- Neka relacija *takes* ima primarni B^+-stablo indeks po atributu ID, a koji ima po 20 ulaza u svakom čvoru indeksa
- S obzirom da *takes* ima 10000 zapisa (u 400 blokova), visina stabla je 4, i još jedan dodatni pristup je potreban da se pronađe odgovarajući podatak
- Relacija student ima 5000 zapisa (u 100 blokova)

- Cena blok ugnježđenog spajanja (u najgorem slučaju):
  400*100+100=40100 blok transfera i još 2*100=200 pretraga

- Cena indeksirane ugnježđene petlje:
  100+5000*5=25100 blok transfera i isto toliko pretraga

Operacija spajanja – objedinjeno spajanje (merge-join)

- Sortirati obe relacije po njihovim atributima spajanja (ako već nisu po tome sortirani)
- Objediniti sortirane relacije kako bi bile spojene
  1. Korak spajanja je sličan fazi objedinjvanja kod sort-merge algoritma
  2. Glavna razlika jeste rad sa duplikatima – svaki par sa istim vrednostima na atributima spajanja moraju biti spojeni
- Može se koristiti samo za ekvi-spajanja i prirodna spajanja
- Svaki blok treba da bude pročitan samo jednom
  (pod pretpostavkom da svi zapisi za bilo koju moguću vrednost mogu stati u memoriji)
- Cena je: b_r+b_s blok transfera i $\lceil b_r/b_b \rceil + \lceil b_s/b_b \rceil$ pretraga (+ cena sortiranja relacija)
  gde je b_b broj blokova u baferu koji je na raspolaganju svakoj od relacija

- Hibridno objedinjeno spajanja (**hybrid merge-join**) ako
  je jedna relacija sortirana, druga ima sekundarni B^+-stablo indeks po atributu spajanja
  - Objediniti sortiranu relaciju sa ulazima u listovima B^+-stabla
  - Sortirati rezultat po adresama zapisa nesortirane relacije
  - Skenirati nesortiranu relaciju po redosledu fizičkih adresa i objediniti sa prethodnim rezultatom kako bi se adrese zamenile konkretnim zapisima

Operacija spajanja – heš spajanje (hash-join)

- Heš spajanje relacija *r* i *s* se obavlja na sledeći način:
  1. Particionisati relaciju *s* koristeći heš funkciju h.
     Dok se obavlja particionisanje relacije, jedan blok rezervisati za potrebe izlaznog bafera
  2. Particionisati relaciju *r* koristeći heš funkciju h.
  3. Za svaku *i*:
     a. Učitati s_i u memoriju i kreirati heš indeks u memoriji koristeći atribute spajanja
        Ovaj heš indeks koristi drugačiju heš funkciju od prethodne (h)
     b. Čitati zapise iz r_i sa diska jedan po jedan. Za svaki zapis t_r locirati jedan odgovarajući zapis t_s u s_i koristeći heš indeks u memoriji.
        Kreirati izlaz kao konkatenaciju njihovih atributa.
- Relacija *s* se naziva ulaz gradnje (**build input**), a relacija *r* se naziva ulaz probe (**probe input**)
- Vrednost *n* i heš funkcija h se biraju tako da svako s_i može da stane u memoriju
  Tipična vrednost za *n* je $\lceil b_s/M \rceil \cdot$ f gde je faktor f tipično oko 1,2 (tzv. **fudge factor**)
  Particije probe relacije ne moraju da stanu u memoriji
- Ukoliko je broj particija *n* veći broja blokova *M* u memoriji (rekurzivno particionisanje)
  - Umesto particionisanja na *n*, raditi particionisanje na *M*-1 particiju za relaciju *s*
  - Dalje particionisati *M*-1 particije koristeći drugačiju heš funkciju
  - Koristiti isti postupak particionisanja i za relaciju *r*
  - Nije potrebno ako je $M > n_h+1$, odnosno $M > b_s/M + 1$, tj. $M > \sqrt{b_s}$

Operacija spajanja – heš spajanje (hash-join)

- Kod heš spajanja može doći do prelivanja (**overflow**) ukoliko s_i ne staje u memoriju
- Particionisanje je iskrivljeno (**skewed**) koliko neke particije imaju značajno više zapisa
- Rezolucija prelivanja se može sprovesti u vreme faze gradnje
  Particije s_i se dalje particionišu koristeći drugačiju heš funkciju
  Particije r_i moraju bite istim postupkom particionisane
- Izbegavanje prelivanja – sprovesti particionisanje pažljivo
  Npr. relaciju *s* particionisati na više particija, pa ih naknadno kombinovati

Operacija spajanja – heš spajanje (hash-join)

- Cena heš spajanja je (ako rekurzivno particionisanje nije potrebno):
  $3(b_r+b_s)+ 4n_h$ blok transfera i $2(\lceil b_r/b_b \rceil + \lceil b_s/b_b \rceil) + 2n_h$ pretraga
  gde je b_b broj blokova u baferu koji je na raspolaganju za ulazni bafer i svaki izlazni bafer
- Cena heš spajanja je (ako rekurzivno particionisanje nije potrebno):
  $3(b_r+b_s)+ 4n_h$ blok transfera i $2(\lceil b_r/b_b \rceil + \lceil b_s/b_b \rceil) + 2n_h$ pretraga
  gde je b_b broj blokova u baferu koji je na raspolaganju za ulazni bafer i svaki izlazni bafer
- Cena heš spajanja je (ako je rekurzivno particionisanje potrebno):
  $2(b_r+b_s)\lceil \log_{M/b_b-1}(b_s/M) \rceil + b_r+b_s$ blok transfera i $2(\lceil b_r/b_b \rceil + \lceil b_s/b_b \rceil)\lceil \log_{M/b_b-1}(b_s/M) \rceil$ pretraga
- Cena heš spajanja ako čitava relacija gradnje (build relation) može da stane u memoriju, pa samim tim particionisanje nije potrebno: b_r+b_s blok transfera i dve pretrage

- *Npr.* Spajanje *student* ⋈ *takes*
  Pretpostaviti da je veličina bafera M=20, da je $b_{student}$=100 i b_{takes}=400
  Koristiti *student* kao build input i u jednom prolazu je particionisati u 5 particija, svaka veličine 20 blokova
  Slično, particionisati *takes* u 5 particija, svaka veličine 80 blokova (u jednom prolazu)
  Ukupna cena: 3(100+400)=1500 blok transfera (zanemerano $4n_h$)
  $2(\lceil 400/3 \rceil + \lceil 100/3 \rceil)$=336 pretraga (zanemareno $2n_h$)
  za baferisanje npr. b_b=3 jer M = $(n_h+1) \cdot b_b$

Operacija spajanja – kompleksna spajanja

- Spajanje sa konjukcijom uslova $r \bowtie_{\theta 1 \wedge \theta 2 \wedge \ldots \theta n} s$
  - Koristiti ugnježdene petlje ili blok ugnježdene petlje, ili
  - Izračunati rezultat jednog jednostavnog spajanja $r \bowtie_{\theta i} s$
    a konačni rezultat formirati proveravajući koji zapisi iz tog međurezultata zadovoljavaju preostale uslove $\theta 1 \wedge \ldots \wedge \theta i\text{-}1 \wedge \theta i\text{+}1 \wedge \ldots \wedge \theta n$
- Spajanje sa disjunkcijom uslova $r \bowtie_{\theta 1 \vee \theta 2 \vee \ldots \theta n} s$
  - Koristiti ugnježdene petlje ili blok ugnježdene petlje, ili
  - Izračunati kao rezultat unije zapisa individualnih spajanja $r \bowtie_{\theta i} s$
    odnosno, $(r \bowtie_{\theta 1} s) \cup (r \bowtie_{\theta 2} s) \cup \ldots \cup (r \bowtie_{\theta n} s)$

- Spajanje po prostornim podacima
  - Nema jednostavnog sortiranog redosleda za prostorna spajanja
  - Inksirane ugnježdene petlje sa prostornim indeksima
    (R-stabla, k-d-B-stabla)

Operacija projekcije

- Operacija projekcije
  - Iz svakog zapisa eliminisati nepotrebne atribute
  - Obaviti eliminaciju duplikata
- Eliminacija duplikata može biti implementirana sortiranjem ili heširanjem
  - Prilikom sortiranja, duplikati će biti susedi, te treba obrisati sve sem jednog
  - Optimizacija: duplikati mogu biti obrisani prilikom kreiranja delove kao i tokom međukoraka objedinjavanja u algoritmu sort-merge
  - Heširanje – duplikati će biti locirani u istim baketima

Operacija agregacije

- Operacija agregacija može biti implementirana slično eliminaciji duplikata
  - Sortiranje ili heširanje dovodi zapise u odgovarajuće grupe, a potom se izračunava agregatna funkcija nad svakom od grupa
  - Optimizacija: parcijalna agregacija
    a. Kombinovati zapise u istoj grupi prilikom kreiranja delova i međukoraka objedinjavanja izračunavanjem parcijalnih vrednosti agregatnih funkcija
    b. Za funkcije *count*, *min*, *max*, *sum*: čuvati parcijalne vrednosti na osnovu zapisa koji su do tog trenutka smešteni u grupu
    c. Za funkciju *avg*, čuvati parcijalne vrednosti za *sum* i *count*, a na kraju uraditi operaciju deljenja

Operacije unije, preseka i razlike skupova

- Skupovne operacije se mogu izračunati koristeći varijante objedinjenog spajanja nakon sortiranja (merge-join) ili heš spajanja (hash-join)
- Npr. Skupovne operacije heširanjem
  1. Particionisati obe relacije koristeći istu heš funkciju
  2. Obraditi svaku particiju *i* na sledeći način:
     1. Koristeći drugačiju heš funkciju, kreirati heš indeks u memoriji nad r_i
     2. Obraditi s_i na sledeći način:

        Za r ∪ s:

        1. Dodati zapise iz s_i u heš indeks, ukoliko nisu već tamo
        2. Kada se završi obrada s_i onda dodati zapise iz heš indeksa u rezultat

        Za r ∩ s:

        1. Dodati zapise iz s_i u rezultat, ukoliko su već u heš indeksu

        Za r – s:

        1. Za svaki zapis iz s_i ukoliko je u heš indeksu, obrisati ga iz heš indeksa
        2. Kada se završi obrada s_i onda dodati zapise iz heš indeksa u rezultat

Operacija spoljnog spajanja (outer join)

- Operacija spoljnog spajanja se može izračunati
  - Spajanje praćeno dodavanjem nedostajućih zapisa dopunjenih sa *null*
  - Modifikacijom algoritma spajanja
- Modifikacija objedinjenog spajanja (merge-join) za izračunavanje $r \ltimes s$ (levo spoljno)
  - Kod $r \ltimes s$ nedostajući zapisi su r – $\boldsymbol{\pi}_R(r \bowtie s)$
  - Modifikovati algoritam:
    Tokom objedinjavanja, za svaki zapis t_r iz r koji nema odgovarajući zapis u s,
    u rezultat dodati t_r dopunjen sa odgovarajućim broj *null*
  - Desno spoljno i puno spoljno spajanje se može izračunati na sličan način
- Modifikacija heš spajanja (hash-join) za izračunavanje $r \ltimes s$
  - Ukoliko je r relacija *probe* relacija, onda zapise za koje se ne pronađe odgovarajući zapis u *build* relacija s, dopuniti sa *null* i staviti u izlaz
  - Ukoliko je r relacije *build* relacija, tokom provera treba voditi evidenciju o tome koji zapisi iz r imaju odgovarajuće zapise iz s. Na kraju obrade s_i u izlaz dodati evidentirane zapise iz r dopunjene sa *null*

Evaluacija celokupnih izraza

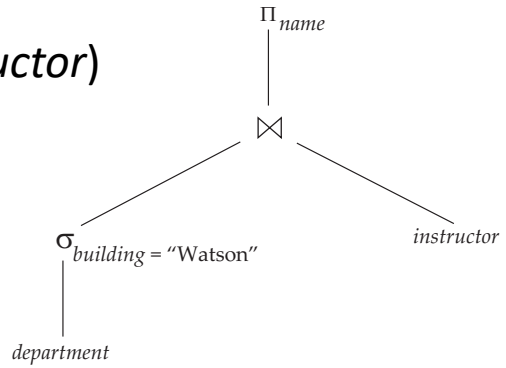
- Evaluacija pojedinih operacija i evaluacija celokupnog izraza
- Evaluacija celokupnog izraza ima dve opcije:
  - Materijalizacija (**materialization**):
    Generisati rezultat izraza čiji su ulazi su relacije koje su prethodno izračunate,
    a sam rezultat snimiti (materijalizovati) na disk.
  - Protočna obrada (**pipelining**):
    Proslediti zapise roditeljskoj operaciji čak i dok se još uvek obavlja izračunavanje

Evaluacija celokupnih izraza - materijalizacija

- Materijalizovana evaluacija: evaluirati jednu po jednu operaciju, počev od najnižeg nivoa. Koristiti materijalizovane međurezultate kako bi se evaluirala naredna operacija.
- Npr. za stablo izvršavanja iskaza

$$\pi_{\text{name}}(\sigma_{\text{building}=\text{„Watson“}}(department) \bowtie instructor)$$

Najpre sračunati selekciju i sačuvati rezultat na disku,
a potom očitati sa diska i sračunati spajanje,
na kraju sračunati projekciju



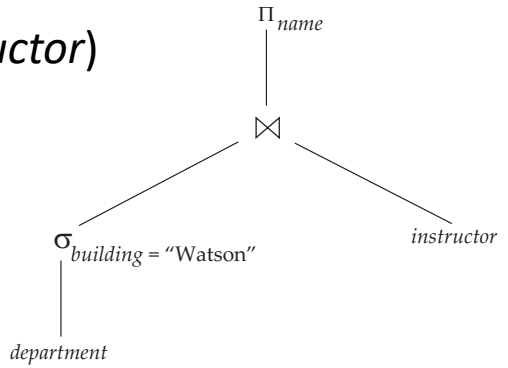
- Materijalizovana evaluacija: uvek moguća
- Cena: cena pojedinih operacija + cena upisa međurezultata na disk
- Duplo baferisanje (**double buffering**): koristiti dva izlazna bafera za svaku operaciju, kada se jena napuni, upisivati ga na disk dok se drugi izlazni bafer puni čime se trošak vremena upisa na disk redukuje

Evaluacija celokupnih izraza – protočna obrada

- Protočna evaluacija: evaluirati nekoliko operacija istovremeno, prosleđujući rezultate jedne operacije drugoj
  Npr. za stablo izvršavanja iskaza

$$\pi_{\text{name}}(\sigma_{\text{building}=\text{„Watson“}}(\textit{department}) \bowtie \textit{instructor})$$

Dok se računa selekcija, rezultat ne čuvati na disk,
već odmah direktno prosleđivati spajanju.
Slično, ne čuvati rezultat spjanja,
već rezultat odmah prosleđivati projekciji



- Daleko jefitnije nego materijalizovana evaluacija
- Protočna evaluacija nije uvek moguća (npr. sort, hash-join)
- Treba koristiti algoritme za operacije koji ne zahtevaju celokupan ulaz pre početka
- Protočna evaluacije se može obaviti na dva načina:
  vođena potražnjom (**demand driven**), vođena proizvođačem (**producer driven**)

Evaluacija celokupnih izraza – protočna obrada

- Evaluacija vođena potražnjom (**demand driven**) je „lenja“ (**lazy evaluation**)
  - Sistem neprestano zahteva naredni zapis za potrebe operacije na najvišem nivou
  - Svaka operacija zahteva naredni zapis od operacije deteta po potrebi
  - Između poziva, operacija mora da održava „stanje“, da zna šta treba da vrati sledeće
  - Svaka operacija se implementira kao *iterator* sa metodama (*open*, *next*, *close*)
- Evaluacija vođena proizvođačem (**producer driven**) je „revnosna“ (**eager evaluation**)
  - Operacije proizvode zapise revnosno i prosleđuju ih operacijama roditelja
    Bafere postoje između operacija, tako da operacije deca dodaju zapise u bafer, a operacije roditelji uklanjaju zapise iz bafera
    Ukoliko je bafer pun, operacija dete čeka dok se u baferu ne oslobodi prostor
  - Sistem pravi raspored izvršavanja operacija na osnovu informacija o prostoru koji imaju u svojim baferima izlaza, odnosno mogućnosti obrade novih ulaza
- Alternativni nazivi: *pull* odnosno *push* modeli protočne evaluacija

Evaluacija celokupnih izraza – protočna obrada

- Neki algoritmi ne mogu generisati rezultat iako je ulaz pristigao
  (npr. merge-join, hash-join), međurezultat se upisuje na disk, a potom čita (materijalizacija)
- Blokirajuća operacija (**blocking operation**) zahteva celokupan ulaz (npr. sort, agregacija)
  Ukoliko blokirajuća operacija ima dve podoeracije, onda ih treba tretirati kao odvojene
  (npr. za sort: generisanje delova i objedinjavanje, za hash-join: particionisanje i build-probe)
- Nivoi protočne evaluacija (**pipeline stages**):
  Sve operacije na istom nivou se izvršavaju konkurektno
  Nivo može otpočeti sa radom tek kada su svi prethodni nivoi završili
- Varijante algoritama tako da mogu da generišu bar neki rezultat u letu („*on the fly*“)
- Tokovi podataka (**data streams**): senzorske mreže, događaji korisnika…
- Kontinualni upiti (**continuous queries**):
  Rezultati se ažuriraju kako pristižu novi podaci
  (npr. agregacija nad prozorom podataka – pristiglim u nekom vremenskom periodu, ili
  ukoliko su razdvojeni delimiterom tzv. **punctuations**)

Algoritmi svesni postojanja keš memorije (cache conscious algorithms)

- Cilj: minimizirati keš promašaja, maksimalno iskoristiti podatke
  koji su već u kešu kao deo keš linije
- Za sortiranje:
  Kreirati delove koji su veliki koliko i L3 keš (nekoliko MB) kako bi se izbegao promašaj tokom sortiranja dela
  Objediniti delove na uobičajan način
- Za heš spajanje:
  Najpre kreirati particije tako da *build-probe* particije staju u memoriju
  Potom dodatno particionisati tako podparticije i indeks staju u L3 keš, čime se *probe* faza značajno ubrazava
- Rasporediti atribute u zapisima tako da se maksimizira upotreba keša
  Npr. Atributi kojima se često prostupa istovremeno treba
  da budu smešteni blizu jedni drugih
- Koristiti više niti za paralelnu obradu upita
  time će keš promašaj zaustaviti jednu nit, ali će ostale nastaviti sa radom