OPTIMIZACIJA UPITA

Gordana Pavlović-Lažetić
Matematički fakultet, Beograd
šk. 2018/19.

Sistemski katalog

- Baza podataka o bazama podataka
- Razne uloge, npr.
  - Provera korektnosti iskaza / upita
  - Optimizacija upita
- Informacije o raznim objektima u sistemu kao što su bazne tabele, pogledi, indeksi, baze podataka, ovlašćenja za pristup objektima, itd.
- Specifični za svaki RSUBP
- Nekoliko desetina tabela

- Sadržaj – tabela svih tabela (po jedan red za svaku), tabela svih kolona, i sl. - podaci o imenu, vlasniku, broju kolona / pripadnosti tabeli i sl.
- Bazne tabele ili pogledi
- Pretraživanje – SQL-om
- Ažuriranje zabranjeno korisnicima
- Ažuriranje - sistemsko
-

Sistemski katalog – DB2

- Neki od pogleda (delimična struktura)

- SYSIBM.SYSTABLES (NAME, CREATOR, TYPE, DBNAME, TSNAME, COLCOUNT, CARD, NPAGES, PARENTS, CHILDREN, KEYCOLUMNS...)
- (red za svaku tabelu, pogled, alias)

- SYSIBM.SYSCOLUMNS (NAME, TBNAME, TBCREATOR, COLNO, COLTYPE, LENGTH, NULLS, COLCARD, ...)
- (red za svaku kolonu svake tabele ili pogleda)

- SYSIBM.SYSINDEXES (INAME, CREATOR, TBNAME, TBCREATOR, FIRSTKEYCARD, COLCOUNT, FULLKEYCARD, NLEAF, NLEVELS, ...)

- Interpretacija

Sistemski katalog – DB2

- Primer pretraživanja

```
SELECT TBNAME, TBCREATOR
FROM SYSIBM.SYSCOLUMNS
WHERE NAME = ‘I_SIF’
```

Sistematski pristup optimizaciji

- Optimizacija neophodna zbog neophodne efikasnosti
- Kod nerelacionih sistema optimizaciju sprovodi korisnik,
- Kod relacionih sistema - automatska optimizacija – prednosti
- Cilj optimizacije – poboljšanje
- Osnovne vrste optimizacije koje se sreću u procesorima upitnih jezika
  - strategija algebarske transformacije upita
  - strategija procene cene.

- Primer pravila algebarske transformacije:
  - Primeniti restrikciju što je moguće ranije u izvršenju upita

Algebarska transformacija upita

- Primer:
- Nazivi izdavača koji izdaju knjigu sa šifrom k2;

```
SELECT DISTINCT I.NAZIV
FROM I JOIN KI ON I.I_SIF = KI.I_SIF
WHERE KI.K_SIF = 'k2'
```

- Npr, 100 izdavača (vrsta u tabeli I) i 10000 izdanja (vrsta u tabeli KI), 20 se odnosi na izdanja knjige sa šifrom k2.
- ***Jedan način izvršavanja upita:***
- 1. Izračunati Dekartov proizvod tabela I i KI; čitanje 10100 vrsta sa diska, konstruisanje privremene tabele sa 1000000 vrsta (disk)
- 2. Izvršiti restrikciju po logičkom izrazu I.I_SIF = KI.I_SIF AND KI.K_SIF = 'k2'; sekvencijalno čitanje 1000000 vrsta sa diska, konstruisanje privremene tabele sa samo 20 vrsta (unutrašnja memorija)
- 3. Projektovati na atribut NAZIV, čime se dobija najviše 20 podataka.

Algebarska transformacija upita

- ***Drugi način izvršavanja upita:***
- 1'. Izvršiti restrikciju tabele KI po uslovu KI.K_SIF = 'k2'; čitanje (sekvencijalno) 10000 vrsta, proizvođenje privremene tabele sa 20 vrsta (unutrašnja memorija)
- 2'. Spojiti tabelu dobijenu u koraku 1' sa tabelom I po jednakim vrednostima atributa I_SIF; čitanje 100 vrsta tabele I sa diska i proizvođenje privremene tabele sa 20 vrsta (unutrašnja memorija)
- 3'. Projektovati privremenu tabelu dobijenu u koraku 2' na atribut NAZIV.

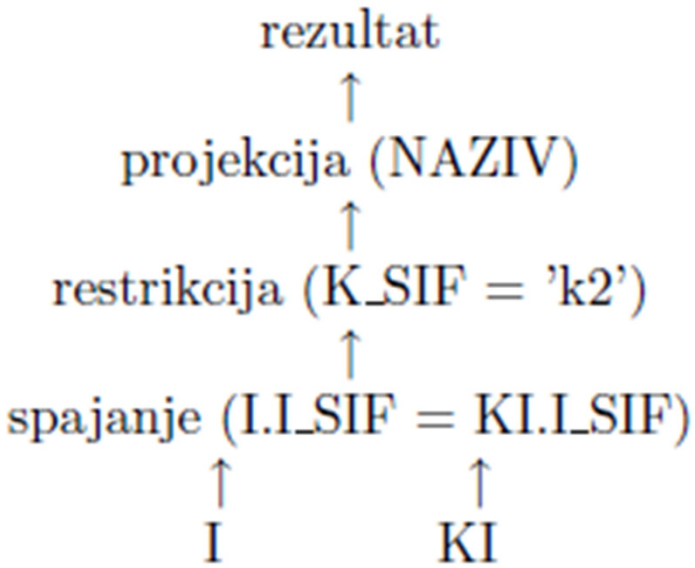
- Ako je mera efikasnosti – broj čitanja sa diska i upisa na disk
- Drugi postupak 200 puta efikasniji od prvog
- Ako je tabela KI indeksirana po atributu K_SIF, broj čitanja u koraku 1' je 20 a ne 10000.
- Indeks može da pomognu i u koraku 2', tako što bi se broj čitanja sa diska smanjio sa 100 na najviše 20

Sistematski pristup optimizaciji

- I Prevesti upit u internu reprezentaciju koja bi ga učinila neosetljivim na detalje specifične sintakse.
- II Prevesti internu reprezentaciju upita u kanoničku formu, kojom bi se anulirale razlike između semantički ekvivalentnih a sintaksno različitih formulacija upita
- III Izabrati kandidate za procedure niskog nivoa kojima se mogu realizovati pojedine operacije sadržane u upitu
- IV Generisati planove izvršavanja kompletnih upita i izabrati najjeftiniji

Prevođenje upita u internu reprezentaciju

- ((I * KI)[K_SIF = 'k2'])[NAZIV]



Kanonička forma

- Definicija (kanonička forma) Neka je dat skup Q objekata i relacija ekvivalencije među objektima. Tada se podskup $C \subset Q$ zove *skup kanoničkih formi* skupa Q pri izabranoj relaciji ekvivalencije ako je svaki objekat $q \in Q$ ekvivalentan nekom (tačno jednom) objektu $c \in C$. Objekat c zove se *kanonička forma* objekta q.

- U kontekstu optimizacije upita, skup Q je skup upita, a relacija ekvivalencije među objektima je relacija ekvivalencije među upitima. Smisao definicije je da omogući da se proučavanjem nekog podskupa C kanoničkih formi, umesto proučavanjem skupa Q svih upita, dokažu bitna svojstva celog skupa Q.

- Semantička ekvivalentnost
- Sintaksna ekvivalentnost: pravila transformacije

Primer pravila transformacije

- Skup pravila:
- Korektnost – sintaksna ekvivalentnost implicira semantičku ekvivalentnost
- Kompletnost –sinatksna ekvivalentnost što bliža semantičkoj ekvivalentnosti

- Primeri pravila:
- Logičke izraze restrikcije prevesti u konjunktivnu normalnu formu, tj. u konjunkciju prostih poređenja i/ili disjunkcija prostih poređenja. Na primer, izraz
- *p* OR (*q* AND *r*) treba zameniti izrazom
- (*p* OR *q*) AND (*p* OR *r*);

Primer pravila transformacije

- Operacijama restrikcije dati viši prioritet (zagradama) u odnosu na operacije spajanja, tj. izraz oblika

- (*R* * *S*)[restrikcija-nad-*S*]
  zameniti izrazom
  (*R**(*S*[restrikcija-nad-*S*])),

- Izraz (*R**S*)[restrikcija-nad-*R* AND restrikcija-nad-*S*]
  zameniti izrazom
  (*R*[restrikcija-nad-*R*]) *(*S*[restrikcija-nad-*S*]);

Primer pravila transformacije

- Niz restrikcija kombinovati u jednu restrikciju, tj. izraz oblika

(*R*[restrikcija-1])[restrikcija-2]

zameniti izrazom

R[restrikcija-1 AND restrikcija-2]

- U nizu projekcija čije su liste atributa, zdesna ulevo, u relaciji inkluzije, eliminisati sve projekcije osim poslednje, tj. izraz oblika

(*R*[lista-atributa-1])[lista-atributa-2], ako je lista-atributa-2 $\subset$ lista-atributa-1,

zameniti izrazom

R[lista-atributa-2]

- Restrikciju projekcije zameniti projekcijom restrikcije, tj. izraz oblika

(*R*[lista-atributa-1])[restrikcija-1]

zameniti izrazom

(*R*[restrikcija-1])[lista-atributa-1]

Pravila transformacije

- Lista pravila nije kompletna
- Razni sistemi je dopunjuju
- Na primer,

ako je *S.A* strani ključ koji se odnosi na primarni ključ *R.A*, onda se izraz

(*S* * *R*)[*A*] može zameniti izrazom *S*[*A*];

na primer, izraz (KI * K)[K_SIF] može da se zameni izrazom KI[K_SIF].

- Sistem INGRES (POSTGRES): pravilo proširenja restrikcije, npr.

A.F1 > B.F2 AND B.F2 = 3

zameniti izrazom

A.F1 > B.F2 AND B.F2 = 3 AND A.F1 > 3;

– izraz I.DRZAVA > P.DRZAVA AND P.DRZAVA =’Amerika’,

zameniti izrazom

I.DRZAVA > P.DRZAVA AND P.DRZAVA =’Amerika’ AND I.DRZAVA > ’Amerika’

Optimizacija u SYSTEM R

- SYSTEM R je prvi kompletni prototipski RSUBP
- To je kompilatorski sistem, što znači da se jednom optimizovani upit može izvršiti veliki broj puta, sve dok se ne promene parametri na osnovu kojih je optimizovani upitni plan sačinjen
- Ova mogućnost značajno opravdava optimizaciju i njenu cenu
- Metoda optimizacije sistema SYSTEM R sa manjim izmenama primenjena je i u svim komercijalnim proizvodima razvijenim iz njega, kao što su DB2 i SQL/DS

Optimizacija u SYSTEM R

- U prvom koraku optimizator sistema odlučuje o redosledu izvršenja SELECT blokova ugnježdenih jedan u drugi; u SYSTEM R izabran je redosled od unutrašnjeg, redom, ka spoljašnjim blokovima
- Zatim se bira najjeftiniji upitni plan za svaki od blokova
- Prvi korak služi kao heuristika koja redukuje broj mogućih planova
- S druge strane, svaki SQL upit koji sadrži ugnježdene blokove može da se transformiše u upit spajanja bez ugnježdenih blokova, pa takva transformacija daje bolje rezultate u optimizaciji celog upita.

Optimizacija u SYSTEM R

- Za dati upitni blok, postoje dva slučaja:
- **a)** Ako je upit jednorelacioni (ne uključuje spajanje), za konstrukciju upitnog plana optimizator koristi statističke informacije iz sistemskog kataloga, formule za procenu veličine međurezultata i cene operacija niskog nivoa
- Sistem izračunava, na osnovu informacija dobijenih iz kataloga, cenu svakog od konačno mnogo načina za izvršavanje upita, i odabira najpovoljniji.
- Statističke informacije u sistemskom katalogu uključuju:
- – broj *n*-torki u svakoj relaciji
- – broj stranica koje zauzima svaka relacija
- – procenat stranica koje svaka relacija zauzima u odnosu na ukupan broj relevantnih stranica u bazi podataka
- – broj različitih vrednosti za svaki indeks
- – broj stranica koje zauzima svaki indeks

Optimizacija u SYSTEM R

- Ovi podaci su statički i ažuriraju se posebnim uslužnim programom koji je potrebno izvršiti posle svakog značajnijeg ažuriranja baze
- U SYSTEM R poziv tog programa je UPDATE STATISTICS
- U sistemu DB2 poziv odgovarajućeg programa je RUNSTATS u kome se, kao parametar, navodi objekat (tabela, indeks, prostor tabela, itd.) čije će se statističke ocene izračunati i upisati u sistemski katalog
- Podaci iz sistemskog kataloga imaju presudnu ulogu u izgradnji plana upita.

- Neka je potrebno izvršiti jednorelacioni upit za koji je iz sistemskog kataloga dobijen broj *n*-torki (*B*) i broj stranica koje zauzima relacija (*S*).
- Upit se izvršava tako što se primenjuje jedan po jedan uslov restrikcije (koja je oblika konjunktivne normalne forme),
- pri čemu se u svakom koraku primenjuje, na prethodno dobijeni međurezultat, najjeftiniji mogući od sledećih postupaka:

Optimizacija u SYSTEM R

- Ako je uslov – poređenje oblika $A = c$, gde je A atribut a c konstanta, primeniti grupišući indeks nad A (ako postoji) da bi se dobile n-torke koje zadovoljavaju uslov.
- Ako je I broj različitih vrednosti indeksa, onda je cena ovog koraka (broj obraćanja disku) S/I.

- Ako je uslov – poređenje oblika $A \Theta c$, gde je $\Theta \in \{<, <=, >, >=\}$, primeniti grupišući indeks nad A (ako postoji) da bi se dobio odgovarajući podskup relacije.
- Cena ovog koraka je $S/2$ jer se pretraži približno polovina relacije. (Poređenje <> nije uključeno, jer se ne očekuje značajna selektivnost; gotovo sve n-torke moraju biti pretražene).

Optimizacija u SYSTEM R

- Ako je uslov – poređenje oblika $A = c$, primeniti negrupišući indeks nad A (ako postoji) da bi se dobile n-torke koje zadovoljavaju uslov.
- Cena koraka je B/I jer je, s obzirom da je indeks negrupišući, položaj n-torki nezavisan.

- Ako nema indeksa, relacija se može čitati redom, primenjujući sve uslove restrikcije, što ima cenu proporcionalnu sa S.

- Ako je uslov – poređenje oblika $A \Theta c$, gde je $\Theta \in \{<, <=, >, >=\}$, primeniti negrupišući indeks nad A (ako postoji).
- Cena ovog koraka je $B/2$.

Optimizacija u SYSTEM R

- **b)** Ako upit uključuje dve ili više relacija koje treba da se spoje, optimizator prvo odabira pristupni put za svaku od relacija na način opisan pod a), a zatim odlučuje o redosledu izvođenja spajanja.
- Ove dve radnje su zavisne, jer se strategija za pristup pojedinačnoj relaciji bira uzimajući u obzir i karakteristike rezultata koje su od značaja za operaciju spajanja (npr. sortiranost neke od relacija rezultata koje učestvuju u spajanju).
- Niz spajanja odvija se strogo sekvencijalno, tj. izraz *A*B*C*D* izvršava se kao izraz (((*A*B*) **C*) **D*), što takođe predstavlja jednu od heuristika kojima se smanjuje broj mogućih planova upita.

Optimizacija u SYSTEM R

- Za izvršenje specifične operacije spajanja, npr. *A * B*, SYSTEM R koristi jednu od dve metode:

- metodu ugnježdenih petlji ili
- metodu sortiranja/ spajanja.

- Izbor metode se vrši na osnovu formula cene.

Optimizacija u SYSTEM R

- Metoda ugnježdenih petlji može se opisati sledećim algoritmom:

BEGIN

FOR svaku *n*-torku *r* relacije *A* DO

FOR svaku *n*-torku *s* relacije *B* DO

IF *n*-torke *r* i *s* imaju jednake vrednosti na atributima spajanja

THEN konstruisati spojenu (*r; s*)-torku

END

- Razna poboljšanja metode postižu se korišćenjem indeksa (grupišućeg ili negrupišućeg) po atributima spajanja.

Optimizacija u SYSTEM R

- Metoda sortiranja/spajanja sastoji se od sortiranja relacija *A;B* po vrednostima atributa spajanja, i spajanja sortiranih relacija po jednakosti atributa.

- Pristup torkama relacija *A* i *B* u fizičkom redosledu sada je najefikasniji, a sam algoritam se može realizovati tako da se spajanje dveju relacija obavi jednim čitanjem svake od njih

Optimizacija u SYSTEM R

- U slučaju višestrukog spajanja, npr. (*A* * *B*) * *C*, obe metode dopuštaju da se svaka proizvedena *n*-torka relacije *A* * *B* preda procesu koji je spaja sa *n*-torkama relacije *C*, bez proizvođenja cele relacije *A* * *B*.

- Cena optimizacije kompleksnih upita u sistemu SYSTEM R je samo nekoliko hiljada bajtova memorije i nekoliko desetih delova sekunde na IBM/370 sistemu, što je zanemarljivo u slučaju da se jednom optimizovani upit izvrši veći broj puta.

Optimizacija u DB2

- Sistem DB2 koristi i dodatne tehnike optimizacije u odnosu na svoj prototip.
- Tako, na primer, upit koji se odnosi na broj *n*-torki u tabeli I koje zadovoljavaju uslov STATUS = 20,

```
SELECT COUNT(*)
FROM I
WHERE STATUS = 20
```

- DB2 izvršava bez pristupa samoj tabeli, brojanjem elemenata podataka indeksa atributa STATUS (za vrednost 20) ako takav indeks postoji.

Optimizacija upita - dekompozicija

- INGRES: višerelacioni upiti
- *dekompozicija upita*

- razlaganje upita nad više *n*-tornih promenljivih u niz podupita nad pojedinačnim *n*-tornim promenljivim,
- primenom postupaka
  - *razdvajanja* i
  - *supstitucije n-torki*

- Razdvajanje je proces uklanjanja komponente upita koja ima tačno jednu zajedničku promenljivu sa ostatkom upita.
- Primer:
- “naći nazive izdavača iz Srbije koji su izdali bar jednu knjigu poezije u tiražu ne manjem od 5000”

Optimizacija upita - dekompozicija

- QUEL:

```
RANGE OF I IS I
RANGE OF K IS K
RANGE OF KI IS KI
(1) RETRIEVE (I.NAZIV) WHERE I.DRZAVA = ’Srbija' AND
I.I_SIF = KI.I_SIF AND KI.TIRAZ >= 5000 AND
KI.K_SIF = K.K_SIF AND K.OBLAST = 'poezija‘
```

Optimizacija upita - dekompozicija

- Razdvajanje:

- Primenjeno na promenljivu K, na primer, razdvajanje proizvodi sledeća dva upita, ekvivalentna polaznom upitu (1):

- (2) RETRIEVE INTO K'(K.K_SIF) WHERE K.OBLAST = 'poezija'

- RETRIEVE (I.NAZIV) WHERE I.DRZAVA = 'Srbija' AND
- (3) I.I_SIF = KI.I_SIF AND KI.TIRAZ >= 5000 AND
- KI.K_SIF = K'.K_SIF

Optimizacija upita - dekompozicija

- Slično se iz upita (3), razdvajanjem po promenljivoj KI, dobija sledeći par upita ekvivalentan upitu (3):

- (4) RETRIEVE INTO KI'(KI.I_SIF, KI.K_SIF) WHERE KI.TIRAZ >= 5000
- (5) RETRIEVE (I.NAZIV) WHERE I.DRZAVA = 'Srbija' AND
- I.I_SIF = KI'.I_SIF AND KI'.K_SIF = K'.K_SIF

Optimizacija upita - dekompozicija

- Na kraju, iz upita (5), razdvajanjem po promenljivoj I dobija se sledeći par upita ekvivalentan upitu (5):

- (6) RETRIEVE INTO I'(I.I_SIF, I.NAZIV) WHERE I.DRZAVA=’Srbija'
- (7) RETRIEVE (I'.NAZIV) WHERE I'.I_SIF=KI'.I_SIF AND KI'.K_SIF=K'.K_SIF

Optimizacija upita - dekompozicija

- Postupak se nastavlja razdvajanjem po dvopromenljivom upitu koji uključuje promenljive KI’ i K’.

- Tako je upit (7) ekvivalentan sledećem paru upita:

- (8) RETRIEVE INTO KI''(KI'.I_SIF) WHERE KI'.K_SIF = K'.K_SIF
- (9) RETRIEVE (I'.NAZIV) WHERE I'.I_SIF = KI''.I_SIF

Optimizacija upita - dekompozicija

- *Supstitucija n-torki* je postupak zamene jedne promenljive u upitu, redom, jednom po jednom *n*-torkom relacije koja odgovara toj promenljivoj.
- Taj postupak se u razmatranom primeru primenjuje na sledeći način:

- Ako, na primer, skup vrednosti atributa K_SIF u relaciji K' jeste {k2, k6},
- onda se dvopromenljivi upit (8) zamenjuje jednopromenljivim upitom u kome obe vrednosti iz tog skupa zamenjuju, redom, komponentu K'.K_SIF.

- Tako se upit (8) zamenjuje upitom

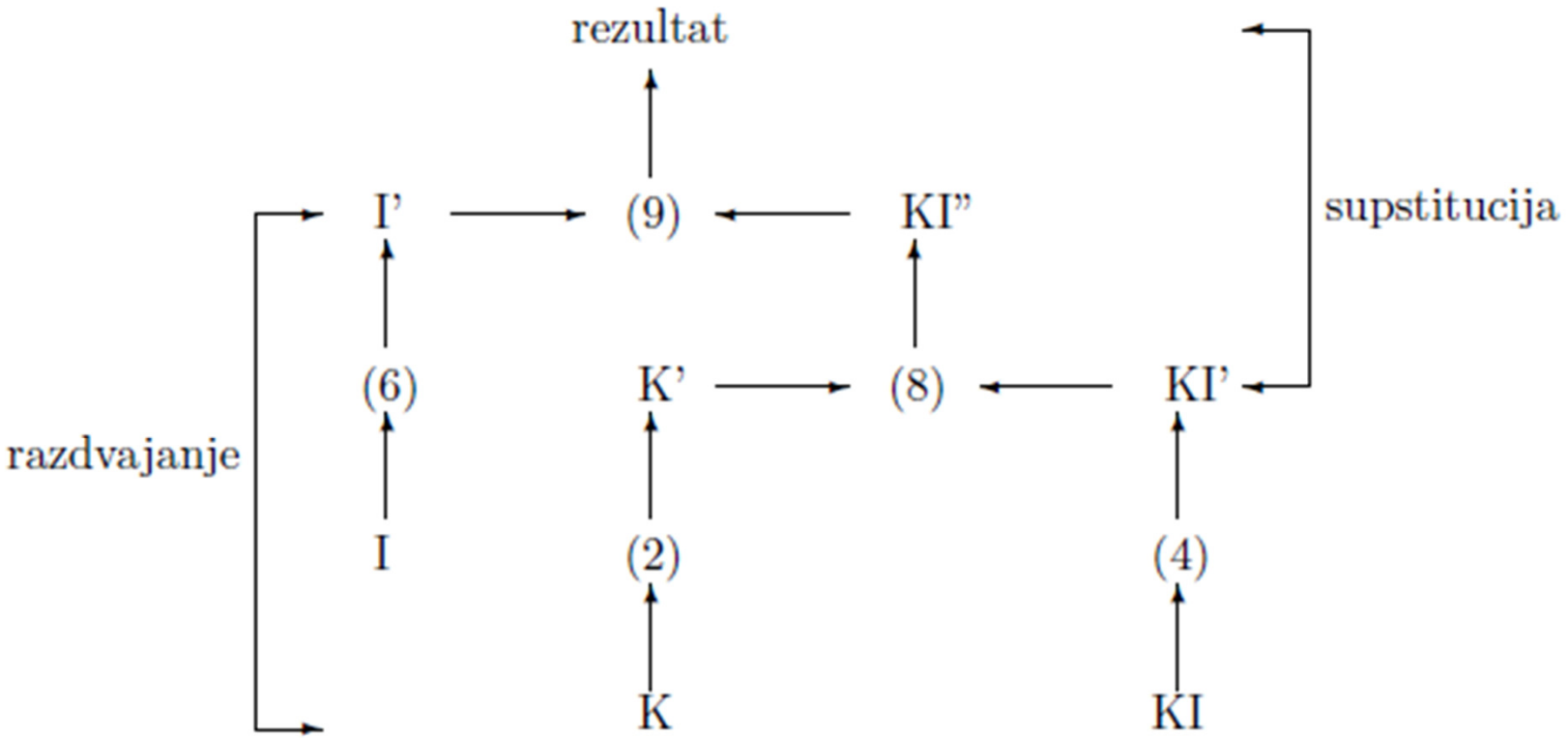
- RETRIEVE INTO KI''(KI'.I_SIF) WHERE KI'.K_SIF='k2' OR KI'.K_SIF='k6'

Optimizacija upita - dekompozicija

- Slično, ako je skup vrednosti atributa I_SIF u relaciji KI” {i1, i4}, dvopromenljivi upit (9) zamenjuje se jednopromenljivim u kome svaka od dve vrednosti iz tog skupa zamenjuje, redom, komponentu KI”.I_SIF.

- Upit (9) se zamenjuje upitom
- RETRIEVE (I'.NAZIV) WHERE I'.I_SIF = 'i1' OR I'.I_SIF = 'i4'

Optimizacija upita - dekompozicija



Slika 7.2: Drvo dekompozicije upita

Optimizacija upita – preporuke za DB2 i Oracle db (IBM Knowledge Center)

- **Optimize queries based on the query optimization guidelines**
- The SQL best practices to ensure query optimization:
  1. ***Index all the predicates in JOIN, WHERE, ORDER BY and GROUP BY clauses***.
     - Without proper indexes, SQL queries can cause table scans, which causes either performance or locking problems.
     - It is recommended that all predicate columns be indexed.
     - The exception being where column data has very low cardinality.

Optimize queries based on the query optimization guidelines

2. ***Avoid using functions in predicates.***
   - The index is not used by the database if there is a function on the column. For example:

     SELECT * FROM TABLE1 WHERE UPPER(COL1)='ABC‘
   - As a result of the function UPPER(), the index on COL1 is not used by database optimizers. If the function cannot be avoided in the SQL, you need to create a function-based index in Oracle or generated columns in DB2 to improve performance.
3. ***Avoid using wildcard (%) at the beginning of a predicate.***
   - The predicate LIKE '%abc' causes full table scan. For example:

     SELECT * FROM TABLE1 WHERE COL1 LIKE '%ABC
   - This is a known performance limitation in all databases.
4. ***Avoid unnecessary columns in SELECT clause.***
   - Specify the columns in the SELECT clause instead of using SELECT *. The unnecessary columns places extra loads on the database, which slows down not just the single SQL, but the whole system.

Optimize queries based on the query optimization guidelines

5. ***Use inner join, instead of outer join if possible***.
   - The outer join should only be used if it is necessary.
   - Using outer join limits the database optimization options which typically results in slower SQL execution.
6. ***DISTINCT and UNION should be used only if it is necessary.***
   - DISTINCT and UNION operators cause sorting, which slows down the SQL execution.
7. ***Use UNION ALL instead of UNION, if possible***, as it is much more efficient.

Optimize queries based on the query optimization guidelines

8. ***The ORDER BY clause is mandatory in SQL if the sorted result set is expected.***
   - The ORDER BY keyword is used to sort the result-set by specified columns. Without the ORDER BY clause, the result set is returned directly without any sorting. The order is not guaranteed. Be aware of the performance impact of adding the ORDER BY clause, as the database needs to sort the result set, resulting in one of the most expensive operations in SQL execution.

Optimizacija upita – preporuke za DB2 i Oracle db

- **Push predicates into the OUTER JOIN clause whenever possible**

- For SQL queries with the LEFT OUTER JOIN, pushing predicates of the right table from the WHERE clause into the ON condition helps the database optimizer generate a more efficient query. Predicates of the left table can stay in the WHERE clause.

- Similarly, for the SQL queries with the RIGHT OUTER JOIN, predicates for the left table should be moved from the WHERE clause into the ON condition.
- For example, the suboptimal query is rewritten by pushing predicates applicable to the table TAB_B into the ON clause. The TAB_A specific predicates in the WHERE clause can either stay, or be pushed into the ON clause:

Push predicates into the OUTER JOIN clause whenever possible

- Suboptimal SQL statement:

```
SELECT TAB_A.COL1, TAB_B.COL1
FROM TAB_A LEFT OUTER JOIN TAB_B
ON TAB_A.COL3 = TAB_B.COL3
WHERE TAB_A.COL1=123 AND TAB_B.COL2=456;
```

- Optimized SQL statement:

```
SELECT TAB_A.COL1, TAB_B.COL1
FROM TAB_A LEFT OUTER JOIN TAB_B
ON TAB_A.COL3 = TAB_B.COL3 AND TAB_B.COL2=456
WHERE TAB_A.COL1=123;
```

Duplicate constant condition for different tables whenever possible

- When two tables, A and B, are joined and there is a constant predicate on one of the joined columns, for example, A.id=B.id and A.id in (10, 12), the constant predicate should be duplicated for the joined column of the second table. That is, A.id=B.id and A.id in (10, 12) and B.id in (10, 12).

- For example, TAB_A has a LEFT OUTER JOIN relationship with TAB_B. If there is a TAB_A specific conditions and a cross table condition with TAB_B, create an extra TAB_B specific condition based on TAB_A requirement and keep cross table conditions in the ON clause:

Duplicate constant condition for different tables whenever possible

- Suboptimal SQL statement:

```
SELECT TAB_A.COL1, TAB_B.COL1
FROM TAB_A LEFT OUTER JOIN TAB_B
ON TAB_A.COL3 = TAB_B.COL3
WHERE TAB_A.COL1 IN (123, 456) AND TAB_B.COL2=TAB_A.COL1;
```

- Optimized SQL statement:

```
SELECT TAB_A.COL1, TAB_B.COL1
FROM TAB_A LEFT OUTER JOIN TAB_B
ON TAB_A.COL3 = TAB_B.COL3 AND TAB_B.COL2 IN (123, 456) AND
TAB_B.COL2=TAB_A.COL1
WHERE TAB_A.COL1 IN (123, 456);
```

Duplicate constant condition for different tables whenever possible

- In particular, if the constant predicate has only 1 value (e.g COL1=123), the second predicate should be converted to a constant predicate too.

- For example:
- Suboptimal SQL statement:

```
SELECT TAB_A.COL1, TAB_B.COL1
FROM TAB_A LEFT OUTER JOIN TAB_B
ON TAB_A.COL3 = TAB_B.COL3
WHERE TAB_A.COL1=123 AND TAB_B.COL2=TAB_A.COL1;
```

- Optimized SQL statement:

```
SELECT TAB_A.COL1, TAB_B.COL1
FROM TAB_A LEFT OUTER JOIN TAB_B
ON TAB_A.COL3 = TAB_B.COL3 AND TAB_B.COL2=123
WHERE TAB_A.COL1=123;
```