Obrada i optimizacija upita kod Oracle sistema za upravljanje bazama podataka

Seminarski rad

Uroš Vukić

Broj indeksa: 1086

Sistemi za upravljanje bazama podataka

Elektronski fakultet u Niš

Sadržaj

1 Uvod

SQL je deklarativni jezik, što znači da nam on govori šta treba uraditi, ali ne i kako. Za jedan upit postoji veliki broj ekvivalentnih izvršenja od čega je veliki broj daje loše performanse. Savremeni sistemi za upravljanje bazama podataka koriste specijalne komponente – Optimizatore – koje pronalaze optimalna rešenja za izvršenje upita. Dati upit je potrebno predstaviti pomoću odgovarajuće reprezentacije koju je moguće dalje obrađivati.

U nastavku rada, biće navedeni osnovni koraci u obradi upita, tehnike za generisanje strukture koja predstavlja upit (operatorsko stablo), kao i tehnike za optimizaciju izvršenja operacija upita. Pored teorijskog pregleda tehnika, daćemo i osnovne detalje implementacije ovih tehnika kod Oracle sistema baza podataka kao i neke praktične primere.

2 Teorijski pregled obrade i optimizacije upita

2.1 Koraci pri obradi upita

Upit napisan u jeziku poput SQL se prvo skenira, parsira i validira (ovde se nećemo detaljno baviti prasiranjem i sintaksnom analizom upita, već ćemo dati samo pregled samih aktivnosti). Skener identifikuje tokene - poput ključnih reči SQL jezika, imena atributa i relacija – koji se javljaju u samom upitu. Zatim parser proverava sintaksu upita da bi utvrdio da li je napisan u skladu sa definisanim pravilima gramatike. Zatim se vrši leksička analiza tj. proverava da li su imena atributa i relacija korektna u pogledu šeme relacije kojoj se upućuje upit. Nakon toga, kreira se interna reprezentacija upita uz pomoću grafa koji se naziva graf upita (query graph). DBMS određuje plan izvršenja upita (execution plan) i način pribavljanja podataka iz samih tabela. Većina upita uglavnom ima više ekvivalentnih planova izvršenja, a posao optimizatora upita (query optimizer) je da nađe najpogodniji (najbrži ili sa najmanjom cenom). Generator koda (code generator) generiše kod za najoptimalnije stablo i prosleđuje ga izvršnom procesoru baze podataka (database runtime processor) na izvršenje.

Optimizator upita najčešće ne pronalazi najoptimalniji plan izvršenja već „dovoljno dobar“ plan izvršenja. Pronalaženje najoptimalnijeg plana izvršenja je vremenski jako zahtevna operacija osim u slučajevima kada je upit jako jednostavan. Pronalaženje najoptimalnijeg plana može zahtevati detaljnije informacije koje nisu prisutne u katalogu DBMS-a.

Relacioni DBMS prilikom obrade upita sistematski generiše više planova izvršenja i bira razumno efikasni plan (razlike u planovima proizilaze od upotrebe različitih algoritama za izvršenje iste operacije kao i različitog redosleda primene operatora).

2.2 Prevođenje SQL upita u operatore relacione algebre

SQL upit se prvo prevodi u reprezentaciju u ekvivalentnu reprezentaciju proširene relacione algebre – predstavljene uz pomoć stabla upita (query tree) koje se kasnije može optimizovati. Tipično se SQL upiti se razlažu u blokove upita (query blocks) koji predstavljaju osnovne jedinice koje se mogu prevesti u algebarske operacije. Blok upita (query block) se sastoji od jednog SELECT-FROM-WHERE izraza kao i GROUP BY i HAVING klauzula ukoliko su one prisutne u upitu. Na ovaj način se ugnježdeni upiti tretiraju kao zasebni blokovi. Agregacioni opratori – poput MIN, MAX, SUM i COUNT – se prevode u odgovarajuće operatore proširene relacione algebre.

Primer:

```
SELECT Lname, Fname
FROM EMPLOYEE
WHERE Salary > ( SELECT MAX (Salary)
                 FROM EMPLOYEE
                 WHERE Dno=5 );
```

Ovaj upit pribavlja sve zaposlene koji zarađuju više od najveće plate u departmanu broj 5. Upit sadrži ugnježdeni upit pa se stoga prevodi uz pomoć dva bloka:

1. Unutrašnji blok koji određuje najveću platu u departmanu broj 5:

```
( SELECT MAX (Salary)
FROM EMPLOYEE
WHERE Dno=5 )
```

2. Spoljašnji upit koji pronalazi sve radnike koji imaju platu veću od najveće u departmanu broj 5:
   SELECT Lname, Fname
   FROM EMPLOYEE
   WHERE Salary > c

C - je konstanta koja predstavlja izračunatu vrednost iz unutrašnjeg bloka - ugnježdenog upita. Unutrašnji blok se prevodi u sledeći izraz proširene relacione algebre (gde $\Im$ predstavlja reprezentaciju max agregacione funkcije u proširenoj relacionoj algebri):

$$\Im_{\text{MAX Salary}}(\sigma_{\text{Dno=5}}(\text{EMPLOYEE}))$$

A spoljašnji upit se prevodi u:

$$\pi_{\text{Lname,Fname}}(\sigma_{\text{Salary>c}}(\text{EMPLOYEE}))$$

Optimizator upita dalje odredio plan izvršenja za svaki blok zasebno.

2.3 Optimizacije upita upotrebom heuristike

Skener i parser SQL upita prvo generišu strukturu koja odgovara inicijalnoj reprezentaciji upita u obliku stabla upita (query tree) koje se kasnije optimizuje. Ovim optimizacijama se dobija optimizovana reprezentacija koja odgovara izvršnom planu upita. Uzimajući to u obzir, izvršni plan se generiše tako da obavlja grupu operacija u zavisnosti od dostupnih načina pristupa podacima koji se koriste u upitu.

Jedna od glavnih pravila heuristike predstavlja primena operatora selekcije i projekcije pre kreiranja spoja (JOIN) ili primene drugih binarnih operacija, jer veličina rezultata binarnih operacija poput spoja zavisi od proizvoda veličina ulaznih podataka (tabela) koje se spajaju. Selekcija i projekcija smanjuju veličinu ulaznih podataka i zbog toga ih treba primeniti pre spojeva i drugih binarnih operacija.

2.4 Reprezentacija stabla upita i grafova upita

Stablo upita (query tree) je struktura podataka koja se koristi za reprezentaciju izraza relacione algebre. Relacije baze podataka koje se koriste u samom upitu su predstavljene listovima stabla, a unutrašnji čvorovi predstavljaju operatore relacione algebre. Izvršenje stabla upita se sastoji od izvršenja operacije unutrašnjeg čvora stabla nad ulaznim relacijama (kad god su one spremne), a zatim se vrši zamena tog čvora rezultujućom relacijom. Redosled izvršenja stabla upita počinje od listova – ulaznih relacija baze podataka i završava se u korenu stabla kada se dobija konačni rezultat upita.

Primer:

```
SELECT   P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
FROM     PROJECT AS P, DEPARTMENT AS D, EMPLOYEE AS E
WHERE    P.Dnum=D.Dnumber AND D.Mgr_ssn=E.Ssn AND
         P.Plocation= 'Stafford';
```

Figure 1. Upit koji za svaki projekat koji se nalazi u Stafford-u, pribavlja broj projekta, broj departmana, prezime menadžera, adresu i rođendan menadžera projekta

Na Figure 1, dat je zapis upita uz pomoć operatora relacione algebre:

$$\pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}(((\sigma_{\text{Plocation='Stafford'}}(\text{PROJECT})) \bowtie_{\text{Dnum=Dnumber}}(\text{DEPARTMENT})) \bowtie_{\text{Mgr\_ssn=Ssn}}(\text{EMPLOYEE}))$$

Figure 2. Reprezentacija prethodnog upita pomoću operatora relacione algebre

Prevođenjem gore navedenog izraza relacione algebre dobija se sledeće stablo na Figure 3.

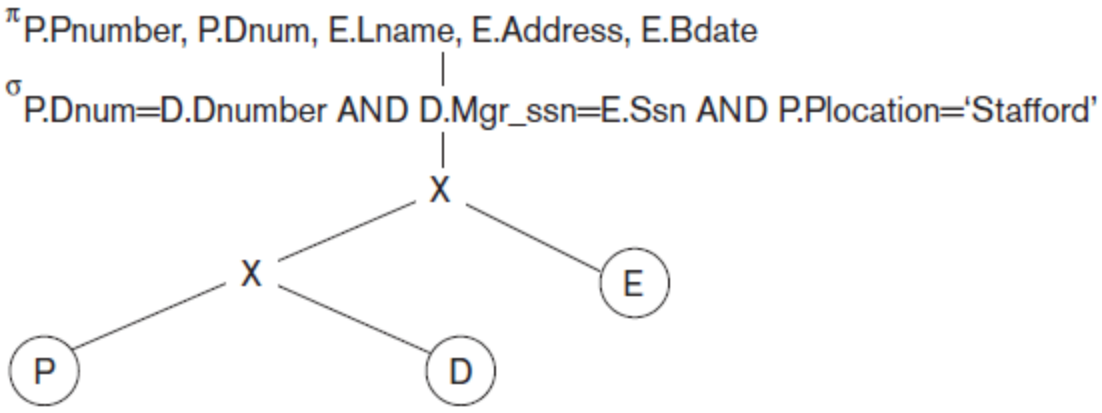

Figure 3. Stablo koje se dobija iz izraza relacione algebre

Daljom njegovom optimizacijom dobijamo stablo na figure 4.

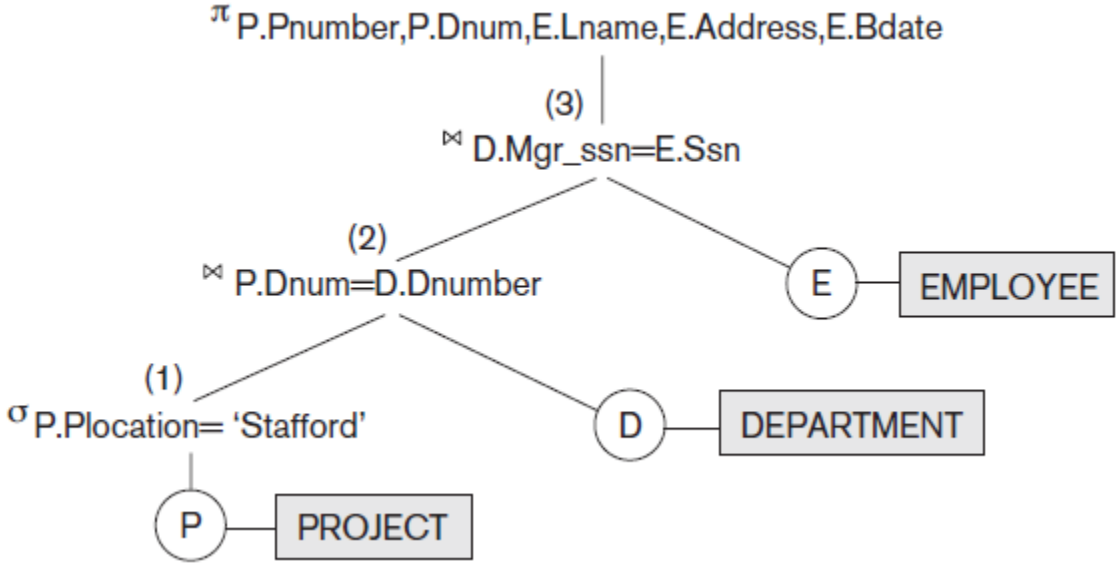

Figure 4. Optimizovano stablo izvršenja upita

Malo prirodnija struktura za reprezentaciju upita je graf upita. Relacije se u ovoj strukturi prikazuju čvorovima grafa (običan krug). Konstante se prikazuju kao konstantni čvorovi (dvostruki krugovi ili ovali). Selekcije i spojevi se prikazuju granama u grafu. Na slici je dat primer grafa gore navedenog upita. Prednost ove metode je to što za svaki upit postoji samo jedan graf. Mana ove reprezentacije je to što ona ne pokazuje redosled izvršenja upita. Iz tog razloga se stabla upita mnogo češće primenjuju u praksi.

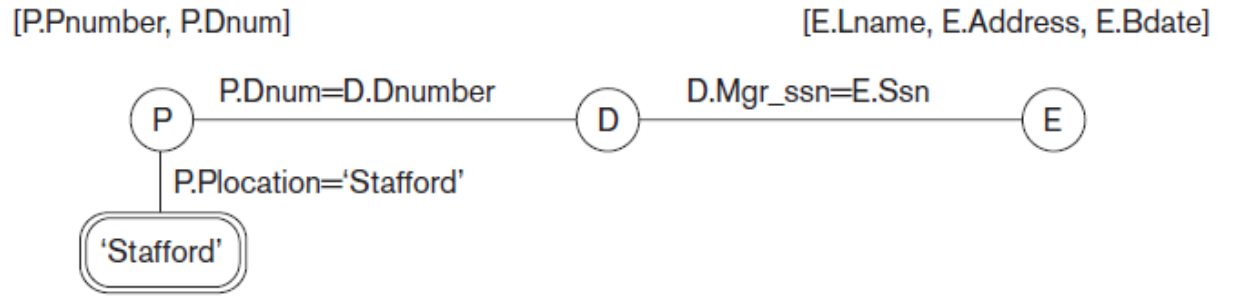

Figure 5. Reprezentacija upita pomoću grafa

2.5 Osnovna pravila transformacije operatora relacione algebre

Postoje puno pravila za transformaciju jedne reprezentacije operatora relacione algebre u drugu koja je njoj ekvivalentna. Sa aspekta optimizacije upita, interesuju nas konkretne operacije koje se obavljaju nad ulazom kao i dobijene rezultujuće relacije. Podsetimo se da je kod relacije redosled atributa nebitan. Ovo pravilo ćemo koristiti ovde prilikom definisanja ekvivalentnih transformacija. Ovde ćemo samo navesti pravila koja se koriste prilikom optimizacije upita bez njihovih dokaza:

1. Kaskada selekcija (σ) – Konjunkcija više selekcija se može razložiti na kaskadu tj. sekvencu pojedinačnih selekcija:
   $\sigma_{c1\ AND\ c2\ ..\ AND\ cn}(R) \equiv \sigma_{c1}(\sigma_{c2}(..(\sigma_{cn}(R))...))$
2. Komutativnost selekcije σ – Operacija selekcije je međusobno komutativna
   $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$
3. Kaskada projekcija (π) – Kaskada (sekvenca) operatora projekcije se može zameniti jednim operatorom projekcije na sledeći način:
   $\pi_{List1}(\pi_{List2}(...(\pi_{Listn}(R))...)) \equiv \pi_{List1}(R)$
4. Zamena mesta selekcije(σ) i projekcije (π) – Ukoliko uslov selekcije c uključuje samo atribute $A_1, ..., A_n$ iz liste projekcije, onda te dve operacije mogu zameniti mesta na sledeći način:
   $\pi_{A1, A2, ..., An}(\sigma_c(R)) \equiv \sigma_c(\pi_{A1, A2, ..., An}(R))$
5. Komutativnost spoja ($\bowtie$) kao i unakrsnog proizvoda ($\times$) – Operacija spoja je komutativna, kao i operacija unakrsnog proizvoda
   $R \bowtie_c S \equiv S \bowtie_c R$
   $R \times S \equiv S \times R$
   Važno je napomenuti da raspored atributa u rezultujućoj relaciji može varirati od redosleda primene spojeva (ili unakrsnih proizvoda). Značenje dobijenih relacija je isto zato što redosled atributa nije bitan prilikom definisanja relacije.

6. Zamena mesta selekcije (σ) i spoja (⋈) (ili unakrsnog proizvoda (×)) – Ukoliko su atributi iz uslova selekcije c uključeni samo atribute relacije R (jedne od relacija koje spajamo), onda ove dve operacije mogu zameniti mesta na sledeći način:
   $\sigma_c (R \bowtie S) \equiv (\sigma_c (R)) \bowtie S$
   Alternativa ovog pravila je sledeće: ako se uslov selekcije c može napisati kao c1 AND c2, gde uslov c1 uključuje samo atribute relacije R a uslov c2 uključuje samo atribute relacije S, operacije mogu zameniti mesta na sledeći način:
   $\sigma_c (R \bowtie S) \equiv (\sigma_{c1}(R)) \bowtie (\sigma_{c2}(S))$
   Ista pravila važe i kada se operacije spoja (⋈) zamene operacijama unakrsnog proizvoda(×).
7. Zamena mesta projekcije (π) i spoja (⋈) (ili unakrsnog proizvoda (×)) – Pretpostavimo da projekcija uključuje samo skup projektivnih atributa $L = \{A_1, ..., A_n, B_1, ..., B_m\}$, pri čemu atributi $A_1, ..., A_n$, pripadaju samo relaciji R, a atributi $B_1, ..., B_m$ pripadaju samo relaciji S. Ako uslov spoja c uključuje samo atribute skupa L, onda operacije projekcije i spoja mogu zameniti mesta na sledeći način:
   $\pi_L (R \bowtie_c S) \equiv (\pi_{A1, ..., An} (R)) \bowtie_c (\pi_{B1, ..., Bm} (S))$
   Ako uslov spoja c sadrži i neke dodatne atribute koji se ne nalaze u skupu L, oni se moraju dodati listi projektivnih atributa i potrebna je još jedna dodatna finalna operacija projekcije. Na primer, neka su $A_{n+1}, ..., A_{n+k}$ dodatni atributi u relaciji R i $B_{m+1}, ..., B_{m+k}$ dodatni atributi u relaciji S. Onda operacije mogu zameniti mesta na sledeći način:
   $\pi_L (R \bowtie_c S) \equiv \pi_L ((\pi_{A1, ..., An, An+1, ..., An+k} (R)) \bowtie_c (\pi_{B1, ..., Bm, Bm+1, ..., Bm+p} (S)))$
   Kod operacije unakrsnog proizvoda nemamo uslov c (uslov spoja), pa kod nje prvo pravilo uvek važi.
8. Komutativnost skupovnih operacija – Operacije unije (∪) i preseka (∩) su komutativne, ali operacija razlike (–) nije.
9. Asocijativnost spoja (⋈), unakrsnog proizvoda (×), unije (∪) i preseka (∩) – Ove četiri operacije su individualno asocijativne tj ako sa θ označimo neku skupovnu operaciju, onda važi sledeće:
   $(R \theta S) \theta T \equiv R \theta (S \theta T)$
10. Zamena mesta selekcije (σ) i skupovnih operacija – Operacija selekcije može zameniti mesta sa nekom od skupovnih operacija (∪,∩, –). Ako sa θ označimo neku skupovnu operaciju, onda važi sledeće:
    $\sigma_c (R \theta S) \equiv (\sigma_c (R)) \theta (\sigma_c (S))$
11. Zamena mesta projekcije (π) i unije (∪) – Operacije projekcije i unije mogu zameniti mesta na sledeći način:
    $\pi_L (R \cup S) \equiv (\pi_L (R)) \cup (\pi_L (S))$
12. Zamena kombinacije selekcija – unakrsni proizvod (σ,×) operacijom spoja (⋈) – Ako uslov selekcije c koja sledi nakon operacije unakrsnog proizvoda odgovara uslovu spoja, onda se par operacija selekcije i unakrsnog proizvoda (σ,×) može zameniti operacijom spoja (⋈) na sledeći način:
    $(\sigma_c (R \times S)) \equiv (R \bowtie_c S)$

2.6 Algoritam optimizovanja izvršenja upita

Na osnovu gore navedenih pravila (a koje ćemo referencirati u daljem tekstu), možemo dati osnovni algoritam za optimizovanje izvršenja upita. Algoritam kao ulaz uzima stablo algebarskih operacija koje

nastaje prevođenjem samog upita, a kao izlaz daje stablo sa optimizovanim redosledom operacija. Koraci algoritma su sledeći:

1. Na osnovu pravila 1, možemo jednu selekciju više uslova povezanih konjunkcijom (SELECT naredba povezana AND uslovima) podeliti na više kaskadnih selekcija. Na taj način dobijamo više slobode kada možemo primeniti selekciju i operaciju možemo spustiti duž više različitih grana operatorskog stabla.
2. Koristeći pravila 2, 4, 6 i 10 koja se odnose na komutativnost operacije selekcije (SELECT) sa ostalim operacijama, pomeramo operacije selekcije na što je moguće niže delove stabla koliko nam to dopuštaju sami atributi uključeni u uslov selekcije. Ako uslov selekcije uključuje samo atribute jedne table, to znači da se uslov selekcije može spustiti sve do listova stabla koji predstavljaju tu tabelu. Ako uslov selekcije uključuje atribute iz dve tabele, što znači da predstavlja uslov spoja, onda se uslov pomera sve do mesta u stablu nakon izvršenja spoja.
3. Koristeći pravila 5 i 9 koja se odnose na komutativnost i asocijativnost binarnih operacija, preurediti listove stabla tako da ispunjavaju sledeće uslove:
    - Pozicionirati listove stabla sa najrestriktivnijim (SELECT) uslovima na taj način da se ona izvršavaju na početku. Pod najrestriktivnijim (SELECT) misli se na one koji proizvode rezultujuću relaciju sa najmanje broja primeraka. Alternativni način objašnjenja bio bi najmanje selektivni uslov. Ova druga definicija je praktičnija jer većina DBMS-ova čuva ovu informaciju u svojim katalozima za svaku tabelu.
    - Postarati se da listovi svojim pozicioniranjem ne proizvode operaciju unakrsnog proizvoda (na primer, ukoliko dve najrestriktivnije SELECT operacije nemaju direktan uslov spoja između sobom, može biti poželjno da se listovi stabla preurede tako da se izbegne unakrsni proizvod)
4. Koristeći pravilo 12, zameniti unakrsni proizvod i selekciju SELECT u stablu operacijom spoja JOIN, ukoliko selekcija predstavlja uslov spoja.
5. Koristeći pravila 3, 4, 7 i 11 koja se odnose na kaskadne projekcije i zamenu mesta projekcije sa drugim operacijama, razbiti operacije projekcije na više operacija i pomerati ih niz stablo što je više moguće i po potrebi kreirati nove operacije projekcije. Samo oni atributi neophodni za rezultat upita kao i oni koji su neophodni za izvršenje operacija na višim nivoima stabla treba ostaviti nakon primene svake operacije projekcije.
6. Identifikovati podstabla koja predstavljaju grupe operacija koje se mogu odraditi zajedno primenom samo jednog algoritma. Na primer, dosta često se algoritmi za implementaciju selekcije koriste za istovremenu projekciju, tj dok tražimo odgovarajuće zapise (record) možemo iz njih pročitati samo one informacije koje se dobijaju nakon primene operacije projekcije.

Intuicija ovog algoritma je da operacije selekcije primenjujemo što je ranije moguće i na taj način smanjimo broj primeraka sa kojima radimo, da operacije projekcije primenimo što je moguće ranije da bi na taj način radili sa što je moguće manje atributa (na ovaj način dobijamo manju memorijsku složenost i skraćujemo vreme izvršenja algoritma). Najselektivnije operacije spoja (JOIN) primenjujemo najranije moguće da bi na taj način radili sa što je moguće manjim skupom podataka. Ovo postižemo preuređivanjem listova izvršnog stabla pri čemu izbegavamo pojavu unakrsnog proizvoda.

2.7 Procena cene prilikom optimizacije izvršenja upita

Optimizatori se na zasnivaju samo na heurističkim pravilima (poput onih predloženih u odeljku iznad). Oni takođe razmatraju i cene izvršenja upita primenom različitih strategija izvršenja i algoritama, i na kraju se opredeljuju za onu sa najnižom cenom. Da bi smo to uradili potrebne su nam određene mere na osnovu kojih ćemo porediti cenu izvršenja upita primenom određene strategije. Osim toga, optimizator mora ograničiti broj razmatranih strategija, u suprotnom previše vremena će potrošiti na procenu cene brojnih strategija izvršenja umesto na samo izvršenje upita (izuzetak od ovoga su prekompajlirani upiti za koje optimizator ima više vremena za odabir najefikasnije strategije koja se čuva i kasnije primenjuje u izvršenju).

Aktivnosti koje ulaze u razmatranje cene izvršenja:

1. Pristup sekundarnoj memoriji – ovo je cena razmene blokova podataka između diska i memorije (poznatija kao I/O cena)
2. Čuvanje među rezultata na disk – cena smeštanja među rezultata generisanih nekom operacijom na disk (najčešće zbog ograničenosti memorije)
3. Cena izvršenja – ovo je cena izvršenja operacija nad podacima koji se nalaze u glavnoj memoriji prilikom izvršenja (npr. sortiranje podataka, spajanje skupova podataka ili merge, obrada primeraka ponaosob). Poznata i kao CPU cena.
4. Cena memorije – broj memorijskih blokova koji se koriste prilikom izvršenja upita
5. Komunikaciona cena – cena transporta podataka od baze podataka do mesta odakle su traženi podaci korišćenjem mreže ili drugih vidova komunikacije

Uglavnom je previše teško uzeti u obzir sve faktore cene izvršenja, pa se određene baze podataka opredeljuju za najkritičniji faktor. Kod velikih baza podataka, najčešće se minimizuju cene pristupa sekundarnim memorijama, dok se cena izvršenja zanemaruje. U tom pogledu, strategije se porede po broju blokova koje treba razmeniti između diska i glavne memorije. Kod malih baza podataka koje sve svoje podatke mogu držati u glavnoj memoriji (in memory baze podataka), za cenu se najčešće uzima cena izvršenja. Kod distribuiranih baza za glavnu cenu se uzima cena komunikacije.

2.8 Korišćenje sistemskog kataloga u proceni cene

Da bi procenili cenu izvršenja neke strategije, potrebne su nam neke dodatne informacije. Ove dodatne informacije čuvaju se u posebnoj tabeli koja se naziva sistemski katalog i kojoj optimizator pristupa. Prva stvar koju moramo znati je veličina svakog fajla. Za svaki fajl kod koga su svi zapisi (record) istog tipa (r), moramo znati broj zapisa (r), prosečnu veličinu zapisa (R), broj fajl blokova koje oni zauzimaju (b) i faktor blokiranja (bfr). Pored toga, moramo znati primarnu organizaciju svakog fajla. Primarna organizacija može biti neuređen fajl, uređen po određenom atributu sa ili bez primarnog ili klaster indeksa, heširan na osnovu nekog atributa (statički ili dinamički). Informacije se čuvaju za sve primarne, sekundarne, klaster indekse kao i informacije o indeksiranim atributima. Čuva se i broj nivoa svakog višenivovskog indeksa (x) kao i broj blokova u prvom nivou.

Drugi važan parametar je broj diskretnih vrednosti nekog atributa (d) i selektivnost atributa (sl) koja predstavlja procenat broja primeraka koji zadovoljavaju uslov jednakosti za taj atribut. To omogućava procenu kardinalnosti skupa primeraka koji zadovoljavaju uslov jednakosti po određenom atributu (ovo je prosečan broj primeraka koji zadovoljavaju uslov i računa se kao s = r * sl). Na primer, za atribut

upotrebljen kao ključ tabele d = r, sl = 1/r i s = 1. Za ne ključni atribut koji ima d diskretnih vrednosti, možemo proceniti sl = 1/d i s = r/d.

Informacije o broju nivoa u indeksima može se lako održavati, zato što se ne menja često. Sa druge strane, neke druge informacije se menjaju svaki put kada upisujemo ili brišemo zapise iz tabele. Iz tog razloga se ovakve informacije ažuriraju periodično, najčešće u periodu kada nije veliko opterećenje nad sistemom baze podataka.

2.9 Histogram

Histogram predstavlja posebnu strukturu podataka koju održava DBMS kako bi aproksimirao raspodelu vrednosti nekog atributa u tabeli. Histogram se dobija tako što se skup vrednosti atributa podeli odgovarajuće intervale („kante“ - bucket) i onda izvrši sumiranje svih zapisa koji pripadaju svakom intervalu zasebno. Postoje dve vrste histograma koje se koriste: hisogrami iste širine i histogrami iste dubine. Kod histograma iste širine, skup vrednosti se deli na intervale jednake dužine. Kod histograma iste dubine, skup vrednosti se deli na intervale tako da je broj primeraka u svakom intervalu podjednak (histogrami iste dubine češće daju bolje rezultate u odnosu na histograme iste širine). Optimizator najčešće koristi hostograme prilikom procene selektivnosti nekog uslova tako što pogleda broj primeraka u intervalu u koji spada selektivni uslov. Korišćenjem histograma dobija se mnogo bolja procena broja primeraka nego bez primene histograma.

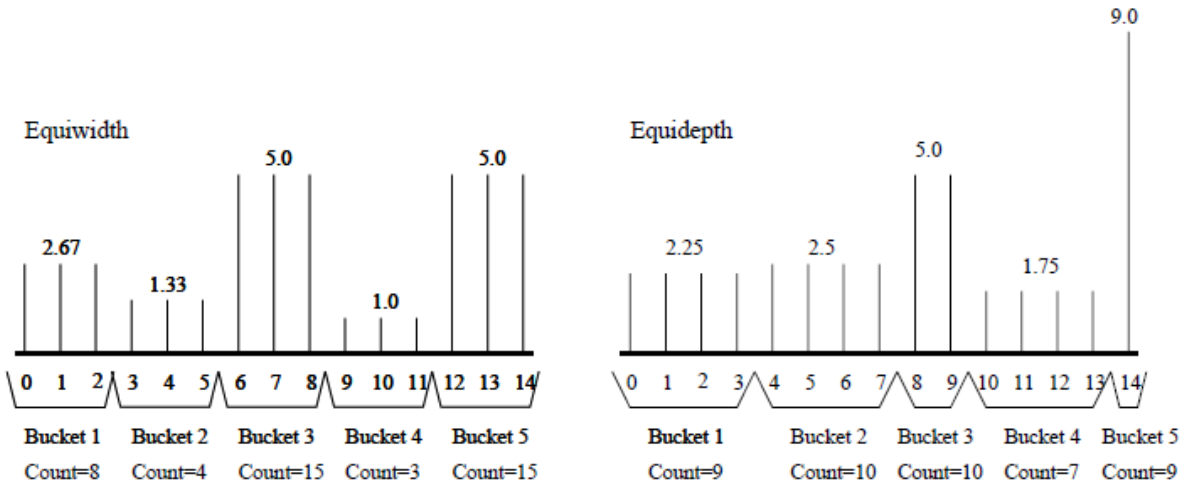

Figure 6. Histogram jednake širine (sa leve strane) i histogram jednake dubine (sa desne strane)

2.10 Spojevi više relacija i uređivanje spojeva (JOIN)

Algebarske transformacije navedene ranije uključuju komutativnost i asocijativnost operacije spoja. Uzimajući to u obzir, mogu se kreirati mnogo ekvivalentnih stabla za izvršenje istog upita. Upit koji uključuje N relacija zahteva N-1 spojeva i stoga može imati veliki broj različitih uređenja spojeva. Procena cene izvršenja za svako moguće stablo izvršenja nekog upita zahteva mnogo vremena. Zbog toga se primenjuju tehnike odbacivanja određenih stabala. Najčešće se od strane optimizatora razmatraju samo leva duboka stabla (ili alternativno desna duboka stabla). Levo duboko stablo je

binarno stablo kod kojeg svaki čvor koji nije list kao desnog potomka ima neku konkretnu tabelu (list stabla). Na kraju se optimizator opredeljuje za neko konkretno stablo levo duboko stablo sa najmanjom cenom izvršenja.

Kod levih dubokih stabala, desni potomak se uvek koristi kao unutrašnja relacija pri spoju sa ugnježdenim petljama. Jedna od prednosti levih (desnih) dubokih stabala je to što omogućavaju primenu pipline-inga (izlaz jedne operacije se odmah prosleđuje narednoj u nizu).

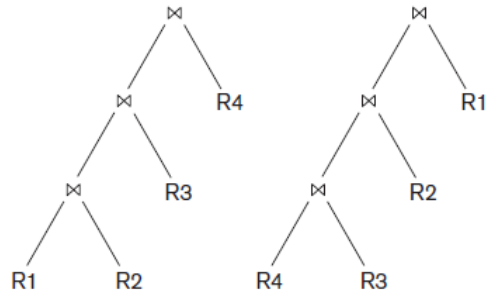

Figure 7. Primer dva leva duboka stabla za isti upit

3 Obrada upita kod Oracle baze podataka

Obrada upita kod Oracle baze obuhvata prasiranje, optimizaciju, generisanje plana izvršenja, generisanje izvora redova i izvršenje same naredbe. U zavisnosti od same naredbe, Oracle baza podataka može da preskoči neke od ovih faza.

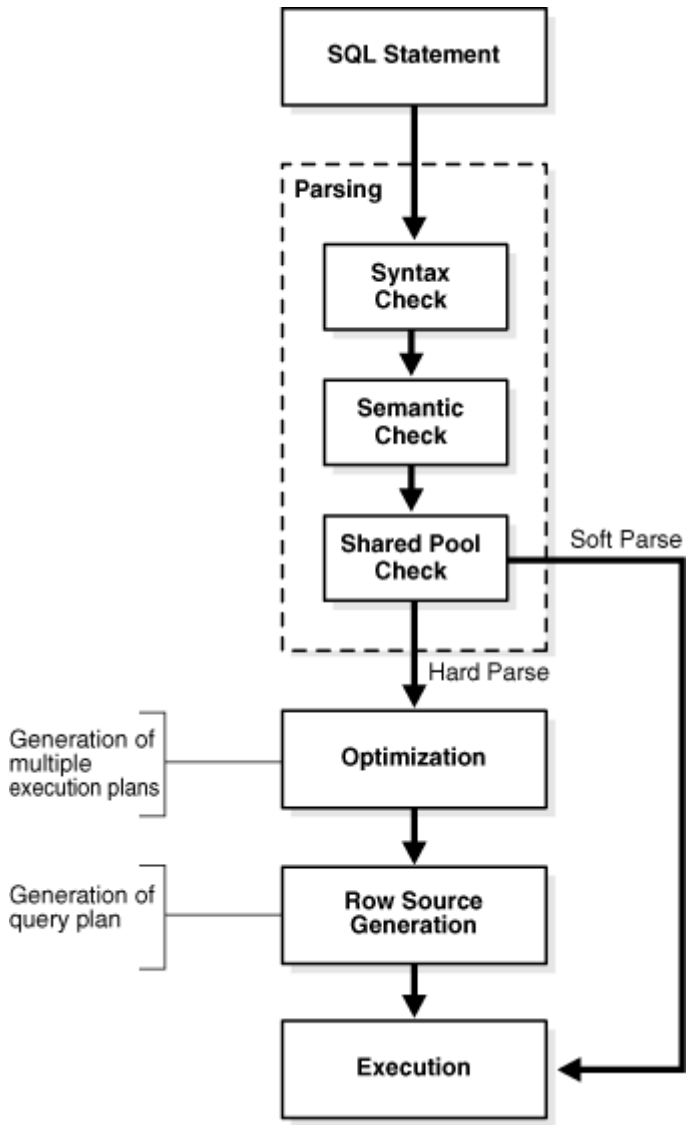

Figure 8. Obrada upita kod Oracle baze podataka

3.1 Parsiranje SQL naredbe

Prva faza pri obradi upita je parsiranje SQL naredbe. Parsiranje predstavlja izdvajanje delova SQL naredbe u posebnu strukturu podataka koje kasnije faze mogu koristiti u svojoj obradi. Baza podataka

obavlja parsiranje naredbe tek onda kada joj je naloženo od strane aplikacije i stoga jedino aplikacija može da smanji broj parsiranja.

Kada aplikacija zatraži izvršenje neke SQL naredbe, ona šalje poziv za parsiranje bazi podataka i na taj način pripremi naredbu za izvršenje. Poziv za parsiranjem kreira kursor (cursor) koji predstavlja način pristupanja privatnoj SQL oblasti sesije koja sadrži parsiranu naredbu i druge podatke koji sluče u obradi. Prilikom poziva za parsiranjem naredbe, baza podataka obavlja sledeće provere:

Sintaksna provera – Oracle baza proverava da li je naredba sintaksno ispravno napisana. Na primer naredba „SELECT * FORM employees“ neće proći ovu proveru jer je ključa reč FROM zamenjena sa FORM

- Semantička provera – Oracle baza proverava da li naredba ima semantičko značenje, tj. proverava da li postoje objekti i kolone navedeni u naredbi. Na primer naredba „SELECT * FROM nonexistent_table“ neće proći ovu proveru jer nonexistent_table ne postoji u kontekstu employee scheme podataka.
- Provera deljenog „pool“-a – Prilikom parsiranja, Oracle koristi hashing algoritam kako bi utvrdio da li može da preskoči neke zahtevne korake za procesiranje.

3.2 Provera deljenog „pool“-a (Shared pool check)

Prilikom obrade upita, korišćenjem hashing algoritma, Oracle generiše hash vrednost za svaku SQL naredbu. Dobijena hash vrednost je SQL_ID koji se može naći u tabeli V$SQL.SQL_ID. Sve Oracle instance generišu istu hash vrednost za istu naredbu.

Prilikom svakog novog upita, baza proverava da li u deljenoj SQL oblasti postoji parsirana naredba sa istom hash vrednošću. Hash vrednost se koristi za dobijanje sledećih podataka:

- Memorijske adrese naredbe – Oracle koristi SQL_ID da pretraži lookup tabelu i na taj način možda pribavlja adresu naredbe (možda jer se više naredbi mogu preslikavati u istu hash vrednost tabele).
- Izvršni plan naredbe – SQL naredba može ima više planova izvršenja. Svaki plan izvršenja ima drugu hash vrednost. Ako jedan SQL_ID ima više hash vrednosti planova izvršenja, onda baza zna da data naredba ima više planova izvršenja.

U zavisnosti od rezultata dobijenih proverom lookup tabela na osnovu hash vrednosti, moguće su dve vrste parsiranja:

1. Teško parsiranje (hard parse) – Ukoliko Oracle ne može da iskoristi neki od već postojećeg koda (promašaj keša), onda mora da generiše novi izvršni kod aplikacije. Prilikom teškog parsiranja baza često pristupa kešu. Da bi baza pristupila kešu, ona koristi „latch“ mehanizam serializaicje nad zahtevanim objektima kako se njihova definicija ne bi promenila. Primena latch-eva povećava vreme izvršenja naredbe i smanjuje konkurentnost. Baza podataka uvek primenjuje teško parsiranje kod DDL (data definition language) naredbi.
2. Meko parsiranje (soft parse) – Ako za datu naredbu već postoji reupotrebljivi deo koda u kešu, ona će ga Oracle iskoristiti (pogodak keša). Meka parsiranja mogu da variraju u odnosu na koliko posla zahtevaju. Ove stvari se mogu konfigurisati i na taj način poboljšati izvršenje. Meka parsiranja su poželjnija od teških jer baza podataka preskače korake optimizacije i generisanje

izvora redova podataka, pa može odmah preći na izvršenje (na Figure 8 se može videti put koji se preskače mekim parsiranjem).

Ako se utvrdi da naredba postoji u deljenom pool-u sa istom hash vrednošću, onda se proverava da li naredbe imaju isto sintaksno i semantičko značenje.

3.3 Komponente optimizatora kod Oracle baze podataka

Optimizator se sastoji iz tri komponente: transformatora upita, estimatora cene i generatora plana. Transformator upita proverava da li je isplativo promeniti formu upita tako da optimizator može da generiše efikasniji plan. Estimator procenjuje cenu svakog plana izvršenja na osnovu statistike. Generator plana generiše više alternativnih planova izvršenja, predi njihove cene i opredeljuje se za najefikasniji plan.

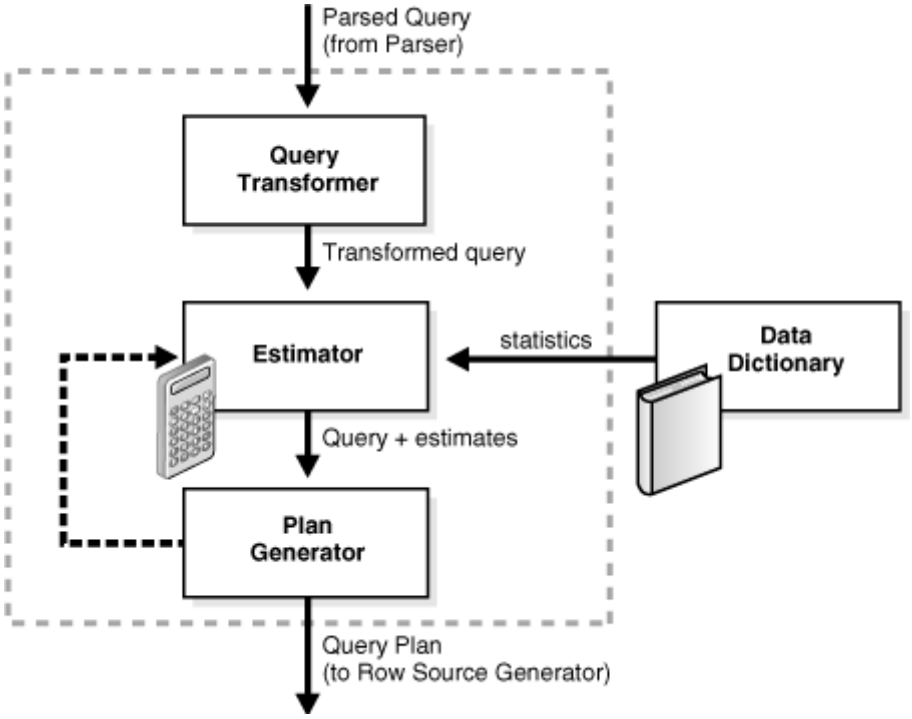

Figure 9. Komponente optimizatora upita

3.4 Transformator upita

Transformator upita proverava da li je isplativo prepisati SQL naredbu u njoj semantički ekvivalentnu koja ima manju cenu izvršenja. Kada postoje više ekvivalentnih alternativa istog upita, baza proverava cenu svake alternative i bira onu sa najmanjom cenom.

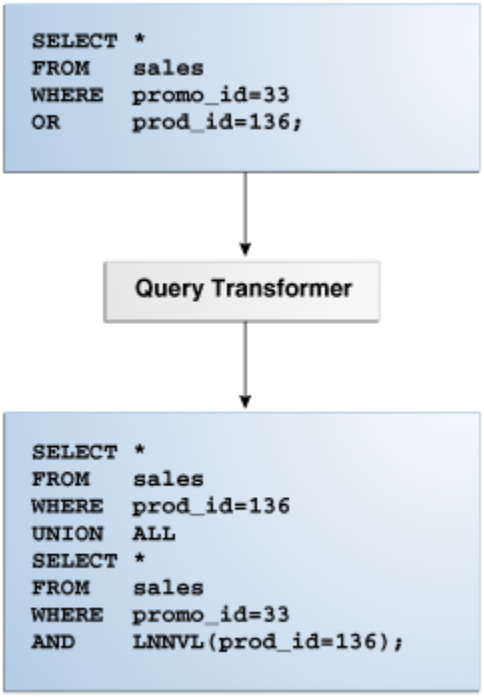

Figure 10. Primer transformacije koju obavlja transformator upita

3.5 Estimator

Esitmator je komponenta koja procenjuje cenu za određeni plan izvršenja. Estimator koristi sledeće tri mere za procenu cene:

- Selektivnost – broj redova koje upit izdvaja. Vrednosti su u opsegu od 0-1 (0 predstavlja da nijedan red nije vraćen upitom). Optimizator procenjuje selektivnost na osnovu dostupne statistike.
- Kardinalnost – broj redova koje svaka operacija u planu izvršenja vraća. Ovo je ključna procena za određivanje dobrog plana. Procenjuje se na osnovu statistike cele tabele i posebnih kolona.
- Cena – mera koja predstavlja cenu izvršenja upita u pogledu na disk I/O, upotreba CPU –a, upotreba memorije.

3.6 Generator plana

Generator plana istražuje različite planove za izvršenje svakog bloka ponaosob, isprobavajući različite metode pristupa podacima, različite metode spojeva kao i redoslede spojeva. Postoji mnogo ekvivalentnih planova izvršenja upita, a optimizator traži onaj sa najnižom cenom.

3.7 Adaptivna obrada upita

3.7.1 Adaptivna optimizacija plana izvršenja

Kod Oracle baze, adaptivna optimizacija upita omogućava optimizatoru da u toku izvršenja napravi neke ispravke u planu izvršenja. Ova opcija je korisna kada o postojeća statistika nije dovoljna za određivanje optimalnog plana izvršenja.

Usled nedostatka statistike, optimizator pravi lošu procenu o kardinalnosti rezultata operacije i zbog toga se opredeljuje za manje optimalni plan. Kada je aktivan adaptivni plan izvršenja, plan izvršenja može sadržati alternative u odnosu na kardinalnost pribavljenih primeraka. U toku izvršenja, kolektor statistike pribavlja informacije o izvršenju i baferuje neke primerke iz pribavljene tabele. Na osnovu izdvojenih primeraka, određuje se procenat primeraka koji zadovoljava odgovarajući uslov upita (npr. selekcija). Dobijena vrednost se poredi sa pretpostavljenom vrednošću prilikom procene cene. Ako vrednosti ne odstupaju jedna od druge drastično, nastavlja se sa prvobitnim planom. Međutim, ukoliko postoji znatna razlika u pretpostavljenim i dobijenim rezultatima, optimizator se može opredeliti za alternativi plan izvršenja (npr. može zameniti plan izvršenja koji koristi spoj ugnježdenih petlji, hash spojem jer se broj rezultata naglo poveća u odnosu na očekivanu vrednost).

Na ovaj način na osnovu dostupnih informacija u toku izvršenja, možemo se opredeliti za efikasniji plan. Kasnije se konačni plan izvršenja dodaje u keš i koristi u naknadnim izvršenjima upita (sve dok ne istekne njegova validnost).

3.7.2 Adaptivna statistika

Optimizator može koristiti adaptivnu statistiku kada su predikati u upitu previše kompleksni da bi se moga osloniti samo na statistiku o tabelama.

3.7.2.1 Dinamička statistika

Prilikom kompajliranja SQL naredbe, optimizator može odlučiti da li je dostupna statistika dovoljna za pronalaženje optimalnog plana. Ako nije, optimizator pokušava da poboljša statistiku. Jedan tip dinamičke statistike je dinamičko semplovanje (dynamic sample) koje se koristi u situacijama kada neka od tabela u upitu nema statistiku. Optimizator može koristiti prilikom skeniranja tabela, pristupa indeksu ili kreiranju spojeva. Dinamičku statistiku ne treba koristiti kao zamenu za običnu statistiku.

3.7.2.2 Automatska reoptimizacija

Kod automatske reoptimizacije, optimizator menja plan narednih izvršenja naredbe nakon inicijalnog izvršenja.

Adaptivna optimizacija planova izvršenja nije uvek primenljiva. Na primer, upit se izvršava sporo zbog lošeg rasporeda spojeva između tabela. Nakon prvog izvršenja, optimizator može uvideti da se informacije o izvršenju drastično razlikuju od procena. Iz tog razloga on pokušava da nađe bolji plan pri narednom izvršenju upita. Optimizator koristi informacije iz prvog izvršenja za pronalaženje optimalnog plana. Jedan upit se može optimizovati više puta, pri čemu se svakog puta dobijaju dodatne informacije.

4 Transformacije upita kod Oracle baze i primeri

4.1 Zamena OR (ILI) izraza

OR izraze u WHERE klauzuli optimizator može transformisati u izraze koji sadrže UNION ALL operator. OR izrazi se mogu zameniti jer se na taj način dobijaju efikasniji pristupi podacima ili metode spoja kojima se izbegavaju unakrsni proizvodi. Zamena se obavlja samo ako je cena transformisanog izraza manja od cene početnog izraza.

```
explain plan for
select *
from orders o, employees e
where (e.employee_id = 62 or o.order_id = 1)
and o.salesman_id = e.employee_id
```

Figure 11. Upit sa OR zamenom

Na Figure 11 možemo je primer upita za koji je optimizator upotrebio OR zamenu, a na Figure 12 plan njegovog izvršenja. Vidimo da je upit razdvojen na dva dela koja zasebno pronalaze rezultate OR uslova i spoja, a zatim se vrši unija dobijenih rezultata.

```
Plan hash value: 1965792772

-----------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                 |    14 |  8932 |     6   (0)| 00:00:01 |
|   1 |  VIEW                         | VW_ORE_93880E11 |    14 |  8932 |     6   (0)| 00:00:01 |
|   2 |   UNION-ALL                   |                 |       |       |            |          |
|   3 |    NESTED LOOPS               |                 |    13 |  1404 |     4   (0)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| EMPLOYEES       |     1 |    84 |     1   (0)| 00:00:01 |
|*  5 |      INDEX UNIQUE SCAN        | SYS_C007429     |     1 |       |     0   (0)| 00:00:01 |
|*  6 |     TABLE ACCESS FULL         | ORDERS          |    13 |   312 |     3   (0)| 00:00:01 |
|   7 |    NESTED LOOPS               |                 |     1 |   108 |     2   (0)| 00:00:01 |
|*  8 |     TABLE ACCESS BY INDEX ROWID| ORDERS          |     1 |    24 |     1   (0)| 00:00:01 |
|*  9 |      INDEX UNIQUE SCAN        | SYS_C007452     |     1 |       |     0   (0)| 00:00:01 |
|  10 |     TABLE ACCESS BY INDEX ROWID| EMPLOYEES       |     1 |    84 |     1   (0)| 00:00:01 |
|* 11 |      INDEX UNIQUE SCAN        | SYS_C007429     |     1 |       |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   5 - access("E"."EMPLOYEE_ID"=62)
   6 - filter("O"."SALESMAN_ID"=62)
   8 - filter("O"."SALESMAN_ID" IS NOT NULL)
   9 - access("O"."ORDER_ID"=1)
  11 - access("O"."SALESMAN_ID"="E"."EMPLOYEE_ID")
       filter(LNNVL("E"."EMPLOYEE_ID"=62))
```

Figure 12. Plan izvršenja upita sa OR zamenom

4.2 Spajanje pogleda (View merging)

Spajanje pogleda predstavlja spajanje blokova upita koji predstavljaju neki pogled u upit u koji sadrži taj pogled (tabela pogleda se menja upitom kojim se dobija pogled). Na taj način se omogućuju dodatne optimizacije poput dodatnih uređenja spojeva, metoda pristupa podacima kao i druge transformacije (npr. nakon spajanja pogleda neka od tabela u pogledu može omogućiti eliminaciju spojeva u ostalom delu upita).

Spajanje pogleda se može podeliti na dva slučajeva:

- Jednostavni slučaj - optimizator spaja jednostavne poglede koji se sastoje samo od selekcije, projekcije i spoja.
- Složeni slučaj – optimizator spaja poglede koji sadrže GROUP BY i DISTINCT klauzule. Optimizator može odložiti izvršenje GROUP BY i DISTINCT operacija nakon izvršenja spojeva. Odlaganje ovih operacija može poboljšati ili pogoršati performanse u zavisnosti od karakteristika podataka.

Na Figure 13 može se videti demonstrativni upit za spajanje pogleda, a njegov plan izvršenja na Figure 14. Figure 15 i Figure 16 pokazuju isti taj upit bez primene spajanja pogleda. Kao što možemo videti, cena izvršenja je znatno veća bez primene ove transformacije.

```
explain plan for
select cs.name, e.first_name, cs.status
from customer_status cs, employees e
where cs.salesman_id = e.employee_id
and e.first_name = 'Freya'
and cs.status = 'Canceled'
```

Figure 13. Primer upita za spajanje pogleda

```
Plan hash value: 529533621

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                |     1 |    45 |     4   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                          |                |     1 |    45 |     4   (0)| 00:00:01 |
|   2 |   NESTED LOOPS                         |                |     1 |    45 |     4   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                        |                |     1 |    26 |     3   (0)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID BATCHED| EMPLOYEES      |     1 |    11 |     2   (0)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN                  | EMPP_FULL_NAME |     1 |       |     1   (0)| 00:00:01 |
|*  6 |     TABLE ACCESS BY INDEX ROWID BATCHED| ORDERS         |     1 |    15 |     1   (0)| 00:00:01 |
|*  7 |      INDEX RANGE SCAN                  | ORDER_STATUS   |    16 |       |     0   (0)| 00:00:01 |
|*  8 |    INDEX UNIQUE SCAN                   | SYS_C007441    |     1 |       |     0   (0)| 00:00:01 |
|   9 |   TABLE ACCESS BY INDEX ROWID          | CUSTOMERS      |     1 |    19 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   5 - access("E"."FIRST_NAME"='Freya')
   6 - filter("O"."SALESMAN_ID"="E"."EMPLOYEE_ID" AND "O"."SALESMAN_ID" IS NOT NULL)
   7 - access("O"."STATUS"='Canceled')
   8 - access("O"."CUSTOMER_ID"="C"."CUSTOMER_ID")

Note
-----
   - this is an adaptive plan
```

Figure 14. Plan izvršenja upita sa spajanjem pogleda

```
explain plan for
select /*+ NO_QUERY_TRANSFORMATION */ cs.name, e.first_name, cs.status
from customer_status cs, employees e
where cs.salesman_id = e.employee_id
and e.first_name = 'Freya'
and cs.status = 'Canceled'
```

Figure 15. Upit upotrebljen za spajanje pogleda sa demonstacijom hint-a optimizatoru da ne vrši transformacije nad upitom

```
Plan hash value: 355738190

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                 |     1 |   165 |     7   (0)| 00:00:01 |
|*  1 |  HASH JOIN                             |                 |     1 |   165 |     7   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED  | EMPLOYEES       |     1 |    11 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                    | EMPP_FULL_NAME  |     1 |       |     1   (0)| 00:00:01 |
|   4 |   VIEW                                 | CUSTOMER_STATUS |    16 |  2464 |     5   (0)| 00:00:01 |
|*  5 |    HASH JOIN                           |                 |    16 |   544 |     5   (0)| 00:00:01 |
|   6 |     TABLE ACCESS BY INDEX ROWID BATCHED| ORDERS          |    16 |   240 |     2   (0)| 00:00:01 |
|*  7 |      INDEX RANGE SCAN                  | ORDER_STATUS    |    16 |       |     1   (0)| 00:00:01 |
|   8 |     TABLE ACCESS FULL                  | CUSTOMERS       |   319 |  6061 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("CS"."SALESMAN_ID"="E"."EMPLOYEE_ID")
   3 - access("E"."FIRST_NAME"='Freya')
   5 - access("O"."CUSTOMER_ID"="C"."CUSTOMER_ID")
   7 - access("O"."STATUS"='Canceled')

Note
-----
   - this is an adaptive plan
```

Figure 16. Plan izvršenja upita bez view merge-a

4.3 Izvršavanje prvo predikata (Predicate pushing)

Kod izvršavanja prvo predikata, optimizator pomera relevantne predikate iz spoljašnjeg bloka u unutrašnji blok pogleda. Kod pogleda koji nisu spojeni sa spoljnim upitom, ova tehnika može poboljšati izvršenje podplana pogleda i omogući upotrebu nekog indeksa ili upotrebu predikata za filtriranje rezultata.

Na Figure 17 kreiran je pogled koji vrši uniju nekih rezultata (ovaj pogled nema mnogo smisla već više služi demonstrativno). Na Figure 18 možemo videti da optimizator pomera selekciju u fazu čitanja vrednosti iz indeksa.

```
create view name_view as
select product_name as name
from products
union
select name
from customers

explain plan for
select *
from name_view
where name like 'A%'
```

Figure 17. Pogled i upit kojim se može demonstrirati izvršenje prvo predikata

```
Plan hash value: 4021016513

-------------------------------------------------------------------------------------------
| Id  | Operation           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |              |    15 |  1935 |     7  (43)| 00:00:01 |
|   1 |  VIEW               | NAME_VIEW    |    15 |  1935 |     7  (43)| 00:00:01 |
|   2 |   SORT UNIQUE       |              |    15 |   297 |     7  (43)| 00:00:01 |
|   3 |    UNION-ALL        |              |       |       |            |          |
|*  4 |     INDEX RANGE SCAN| PROD_NAME    |     8 |   192 |     2   (0)| 00:00:01 |
|*  5 |     INDEX RANGE SCAN| CUSOMER_NAME |     7 |   105 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("PRODUCT_NAME" LIKE 'A%')
       filter("PRODUCT_NAME" LIKE 'A%')
   5 - access("NAME" LIKE 'A%')
       filter("NAME" LIKE 'A%')
```

Figure 18. Plan izvršenja za upit koji demonstira izvršenje prvo predikata

4.4 Odgnježdavanje podupita

Odgnježdavanje upita je transformacija kojom optimizator transformiše upit koji sadrži ugnježdeni upit u nemu ekvivalentan upit zapisan pomoću spoja. Ova transformacija omogućava optimizatoru da razmotri nove načine pristupa podacima, uređenjima spojeva kao i same metode spojeva. Odgnježdavanje upita se može primeniti samo ako rezultat vraća isti broj redova kao i originalni upit.

Na Figure 19 imamo primer ugnježdenog upita. Kao što se može videti na Figure 20, ugnježdeni upit je zamenjen spojem sa tabelom customers.

```
explain plan for
select *
from orders
where customer_id in (select customer_id from customers where customers.credit_limit < 200)
```

Figure 19. Primer ugnježdenog upita

```
Plan hash value: 3042513348

-------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |    36 |  1116 |     6   (0)| 00:00:01 |
|*  1 |  HASH JOIN         |           |    36 |  1116 |     6   (0)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| CUSTOMERS |    16 |   112 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| ORDERS    |   105 |  2520 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("CUSTOMER_ID"="CUSTOMER_ID")
   2 - filter("CUSTOMERS"."CREDIT_LIMIT"<200)
```

Figure 20. Plan izvršenja ugnježedenog upita

4.5 Prepisivanje upita materijalizovanim pogledom

Materijalizovani pogled je upit koji baza podataka materijalizuje i smešta rezultat u tabelu. Kada optimizator pronađe upit koji se može asocirati sa nekim materijalizovanim pogledom, on izmenjuje upit tako da koristi pogled. Ova tehnika poboljšava izvršenje jer su rezultati većeg dela upita već izračunati i sačuvani u pogledu.

Na Figure 21 možemo videti naredbu koja kreira materijalizovani pogled koji prikazuje ukupnu vrednost prodate robe svakom kupcu ponaosob kao i broj porudžbina kupaca. Možemo videti da se upit koji je jako sličan upitu za kreiranje pogleda svodi na prosto čitanje iz pogleda - Figure 22. Kod Oracle baze prilikom kreiranja materijalizovanih pogleda potrebno je navesti da se pogled može koristiti u prepisivanju upita.

```sql
create materialized view customer_value
enable query rewrite
as
select c.customer_id, c.name,
count(*) as order_count, sum(oi.quantity*oi.unit_price) as total_value
from customers c
inner join orders o on o.customer_id = c.customer_id
inner join order_items oi on o.order_id = oi.order_id
group by c.customer_id, c.name
order by total_value desc;

explain plan for
select c.name, sum(oi.quantity*oi.unit_price) as total_value
from customers c
inner join orders o on o.customer_id = c.customer_id
inner join order_items oi on o.order_id = oi.order_id
group by c.customer_id, c.name;
```

Figure 21. Naredba za kreiranje materijalizovanog pogleda i upit koji se prevodi u naredbu čitanja iz pogleda

```
Plan hash value: 3150927990

-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |    47 |  1034 |     3   (0)| 00:00:01 |
|   1 |  MAT_VIEW REWRITE ACCESS FULL| CUSTOMER_VALUE |    47 |  1034 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

Figure 22. Plan izvršenja gore navedenog upita

5 Zaključak

Kreiranje indeksa nema mnogo smisla ukoliko optimizator ne kreira plan koji će ih upotrebiti. Zbog toga obrada i optimizacija upita igraju važnu ulogu za efikasno izvršenje upita u svim sistemima za upravljanje bazama podataka.

Videli smo da su osnovni koraci u obradi upita sintaksna analiza (provera sintaksne ispravnosti upita), semantička analiza upita (provera da li upit ima semantičko značenje) i prevođenje upita u reprezentaciju operatora relacione algebre. Postoje dve takve reprezentacije: pomoću stabla i pomoću grafa. Optimizacija upita je višestruko lakša nad reprezentacijom uz pomoću stabla pa se iz tog razloga samo ona koristi. Videli smo 12 osnovnih pravila relacione algebre na kojima se zasniva optimizacija upita. Dali smo pregled osnovnog algoritma za optimizaciju upita koji čija je suština da selekcije i projekcije obavljamo što je moguće bliže samim listovima stabala, preuredimo spojeve tako da damo prednost onim koji su restriktivniji i generišu manje rezultata, grupišemo operacije tako da više njih obavljamo u jednom prolazu nad podacima. Videli smo neke osnovne načine za procenu cene na osnovu kardinalnosti i selektivnosti, kao i neke osnovne informacije o katalogu baze podataka. Spomenuli smo osnovne statistike koje sistem baza podataka održava i dali smo pregled rada histograma.

Oracle je jedna od najpoznatijih relacionih baza podataka. Pored osnovnih koraka u obradi upita, nakon Oracle uključuje još jedan korak – keširanje. Nakon semantičke analize, vrši se provera da li postoji plan izvršenja u kešu. Ukoliko to nije slučaj nastavlja se sa generisanjem operatorskog stabla i njegovom optimizacijom (ovaj slučaj je poznat kao hard parse). Ako je plan već prisutan u kešu, on se iz njega uzima i prosleđuje se na izvršenje. Optimizacija upita kod Oracle-a prolazi kroz korake transformacije upita, procene cene plana i generisanje alternativnih planova. Pored ovih koraka Oracle uključuje i neke specifične korake poput adaptivne optimizacije upita – plan izvršenja upita se može promeniti i u toku izvršenja na osnovu nekih novih podataka koji su prikupljeni.

Pored svega ovoga, Oracle implementira i brojne dodatne funkcionalnosti koje prevazilaze opsege ovog rada od kojih su najpoznatije one za implementaciju Data Warehouse rešenja. Zbog svega ovoga Oracle je stekao značajnu poziciju na svetskom tržištu sistema za upravljanje bazama podataka.

6 Literatura

[1] Ramakrishnan, "Database Management Systems 3rd Edition," [Online].

[2] N. Elmasri, "Fundamentals of Database Systems 6th Edition," [Online].

[3] "Oracle documentation - SQL Processing," [Online]. Available: https://docs.oracle.com/database/121/TGSQL/tgsql_sqlproc.htm#TGSQL175.

[4] "Oracle documentation - Query Transformations," [Online]. Available: https://docs.oracle.com/database/121/TGSQL/tgsql_transform.htm#TGSQL94896.

[5] "Oracle documentation - Query Optimizer Concepts," [Online]. Available: https://docs.oracle.com/database/121/TGSQL/tgsql_optcncpt.htm#TGSQL192.