Optimizacija upita kod Oracle baze podataka

Student: Uroš Pešić 1465
Profesor: Aleksandar Stanimirović

Optimizacija upita

- Optimizacija upita predstavlja proces određivanja najbolje dostupne strategije za izvršenje SQL upita. Predstavlja jedan od ključnih zadataka DBMS-a, jer bitno utiče na performanse izvršenja upita, a samim tim i na performanse softverskih sistema i celokupno korisničko iskustvo.

- Za optimizaciju upita je zadužena specijalna komponenta DBMS-a koja se zove **optimizator**.

Metode optimizacije upita

Postoje dva različita pristupa optimizacije upita:

- **Heurističke metode** – Za optimizaciju upita se koriste unapred definisana pravila koja se primenjuju za transformaciju upita. Ovde metode generalno dovode do optimalnijeg rešenja, mada to ne mora uvek biti slučaj.

- **Metode zasnovane na ceni (cost-based) –** za svaku transformaciju i potencijalnu strategiju se računa cena izvršenja. Kao finalni plan izvršenja se bira onaj sa najmanjom cenom.

Heurističke metode

- SQL upit se interno transformiše u *izraz proširene relacione algebre* koji je moguće predstaviti *sturkturom stabla*. **Heurističkim metodama se inicijalno stablo upita, primenom definisanih plavila transformiše u semantički ekvivalentna stabla, sve dok se ne dobije stablo upita koje je optimalno za izvršenje.**

- Generalna ideja heurističkih metoda je transformacija upita promenom redosleda relacionih operacija, tako da se operacije selekcije i projekcije koje smanjuju kardinalnost među-relacija I broj njihovih atributa, izvrše što je moguće ranije. Na taj način se najčešće obezbeđuje optimalnije izvršenje nekih skupiljih operacija, kao što su različite vrste spojeva, Dekartov proizvod, itd.

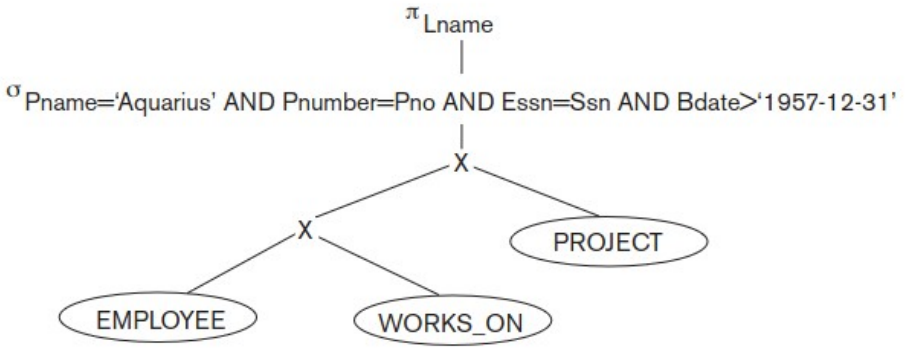
Stablo upita

- Na slici je prikazan primer stabla za izraz relacione algebre koji odgovara upitu

SELECT E.Lname

FROM EMPLOYEE E, WORKS_ON W, PROJECT P

WHERE P.Pname=“Aquariu



Pravila za transformaciju upita

Neka pravila koje je moguće primeniti, koja ne menjaju semantiku upita su sledeća:

- **Nadovezivanje selekcija**

  $\sigma_{c1\ \text{AND}\ c2\ \text{AND}\ \ldots\ \text{AND}\ cn}(R) = \sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R)))$

- **Komutativnost selekcije**

  $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$

- **Nadovezivenje projekcija**

  $\pi_{list1}(\pi_{list2}\ldots(\pi_{listn}(R))) = \pi list1(R)$

- **Distributivnost selekcije prema ⋈ i x**

  $\sigma_{c1}(R \bowtie S) = \sigma_{c1}(R) \bowtie S$

- **Distributivnost projekcije prema ⋈ i x**

  $\pi_{c1}(R \bowtie S) = \pi_{A1, A2, \ldots, An}(R) \bowtie \pi_{B1, B2, \ldots, Bm}(S)$

- **Asocijativnost operacija ⋈, x, U, ∩**

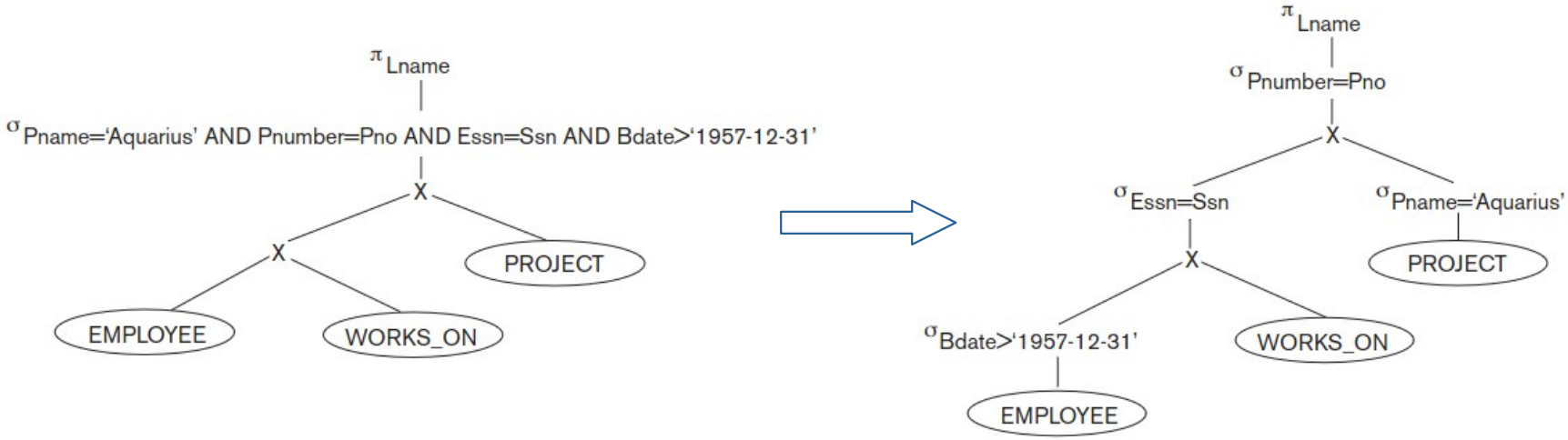
  (R *op* S) *op* P = R *op* (S *op* P), gde je *op* neka od operacija iz navedenog skupa

- **Distributivnost selekcije u odnosu na skupovne operacije**

  σ(R *op* S) = σ(R) *op* σ(S), gde *op* označava bilo koju skupovnu operaciju

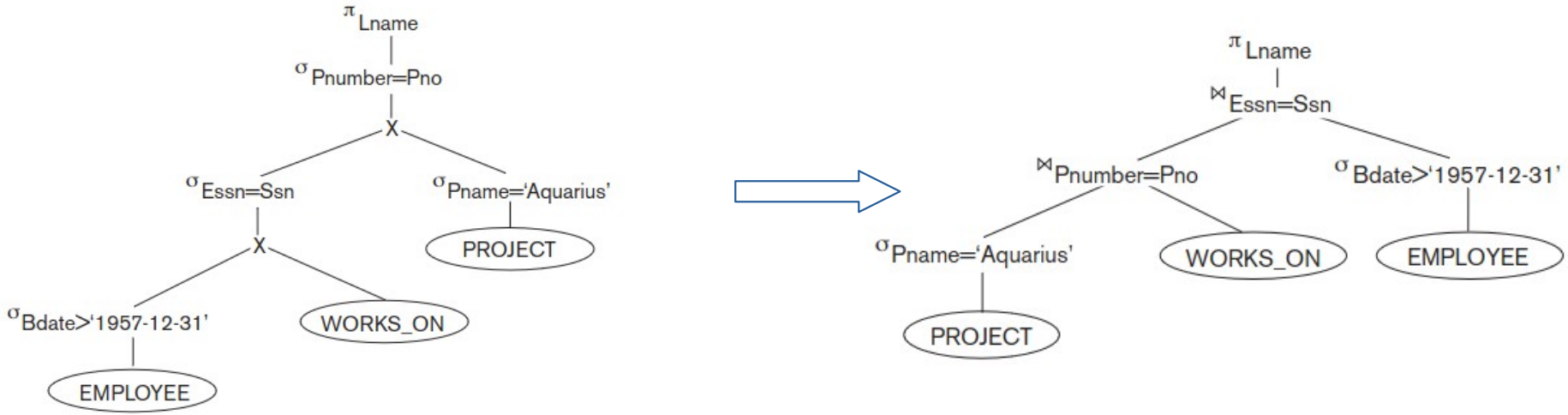
- **Konverzija sekvence (σ, x) u prirodni spoj**

Heuristička optimizacija inicijalnog stabla



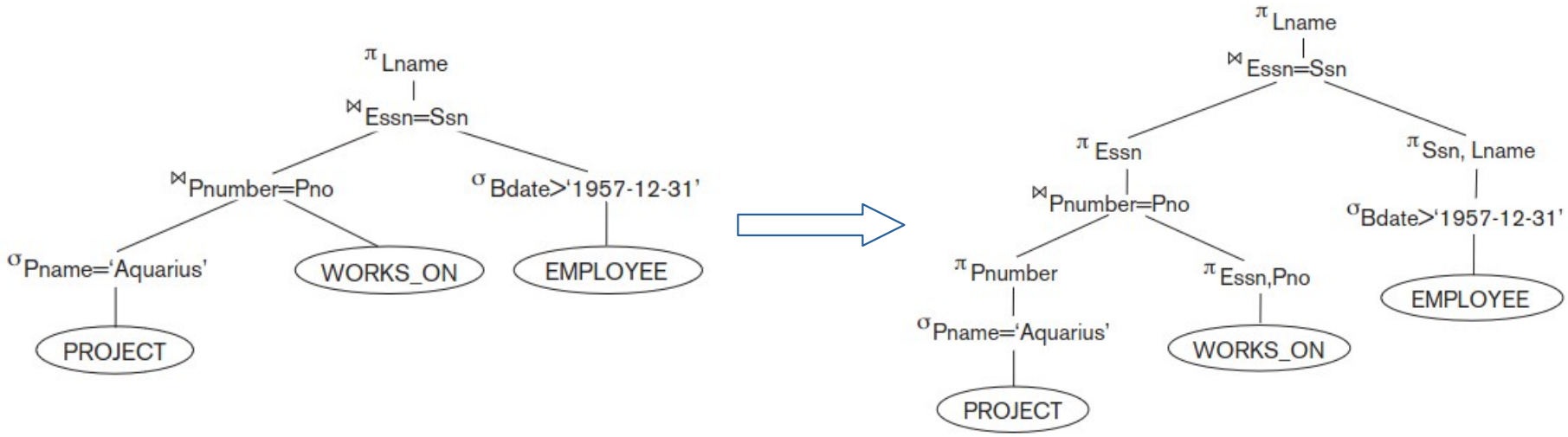
- Prvi korak algoritma je razbijanje konjunkcija u uslovima selekcije na individualne selekcije, a zatim pomeranje tih selekcija što bliže listovima stabla (početnim relacijama).

Heuristička optimizacija inicijalnog stabla



- Zatim je potrebno eventualno promeniti redosled listova stabla, tako da se najrestriktivnije selekcije prve izvrše. Pritom je potrebno voditi računa da se promenom redosleda ne javi potreba za operacijom Dekartovog spoja – koja je najskuplja za izvršenje

Heuristička optimizacija inicijalnog stabla



- Primenom poslednjeg navedenog pravila, svaku operaciju x koja je praćena uslovom selekcije (koji je u stvari uslov spoja), pretvoriti u operaciju ⋈. Na kraju, sve projekcije pomeriti što je moguće bliže listovima, u zavisnosti od potrebnih atributa za izvršenje operacija.

Cost–based metode

- Kod cost–based metoda se problem opzimizacije upita posmatra kao tradicionalni problem minimizacije ciljne funkcije (cost funkcije). Cost funkcija predstavlja kvantitativnu meru performansi izvršenja nekog plana i obrnuto je srazmerna potrebnom vremenu i resursima za izvršenje upita. Na cenu može da utiču neki od sledećih faktora:
  - **Cena pristupa sekundarnom skladištu**
  - **Cena skladištenja privremenih rezultata**
  - **Cena izvršenja (CPU cena)**
  - **Cena zauzeća memorije**
  - **Cena komunikacije**

Primer cost funkcije za JOIN operaciju

- Za JOIN operaciju koja je implementirana brute-force algoritmom (dve ugnježdene petlje) cena izvršenja u pogledu broja blokova koje je potrebno preneti sa diska u glavnu memoriju je:

  $$\mathbf{C1 = b_r + [ceiling(b_r / n_b - 2) * b_s] + (js * r_r * r_s) / bfr_{res},}$$

  Gde $\mathbf{b_r}$ označava broj blokova fajla relacije **r** na disku, $\mathbf{n_b}$ broj blokova koji staju u glavnu memoriju, **js** je selektivnost spoja, a $\mathbf{bfr_{res}}$ broj slogova po bloku za rezultat.

DBMS katalog

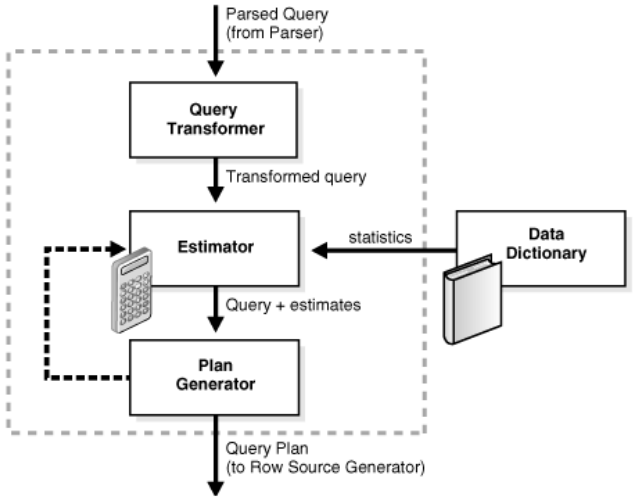
- Kao što se vidi i na prethodnom primeru, optimizator mora da koristi različite informacije o stanju sistema, kako bi procena cene izvršenja plana bila što tačnija. Ove informacije se čuvaju u **katalogu DBMS-a**.
- Neke od informacija koje se čuvaju za svaku relaciju su:
  - **Broj slogova relacije**
  - **Prosečna veličina sloga**
  - **Broj blokova na disku**
  - **Broj slogova po bloku**
  - **Indkesi i tipovi indeksa**
  - **Selektivnosti za različite uslove**
  - **Kardinalnost selekcije za različite uslove**

Histogrami

- DBMS takođe kreira i čuva posebne strukture sa informacijama o statističkoj raspodeli atributa u relacijama, koje se nazivaju **histogrami**. Zahvaljujući histogramima je moguće bolje aproksimirati selektivnost za različite atribute (bez njih bismo morali da pretpostavimo da svi atributi imaju uniformnu raspodelu – što najčešće nije slučaj). Kako histogrami obezbeđuju tačniju procenu cene za planove izvršenja – predstavljaju bitan faktor za postizanje dobrih performansi sistema.

ORACLE – Optimizacija upita

- Optimizator upita Oracle baze podatka je cost-based optimizaor (u ranijim verzijama je imao I rule-based komponentu od koje se danas odustaje). Glavni faktori koji utiču na cenu izvršenja plana su cena pristupa disku, CPU cena i cena komunikacije usled paralelizacije izvršenja upita. Optimizator sadrži 3 komponente:
  - **Transformer upita**
  - **Estimator**
  - **Plan generator**



Transformer upita

- **Transformer upita** – transformiše inicijalni parsirani SQL upit u semantički ekvivalenti upit koji je pogodniji za izvršenje. Neke od tehnika koje transforormer upita koristi su:
  - **OR ekspanzija**
  - **View Merging**
  - **Query unnesting**
  - **Faktorizacija operacija**

OR ekspanzija

- OR ekspanzija je tehnika razbijanja disjunkcije predikata na više upita sa prostim predikatima koji su povezani UNION ALL operacijom.

```
SELECT * FROM employees e,
departments d
WHERE (e.email='SSTILES' OR
d.department_name='Sales')
AND e.department_id =
d.department_id;
```

```
SELECT * FROM employees e, departments d
WHERE e.email='SSTILES'
AND e.department_id = d.department_id;
UNION ALL
SELECT * FROM employees e, departments d
WHERE  d.department_name='Sales'
AND e.department_id = d.department_id;
```

OR Ekspanzija – Primer plana izvršenja

```
3  ------------------------------------------------------------------------------------------------------
4  | Id  | Operation                     | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
5  ------------------------------------------------------------------------------------------------------
6  |   0 | SELECT STATEMENT              |                   |    11 |  2079 |     6   (0)| 00:00:01 |
7  |   1 |  VIEW                         | VW_ORE_19FF4E3E   |    11 |  2079 |     6   (0)| 00:00:01 |
8  |   2 |   UNION-ALL                   |                   |       |       |            |          |
9  |   3 |    NESTED LOOPS               |                   |     1 |    90 |     2   (0)| 00:00:01 |
10 |   4 |     TABLE ACCESS BY INDEX ROWID| EMPLOYEES         |     1 |    69 |     1   (0)| 00:00:01 |
11 |*  5 |      INDEX UNIQUE SCAN        | EMP_EMAIL_UK      |     1 |       |     0   (0)| 00:00:01 |
12 |   6 |     TABLE ACCESS BY INDEX ROWID| DEPARTMENTS       |     1 |    21 |     1   (0)| 00:00:01 |
13 |*  7 |      INDEX UNIQUE SCAN        | DEPT_ID_PK        |     1 |       |     0   (0)| 00:00:01 |
14 |   8 |    NESTED LOOPS               |                   |    10 |   900 |     4   (0)| 00:00:01 |
15 |   9 |     NESTED LOOPS              |                   |    10 |   900 |     4   (0)| 00:00:01 |
16 |* 10 |      TABLE ACCESS FULL        | DEPARTMENTS       |     1 |    21 |     3   (0)| 00:00:01 |
17 |* 11 |      INDEX RANGE SCAN         | EMP_DEPARTMENT_IX |    10 |       |     0   (0)| 00:00:01 |
18 |* 12 |     TABLE ACCESS BY INDEX ROWID| EMPLOYEES         |    10 |   690 |     1   (0)| 00:00:01 |
19 ------------------------------------------------------------------------------------------------------
20
21 Predicate Information (identified by operation id):
22 ---------------------------------------------------
23
24    5 - access("E"."EMAIL"='SSTILES')
25    7 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
26   10 - filter("D"."DEPARTMENT_NAME"='Sales')
27   11 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
28   12 - filter(LNNVL("E"."EMAIL"='SSTILES'))
```

View merging

- Optimizacija upita se vrši na nivou SQL bloka. Generisanje planova se vrši prvo za najugnježdeniji blok. Nekad je pogodnije ovakve upite transformisati u ekvivalentni upit sa jednim blokom, ukoliko je to moguće.

- View merging je upravo jedna ovakva transformacija, koja se koristi kada se unutrašnji SQL blok nalazi u okviru naredbe FROM spoljašnjeg upita – tj. Kada unutrašnju blok predstavlja pogled, koji može biti eksplicitno ili implicitno definisan.

View merging - Primer

```
SELECT e.first_name, e.last_name, dept_locs_v.street_address,
dept_locs_v.postal_code
FROM employees e,
    ( SELECT d.department_id, d.department_name, l.street_address,
    l.postal_code
    FROM departments d, locations l
    WHERE d.location_id = l.location_id ) dept_locs_v
WHERE dept_locs_v.department_id = e.department_id
AND e.last_name = 'Smith';
```

```
SELECT e.first_name, e.last_name,
l.street_address, l.postal_code
FROM   employees e, departments d, locations l
WHERE  d.location_id = l.location_id
AND    d.department_id = e.department_id
AND    e.last_name = 'Smith';
```

- Spajanjem pogleda smo pružili više mogućnosti optimizatoru za implementiranje spojeva tabela. U prvom slučaju postoje samo 4 moguće permutacije, dok u drugom slučaju imamo 6 mogućih načina za redosled spajanja tabela. Samim tim su optimizator ima više mogućnosti za nalaženje najoptimalnijeg rešenja.

View merging – Primer plana izvršenja

```
Plan hash value: 257295346

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |              |     1 |    56 |     4   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                          |              |     1 |    56 |     4   (0)| 00:00:01 |
|   2 |   NESTED LOOPS                         |              |     1 |    56 |     4   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                        |              |     1 |    25 |     3   (0)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID BATCHED| EMPLOYEES    |     1 |    18 |     2   (0)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN                  | EMP_NAME_IX  |     1 |       |     1   (0)| 00:00:01 |
|   6 |     TABLE ACCESS BY INDEX ROWID        | DEPARTMENTS  |     1 |     7 |     1   (0)| 00:00:01 |
|*  7 |      INDEX UNIQUE SCAN                 | DEPT_ID_PK   |     1 |       |     0   (0)| 00:00:01 |
|*  8 |    INDEX UNIQUE SCAN                   | LOC_ID_PK    |     1 |       |     0   (0)| 00:00:01 |
|   9 |   TABLE ACCESS BY INDEX ROWID          | LOCATIONS    |     1 |    31 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

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

   5 - access("E"."LAST_NAME"='Smith')
   7 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
   8 - access("D"."LOCATION_ID"="L"."LOCATION_ID")
```

Query Unnesting – Odmotavanje upita

- Ideja odmotavanja upita je ista kao i kod spajanja pogleda – upit koji sadrži više SQL blokova se transformiše u upit sa jednim blokom. Međutim, odmotavanje upita se koristi kada se ugnježdeni upit nalazi u WHERE delu spoljašnjeg upita.

```
SELECT department_name
FROM DEPARTMENTS d
WHERE d.department_id IN (SELECT
e.department_id
FROM EMPLOYEES e
WHERE e.salary > 10000);
```

```
SELECT department_name
FROM DEPARTMENTS d, EMPLOYEES e
WHERE d.department_id S=e.department_id
AND e.salary > 10000;
```

Plan izvršenja, SEMI JOIN

```
Plan hash value: 2188966913

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |     9 |   207 |     6  (17)| 00:00:01 |
|   1 |  MERGE JOIN SEMI              |              |     9 |   207 |     6  (17)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPARTMENTS  |    27 |   432 |     2   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN            | DEPT_ID_PK   |    27 |       |     1   (0)| 00:00:01 |
|*  4 |   SORT UNIQUE                 |              |    14 |    98 |     4  (25)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL          | EMPLOYEES    |    14 |    98 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   4 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
       filter("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
   5 - filter("E"."SALARY">10000)
```

- Odmotavanje upita se postiže tako što se ugnježdeni upit menja adekvatnom operacijom spoja. U ovom primeru imamo specifičan tip spoja – **SEMI JOIN.** Ovaj spoj funkcioniše kao prirodni spoj, s tim što se rezultujući slog generiše čim se za njega nađe odgovarajući spoj u drugoj tabeli. Zbog toga je ovaj spoj pogodan za odmotavanje upita u slučaju operacije IN, EXISTS, ANY. Spoj koji se suprotan ovom – **ANTI JOIN** se koristi u slučaju operacija NOT IN, NOT EXISTS, ALL.

Join factorization

- Ova transformacija se koristi za optimizaciju izvršavanja SQL upita koji sadrži više grana povezanih UNION ALL operacijom. Tehnikom faktorizacije se prepoznaju izračunavanja koja su zajednička za sve grane ovakvih upita, a zatim se takva izračunavanja izvajaju kao top-level izračunavanja, van UNION ALL operacije.

- Ovakvim pristupom se sprečavaju višestruka izračunavanja za iste operacije, što obezbeđuje drastično poboljšanje performansi.

Join factorisation - primer

```
SELECT e.first_name, e.salary, d.location_id
FROM EMPLOYEES e, DEPARTMENTS d, LOCATIONS l
WHERE e.department_id = d.department_id AND
d.location_id = l.location_id
AND e.salary > 5000
AND l.postal_code IS NOT NULL
UNION ALL
SELECT e.first_name, e.salary, d.location_id
FROM EMPLOYEES e, DEPARTMENTS d, EMPLOYEES m
WHERE e.department_id = d.department_id AND
d.manager_id = m.employee_id
AND e.salary > 5000
AND m.commission_pct IS NOT NULL;
```

```
SELECT e.first_name, e.salary, view.ITEM_1
FROM EMPLOYEES e, (SELECT d.location_id,
d.department_id
    FROM DEPARTMENTS d, LOCATIONS l
    WHERE d.location_id = l.location_id AND
    l.postal_code IS NOT NULL
    UNION ALL
    SELECT d.location_id, d.department_id
    FROM DEPARTMENTS d, EMPLOYEES m
    WHERE d.manager_id = m.employee_id
    AND m.commission_pct IS NOT NULL)
WHERE e.department_id = view.ITEM_2 AND
e.salary > 5000;
```

Join factorisation – primer plana izvršenja

```
| Id  | Operation                     | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
|   0 | SELECT STATEMENT              |                     |    80 |  3200 |    14   (8)| 00:00:01 |
|*  1 |  HASH JOIN                    |                     |    80 |  3200 |    14   (8)| 00:00:01 |
|   2 |   VIEW                        | VW_JF_SET$C68A813A  |    38 |   988 |    11  (10)| 00:00:01 |
|   3 |    UNION-ALL                  |                     |       |       |            |          |
|   4 |     MERGE JOIN                |                     |    27 |   459 |     5  (20)| 00:00:01 |
|*  5 |      TABLE ACCESS BY INDEX ROWID| LOCATIONS         |    22 |   220 |     2   (0)| 00:00:01 |
|   6 |       INDEX FULL SCAN         | LOC_ID_PK           |    23 |       |     1   (0)| 00:00:01 |
|*  7 |      SORT JOIN                |                     |    27 |   189 |     3  (34)| 00:00:01 |
|   8 |       VIEW                    | index$_join$_002    |    27 |   189 |     2   (0)| 00:00:01 |
|*  9 |        HASH JOIN              |                     |       |       |            |          |
|  10 |         INDEX FAST FULL SCAN  | DEPT_ID_PK          |    27 |   189 |     1   (0)| 00:00:01 |
|  11 |         INDEX FAST FULL SCAN  | DEPT_LOCATION_IX    |    27 |   189 |     1   (0)| 00:00:01 |
|* 12 |     HASH JOIN                 |                     |    11 |   176 |     6   (0)| 00:00:01 |
|* 13 |      TABLE ACCESS FULL        | DEPARTMENTS         |    11 |   110 |     3   (0)| 00:00:01 |
|* 14 |      TABLE ACCESS FULL        | EMPLOYEES           |    35 |   210 |     3   (0)| 00:00:01 |
|* 15 |   TABLE ACCESS FULL           | EMPLOYEES           |    57 |   798 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   1 - access("E"."DEPARTMENT_ID"="ITEM_1")
   5 - filter("L"."POSTAL_CODE" IS NOT NULL)
   7 - access("D"."LOCATION_ID"="L"."LOCATION_ID")
       filter("D"."LOCATION_ID"="L"."LOCATION_ID")
   9 - access(ROWID=ROWID)
  12 - access("D"."MANAGER_ID"="M"."EMPLOYEE_ID")
  13 - filter("D"."MANAGER_ID" IS NOT NULL)
  14 - filter("M"."COMMISSION_PCT" IS NOT NULL)
  15 - filter("E"."SALARY">5000)
```

Još neke transformacije

- Osim navedenih transformacija, transformer Oracle optimizatora koristi još neke tehnike za transformaciju upita, kao što su:
  - **Predicate pushing**
  - **Star transformation**
  - **Query rewrite with materialized views**
- Poslednje dve transformacije imaju veliki značaj u Data Warehouse sistemima.

Adaptivna optimizacija upita

- Mehanizam adaptivne optimizacije upita omogućava optimizatoru da prilagođava planove u toku samog izvršenja upita, na osnovu dostupnih informacija o upitu i adaptivnih statistika o kojima se vodi evidencija.
- Podrazumevano je svaki plan adaptivni – sastoji se od više podplanova. Podplan koji će se koristiti za određenu operaciju se određuje u toku izvršenja upita, na osnovu informacija koje se “pridružuju” upitu zahvaljujući **kolektoru statistika**.

Adaptivna optimizacija upita - Primer

```
14 ----------------------------------------------------------------------------------------------------
15 |   Id  | Operation                          | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
16 ----------------------------------------------------------------------------------------------------
17 |     0 | SELECT STATEMENT                   |              |       |       |     8 (100)|          |
18 |     1 |  SORT UNIQUE                       |              |     5 |   112 |     7  (15)| 00:00:01 |
19 |     2 |   UNION-ALL                        |              |       |       |            |          |
20 |  *  3 |    CONNECT BY WITH FILTERING (UNIQUE)|            |       |       |            |          |
21 |- *  4 |     HASH JOIN                      |              |     1 |    16 |     2   (0)| 00:00:01 |
22 |     5 |      NESTED LOOPS                  |              |     1 |    16 |     2   (0)| 00:00:01 |
23 |-    6 |       STATISTICS COLLECTOR         |              |       |       |            |          |
24 |  *  7 |        INDEX RANGE SCAN            | I_CODEAUTH1  |     1 |     8 |     1   (0)| 00:00:01 |
25 |  *  8 |       INDEX RANGE SCAN             | I_SYSAUTH1   |     1 |     8 |     1   (0)| 00:00:01 |
26 |- *  9 |      INDEX FAST FULL SCAN          | I_SYSAUTH1   |     1 |     8 |     1   (0)| 00:00:01 |
27 |- * 10 |     HASH JOIN                      |              |     3 |    63 |     3   (0)| 00:00:01 |
28 |    11 |      NESTED LOOPS                  |              |     3 |    63 |     3   (0)| 00:00:01 |
29 |-   12 |       STATISTICS COLLECTOR         |              |       |       |            |          |
30 |    13 |        CONNECT BY PUMP             |              |       |       |            |          |
31 |  * 14 |       INDEX RANGE SCAN             | I_SYSAUTH1   |     3 |    24 |     1   (0)| 00:00:01 |
32 |- * 15 |      INDEX FAST FULL SCAN          | I_SYSAUTH1   |     3 |    24 |     1   (0)| 00:00:01 |
33 |  * 16 |    INDEX RANGE SCAN                | I_CODEAUTH1  |     1 |     8 |     1   (0)| 00:00:01 |
34 ----------------------------------------------------------------------------------------------------
35
```

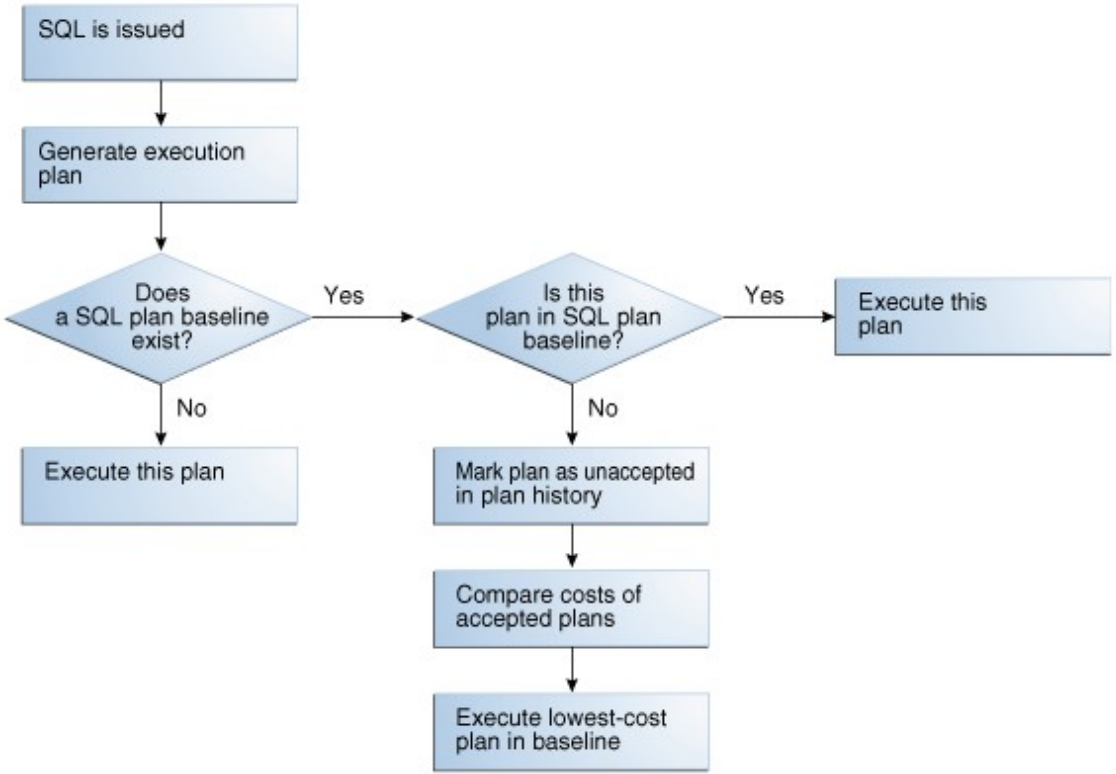
Aproksimacija rezultata upita

- Često je za potrebe analize podataka potrebno primenjivati agregacione funkcije, kao što su AVG, COUNT, MEDIAN, nad ogromnom količinom podataka. Kako se ovakvi upiti često koriste da bi se uvideli određeni trendovi i obrasci u podacima, korisnicima nije presudno da uvek dobiju 100% tačne rezultate. Za ovakve potrebe, Oracle nudi set funkcija kojima se vrednosti rezultata ovakvih upita aproksimiraju, sa tačnošću od oko 97% i izvršavaju se jako brzo. Primeri ovakvih funkcija su **APPROX_SUM, APPROX_DISTINCT, APPROX_COUNT, APPROX_MEDIAN**, itd.

SQL Plan management

- SQL Plan Management je mehanizam koji vodi evidenciju o skupu planova za različite SQL upite – koji se zove SQL Plan Baseline, kojim je za svaki upis definisana donja granica performansi izvršenja. Glavne komponente SQL Plan menadžera su:
  - **Plan Capture** - Komponenta zadužena za snimanje svih podataka o planu izvršenja SQL upita.
  - **Plan Seletion** - Komponenta koja je zadužena za selekciju plana za izvršenje, tako da ne dođe do gubitka performansi.
  - **Plan Evolution** – Procenjuje nove planove u fazi *verifikacije* i ukoliko su dovoljno dobri dodaje ih u baseline, u suprotnom ih odbacuje

Plan selection algoritam



Hvala na pažnji!