

Optimizatorul Oracle:

Referat Sisteme de gestiune a bazelor de date

Tender Laura - Maria

Grupa 234

December 24, 2020

Cuprins

1	Introducere	1
2	Prezentare generală a Optimizatorului Oracle	2
3	Rule based optimizer, cost based optimizer	3
3.1	RBO	3
3.2	Componentele CBO	4
4	Alegerea scopului optimizatorului	5
5	Etapele optimizării	7
6	Metode de acces ale tabelelor	7
7	Tipuri de Join	8
8	Plan de execuție - generare și analiză	9
8.1	Butonul Explain Plan	10
8.2	Comanda Explain Plan For	11
8.3	Exportarea rezultatelor unei cereri în format PDF	11
8.4	Coloanele tabelului PLAN TABLE	12
8.5	Reprezentarea planului sub formă de arbore	14
9	Hints	15
10	Bibliografie	17

1 Introducere

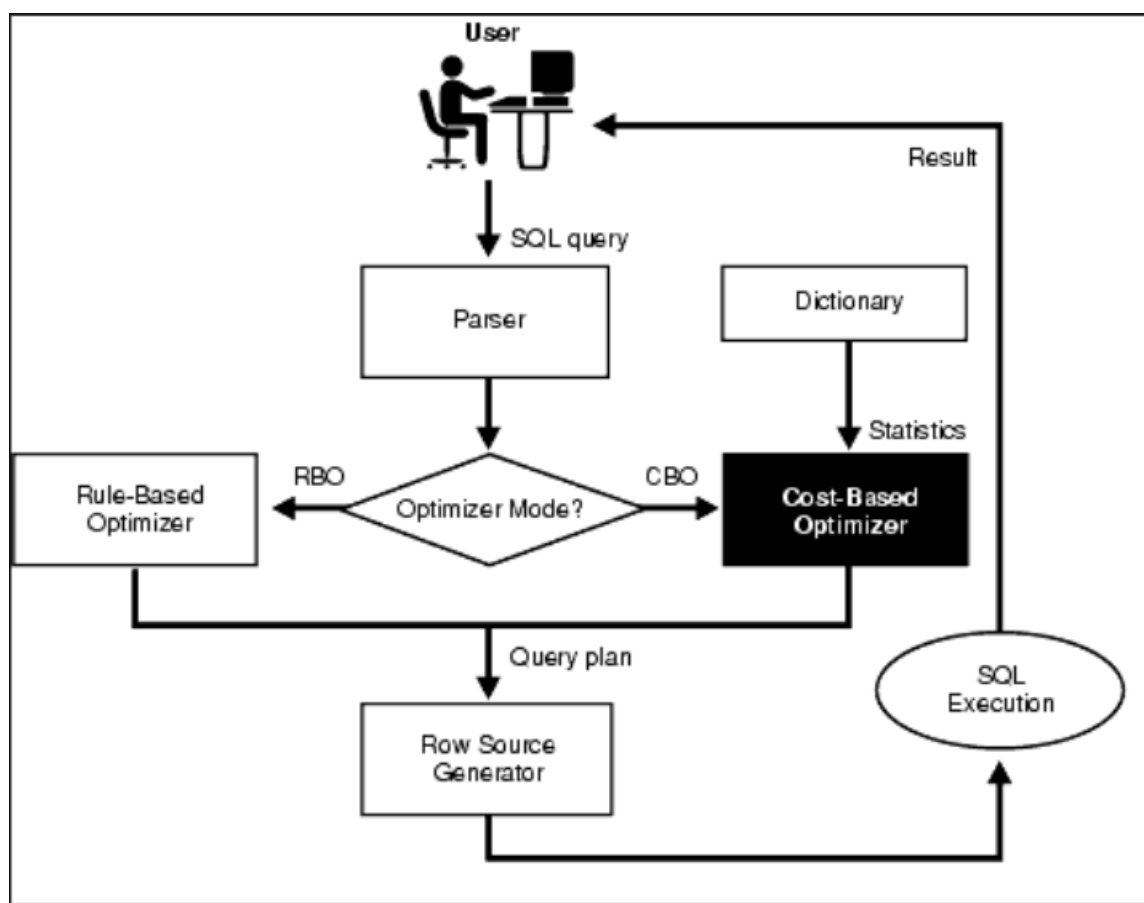
În acest referat voi descrie cum funcționează Optimizatorul Oracle, pașii pe care acesta îi face, modurile în care poate alege cum să execute o cerere (prin reguli sau prin cost), ce reprezintă costul, cum putem vedea planul de execuție al unei cereri și cum îl

putem analiza.

Acest referat are ca suport prezentarea din cadrul Workshopului Oracle Tuning, unde am aflat mai multe informații despre optimizările pe care Oracle le face în spatele cererilor scrise de noi. Înțelegând aceste procese putem îmbunătăți performanța codului nostru în funcție de scopurile aplicației.

2 Prezentare generală a Optimizatorului Oracle

Optimizatorul determină cel mai eficient mod de a executa o instrucțiune SQL după ce analizează obiectele specificate în interogare. Această etapă este importantă în procesarea oricărei instrucțiuni SQL și poate afecta foarte mult timpul de execuție.



3 Rule based optimizer, cost based optimizer

3.1 RBO

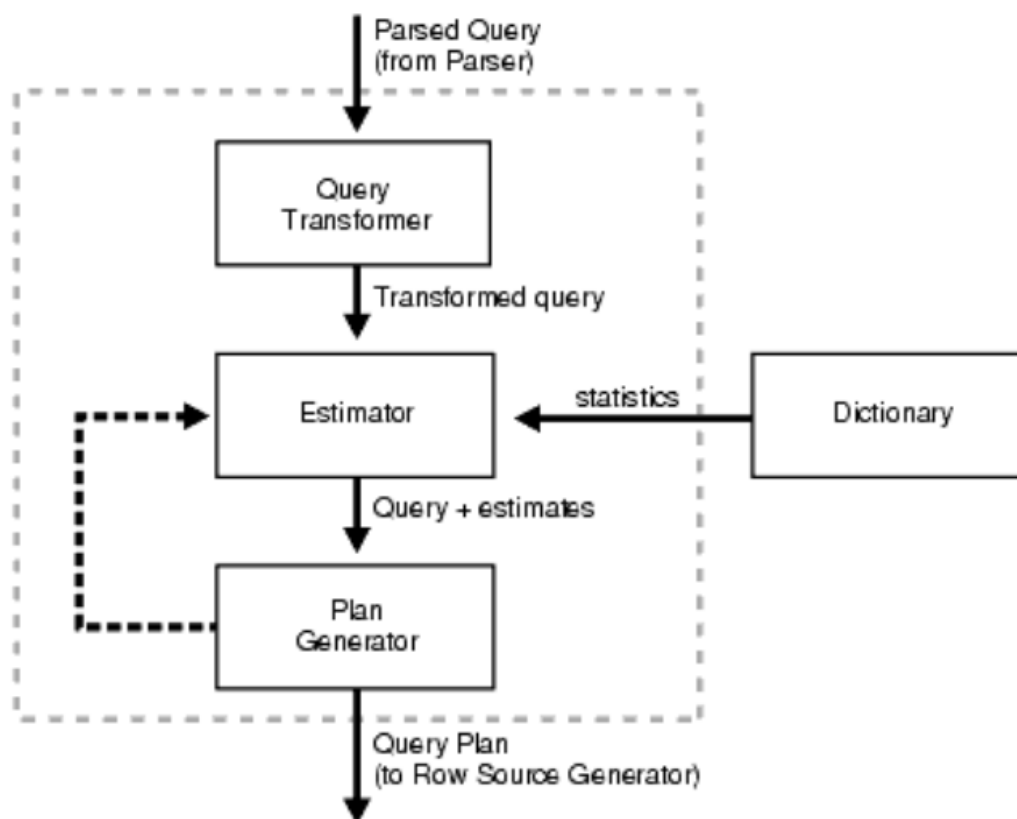
Cu mult timp în urmă, singurul optimizator din baza de date Oracle era Optimizatorul bazat pe reguli (RBO). Practic, RBO folosește un set de reguli pentru a determina cum să execute o interogare. Dacă exista un index pentru un tabel, regulile RBO spuneau să folosească întotdeauna indexul. Există cazuri în care utilizarea unui index încetinește o interogare.

Dacă o interogare returnează aproximativ jumătate din rânduri, atunci utilizarea unui index încetinește execuția. Ar fi mai rapidă accesarea întregul tabel și eliminarea rândurilor care nu satisfac condițiile cererii. Experții în optimizarea interogărilor Oracle au ajuns la concluzia că dacă numărul de rânduri returnate este mai mare decât 5-10 procente din volumul total al tabelului, utilizarea unui index ar încetini execuția.

Astfel, RBO nu a luat întotdeauna cele mai bune decizii. Cea mai mare problemă cu RBO a fost că nu a ținut cont de distribuția datelor. Astfel s-a născut Optimizatorul bazat pe costuri (CBO). CBO utilizează statistici despre tabele, indecșii lor și distribuția datelor pentru a alege cum execută o cerere. RBO există pentru compatibilitatea cu versiunile anterioare. Oracle ne sfătuiește să folosim CBO, RBO va fi înlăturat în viitor.

3.2 Componentele CBO

Componentele CBO sunt reprezentate în următoarea figură:



Estimatorul folosește trei măsurători: selectivitatea, cardinalitatea și costul. Selectivitatea are următoarea formulă și astfel ia valori între 0 și 1 sau poate fi calculată procentual.

$$\text{Selectivity} = \frac{\text{Number of rows satisfying a condition}}{\text{Total number of rows}}$$

Cardinalitatea reprezintă numărul de înregistrări dintr-o mulțime de înregistrări și poate fi determinată prin formula:

$$\text{Cardinality} = \text{Total number of rows} * \text{Selectivity}$$

Costul reprezintă de câte resurse estimează optimizatorul ca va avea nevoie query-ul. CBO ține cont de costurile de I/O, CPU și memorie. Unitatea de măsură a costului este standardized single-block random read, 1 cost unit = 1 SRd.

$$\text{Cost} = \frac{\text{Single-block I/O cost} + \text{Multiblock I/O cost} + \text{CPU cost}}{\text{sreadtim}}$$

#SRds: Number of single-block reads sreadtim: Single-block read time
 #MRds: Number of multiblock reads mreadtim: Multiblock read time
 #CPUCycles: Number of CPU Cycles Cpuspeed: Millions instructions per second

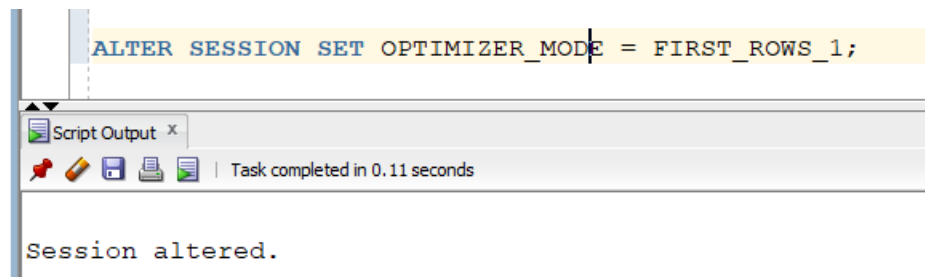
Generatorului de planuri încearcă diferite planuri posibile pentru o anumită interogare și îl alege pe cel care are cel mai mic cost. Multe planuri diferite sunt posibile întrucât există multe combinații de căi de acces, metode de join și ordini în care are loc join-urile.

Observăm că estimatorul din CBO folosește statistici din dicționarul datelor. Statisticile generate includ date: despre tabele - număr de rânduri, numărul de blocuri, lungimea medie a unei înregistrări, despre coloane - numărul de valori distincte din coloană, numărul de valori nule din coloană, distribuția datelor (histograma), despre sistem - performanța I/O, performanța procesorului.

4 Alegerea scopului optimizatorului

Optimizatorul are ca scop să folosească cât mai puține resurse. Oracle poate de asemenea să optimizeze o cerere astfel încât să obțină un timp de răspuns minim. Astfel, planul de execuție poate varia în funcție de scopul optimizatorului. Pentru aplicații interactive, unde utilizatorul așteaptă să vadă primele rezultate ar fi indicat să optimizăm pentru cel mai bun timp de răspuns.

Putem schimba comportamentul optimizatorului pentru toate cererile SQL dintr-o sesiune printr-o comanda precum:



Parametrul OPTIMIZER MODE poate lua următoarele valori:

- CHOOSE, aceasta este valoarea default, optimizatorul alege între o optimizare prin cost sau prin reguli în funcție de statisticile pe care le are. Dacă dicționarul datelor nu conține nicio statistică despre tabelele accesate atunci optimizatorul alege optimizarea prin reguli.
- ALL ROWS, optimizatorul alege o optimizare prin cost pentru toate cererile SQL din sesiune în scopul de a folosi cât mai puține resurse.
- FIRST ROWS N, optimizatorul alege o optimizare prin cost pentru primele N rânduri în scopul de a folosi cât mai puține resurse. N poate fi 1, 10, 100 sau 1000. Această abordare va optimiza timpul de răspuns.
- FIRST ROWS, optimizatorul folosește și costul și euristică pentru a găsi cel mai bun plan. Această abordare poate genera planuri cu costuri mai mari și este valabilă pentru compatibilitatea cu versiuni anterioare.
- RULE Optimizatorul alege să optimizeze prin reguli toate cererile SQL din sesiune indiferent de statistici.

5 Etapele optimizării

- Evaluarea expresiilor și condițiilor care conțin constante;
- Transformarea cererii - pentru cereri complexe, optimizatorul poate transforma cererea originală în alta echivalentă;
- Alegerea scopului - optimizatorul alege în ce scop optimizează cererea așa cum am detaliat secțiunea anterioară;
- Alegerea acces path-urilor - pentru fiecare tabel din cerere, optimizatorul alege cum să acceseze datele din acesta;
- Alegerea tipului de JOIN - pentru fiecare operație de tip JOIN, optimizatorul alege ce tip de JOIN este optim;
- Alegerea ordinii de JOIN - optimizatorul alege în ce ordine are loc operația de JOIN între mai multe tabele.

6 Metode de acces ale tabelelor

Datele dintr-un tabel pot fi accesate prin mai multe metode, printre care următoarele:

- Full Table Scan - toate liniile din tabel sunt citite și sunt eliminate cele care nu respectă criteriile. Optimizatorul folosește un Full Table Scan atunci când nu poate folosi un index, când consideră că are nevoie de majoritatea datelor din tabel sau când tabelul este mic.
- Rowid Scans - liniile sunt accesate prin rowid. Rowid-urile sunt o reprezentare internă a locului în care baza de date stochează datele. Rowid-urile se pot schimba între versiuni, rândurile se pot muta în urma a diferite operații.

- Index Scan - această operație poate fi de mai multe tipuri (Index Unique Scan, Index Range Scan, Index Range Scan Descending, Index Skip Scan, Bitmap Index). O înregistrare este obținută prin index, folosind valorile coloanei indexate precizate în cerere.

Optimizatorul alege metoda de acces asupra unui tabel având în vedere modalitățile posibile, costul estimat, hinturile date optimizatorului (acestea vor fi descrise într-o secțiune ulterioară) și statisticile.

7 Tipuri de Join

Prin operația de JOIN, obținem date din mai multe tabele în cadrul aceleiași cereri.

Într-un Join, unul dintre tabele este inner, celălalt este outer. Operațiile de tip Join se pot clasifica astfel:

- Nested Loop Join - în acest tip de Join, pentru fiecare înregistrare a tabelului cu mai puține date este verificată fiecare înregistrare a celuilalt tabel. Optimizatorul folosește acest tip de Join atunci când există puține înregistrări și o condiție bună de Join între cele două tabele.
- Hash Join - optimizatorul folosește cel mai mic dintre tabele pentru a construi un Hash Table pe cheia de Join în memorie. Dacă tabelul este prea mare pentru a avea loc în memorie, atunci optimizatorul îl împarte în partiții. Optimizatorul folosește Hash Joins atunci când sunt multe înregistrări.
- Sort Merge Join - Ambele mulțimi de date sunt sortate după cheia de Join, apoi sunt interclasate. Acest tip de Join poate fi folosit pentru Join-ul între înregistrări din surse independente. Obține performanțe bune atunci când mulțimile sunt deja

sortate sau sortarea nu este necesară. Acest tip de Join este util când condiția de Join include o inegalitate.

- Cartesian Join - Este un produs cartezian între cele două tabele și este folosit atunci când nu există o condiție de Join.

O comparație între costurile metodelor de Join este următoarea:

Fie A primul tabel și B al doilea tabel.

cost nested loop join = access cost of A + (access cost of B * number of rows from A)

cost hash join = (access cost of A * number of hash partitions of B) + access cost of B

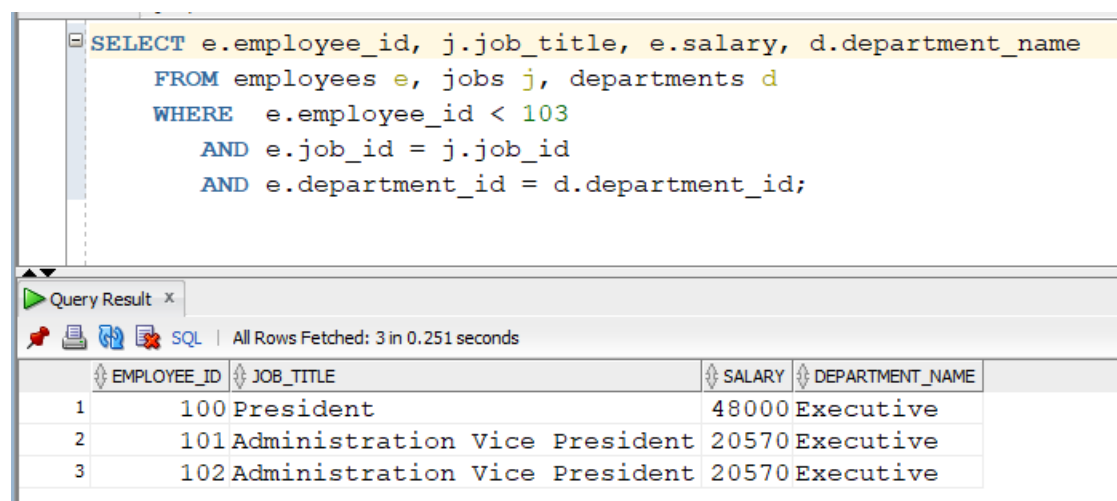
cost sort merge join = access cost of A + access cost of B +(sort cost of A + sort cost of B)

Când datele sunt deja sortate, costurile sortării sunt zero.

8 Plan de execuție - generare și analiză

Un plan de execuție cuprinde detaliile optimizării: modul în care au fost accesate tabelele, tipurile de Join și ordinea în care acestea au fost efectuate. Voi descrie cum putem vedea planul de execuție al unei cereri în SQL Developer și cum interpretăm acest plan.

Fie cererea:

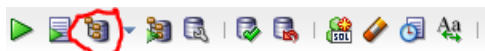


The screenshot shows the SQL Developer interface. The top pane displays a SQL query: `SELECT e.employee_id, j.job_title, e.salary, d.department_name FROM employees e, jobs j, departments d WHERE e.employee_id < 103 AND e.job_id = j.job_id AND e.department_id = d.department_id;`. The bottom pane, titled 'Query Result', shows the results of the query. It indicates 'All Rows Fetched: 3 in 0.251 seconds'. The results are displayed in a table with four columns: EMPLOYEE_ID, JOB_TITLE, SALARY, and DEPARTMENT_NAME. There are three rows of data.

	EMPLOYEE_ID	JOB_TITLE	SALARY	DEPARTMENT_NAME
1	100	President	48000	Executive
2	101	Administration Vice President	20570	Executive
3	102	Administration Vice President	20570	Executive

8.1 Butonul Explain Plan

Pentru a vedea planul de execuție al acestei cereri putem apăsa butonul explain plan.



Astfel vom obține următorul plan de execuție:

Query Result x Explain Plan x				
SQL 0.111 seconds				
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				9
NESTED LOOPS				9
NESTED LOOPS				9
NESTED LOOPS				6
TABLE ACCESS	EMPLOYEES	FULL		3
Filter Predicates				
E.EMPLOYEE_ID < 103				
TABLE ACCESS	JOBS	BY INDEX ROWID		1
INDEX	JOB_ID_PK	UNIQUE SCAN		0
Access Predicates				
E.JOB_ID=J.JOB_ID				
INDEX	DEPT_ID_PK	UNIQUE SCAN		0
Access Predicates				
E.DEPARTMENT_ID=D.DEPARTMENT_ID				
TABLE ACCESS	DEPARTMENTS	BY INDEX ROWID		1
Other XML				
{info}				
info type="db_version"				
11.1.0.6				
info type="parse_schema"				
"GRUPA234"				
info type="dynamic_sampling"				
yes				
info type="plan_hash"				
3732603216				
info type="plan_hash_2"				
3596485099				
{hint}				
NLJ_BATCHING(@"SEL\$1" "D"@"SEL\$1")				
USE_NL(@"SEL\$1" "D"@"SEL\$1")				
USE_NL(@"SEL\$1" "J"@"SEL\$1")				
LEADING(@"SEL\$1" "E"@"SEL\$1" "J"@"SEL\$1" "D"@"SEL\$1")				
INDEX(@"SEL\$1" "D"@"SEL\$1" ("DEPARTMENTS"."DEPARTMENT_ID"))				
INDEX_RS_ASC(@"SEL\$1" "J"@"SEL\$1" ("JOBS"."JOB_ID"))				
FULL(@"SEL\$1" "E"@"SEL\$1")				
OUTLINE_LEAF(@"SEL\$1")				
ALL_ROWS				
DB_VERSION("11.1.0.6")				
OPTIMIZER_FEATURES_ENABLE("11.1.0.6")				
IGNORE_OPTIM_EMBEDDED_HINTS				

Analizând planul putem observa că optimizatorul a ales nested loops, acces full la tabelul employees, index scan la tabelele jobs și departments. Ordinea în care a avut loc operația de Join este employees, jobs, departments.

8.2 Comanda Explain Plan For

O alta modalitate prin care putem vedea planul de execuție este prin următoarea comandă:

```
EXPLAIN PLAN FOR
```

```
SELECT e.employee_id , j.job_title , e.salary , d.department_name
```

```
FROM employees e, jobs j, departments d
```

```
WHERE e.employee_id < 103
```

```
AND e.job_id = j.job_id
```

```
AND e.department_id = d.department_id;
```

Apoi prin

```
SELECT *
```

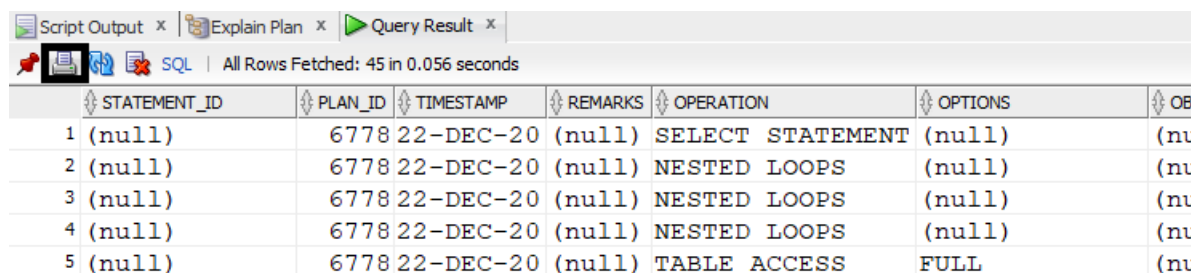
```
FROM PLAN_TABLE;
```

Obținem datele astfel:

STATEMENT_ID	PLAN_ID	TIMESTAMP	REMARKS	OPERATION	OPTIONS	OBJECT_NODE	OBJECT_OWNER	OBJECT_NAME	OBJECT_ALIAS	OBJECT_INSTANCE	OBJECT_TYPE	OPTIMIZER	SEARCH_COLUMNS	ID	PAREN
11608744374295	678623-DEC-20	(null)	SELECT STATEMENT	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	ALL_ROWS	(null)	0	(null)
21608744374295	678623-DEC-20	(null)	NESTED LOOPS	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	1	(null)
31608744374295	678623-DEC-20	(null)	NESTED LOOPS	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	2	(null)
41608744374295	678623-DEC-20	(null)	NESTED LOOPS	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	(null)	3	(null)
51608744374295	678623-DEC-20	(null)	TABLE ACCESS	FULL	(null)	GRUPA234	EMPLOYEES	E@SEL\$1			1 TABLE	(null)	(null)	4	(null)
61608744374295	678623-DEC-20	(null)	TABLE ACCESS	BY INDEX ROWID	(null)	GRUPA234	JOBS	J@SEL\$1			2 TABLE	(null)	(null)	5	(null)
71608744374295	678623-DEC-20	(null)	INDEX	UNIQUE SCAN	(null)	GRUPA234	JOB_ID_PK	J@SEL\$1		(null)	INDEX (UNIQUE)	(null)		6	(null)
81608744374295	678623-DEC-20	(null)	INDEX	UNIQUE SCAN	(null)	GRUPA234	DEPT_ID_PK	D@SEL\$1		(null)	INDEX (UNIQUE)	(null)		7	(null)
91608744374295	678623-DEC-20	(null)	TABLE ACCESS	BY INDEX ROWID	(null)	GRUPA234	DEPARTMENTS	D@SEL\$1			3 TABLE	(null)	(null)	8	(null)

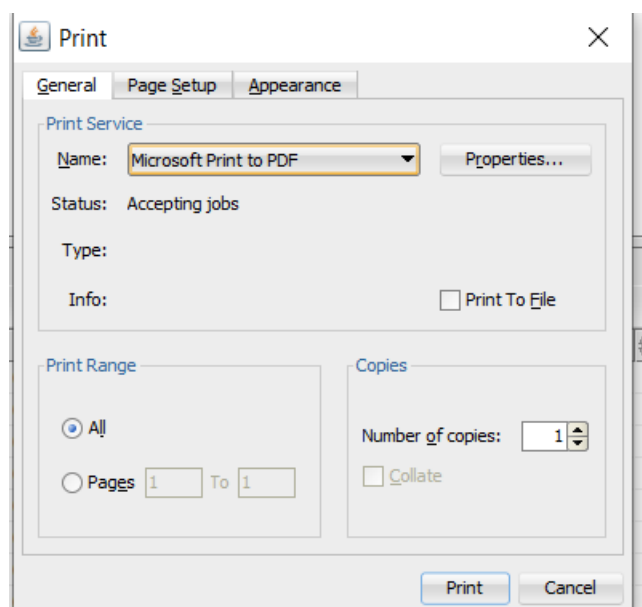
8.3 Exportarea rezultatelor unei cerei în format PDF

Rezultatele unui query pot fi salvate în computer în format PDF. Apasăm pe pictograma Print.



	STATEMENT_ID	PLAN_ID	TIMESTAMP	REMARKS	OPERATION	OPTIONS	OB
1	(null)	6778	22-DEC-20	(null)	SELECT STATEMENT	(null)	(nu
2	(null)	6778	22-DEC-20	(null)	NESTED LOOPS	(null)	(nu
3	(null)	6778	22-DEC-20	(null)	NESTED LOOPS	(null)	(nu
4	(null)	6778	22-DEC-20	(null)	NESTED LOOPS	(null)	(nu
5	(null)	6778	22-DEC-20	(null)	TABLE ACCESS	FULL	(nu

Alegem opțiunea Microsoft Print to PDF și apasam pe butonul Print care ne va conduce la salvarea fișierului în format PDF în calculator.



8.4 Coloanele tabelului PLAN TABLE

Tabelul PLAN TABLE conține următoarele coloane:

- STATEMENT ID - parametru opțional în cererea EXPLAIN PLAN
- PLAN ID - identificator unic al planului
- TIMESTAMP - data la care a fost generat planul
- REMARKS - comentarii care pot fi adăugate la fiecare pas al planului
- OPERATION - numele operației care a avut loc

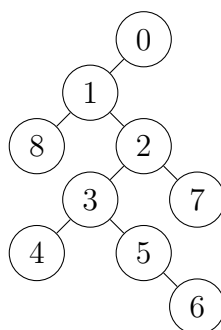
- OPTIONS - detalii despre cum a fost efectuată operația
- OBJECT NODE
- OBJECT OWNER - proprietarul tabelului sau indexului
- OBJECT NAME - numele tabelului sau indexului
- OBJECT ALIAS - alias unic al tabelelor sau viewurilor într-o cerere SQL, pentru indecși este aliasul tabelului la care sunt asociați
- OBJECT INSTANCE - ordinea obiectelor în cererea inițială
- OBJECT TYPE - informații despre tipul obiectului
- OPTIMIZER - modul curent de optimizare
- SEARCH COLUMNS - numărul coloanei care este folosită
- ID - un număr asignat fiecărui pas din planul de execuție
- PARENT ID - id-ul următorului proces care folosește rezultatele procesului curent
- DEPTH - adâncimea unei operații în arborele planului (nivelul)
- POSITION - costul estimat de optimizator pentru prima linie din rezultat
- COST - costul estimat al pasului
- CARDINALITY - numărul de linii estimat că va fi accesat de operație
- BYTES - numărul de bytes estimat pentru operație
- OTHER TAG
- PARTION START
- PARTITION STOP
- PARTITION ID
- OTHER
- OTHER XML

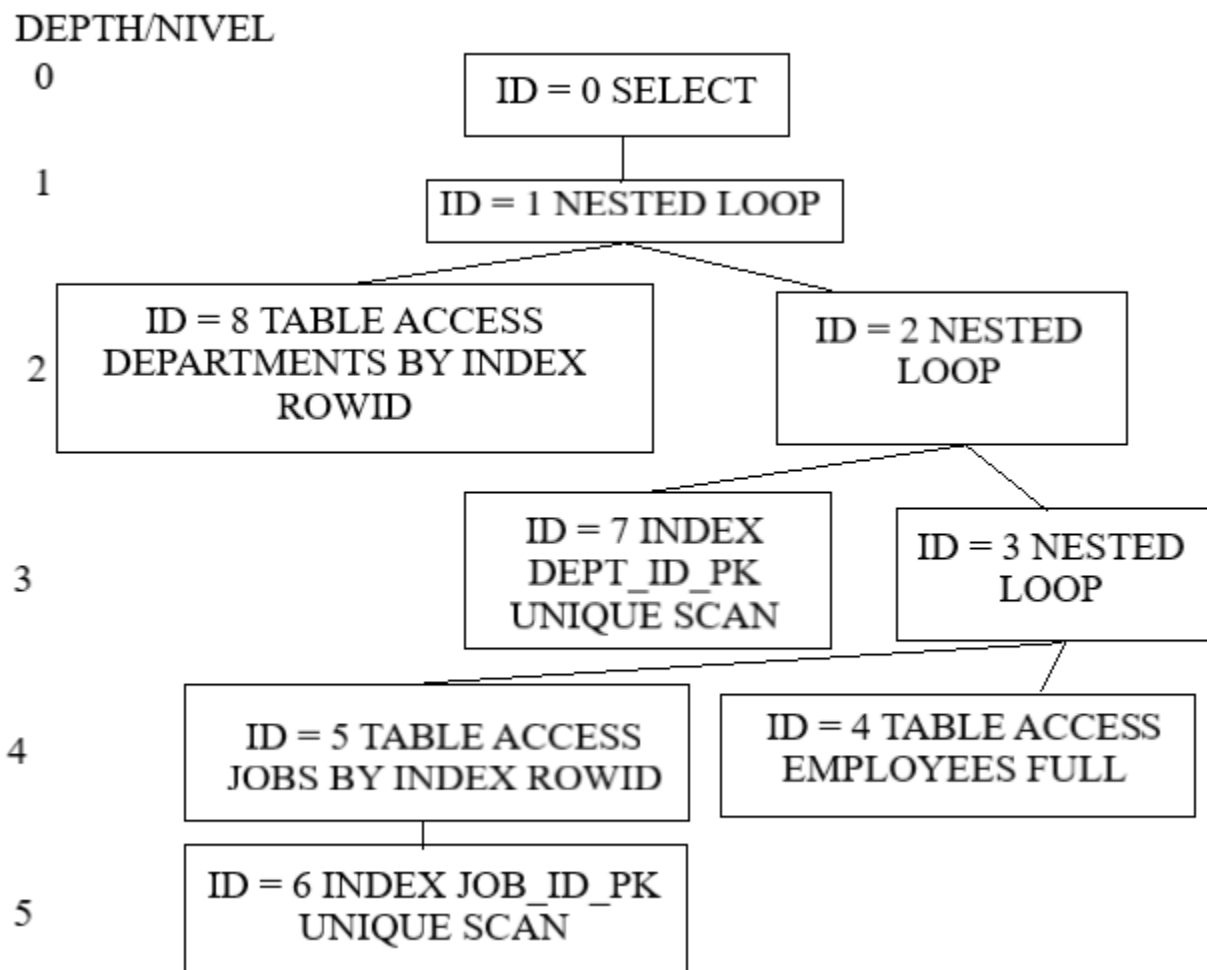
- DISTRIBUTION
- CPU COST - costul CPU estimat
- IO COST - costul I/O estimat
- TEMP SPACE - spațiul temporar estimat
- ACCES PREDICATES - predicate folosite pentru a accesa un obiect
- FILTER PREDICATES - predicate folosite pentru a filtra liniile
- PROJECTION - expresii produse de operații
- TIME - timp estimat în secunde
- QBLOCK NAME - numele blocului înregistrării

8.5 Reprezentarea planului sub formă de arbore

Coloanele ID, PARENT ID, DEPTH indică structura de arbore a planului de execuție.

Astfel vom reprezenta arborele acestui plan. Numărul nodului reprezintă operația cu acel ID.





9 Hints

Hinturile de optimizare pot fi folosite împreună cu cereri SQL pentru a schimba planurile de execuție. Ca designer al aplicației, poți cunoaște informații despre date pe care optimizatorul nu le cunoaște. Prin hinturi poți preciza tipul sau scopul optimizării, cum să fie accesat un tabel, metoda de join, ordinea de join.

Sintaxa pentru a adăuga un hint este următoarea:

```
{DELETE|INSERT|SELECT|UPDATE} /*+ hint [text] [hint [text]]... */
```

sau

{DELETE|INSERT|SELECT|UPDATE} —+ hint [text] [hint [text]]...

Fie următorul exemplu în care i-am precizat optimizatorului să facă un merge join. Analizând planul de execuție putem observa că a respectat hintul.

```
SELECT /*+USE_MERGE(employees departments)*/ *
FROM employees, departments
WHERE employees.department_id = departments.department_id;
```

Query Result x Explain Plan x

SQL | 0.265 seconds

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
MERGE JOIN		
SORT		JOIN
TABLE ACCESS	DEPARTMENTS	FULL
SORT		JOIN
Access Predicates		
EMPLOYEES.DEPARTMENT_ID=DEPARTMENTS.DEPARTMENT_ID		
Filter Predicates		
EMPLOYEES.DEPARTMENT_ID=DEPARTMENTS.DEPARTMENT_ID		
TABLE ACCESS	EMPLOYEES	FULL

Other XML

în absența hintul putem observa că optimizatorul alege să folosească un hash join.

```
SELECT *
FROM employees, departments
WHERE employees.department_id = departments.department_id;
```

Query Result x Explain Plan x

SQL | 0.056 seconds

OPERATION	OBJECT_NAME	OPTIONS
SELECT STATEMENT		
HASH JOIN		
Access Predicates		
EMPLOYEES.DEPARTMENT_ID=DEPARTMENTS.DEPARTMENT_ID		
TABLE ACCESS	DEPARTMENTS	FULL
TABLE ACCESS	EMPLOYEES	FULL

10 Bibliografie

Workshop Oracle Tuning, 26 octombrie 2020

TechTarget, SearchOracle, Rule-based vs Cost-based optimization

Oracle Database Performance Tuning Guide 11g Release 2

Oracle9i Database Performance Tuning Guide and Reference Release 2

Oracle9i Database Performance Tuning Guide and Reference Release 2, Gathering
Optimizer Statistics

Oracle9i Database Performance Tuning Guide and Reference Release 2, Using EXPLAIN
PLAN

Oracle9i Database Performance Tuning Guide and Reference Release 2, Using the
Rule-Based Optimizer

PLAN TABLE

Understanding When Oracle Nested Loop Joins Are Ideal

Optimizer Hints