LABORATOR 8:
INTEROGĂRI IERARHICE

INTEROGĂRI IERARHICE

Interogările ierarhice permit regăsirea datelor pe baza unei relaţii ierarhice care există între liniile tabelului. Dacă există o relaţie ierarhică între liniile unui tabel, un proces de parcurgere a unui arbore (*tree walking*) permite construirea ierarhiei. O cerere ierarhică este o metodă de raportare, în ordine a ramurilor arborelui.
Clauzele instrucţiunii *SELECT* care intervin în formularea unei cereri ierarhice sunt *START WITH* şi *CONNECT BY*.

START WITH specifică o condiţie care identifică liniile ce urmează să fie considerate ca rădăcini ale cererii. Dacă se omite această clauză, sistemul *Oracle* utilizează toate liniile din tabel drept linii rădăcină.

Clauza **CONNECT BY** specifică o condiţie care identifică relaţia dintre liniile „părinte" şi „copil" ale ierarhiei.
Operatorul **PRIOR** face referinţă la linia părinte şi poate fi plasat în faţa oricărui membru al condiţiei specificate de clauza *CONNECT BY*. Plasarea acestui operator determină direcţia interogării, dinspre „părinte" spre „copil" (*top-down*) sau invers (*bottom-up*).

Traversarea *topdown*, respectiv *bottom-up* a arborelui se realizează prin specificări de forma următoare:
Top-down: ***CONNECT BY PRIOR*** *cheie_parinte = cheie_copil*;
Bottom-up: ***CONNECT BY*** *cheie_copil =* ***PRIOR*** *cheie_parinte*;

Liniile „părinte" ale interogării sunt identificate de clauza *START WITH*. Pentru a găsi liniile „copil", *server*-ul evaluează expresia din dreptul operatorului PRIOR pentru linia „părinte", şi cealaltă expresie pentru fiecare linie a tabelului. Înregistrările pentru care condiţia este adevărată vor fi liniile „copil".

Pseudocoloana **LEVEL** determină lungimea drumului de la rădăcină la un nod.

Ordinea de parcurgere a arborelul, în adâncime, este ilustrată mai jos:
Clauza *ORDER BY* va înlocui parcurgerea în adâncime a arborelui cu o afişare a liniilor în ordinea specificată. Pentru a ordona doar fii unui anumit nod după expresiile de la *ORDER BY* se va utiliza **ORDER SIBLINGS BY**.

SYS_CONNECT_BY_PATH returnează calea de la rădacina la un nod. Din fiecare linie nod se va alege coloana specificată de către primul parametru. Valorile coloanelor selectate sunt separate de caracterul specificat de al doilea parametru.

CONNECT_BY_ROOT returnează informaţii preluate din linia rădăcină a linie curente.

Exerciţiul 1: Să se obţină ierarhia bazată pe relaţia angajat – manager considerând angajatul cu codul 100 ca rădăcină (clauza *START WITH*). Să se ordoneze după lungimea drumului de la rădăcină la fiecare nod (pseudocoloana *LEVEL*).

Exerciţiul 2: Scrieţi o cerere ierarhică pentru a afişa codul salariatului, codul managerului şi numele salariatului, pentru angajaţii care sunt cu 2 niveluri sub De Haan. Afişaţi, de asemenea, nivelul angajatului în ierarhie.

Exerciţiul 3: Să se afişeze codul, numele şi salariul angajatului însoţit de codul managerului său, pentru angajaţii al căror salariu este mai mic decât 15000. Se va considera ca rădăcină salariatul având codul 206. Să se indenteze rezultatul pentru a se sugera nivelul pe care se află în arbore fiecare linie.

Exemplul 4: Să se afişeze pentru fiecare angajat toţi superiorii săi pornind de la managerul general. Angajaţii subordonaţi aceluiaşi manager vor fi ordonaţi după salariu.

Exercițiul 5: Să se obţină ierarhia bazată pe relaţia angajat – manager considerând ca rădăcină, angajatul care are cel mai mare salariu.

Alternativa: Sql recursiv

```
with emp_rec
 as
 (select employee_id, last_name, null manager
 from employees
 where manager_id is null
 union all
 select last_name, last_name, er.employee_id
 from employees e join emp_rec er on ( e.manager_id  = er.employee_id)
 )
 select employee_id, last_name,  manager
 from emp_rec;
```

PLAN DE EXECUTIE

EXPLAIN PLAN descrie planul ales de optimizor pentru executia unei cereri.

Exemplul 6:

```
EXPLAIN PLAN
set statement_id = 'leading'
for SELECT /*+ LEADING(d e) */ *  ….


SELECT  LPAD(' ',2*(LEVEL-1))||p.operation operation, p.options,
      p.object_name, p.object_alias, p.position , p.parent_id
   FROM plan_table p
   START WITH p.id = 0 AND p.statement_id = 'leading'
   CONNECT BY PRIOR p.id = parent_id AND p.statement_id = 'leading'
   ORDER BY p.id;
```

Exercitiul 7:
Modificati exemplul 6 astfel incat hint-ul sa indice ordinea LEADING(e d). Difera tipul de join efectuat? Modificati exemplul 6 astfel incat sa se sugereze utilizarea tipului de join **nested loops.**

```
SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY('plan_table',
NULL, 'TYPICAL'));

SELECT * FROM V$SQL;

SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR('  ', 0 ,
'TYPICAL'));
```

CUM se modifica planul de executie?

hint-uri /*+ LEADING(d e j) */ **USE_NL, USE_MERGE, USE_HASH**
INDEX (employees emp_department_ix)

parametrii

```
ALTER SESSION SET OPTIMIZER=FIRST_ROWS;

ALTER SESSION SET OPTIMIZER=ALL_ROWS;
```

statistici

```
EXEC DBMS_STATS.gather_index_stats('grupa33', 'EMP_EMP_ID_PK');
```

```
GRANT SELECT ANY DICTIONARY TO grupa33;
GRANT ANALYZE ANY  TO grupa33;


BEGIN
DBMS_AUTO_TASK_ADMIN.DISABLE( client_name => 'auto optimizer stats collection',
operation => NULL, window_name => NULL);
END;
/
```