

Operatorii ROLLUP și CUBE. Clauza GROUPING SETS. Funcția GROUPING.
Subcereri corelate.
Cereri ierarhice.
Analiza top-n.
Clauza WITH.

I. [Operatorii ROLLUP și CUBE. Clauza GROUPING SETS. Funcția GROUPING.]

Am introdus, în laboratorul 4, operatorii ROLLUP și CUBE. Aceștia se utilizează în cadrul clauzei GROUP BY pentru generarea de linii **superagregat**.

➤ Reamintim că:

- **GROUP BY ROLLUP** (expr_1, expr_2, ..., expr_n) generează **n+1 tipuri de linii**, corespunzătoare următoarelor grupări:
 - GROUP BY (expr_1, expr_2, ..., expr_n-1, expr_n)
 - GROUP BY (expr_1, expr_2, ..., expr_n-1)
 - ...
 - GROUP BY (expr_1, expr_2)
 - GROUP BY (expr_1)
 - GROUP BY () – corespunzător absenței clauzei GROUP BY și deci, calculului funcțiilor grup din cerere pentru întreg tabelul.

Obs:

- Lista de expresii care urmează operatorului ROLLUP este parcursă de la dreapta la stânga, suprimându-se câte o expresie .
- O cerere în care apare un astfel de operator este echivalentă cu reuniunea (UNION ALL) a n+1 cereri.

- **GROUP BY CUBE** (expr_1, expr_2, ..., expr_n) generează 2^n tipuri de linii, corespunzătoare tuturor combinațiilor posibile de expresii din lista.

➤ Pentru determinarea modului în care a fost obținută o valoare totalizatoare cu ROLLUP sau CUBE, se utilizează funcția:

- **GROUPING**(expresie)

Aceasta întoarce:

- valoarea 0, dacă expresia a fost utilizată pentru calculul valorii agregat
- valoarea 1, dacă expresia nu a fost utilizată.

➤ Dacă se dorește obținerea numai a anumitor grupări superagregat, acestea pot fi precizate prin intermediul clauzei :

- **GROUPING SETS** ((expr_11, expr_12, ..., expr_1n), (expr_21, expr_22, ...expr_2m), ...)

Exerciții:

1. a) Să se afișeze numele departamentelor, titlurile job-urilor și valoarea medie a salariilor, pentru:
 - fiecare departament și, în cadrul său pentru fiecare job;
 - fiecare departament (indiferent de job);
 - întreg tabelul.

b) Analog cu a), afișând și o coloană care arată intervenția coloanelor *department_name*, *job_title*, în obținerea rezultatului.

Obs: Pentru formatarea coloanelor numerice în SQL*Plus, este utilă comanda:
COLUMN <coloana> FORMAT '99'

2. a) Să se afișeze numele departamentelor, titlurile job-urilor și valoarea medie a salariilor, pentru:
 - fiecare departament și, în cadrul său pentru fiecare job;
 - fiecare departament (indiferent de job);
 - fiecare job (indiferent de departament)
 - întreg tabelul.
- b) Cum intervin coloanele în obținerea rezultatului? Să se afișeze 'Dep', dacă departamentul a intervenit în agregare, și 'Job', dacă job-ul a intervenit în agregare.
3. Să se afișeze numele departamentelor, numele job-urilor, codurile managerilor, maximul și suma salariilor pentru:
 - fiecare departament și, în cadrul său, fiecare job;
 - fiecare job și, în cadrul său, pentru fiecare manager;
 - întreg tabelul.
4. Să se afișeze salariul maxim al angajaților doar dacă acesta este mai mare decât 15000.

II. [Subcereri corelate (sincronizate)]

O subcerere (cerere imbricată sau încuibărită) corelată poate avea forma următoare:

```
SELECT nume_coloană_1[, nume_coloană_2 ...]
FROM   nume_tabel_1 extern
WHERE  expresie operator
        (SELECT nume_coloană_1 [, nume_coloană_2 ...]
         FROM   nume_tabel_2
         WHERE  expresie_1 = extern.expresie_2);
```

Modul de execuție este următorul :

- cererea externă determină o linie candidat;
- cererea internă este executată utilizând valoarea liniei candidat;
- valorile rezultate din cererea internă sunt utilizate pentru calificarea sau descalificarea liniei candidat;
- pașii precedenți se repetă până când nu mai există linii candidat.

Obs: operator poate fi:

- *single-row operator* (>, =, >=, <, <>, <=), care poate fi utilizat dacă subcererea returnează o singură linie;
- *multiple-row operator* (IN, ANY, ALL), care poate fi folosit dacă subcererea returnează mai mult de o linie.

Obs: O subcerere (corelată sau necorelată) poate apărea în clauzele:

- SELECT
- FROM (vezi laboratorul 4)
- WHERE
- HAVING (vezi laboratorul 4)
- START WITH (vezi mai jos – la cereri ierarhice)

Operatorul EXISTS

- În instrucțiunile *SELECT* imbricate, este permisă utilizarea oricărui operator logic.
- Pentru a testa dacă valoarea recuperată de cererea externă există în mulțimea valorilor regăsite de cererea internă corelată, se poate utiliza operatorul *EXISTS*. Dacă subcererea returnează cel puțin o linie, operatorul returnează valoarea *TRUE*. În caz contrar, va fi returnată valoarea *FALSE*.
- Operatorul *EXISTS* asigură că nu mai este continuată căutarea în cererea internă după ce aceasta regăsește o linie.

Soluția 3: Subcerere în clauza FROM (temă)

8. Pentru fiecare departament, să se obțină numele salariatului având cea mai mare vechime din departament. Să se ordoneze rezultatul după numele departamentului.
9. Să se obțină numele salariaților care lucrează într-un departament în care există cel puțin 1 angajat cu salariul egal cu salariul maxim din departamentul 30.

```
SELECT last_name, salary
FROM employees e
WHERE EXISTS (SELECT 1
              FROM employees
              WHERE e.department_id = department_id
              AND salary = (SELECT MAX(salary)
                           FROM employees
                           WHERE department_id = 30));
```

Obs: Deoarece nu este necesar ca instrucțiunea *SELECT* interioară să returneze o anumită valoare, se poate selecta o constantă ('x', '', 1 etc.). De altfel, din punct de vedere al performanței, selectarea unei constante asigură mai multă rapiditate decât selectarea unei coloane.

10. Să se obțină numele primilor 3 angajați având salariul maxim. Rezultatul se va afișa în ordine crescătoare a salariilor.

Soluția 1: subcerere sincronizată

Soluția 2: vezi analiza top-n (mai jos)

11. Să se afișeze codul, numele și prenumele angajaților care au cel puțin doi subalterni.

12. Să se determine locațiile în care se află cel puțin un departament.

Obs: Ca alternativă a lui *EXISTS*, poate fi utilizat operatorul *IN*. Scrieți și această variantă de rezolvare.

13. Să se determine departamentele în care nu există nici un angajat.

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'x'
                  FROM employees
                  WHERE department_id = d.department_id);
```

Obs: Acest exemplu poate fi rezolvat și printr-o subcerere necorelată, utilizând operatorul *NOT IN* (vezi și laboratorul 3). Atenție la valorile *NULL*! (fie puneți condiția *IS NOT NULL* în subcerere, fie utilizați funcția *NVL*). Scrieți și această variantă de rezolvare.

III.[Subcereri ierarhice]

➤ Clauzele **START WITH** și **CONNECT BY** se utilizează în formularea cererilor ierarhice.

- *START WITH* specifică o condiție care identifică liniile ce urmează să fie considerate ca rădăcini ale cererii ierarhice respective. Dacă se omite această clauză, sistemul *Oracle* utilizează toate liniile din tabel drept linii rădăcină.
- *CONNECT BY* specifică o condiție care identifică relația dintre liniile „părinte” și „copil” ale ierarhiei. Condiția trebuie să conțină operatorul *PRIOR* pentru a face referință la linia „părinte”.
- Operatorul *PRIOR* face referință la linia „părinte”. Plasarea acestui operator determină direcția interogării, dinspre „părinte” spre „copil” (*top-down*) sau invers (*bottom-up*). Traversarea *top-down*, 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 PRIOR** cheie_copil = cheie_parinte;

Obs: Operatorul *PRIOR* poate fi plasat în fața oricărui membru al condiției specificate în clauza

CONNECT BY.

Obs: Liniile „părinte“ ale interogării sunt identificate prin clauza *START WITH*. Pentru a găsi liniile „copil“, server-ul evaluează expresia din dreptul operatorului *PRIOR* pentru linia „părinte“, și cealaltă expresie pentru fiecare linie a tabelului. Înregistrările pentru care condiția este adevărată vor fi liniile „copil“. Spre deosebire de *START WITH*, în clauza *CONNECT BY* nu pot fi utilizate subcereri.

- Pseudocoloana **LEVEL** poate fi utilă într-o cerere ierarhică. Aceasta determină lungimea drumului de la rădăcină la un nod.

Exerciții:

14. Să se afișeze codul, numele, data angajării, salariul și managerul pentru:

a) subalternii directi ai lui De Haan;

b) ierarhia arborescenta de sub De Haan.

```
SELECT     employee_id, last_name, hire_date, salary, manager_id
FROM       employees
START WITH employee_id=( SELECT employee_id
                          FROM   employees
                          WHERE  LOWER(last_name)='de haan')
CONNECT BY manager_id = PRIOR employee_id;
```

Obs: Traversarea precedentă este *top-down*. Faceți modificarea necesară obtinerii unei traversari *bottom-up*. Interpretați rezultatul.

15. Să se obțină ierarhia șef-subaltern, considerând ca rădăcină angajatul având codul 114.

16. Scrieti o cerere ierarhica pentru a afisa codul salariatului, codul managerului si numele salariatului, pentru angajatii care sunt cu 2 niveluri sub De Haan. Afisati, de asemenea, nivelul angajatului în ierarhie.

17. Pentru fiecare linie din tabelul EMPLOYEES, se va afisa o structura arborescenta in care va apărea angajatul, managerul său, managerul managerului etc. Coloanele afișate vor fi: codul angajatului, codul managerului, nivelul în ierarhie (LEVEL) si numele angajatului. Se vor folosi indentari.

Obs: Pentru formatarea afișării coloanei *nume* din cerere, dați comenzile SQL*Plus următoare:

```
SET LINESIZE 100
```

```
COLUMN name FORMAT a25;
```

```
SELECT     employee_id, manager_id, LEVEL, last_name,
          LPAD(last_name, length(last_name)+level*2-2, '_') name
FROM       employees
CONNECT BY employee_id=prior manager_id;
```

18. Să se afișeze ierarhia de sub angajatul având salariul maxim, reținând numai angajații al căror salariu este mai mare de 5000. Se vor afișa codul, numele, salariul, nivelul din ierarhie și codul managerului.

```
SELECT     employee_id, last_name, salary, LEVEL, manager_id
FROM       employees
START WITH salary=( SELECT MAX(salary)
                     FROM   employees)
CONNECT BY PRIOR employee_id = manager_id
          AND salary > 5000 ;
```

Obs: În clauza *CONNECT BY*, coloana *employee_id* este evaluată pentru linia „părinte“, iar coloanele *manager_id* și *salary* sunt evaluate pentru linia „copil“. Pentru a introduce, de exemplu, condiția ca salariul managerilor să fie mai mare decât 15000, se scrie:

```
PRIOR salary > 15000
```

IV.[Clauza WITH]

- Cu ajutorul clauzei *WITH* se poate defini un bloc de cerere înainte ca acesta să fie utilizat într-o interogare.
- Clauza permite reutilizarea aceluiași bloc de cerere într-o instrucțiune *SELECT* complexă. Acest lucru este util atunci când o cerere face referință de mai multe ori la același bloc de cerere, care conține operații *join* și funcții agregat.

Exerciții:

19. Utilizând clauza *WITH*, să se scrie o cerere care afișează numele departamentelor și valoarea totală a salariilor din cadrul acestora. Se vor considera departamentele a căror valoare totală a salariilor este mai mare decât media valorilor totale ale salariilor tuturor angajaților.

```
WITH val_dep AS (SELECT    department_name, SUM(salary) AS total
                    FROM      departments d, employees e
                    WHERE      d.department_id = e.department_id
                    GROUP BY department_name),
val_medie AS (SELECT    SUM(total)/COUNT(*) AS medie
               FROM      val_dep)
SELECT *
FROM    val_dep
WHERE   total > (SELECT    medie
                  FROM      val_medie)
ORDER BY department_name;
```

20. Să se afișeze ierarhic codul, prenumele și numele (pe aceeași coloană), codul job-ului și data angajării, pornind de la subordonații direcți ai lui Steven King care au cea mai mare vechime. Rezultatul nu va conține angajații în anul 1970.

V . [Analiza top-n]

Pentru aflarea primelor *n* rezultate ale unei cereri, este utilă pseudocoloana *ROWNUM*. Aceasta returnează numărul de ordine al unei linii în rezultat.

Exerciții:

21. Să se determine primii 10 cei mai bine plătiți angajați.
*SELECT * FROM (SELECT * FROM employees ORDER BY salary DESC)*
WHERE ROWNUM < 11;
22. Să se determine cele mai prost plătite 3 job-uri, din punct de vedere al mediei salariilor.

VI.[Exerciții – utilizarea alternativă a funcției DECODE sau a structurii CASE; din nou NVL și NVL2; COALESCE; NULLIF]

Obs:

- *NVL(a, b)* – întoarce *a*, dacă *a* este NOT NULL, altfel întoarce *b*;
- *NVL2(a, b, c)* - întoarce *b*, dacă *a* este NOT NULL, altfel întoarce *c*;
- *COALESCE (expr_1, expr_2, ...expr_n)* – întoarce prima expresie NOT NULL din listă;
- *NULLIF(a, b)* – întoarce *a*, dacă *a*!=*b*; altfel întoarce NULL ;
- *DECODE (expresie, val_1, val_2, val_3, val_4, ..., val_2n-1, val_2n, default)* – dacă *expresie* = *val_1*, întoarce *val_2*; dacă *expresie* = *val_3*, întoarce *val_4*; ...; altfel întoarce *default*.
- *DECODE* este echivalent cu *CASE*, a cărui structură este:

```
CASE expresie
    WHEN val_1 THEN val_2
    WHEN val_3 THEN val_4
    ...
    ELSE default
END
```

CASE poate avea si forma:

```

CASE
    WHEN expr_logica_1 THEN val_2
    WHEN expr_logica_3 THEN val_4
    ...
    ELSE default
END

```

23. Să se afișeze informații despre departamente, în formatul următor: „Departamentul <department_name> este condus de {<manager_id> | nimeni} și {are numărul de salariați <n> | nu are salariați}”.

24. Să se afișeze numele, prenumele angajaților și lungimea numelui pentru înregistrările în care aceasta este diferită de lungimea prenumelui.

```

SELECT last_name, first_name,
       NULLIF (LENGTH(last_name), LENGTH(first_name)) Rezultat
FROM   employees;

```

25. Să se afișeze numele, data angajării, salariul și o coloană reprezentând salariul după ce se aplică o mărire, astfel: pentru salariații angajați în 1989 creșterea este de 20%, pentru cei angajați în 1990 creșterea este de 15%, iar salariul celor angajați în anul 1991 crește cu 10%. Pentru salariații angajați în alți ani valoarea nu se modifică.

```

SELECT last_name, hire_date, salary,
       CASE TO_CHAR(hire_date, 'yyyy')
         WHEN '1989' THEN salary * 1.20
         WHEN '1990' THEN salary * 1.15
         WHEN '1991' THEN salary * 1.10
         ELSE salary
       END "Salariu marit"
FROM   employees;

```

Instrucțiunea din acest exemplu poate fi rescrisă utilizând funcția *DECODE* în modul următor:

```

SELECT last_name, hire_date, salary,
       DECODE (TO_CHAR(hire_date, 'yyyy'),
               '1989', salary * 1.20,
               '1990', salary * 1.15,
               '1991', salary * 1.10,
               salary) "Salariu marit"
FROM   employees;

```

26. Să se afișeze:

- suma salariilor, pentru job-urile care încep cu litera S;
- media generală a salariilor, pentru job-ul având salariul maxim;
- salariul minim, pentru fiecare din celelalte job-uri.

```

SELECT job_id,
       ( CASE
         WHEN UPPER(job_id) LIKE 'S%' THEN SUM(salary)
         WHEN job_id= (SELECT job_id
                       FROM   employees
                       WHERE  salary =(SELECT MAX(salary)
                                       FROM   employees))
         THEN (SELECT AVG(salary) FROM employees)
         ELSE MIN(salary)
       END) calcul
FROM   employees
GROUP BY job_id;

```

Se poate folosi *DECODE*?