Baze de date-Anul 2
Laborator 5 SQL

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

I. Operatorii ROLLUP şi CUBE. Clauza GROUPING SETS. Funcţia GROUPING.

Am introdus în laboratorul 4 operatorii ROLLUP şi CUBE. Aceştia 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 absenţei clauzei GROUP BY şi deci, calculului funcţiilor 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 combinaţiilor posibile de expresii din listă.

- Pentru determinarea modului în care a fost obţinută o valoare totalizatoare cu ROLLUP sau CUBE, se utilizează funcţia:

  - ***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 doreşte obţinerea 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), …)

Exerciţii:

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

b) Analog cu a), afişând şi o coloană care arată intervenţia coloanelor *department_name*, *job_title*, în obţinerea rezultatului.

2. a) Să se afişeze numele departamentelor, titlurile job-urilor şi valoarea medie a salariilor, pentru:

- fiecare departament şi, în cadrul său pentru fiecare job;
- fiecare departament (indiferent de job);
- fiecare job (indiferent de departament)
- întreg tabelul.

b) Cum intervin coloanele în obţinerea rezultatului? Să se afişeze 'Dep', dacă departamentul a intervenit în agregare, şi 'Job', dacă job-ul a intervenit în agregare.

3. Să se afişeze numele departamentelor, numele job-urilor, codurile managerilor, maximul şi suma salariilor pentru:
- fiecare departament şi, în cadrul său, fiecare job;
- fiecare job şi, în cadrul său, pentru fiecare manager;
- întreg tabelul.

4. Să se afişeze salariul maxim al angajatilor doar daca 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 execuţie 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;
- paşii precedenţi 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 decât 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 instrucţiunile *SELECT* imbricate, este permisă utilizarea oricărui operator logic.
- Pentru a testa dacă valoarea recuperată de cererea externă există în mulţimea valorilor regăsite de cererea internă corelată, se poate utiliza operatorul *EXISTS*. Dacă subcererea returnează cel puţin 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 regăseşte o linie.

Exerciţii:

5. a) Să se afişeze informaţii despre angajaţii al căror salariu depăşeşte valoarea medie a salariilor colegilor săi de departament.

```
SELECT   last_name, salary, department_id
FROM     employees e
WHERE    salary > (SELECT   AVG(salary)
                   FROM     employees
                   WHERE    department_id = e.department_id);
```

b) Analog cu cererea precedentă, afişându-se şi numele departamentului şi media salariilor acestuia şi numărul de angajaţi. Se cer 2 soluții (cu subcerere necorelată în clauza FROM și cu subcerere corelată în clauza SELECT).

6. Să se afişeze numele şi salariul angajaţilor al căror salariu este mai mare decât salariile medii din toate departamentele. Se cer 2 variante de rezolvare: cu operatorul *ALL* și cu funcţia *MAX*.
7. Să se afișeze numele și salariul celor mai puțin plătiți angajați din fiecare departament (3 soluții: cu și fără sincronizare, subcerere în clauza FROM).
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 instrucţiunea *SELECT* interioară să returneze o anumită valoare, se poate selecta o constantă ('x', '', 1 etc.). De altfel, din punct de vedere al performanţei, 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.

Solutia 1: subcerere sincronizată
Solutia 2: vezi analiza top-n (mai jos)

11. Să se afişeze codul, numele şi prenumele angajaţilor care au cel puţin doi subalterni.
12. Să se determine locaţiile în care se află cel puţin un departament.

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

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

Obs: Se va utiliza *NOT EXISTS*. Acest exemplu poate fi rezolvat şi printr-o subcerere necorelată, utilizând operatorul *NOT IN* (vezi laboratorul 3). Atenţie la valorile NULL! (fie puneţi condiţia IS NOT NULL în subcerere, fie utilizaţi funcţia NVL). Scrieţi şi această variantă de rezolvare.

III. Cereri ierarhice

- Clauzele ***START WITH*** şi ***CONNECT BY*** se utilizează în formularea cererilor ierarhice.
  - *START WITH* specifică o condiţie 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 condiţie care identifică relaţia dintre liniile „părinte“ şi „copil“ ale ierarhiei. Condiţia trebuie să conţină operatorul *PRIOR* pentru a face referinţă la linia „părinte“.
- Operatorul *PRIOR* face referinţă la linia „părinte“. Plasarea acestui operator determină direcţia 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 faţa oricărui membru al condiţiei 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“, şi cealaltă expresie pentru fiecare linie a tabelului. Înregistrările pentru care condiţia 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.

Exerciţii:

14. Să se afişeze codul, numele, data angajării, salariul şi managerul pentru:
    a) subalternii directi ai lui De Haan;
    b) ierarhia arborescenta de sub De Haan.

Obs: Traversarea precedentă este *top-down*. Faceţi modificarea necesară obtinerii unei traversări *bottom-up*. Interpretaţi rezultatul.

15. Să se obţină ierarhia şef-subaltern, considerând ca rădăcină angajatul având codul 114.

16. Scrieți o cerere ierarhică pentru a afișa codul salariatului, codul managerului și numele salariatului, pentru angajații 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 afişate vor fi: codul angajatului, codul managerului, nivelul în ierarhie (LEVEL) si numele angajatului. Se vor folosi indentari.

Obs: Se vor adăuga câte 2 caractere „_” în fața numelui, pentru fiecare nivel (LPAD).

18. Să se afişeze ierarhia de sub angajatul având salariul maxim, reţinând numai angajaţii al căror salariu este mai mare de 5000. Se vor afişa codul, numele, salariul, nivelul din ierarhie şi codul managerului.

```
.....
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* şi *salary* sunt evaluate pentru linia „copil“. Pentru a introduce, de exemplu, conditia ca salariul managerilor sa 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 aceluiaşi bloc de cerere într-o instrucţiune *SELECT* complexă. Acest lucru este util atunci când o cerere face referinţă de mai multe ori la acelaşi bloc de cerere, care conţine operaţii *join* şi funcţii agregat.

Exerciţii:

19. Utilizând clauza *WITH*, să se scrie o cerere care afişează numele departamentelor şi 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 angajatilor.

```
WITH val_dep AS (...),
val_medie  AS (...)
SELECT   *
FROM     val_dep
WHERE   total > (SELECT   medie
                 FROM     val_medie)
ORDER BY department_name;
```

20. Să se afişeze ierarhic codul, prenumele şi numele (pe aceeaşi coloană), codul job-ului şi data angajării, pornind de la subordonaţii direcţi ai lui Steven King care au cea mai mare vechime. Rezultatul nu va conţine angajaţii 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.

Exerciţii:

21. Să se detemine primii 10 cei mai bine plătiţi angajaţi.

22. Să se determine cele mai prost plătite 3 job-uri, din punct de vedere al mediei salariilor.

VI. [Exerciţii – utilizarea alternativă a funcţiei DECODE sau a structurii CASE; din nou NVL şi 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 afişeze informaţii despre departamente, în formatul următor: „Departamentul <*department_name*> este condus de {<*manager_id*> | *nimeni*} şi {are numărul de salariaţi <*n*> | nu are salariati}“.
24. Să se afişeze numele, prenumele angajaţilor şi lungimea numelui pentru înregistrările în care aceasta este diferită de lungimea prenumelui.
25. Să se afişeze numele, data angajării, salariul şi o coloană reprezentând salariul după ce se aplică o mărire, astfel: pentru salariaţii angajaţi în 1989 creşterea este de 20%, pentru cei angajaţi în 1990 creşterea este de 15%, iar salariul celor angajaţi în anul 1991 creşte cu 10%. Pentru salariaţii angajaţi în alţi ani valoarea nu se modifică. (2 soluții: CASE și DECODE)
26. Să se afişeze:

- suma salariilor, pentru job-urile care incep cu litera S;
- media generala a salariilor, pentru job-ul avand salariul maxim;
- salariul minim, pentru fiecare din celelalte job-uri.

Se poate folosi DECODE?