

## **I. Operatorul *DIVISION*.**

## **II. *SQL\*Plus***

### **I. Implementarea operatorului *DIVISION* în *SQL***

Diviziunea este o operație binară care definește o relație ce conține valorile atributelor dintr-o relație care apar **în toate** valorile atributelor din cealaltă relație.

Operatorul *DIVISION* este legat de cuantificatorul universal ( $\forall$ ) care nu există în *SQL*. Cuantificatorul universal poate fi însă simulat cu ajutorul cuantificatorului existențial ( $\exists$ ) utilizând relația:

$$\forall x P(x) \equiv \neg \exists x \neg P(x).$$

Prin urmare, operatorul *DIVISION* poate fi exprimat în *SQL* prin succesiunea a doi operatori *NOT EXISTS*. Alte modalități de implementare a acestui operator vor fi prezentate în exemplul de mai jos.

Extindem diagrama *HR* cu o nouă entitate, *PROJECT*, și o nouă asociere: „angajat lucrează în cadrul unui proiect”, între entitățile *EMPLOYEES* și *PROJECT*. Aceasta este o relație *many-to-many*, care va conduce la apariția unui tabel asociativ, numit *WORKS\_ON*.

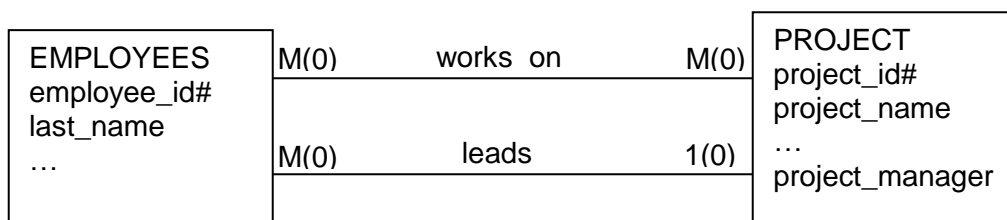
O altă asociere între entitățile *EMPLOYEES* și *PROJECT* este „angajat conduce proiect”. Aceasta este o relație *one-to-many*.

Noile tabele au următoarele scheme relaționale:

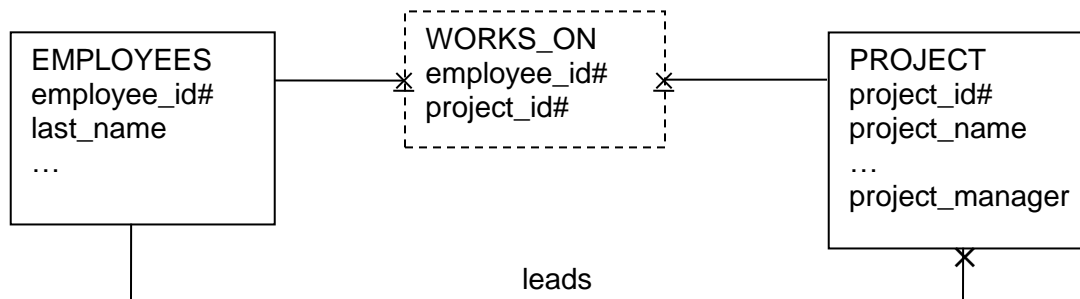
- 1) *PROJECT*(project\_id#, project\_name, budget, start\_date, deadline, delivery\_date, project\_manager)
  - project\_id reprezintă codul proiectului și este cheia primară a relației *PROJECT*
  - project\_name reprezintă numele proiectului
  - budget este bugetul alocat proiectului
  - start\_date este data demarării proiectului
  - deadline reprezintă data la care proiectul trebuie să fie finalizat
  - delivery\_date este data la care proiectul este livrat efectiv
  - project\_manager reprezintă codul managerului de proiect și este cheie externă. Pe cine referă această coloană ? Ce relație implementează această cheie externă?
- 2) *WORKS\_ON*(project\_id#, employee\_id#, start\_date, end\_date)
  - cheia primară a relației este compusă din attributele employee\_id și project\_id.

Scriptul pentru crearea noilor tabele și inserarea de date în acestea este *hr\_project.sql*.

Diagrama entitate-relație corespunzătoare modelului *HR* va fi extinsă, pornind de la entitatea *EMPLOYEES*, astfel:



Partea din diagrama conceptuală corespunzătoare acestei extinderi a modelului este următoarea:



**Exemplu:** Să se obțină codurile salariaților atașați tuturor proiectelor pentru care s-a alocat un buget egal cu 10000.

*Metoda 1* (utilizând de 2 ori *NOT EXISTS*):

```

SELECT    DISTINCT employee_id
FROM      works_on a
WHERE NOT EXISTS
    (SELECT    1
     FROM      project p
     WHERE     budget=10000
     AND NOT EXISTS
        (SELECT    'x'
         FROM      works_on b
         WHERE     p.project_id=b.project_id
         AND       b.employee_id=a.employee_id));
    
```

*Metoda 2* (simularea diviziunii cu ajutorul funcției *COUNT*):

```

SELECT    employee_id
FROM      works_on
WHERE     project_id IN
    (SELECT    project_id
     FROM      project
     WHERE     budget=10000)
GROUP BY  employee_id
HAVING    COUNT(project_id)=
    (SELECT    COUNT(*)
     FROM      project
     WHERE     budget=10000);
    
```

*Metoda 3* (operatorul *MINUS*):

```

SELECT DISTINCT employee_id
FROM  works_on
MINUS
SELECT employee_id from
    ( SELECT employee_id, project_id
    
```

```

FROM (SELECT employee_id FROM works_on) t1,
      (SELECT project_id FROM project WHERE budget=10000) t2
MINUS
SELECT employee_id, project_id FROM works_on
) t3;

```

Metoda 4 ( $A \text{ include } B \Rightarrow B \setminus A = \emptyset$ ):

```

SELECT      DISTINCT employee_id
FROM        works_on a
WHERE NOT EXISTS (
    (SELECT      project_id
      FROM        project p
      WHERE       budget=10000)
  MINUS
  (SELECT      project_id
    FROM        project p, works_on b
    WHERE       p.project_id=b.project_id
    AND         b.employee_id=a.employee_id));

```

### Exerciții (*DIVISION* + alte cereri):

- Să se listeze informații despre angajații care au lucrat în toate proiectele demarate în primele 6 luni ale anului 2006. Implementați toate variantele.
- Să se obțină numele angajaților care au lucrat **cel puțin** pe aceleași proiecte ca și angajatul având codul 200.  
**Obs:** Incluziunea dintre 2 mulțimi se testează cu ajutorul proprietății „ $A \text{ inclus în } B \Rightarrow A - B = \emptyset$ ”. Cum putem implementa acest lucru în SQL?  
 Pentru rezolvarea exercițiului, trebuie selectați angajații pentru care este vidă lista proiectelor pe care a lucrat angajatul 200 mai puțin lista proiectelor pe care au lucrat acei angajați.
  - Să se obțină numele angajaților care au lucrat **cel mult** pe aceleași proiecte ca și angajatul având codul 200.
  - Să se obțină angajații care au lucrat pe aceleași proiecte ca și angajatul având codul 200.  
**Obs:** Egalitatea între două mulțimi se testează cu ajutorul proprietății „ $A=B \Rightarrow A-B=\emptyset$  și  $B-A=\emptyset$ ”.
- Pentru fiecare țară, să se afișeze numărul de angajați din cadrul acesteia.
- Să se afișeze angajații care lucrează în același departament cu cel puțin un manager de proiect.
- Să se afișeze lista angajaților care au lucrat numai pe proiecte conduse de managerul de proiect având codul 102.

## II. *SQL\*Plus*

### Variabile de substitutie

- Variabilele de substitutie sunt utile în crearea de comenzi/script-uri dinamice (care depind de niste valori pe care utilizatorul le furnizează la momentul rularii).

- Variabilele de substitutie se pot folosi pentru stocarea temporara de valori, transmiterea de valori intre comenzi SQL etc. Ele pot fi create prin:
  - comanda *DEFINE*.( *DEFINE* variabila = valoare )
  - Prefixarea cu & (indica existenta unei variabile intr-o comanda SQL, daca variabila nu exista, atunci *SQL\*Plus* o creeaza).
  - Prefixarea cu && (indica existenta unei variabile intr-o comanda SQL, daca variabila nu exista, atunci *SQL\*Plus* o creeaza). Deosebirea fata de & este ca, daca se foloseste &&, atunci referirea ulterioara cu & sau && nu mai cere ca utilizatorul sa introduca de fiecare data valoarea variabilei. Este folosita valoarea data la prima referire.

Variabilele de substitutie pot fi eliminate cu ajutorul comenzii *UNDEFINE*

## Comanda *DEFINE*

Forma comenzii	Descriere
DEFINE variabila = valoare	Creeaza o variabila utilizator cu valoarea de tip sir de caracter precizata.
DEFINE variabila	Afiseaza variabila, valoarea ei si tipul de data al acesteia.
DEFINE	Afiseaza toate variabilele existente in sesiunea curenta, impreuna cu valorile si tipurile lor de date.

### Observatii:

- Variabilele de tip *DATE* sau *CHAR* trebuie sa fie incluse intre apostrofuri in comanda *SELECT*.
- Dupa cum le spune si numele, variabilele de sustitutie inlocuiesc/substituie in cadrul comenzii SQL variabila respectiva cu sirul de caractere introdus de utilizator.
- Variabilele de sustitutie pot fi utilizate pentru a inlocui la momentul rularii:
  - conditii *WHERE*;
  - clauza *ORDER BY*;
  - expresii din lista *SELECT*;
  - nume de tabel;
  - o intreaga comanda SQL;
- Odata definita, o variabila ramane pana la eliminarea ei cu o comanda *UNDEF* sau pana la terminarea sesiunii *SQL\*Plus* respective.
- Comanda *SET VERIFY ON | OFF* permite afisarea sau nu a comenzii inainte si dupa inlocuirea variabilei de substitutie.

## Comenzi interactive in *SQL\*Plus*

Comanda	Descriere
<i>ACC[EPT]</i> variabila [tip] [ <i>PROMPT</i> text]	Citește o linie de intrare și o stochează într-o variabilă utilizator.
<i>PAU[SE]</i> [text]	Afișează o linie vidă, urmată de o linie conținând text, apoi așteaptă ca utilizatorul să apese tasta <i>return</i> . De asemenea, această comandă poate lista două linii

	vide, urmate de aşteptarea răspunsului din partea utilizatorului.
<b>PROMPT</b> [text]	Afişează mesajul specificat sau o linie vidă pe ecranul utilizatorului.

6. Sa se afiseze codul, numele, salariul si codul departamentului din care face parte pentru un angajat al carui cod este introdus de utilizator de la tastatura. Analizati diferentele dintre cele 4 posibilitati prezentate mai jos :

I.

```
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &p_cod;
```

II.

```
DEFINE p_cod; // Ce efect are?
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &p_cod;
UNDEFINE p_cod;
```

III.

```
DEFINE p_cod=100;
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &&p_cod;
UNDEFINE p_cod;
```

IV.

```
ACCEPT p_cod PROMPT "cod= ";
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &p_cod;
```

7. Sa se afiseze numele, codul departamentului si salariul anual pentru toti angajatii care au un anumit job.
8. Sa se afiseze numele, codul departamentului si salariul anual pentru toti angajatii care au fost angajati dupa o anumita data calendaristica.
9. Sa se afiseze o coloana aleasa de utilizator, dintr-un tabel ales de utilizator, ordonand dupa aceeasi coloana care se afiseaza. De asemenea, este obligatorie precizarea unei conditii WHERE.

```
SELECT    &p_coloana -- && determina ca valoarea lui p_coloana san nu mai
           --fie ceruta si pentru clauza ORDER BY, urmand sa
           --fie utilizata valoarea introdusa aici pentru toate
           --aparitiile ulterioare ale lui &p_coloana
FROM      &p_tabel
WHERE     &p_where
ORDER BY  &p_coloana;
```

10. Sa se afiseze numele angajatului, codul job-ului, salariul si numele departamentului pentru salariatii care lucreaza intr-o locatie data de utilizator. Va fi permisa cautarea case-insensitive.

11. Să se citească două date calendaristice de la tastatură și să se afișeze zilele dintre aceste două date.

**Exemplu:** Dacă se introduc datele 1-apr-2008 și 14-apr-2008, rezultatul cererii va fi:

01-apr-2008

02-apr-2008

...

14-apr-2008

Modificați cererea anterioară astfel încât să afișeze doar zilele lucrătoare dintre cele două date calendaristice introduse.