

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)

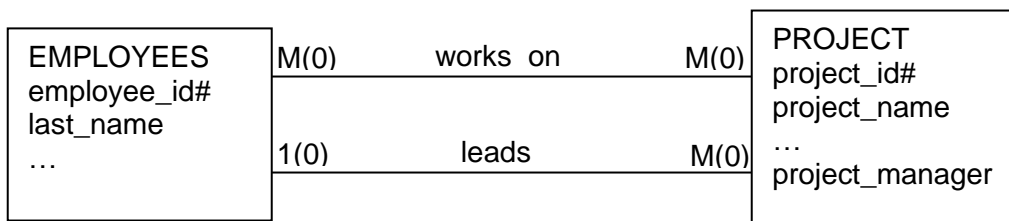
- **project_id** reprezintă codul proiectului și este cheia primară a relației ***PROJECT***
- **project_name** reprezintă numele proiectului
- **budget** este bugetul alocat proiectului
- **start_date** este data demarării proiectului
- **deadline** reprezintă data la care proiectul trebuie să fie finalizat
- **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)

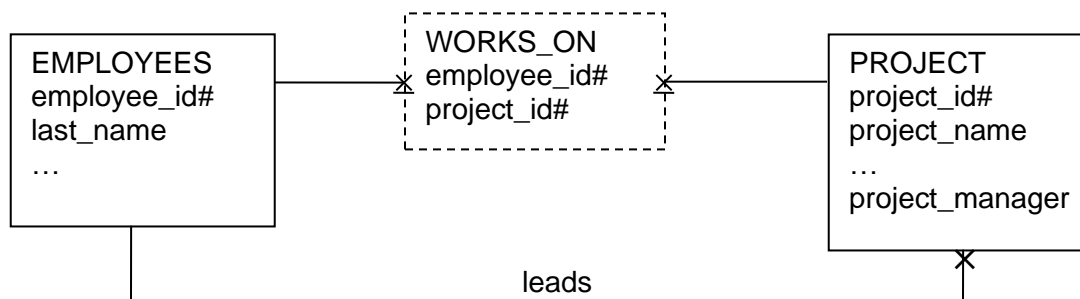
- **cheia primară** a relației este compusă din attributele **employee_id** și **project_id**.

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 employee_id  
FROM works_on
```

MINUS

```
SELECT employee_id from  
( SELECT employee_id, project_id  
  FROM (SELECT DISTINCT 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 p.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):

1. Să se listeze **informații despre angajații** care au lucrat în **toate** proiectele demarate în primele 6 luni ale anului 2006.
2. Să se listeze **informații** despre proiectele la care au participat **toți** angajații care au deținut alte 2 posturi în firmă.
3. Să se obțină **numărul de angajați** care au avut **cel puțin trei job-uri**, luându-se în considerare și job-ul curent.
4. Pentru fiecare țară, să se afișeze **numărul de angajați** din cadrul acesteia.
5. Să se listeze **codurile angajaților și codurile proiectelor** pe care au lucrat. Listarea va cuprinde și angajații care nu au lucrat pe nici un proiect.
6. Să se afișeze **angajații** care **lucrează** în același departament cu **cel puțin** un manager de proiect.
7. Să se afișeze **angajații** care **nu lucrează** în același departament cu nici un manager de proiect.
8. Să se determine **departamentele** având media salariilor mai mare decât **un număr dat**.

Obs : Este necesară o **variabilă de substituție**. Apariția acesteia este indicată prin caracterul „&”. O prezentare a variabilelor de substituție va fi făcută în a doua parte a acestui laborator.

...
HAVING AVG(salary) > &p;

9. Să se afișeze **lista angajaților** care au lucrat **numai** pe proiecte conduse de managerul de proiect având codul 102.
10. a) 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 inclus în B => A-B = Ø”. 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.

b) Să se obțină numele angajaților care au lucrat **cel mult** pe aceleași proiecte ca și angajatul având codul 200.

11. 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 $B-A=\emptyset$ ”.

12. Modelul HR conține un tabel numit **JOB_GRADES**, care conține grilele de salarizare ale companiei.

a) Afișați structura și conținutul acestui tabel.

b) Pentru fiecare angajat, afișați numele, prenumele, salariul și grila de salarizare corespunzătoare.

II. SQL*Plus

Variabile de substitutie

- Variabilele de substitutie sunt utile in crearea de comenzi/script-uri dinamice (care depind de niste valori pe care utilizatorul le furnizeaza 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 creaza). 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 întreaga comandă *SQL*;
- Odata definită, o variabilă ramane până la eliminarea ei cu o comandă **UNDEF** sau până la terminarea sesiunii *SQL*Plus* respective.
- Comanda **SET VERIFY ON | OFF** permite afisarea sau nu a comenzii înainte și după înlocuirea variabilei de substituție.

Comenzi interactive în SQL*Plus

Comanda	Descriere
ACC [EPT] <i>variabila</i> [tip] [PROMPT <i>text</i>]	Citește o linie de intrare și o stochează într-o variabilă utilizator.
PAU [SE] [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 așteptarea răspunsului din partea utilizatorului.
PROMPT [text]	Afișează mesajul specificat sau o linie vidă pe ecranul utilizatorului.

Cum se creează un fișier script ?

De obicei, un fișier script constă în comenzi *SQL*Plus* și cel puțin o instrucțiune *SELECT*. Crearea unui fișier script simplu se poate realiza urmând etapele expuse în continuare.

- 1) Se redactează instrucțiunea *SELECT* la *prompt*-ul *SQL* sau în regiunea de editare din *iSQL*Plus*.
- 2) Se salvează instrucțiunea *SELECT* într-un fișier *script*.
- 3) Se editează fișierul *script*, adăugându-se comenzile *SQL*Plus* corespunzătoare.
- 4) Se verifică dacă instrucțiunea *SELECT* este urmată de un caracter pentru execuție („;“ sau „/“).
- 5) Se salvează fișierul *script*.

Se execută fișierul *script* (prin comenzile @ sau *START*). În *SQL Developer*, se încarcă fișierul și se acționează butonul *Run Script*.

Exercitii (SQL*Plus)

13. Care sunt setările actuale pentru dimensiunea paginii și a liniei în interfața *SQL*Plus*? Setati dimensiunea liniei la 100 de caractere și pe cea a paginii la 24 de linii.

```
SHOW LINESIZE
SHOW PAGESIZE
```

```
SET LINESIZE 100
SET PAGESIZE 24
```

14. 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;
```

15. 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 sa 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;
```

16. Să se realizeze un script (fișier SQL*Plus) prin care să se afișeze numele, job-ul și data angajării salariaților care au început lucrul între 2 date calendaristice introduse de utilizator. Să se concateneze numele și job-ul, separate prin spațiu și virgulă, și să se eticheteze coloana "Angajati". Se vor folosi comanda ACCEPT și formatul pentru data calendaristica MM/DD/YY.

```
ACCEPT data_inceput PROMPT 'Introduceti data de inceput '
ACCEPT data_sfarsit PROMPT 'Introduceti data de sfarsit '
SELECT .... Angajati, hiredate
FROM ...
WHERE hire_date BETWEEN TO_DATE('&data_inceput', 'mm/dd/yy')
                        AND TO_DATE('&data_sfarsit', 'mm/dd/yy');
```