

## Baze de date-Anul 1

### Laborator 1

---

#### 1. Introducere

1. Ce este o bază de date ? Dar un sistem de gestiune a bazelor de date? Dați exemple.

- **Baza de date** este un ansamblu structurat de date coerente, fără redundanță inutilă, care pot fi accesate în mod concurrent de către mai mulți utilizatori.
- **Un sistem de gestiune a bazelor de date (SGBD)** este un produs software care asigură interacțiunea cu o bază de date, permitând definirea, consultarea și actualizarea datelor din baza de date.

#### 2. Ce este SQL?

- **SQL (Structured Query Language)** este un *limbaj* neprocedural pentru interrogarea și prelucrarea informațiilor din baza de date.
  - Compilatorul limbajului SQL generează automat o procedură care accesează baza de date și execută comanda dorită.
  - SQL permite:
    - definirea datelor (LDD)
    - prelucrarea și interrogarea datelor (LMD)
    - controlul accesului la date (LCD).
  - Comenzile SQL pot fi integrate în programe scrise în alte limbaje, de exemplu C, C++, Java etc.

#### 3. Ce este SQL\*Plus? Comenzile SQL\*Plus accesează baza de date ?

- **SQL\*Plus** este un *utilitar Oracle*, având comenzi proprii specifice, care recunoaște instrucțiunile SQL și le trimit server-ului Oracle pentru execuție.
  - Dintre funcționalitățile mediului SQL\*Plus, se pot enumera:
    - editarea, executarea, salvarea și regăsirea instrucțiunilor SQL și a blocurilor PL/SQL;
    - calculul, stocarea și afișarea rezultatelor furnizate de cereri;
    - listarea structurii tabelelor.
  - Tabelul următor evidențiază diferențele dintre instrucțiunile SQL și cele SQL\*Plus:

<b>SQL</b>	<b>SQL*Plus</b>
Este un <b>limbaj</b> de comunicare cu server-ul Oracle pentru accesarea datelor.	Recunoaște instrucțiunile SQL și le transferă server-ului Oracle.
Se bazează pe <b>standardul ANSI</b> pentru SQL.	Este o interfață specifică sistemului Oracle pentru execuția instrucțiunilor SQL.
Prelucreză date și definește obiecte din baza de date.	Nu permite prelucrarea informațiilor din baza de date.
Utilizează funcții pentru a efectua formatari.	Utilizează comenzi pentru a efectua formatari.
Instrucțiunile nu pot fi abreviate.	Comenzile pot fi abreviate.
Nu are un caracter de continuare a instrucțiunilor scrise pe mai multe linii.	Acceptă „-“ drept caracter de continuare pentru comenzile scrise pe mai multe linii.
Caracterul de terminare a unei comenzi este „;“	Nu necesită caracter de terminare a unei comenzi.

4. Comenzile SQL\*Plus acceptă abrevieri? Este necesar vreun caracter de încheiere a comenzi? (vezi tabelul de mai sus)

Care sunt regulile de scriere a comenziilor SQL (acceptă abrevieri, e nevoie de caracter de terminare)?

5. Care sunt limbajele SQL?

- În funcție de tipul acțiunii pe care o realizează, instrucțiunile SQL se împart în mai multe categorii. Datorită importanței pe care o au comenziile componente, unele dintre aceste categorii sunt evidențiate ca limbaje în cadrul SQL, și anume:
  - limbajul de definire a datelor (*LDD*) – comenziile *CREATE*, *ALTER*, *DROP*;
  - limbajul de prelucrare a datelor (*LMD*) – comenziile *INSERT*, *UPDATE*, *DELETE*, *SELECT*;
  - limbajul de control al datelor (*LCD*) – comenziile *COMMIT*, *ROLLBACK*, *SAVEPOINT*.
- Pe lângă instrucțiunile care alcătuiesc aceste limbaje, SQL cuprinde și alte tipuri de instrucțiuni:
  - instrucțiuni pentru controlul sesiunii;
  - instrucțiuni pentru controlul sistemului;
  - instrucțiuni SQL încapsulate.

6. Analizați sintaxa simplificată a comenzi SELECT:

```
SELECT { [ {DISTINCT | UNIQUE} | ALL] lista_campuri | *}
FROM [nume_schemă.]nume_object ]
[ , [nume_schemă.]nume_object ...]
[WHERE condiție_clauza_where]
[START WITH condiție_clauza_start_with
  CONNECT BY condiție_clauza_connect_by]
[GROUP BY expresie [, expresie ...]
  [HAVING condiție_clauza_having]]
[ORDER BY {expresie | poziție} [, {expresie | poziție} ...] ]
[FOR UPDATE
  [OF [ [nume_schemă.]nume_object.]nume_colonă
    [, [nume_schemă.]nume_object.]nume_colonă] ...]
  [NOWAIT | WAIT număr_intreg] ];
```

Sintaxa completă:

[https://docs.oracle.com/cd/E11882\\_01/server.112/e41084/statements\\_10002.htm#SQLRF\\_01702](https://docs.oracle.com/cd/E11882_01/server.112/e41084/statements_10002.htm#SQLRF_01702)

#### Observații:

- Un element din *lista\_campuri* are forma: *expresie* [*AS*] *alias*.
- Dacă un alias conține *blank-uri*, el va fi scris obligatoriu între ghilimele. Altfel, ghilimelele pot fi omise.
- *Alias-ul* apare în rezultat, ca și cap de coloană pentru expresia respectivă. Doar cele specificate între ghilimele sunt *case-sensitive*, celelalte fiind scrise implicit cu majuscule.

7. Care dintre clauze (în sintaxa simplificată) sunt obligatorii?

In instructiunea urmatoare sunt 2 erori. Care sunt acestea?

```
SELECT employee_id, last_name
salary * 12 ANNUAL SALARY
FROM employees;
```

**Observatie:** *ANNUAL SALARY* este un alias pentru câmpul reprezentând salariul anual.

## 2. Exerciții

1. a) Consultați diagrama exemplu *HR* (Human Resources) pentru lucrul în cadrul laboratoarelor de baze de date.  
b) Identificați cheile primare și cele externe ale tabelelor existente în schemă, precum și tipul relațiilor dintre aceste tabele.
2. Să se inițieze o sesiune *SQL\*Plus / SQL Developer* folosind informațiile de conectare indicate.
3. Să se listeze **structura** tabelelor din schema *HR* (*EMPLOYEES, DEPARTMENTS, JOBS, JOB\_HISTORY, LOCATIONS, COUNTRIES, REGIONS*), observând tipurile de date ale coloanelor.

**Obs:** Se va utiliza comanda *DESC[RIBE] nume\_tabel*.

4. Să se listeze **conținutul** tabelelor din schema considerată, afișând valorile tuturor câmpurilor.

**Obs:** *SELECT \* FROM nume\_tabel;*

5. Să se afișeze codul angajatului, numele, codul job-ului, data angajării. Ce fel de operație este aceasta (selecție sau proiecție)?
6. Modificați cererea anterioară astfel încât, la rulare, capetele coloanelor să aibă numele *cod, nume, cod job, data angajării*.
7. Să se listeze, cu și fără duplicate, codurile job-urilor din tabelul *EMPLOYEES*.

**Obs:** Se va utiliza opțiunea *DISTINCT*.

8. Să se afișeze numele concatenat cu job\_id-ul, separate prin virgula și spatiu. Etichetați coloana “Angajat si titlu”.

**Obs:** Operatorul de concatenare este “||”. Sirurile de caractere se specifică între apostrofuri (NU ghilimele, caz în care ar fi interpretate ca alias-uri).

9. Creați o cerere prin care să se afișeze toate datele din tabelul *EMPLOYEES* pe o singură coloană. Separați fiecare coloană printr-o virgulă. Etichetati coloana “Informatii complete”.
10. Să se listeze numele și salariul angajaților care câștigă mai mult de 2850.
11. Să se creeze o cerere pentru a afișa numele angajatului și codul departamentului pentru angajatul având codul 104.
12. Să se afișeze numele și salariul angajaților al căror salar nu se află în intervalul [1500, 2850].

**Obs:** Pentru testarea apartenenței la un domeniu de valori se poate utiliza operatorul *[NOT] BETWEEN valoare1 AND valoare2*.

13. Să se afișeze numele, job-ul și data la care au început lucrul salariații angajați între 20 Februarie 1987 și 1 Mai 1989. Rezultatul va fi ordonat crescător după data de început.

```
SELECT __, __, __
FROM __
WHERE __ BETWEEN '20-FEB-1987' __ '1-MAY-1989'
ORDER BY __;
```

14. Să se afișeze numele salariaților și codul departamentelor pentru toți angajații din departamentele 10, 30 și 50 în ordine alfabetică a numelor.

**Obs:** Apartenența la o mulțime finită de valori se poate testa prin intermediul operatorului *IN*, urmat de lista valorilor (specificate între paranteze și separate prin virgule):

*expresie IN (valoare\_1, valoare\_2, ..., valoare\_n)*

15. Să se listeze numele și salariile angajaților care câștigă mai mult decât 1500 și lucrează în departamentul 10, 30 sau 50. Se vor eticheta coloanele drept *Angajat* și *Salariu lunar*.

16. Care este data curentă? Afisați diferite formate ale acesteia.

**Obs:**

- Functia care returnează data curentă este *SYSDATE*. Pentru completarea sintaxei obligatorii a comenzi *SELECT*, se utilizează tabelul *DUAL*:

*SELECT SYSDATE*

*FROM dual;*

- Datele calendaristice pot fi formatare cu ajutorul funcției *TO\_CHAR(data, format)*, unde formatul poate fi alcătuit dintr-o combinație a următoarelor elemente:

Element	Semnificație
D	Numărul zilei din săptămâna (duminica=1; luni=2; ...sâmbătă=6)
DD	Numărul zilei din lună.
DDD	Numărul zilei din an.
DY	Numele zilei din săptămână, printr-o abreviere de 3 litere (MON, THU etc.)
DAY	Numele zilei din săptămână, scris în întregime.
MM	Numărul lunii din an.
MON	Numele lunii din an, printr-o abreviere de 3 litere (JAN, FEB etc.)
MONTH	Numele lunii din an, scris în întregime.
Y	Ultima cifră din an
YY, YYYY, YYYY	Ultimile 2, 3, respectiv 4 cifre din an.
YEAR	Anul, scris în litere (ex: <i>two thousand four</i> ).
HH12, HH24	Orele din zi, între 0-12, respectiv 0-24.
MI	Minutele din oră.
SS	Secundele din minut.
SSSS	Secundele trecute de la miezul nopții.

17. Să se afișeze numele și data angajării pentru fiecare salariat care a fost angajat în 1987. Se cer 2 soluții: una în care se lucrează cu formatul implicit al datei și alta prin care se formatează data.

**Varianta 1:**

.....  
WHERE *hire\_date* LIKE ('%87%');

**Varianta 2:**

.....  
WHERE *TO\_CHAR(hire\_date, 'YYYY')*='1987';

Sunt obligatorii ghilimelele de la sirul de caractere '1987'? Ce observați?

**Varianta 2':**

.....  
WHERE *EXTRACT(YEAR from hire\_date)=1987*;

**Obs:** Elementele (câmpuri ale valorilor de tip *datetime*) care pot fi utilizate în cadrul acestei funcții sunt: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND.

18. Să se afișeze numele, prenumele și data angajării persoanelor care au început activitatea într-o zi a lunii egală cu cea a datei curente.

19. Să se afișeze numele și job-ul pentru toți angajații care nu au manager.

SQL> *SELECT \_\_\_\_\_, \_\_\_\_\_*  
*FROM \_\_\_\_\_*  
*WHERE manager\_id IS NULL;*

20. Să se afișeze numele, salariul și comisionul pentru toti salariații care câștigă comision (se presupune că aceasta înseamnă prezența unei valori nenule în coloana respectivă). Să se sorteze datele în ordine descrescătoare a salariilor și comisioanelor.
  21. Eliminați clauza *WHERE* din cererea anterioară. Unde sunt plasate valorile *NULL* în ordinea descrescătoare?
  22. Să se listeze numele tuturor angajaților care au a treia literă din nume 'A'.
- Obs:** Pentru compararea sirurilor de caractere, împreună cu operatorul *LIKE* se utilizează caracterele *wildcard*:
- % - reprezentând orice sir de caractere, inclusiv sirul vid;
  - \_ (*underscore*) – reprezentând un singur caracter și numai unul.
23. Să se listeze numele tuturor angajatilor care au 2 litere 'L' în nume și lucrează în departamentul 30 sau managerul lor este 102.
  24. Să se afișeze numele, job-ul și salariul pentru toti salariatii al caror job conține sirul "CLERK" sau "REP" și salariul nu este egal cu 1000, 2000 sau 3000. (operatorul *NOT IN*)
  25. Să se afișeze numele departamentelor care nu au manager.

## Baze de date-Anul 1

### Laborator 2

#### Functii SQL. Cereri multi-relatie (introducere)

##### I. [Functii SQL]

Functiile SQL sunt predefinite in sistemul Oracle si pot fi utilizate in instructiuni SQL. Ele nu trebuie confundate cu functiile definite de utilizator, scrise in PL/SQL.

Daca o functie SQL este apelata cu un argument avand un alt tip de date decat cel asteptat, sistemul converteste implicit argumentul inainte sa evalueze functia.

De obicei, daca o functie SQL este apelata cu un argument null, ea returneaza valoarea null. Functiile care nu urmeaza aceasta regula sunt CONCAT, NVL si REPLACE.

Functiile SQL pot fi clasificate in urmatoarele categorii:

- Functii single-row
- Functii multiple-row (functii agregat)

**1. Functiile single row** returneaza cate o singura linie rezultat pentru fiecare linie a tabelului sau vizualizarii interogate. Aceste functii pot aparea in:

- liste de expresii din clauza SELECT
- clauzele WHERE, START WITH, CONNECT BY si HAVING.

In ceea ce privesc tipul argumentelor asupra carora opereaza si al rezultatelor furnizate, functiile single row pot fi clasificate in categorii corespunzatoare.

□ **Functiile de conversie** cele mai importante sunt:

Funcție	Descriere	Exemplu conversie
TO_CHAR	convertește (sau formatează) un număr sau o dată calendaristică în sir de caractere	TO_CHAR(7) = '7' TO_CHAR(-7) = '-7' TO_CHAR(SYSDATE, 'DD/MM/YYYY') = '26/02/2019'
TO_DATE	convertește (sau formatează) un număr sau un sir de caractere în dată calendaristică	TO_DATE('26-MAR-2019','dd-mon-yyyy')
TO_NUMBER	convertește (sau formatează) un sir de caractere în număr	TO_NUMBER ('-25789', 'S99,999') = -25,789

**Obs:** Există două tipuri de conversii:

- **implicite**, realizate de sistem atunci cand este necesar;
- **explicite**, indicate de utilizator prin intermediul functiilor de conversie.

Conversiile implicite asigurate de server-ul Oracle sunt:

- de la VARCHAR2 sau CHAR la NUMBER;
- de la VARCHAR2 sau CHAR la DATE;
- de la NUMBER la VARCHAR2 sau CHAR;
- de la DATE la VARCHAR2 sau CHAR.

□ Dintre functiile pentru prelucrarea sirurilor de caractere amintim:

Funcție	Descriere	Exemplu
<i>LENGTH(string)</i>	întoarce lungimea sirului de caractere <i>string</i>	<i>LENGTH('Informatica')=11</i>
<i>SUBSTR(string, start [n])</i>	întoarce subșirul lui <i>string</i> care începe pe poziția <i>start</i> și are lungimea <i>n</i> ; dacă <i>n</i> nu este specificat, subșirul se termină la sfârșitul lui <i>string</i> ;	<i>SUBSTR('Informatica', 1, 4) = 'Info'</i> <i>SUBSTR('Informatica', 6) = 'matica'</i> <i>SUBSTR('Informatica', -5) = 'atica'</i> (ultimele 5 caractere)
<i>LTRIM(string [, 'chars'])</i>	șterge din stânga sirului <i>string</i> orice caracter care apare în <i>chars</i> , până la găsirea primului caracter care nu este în <i>chars</i> ; în cazul în care <i>chars</i> nu este specificat, se șterg spațiile libere din stânga lui <i>string</i> ;	<i>LTRIM (' info') = 'info'</i>
<i>RTRIM(string [, 'chars'])</i>	este similar funcției <i>LTRIM</i> , cu excepția faptului că ștergerea se face la dreapta sirului de caractere;	<i>RTRIM ('infoXXXX', 'X') = 'info'</i>
<i>TRIM (LEADING / TRAILING / BOTH chars FROM expresie)</i>	elimină caracterele specificate ( <i>chars</i> ) de la începutul ( <i>leading</i> ), sfârșitul ( <i>trailing</i> ) sau din ambele părți, dintr-o expresie caracter dată.	<i>TRIM (LEADING 'X' FROM 'XXXInfoXXX') = 'InfoXXX'</i> <i>TRIM (TRAILING 'X' FROM 'XXXInfoXXX') = 'XXXInfo'</i> <i>TRIM ( BOTH 'X' FROM 'XXXInfoXXX') = 'Info'</i> <i>TRIM ( BOTH FROM ' Info ') = 'Info'</i>
<i>LPAD(string, length [, 'chars'])</i>	adaugă <i>chars</i> la stânga sirului de caractere <i>string</i> până când lungimea noului sir devine <i>length</i> ; în cazul în care <i>chars</i> nu este specificat, atunci se adaugă spații libere la stânga lui <i>string</i> ;	<i>LPAD (LOWER('iNfO'),6) = ' info'</i>
<i>RPAD(string, length [, 'chars'])</i>	este similar funcției <i>LPAD</i> , dar adăugarea de caracter se face la dreapta sirului;	<i>RPAD (LOWER('InfO'), 6, 'X') = 'infoXX'</i>
<i>REPLACE(string1, string2 [,string3])</i>	întoarce <i>string1</i> cu toate aparițiile lui <i>string2</i> înlocuite prin <i>string3</i> ; dacă <i>string3</i> nu este specificat, atunci toate aparițiile lui <i>string2</i> sunt șterse;	<i>REPLACE ('\$b\$bb','\$', 'a') = 'ababb'</i> <i>REPLACE ('\$b\$bb','\$b', 'ad') = 'adadb'</i> <i>REPLACE ('\$aa','\$') = 'aaa'</i>
<i>UPPER(string), LOWER(string)</i>	transformă toate literele sirului de caractere <i>string</i> în majuscule, respectiv minuscule;	<i>LOWER ('InFo') = 'info'</i> <i>UPPER ('INFO') = 'INFO'</i>
<i>INITCAP(string)</i>	transformă primul caracter al sirului în majusculă, restul caracterelor fiind transformate în minuscule	<i>INITCAP ('iNfO') = 'Info'</i>

<code>INSTR(string, 'chars' [,start [,n]])</code>	caută în <i>string</i> , începând de la poziția <i>start</i> , a <i>n</i> -a apariție a secvenței <i>chars</i> și întoarce poziția respectivă; dacă <i>start</i> nu este specificat, căutarea se face de la începutul sirului; dacă <i>n</i> nu este specificat, se caută prima apariție a secvenței <i>chars</i> ;	$INSTR(LOWER('AbC aBcDe'), 'ab', 5, 2) = 0$ $INSTR(LOWER('AbCdE aBcDe'), 'ab', 5) = 7$
<code>ASCII(char)</code>	furnizează codul ASCII al primului caracter al unui sir	$ASCII('alfa') = ASCII('a') = 97$
<code>CHR(num)</code>	întoarce caracterul corespunzător codului ASCII specificat	$CHR(97) = 'a'$
<code>CONCAT(string1, string2)</code>	realizează concatenarea a două siruri de caractere	$CONCAT('In', 'fo') = 'Info'$
<code>TRANSLATE(string, source, destination)</code>	fiecare caracter care apare în sirurile de caractere <i>string</i> și <i>source</i> este transformat în caracterul corespunzător (aflat pe aceeași poziție ca și în <i>source</i> ) din sirul de caractere <i>destination</i>	$TRANSLATE('$a$aa','$','b') = 'babaa'$ $TRANSLATE('$a$aaa','$a','bc') = 'bcbccc'$

**Obs:** Testarea funcțiilor prezentate se face de maniera : `SELECT apel_functie FROM dual;` astfel că vom omite comanda `SELECT` și vom da numai apelul funcției și rezultatul returnat.

□ **Funcțiile aritmetice single-row** pot opera asupra:

- unei singure valori, și aceste funcții sunt: *ABS* (valoarea absolută), *CEIL* (partea întreagă superioară), *FLOOR* (partea întreagă inferioară), *ROUND* (rotunjire cu un număr specificat de zecimale), *TRUNC* (trunchiere cu un număr specificat de zecimale), *EXP* (ridicarea la putere a lui e), *LN* (logaritm natural), *LOG* (logaritm într-o bază specificată), *MOD* (restul împărțirii a două numere specificate), *POWER* (ridicarea la putere), *SIGN* (semnul unui număr), *COS* (cosinus), *COSH* (cosinus hiperbolic), *SIN* (sinus), *SINH* (sinus hiperbolic), *SQRT* (rădăcina pătrată), *TAN* (tangent), *TANH* (tangent hiperbolic);
- unei liste de valori, iar acestea sunt funcțiile *LEAST* și *GREATEST*, care întorc cea mai mică, respectiv cea mai mare valoare a unei liste de expresii.

□ **Functiile pentru prelucrarea datelor calendaristice** sunt:

Funcție	Descriere	Exemplu
<code>SYSDATE</code>	întoarce data și timpul curent	<code>SELECT SYSDATE FROM dual;</code> (de revăzut utilizarea acestei funcții împreună cu <i>TO_CHAR</i> în cadrul laboratorului 1)
<code>ADD_MONTHS(expr_date, nr_luni)</code>	întoarce data care este după <i>nr_luni</i> luni de la data <i>expr_date</i> ;	$ADD\_MONTHS('02-MAR-2016', 3) = '02-JUN-2016'$ .
<code>NEXT_DAY(expr_date, day)</code>	întoarce următoarea dată după data <i>expr_date</i> , a cărei zi a săptămânii este cea specificată prin sirul de caractere <i>day</i>	$NEXT\_DAY('02-MAR-2016', 'Monday') = '07-MAR-2007'$
<code>LAST_DAY(expr_date)</code>	întoarce data corespunzătoare ultimei zile a lunii din care data <i>expr_date</i> face parte	$LAST\_DAY('02-MAR-2016') = '31-MAR-2016'$

<code>MONTHS_BETWEEN(expr_date1, expr_date2)</code>	întoarce numărul de luni dintre cele două date calendaristice specificate. Valoarea mai mare trebuie specificată în primul argument, altfel rezultatul este negativ.	<code>MONTHS_BETWEEN('02-DEC-2014', '10-OCT-2011') = 37.7419355</code> <code>MONTHS_BETWEEN('10-OCT-2011', '02-DEC-2014') = -37.7419355</code>
<code>TRUNC(expr_date)</code>	întoarce data <code>expr_date</code> , dar cu timpul setat la ora 12:00 AM (miezul nopții)	<code>TO_CHAR(TRUNC(SYSDATE), 'dd/mm/yy HH24:MI') = '26/02/19 00:00'</code>
<code>ROUND(expr_date)</code>	dacă data <code>expr_date</code> este înainte de miezul zilei, întoarce data <i>d</i> cu timpul setat la ora 12:00 AM; altfel, este returnată data corespunzătoare zilei următoare, cu timpul setat la ora 12:00 AM	<code>TO_CHAR(ROUND(SYSDATE), 'dd/mm/yy hh24:mi am') = '26/02/19 00:00 AM'</code>
<code>LEAST(d1, d2, ..., dn), GREATEST(d1, d2, ..., dn)</code>	dintr-o listă de date calendaristice, funcțiile întorc prima, respectiv ultima dată în ordine cronologică	<code>LEAST(SYSDATE, SYSDATE + 3, SYSDATE - 5) = SYSDATE-5</code> <code>GREATEST(SYSDATE, SYSDATE + 3, SYSDATE - 5) = SYSDATE + 3</code>

Operațiile care se pot efectua asupra datelor calendaristice sunt următoarele:

Operație	Tipul de date al rezultatului	Descriere
<code>expr_date -/+ expr_number</code>	Date	Scade/adună un număr de zile dintr-o / la o dată. Numărul de zile poate să nu fie întreg (putem adăuga, de exemplu, un număr de minute sau de ore).
<code>expr_date1 - expr_date2</code>	Number	Întoarce numărul de zile dintre două date calendaristice. Data <code>expr_date1</code> trebuie să fie mai recentă decât <code>expr_date2</code> , altfel rezultatul este negativ.

#### □ Funcții diverse:

Funcție	Descriere	Exemplu
<code>DECODE(value, if1, then1, if2, then2, ..., ifN, thenN, else)</code>	returnează <code>then1</code> dacă <code>value</code> este egală cu <code>if1</code> , <code>then2</code> dacă <code>value</code> este egală cu <code>if2</code> etc.; dacă <code>value</code> nu este egală cu nici una din valorile <code>if</code> , atunci funcția întoarce valoarea <code>else</code> ;	<code>DECODE ('a', 'a', 'b', 'c') = 'b'</code> <code>DECODE ('b', 'a', 'b', 'c') = 'c'</code> <code>DECODE ('c', 'a', 'b', 'c') = 'c'</code>
<code>NVL(expr_1, expr_2)</code>	dacă <code>expr_1</code> este <code>NULL</code> , întoarce <code>expr_2</code> ; altfel, întoarce <code>expr_1</code> . Tipurile celor două expresii trebuie să fie compatibile sau <code>expr_2</code> să poată fi convertit implicit la <code>expr_1</code>	<code>NVL(NULL, 1) = 1</code> <code>NVL(2, 1) = 2</code> <code>NVL('a', 1) = 'a' -- conversie implicită</code> <code>NVL(1, 'a') -- eroare --nu are loc conversia implicită</code>

<code>NVL2(expr_1, expr_2, expr_3)</code>	dacă <code>expr_1</code> este <code>NOT NULL</code> , întoarce <code>expr_2</code> , altfel întoarce <code>expr_3</code>	$NVL2(1, 2, 3) = 2$ $NVL2(NULL, 1, 2) = 2$
<code>NULLIF(expr_1, expr_2)</code>	Daca <code>expr_1 = expr_2</code> atunci funcția returnează <code>NULL</code> , altfel returnează expresia <code>expr_1</code> . Echivalent cu <code>CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END</code>	$NULLIF(1, 2) = 1$ $NULLIF(1, 1) = NULL$
<code>COALESCE(expr_1, expr_2, ..., expr_n)</code>	Returnează prima expresie NOT <code>NULL</code> din lista de argumente.	<code>COALESCE(NULL, NULL, 1, 2, NULL) = 1</code>
<code>UID, USER</code>	întorc <code>ID</code> -ul, respectiv <code>username</code> -ul utilizatorului <code>ORACLE</code> curent	<code>SELECT USER FROM dual;</code>
<code>VSIZE(expr)</code>	întoarce numărul de octeți ai unei expresii de tip <code>DATE</code> , <code>NUMBER</code> sau <code>VARCHAR2</code>	<code>SELECT VSIZE(salary) FROM employees WHERE employee_id=200;</code>

Utilizarea funcției `DECODE` este echivalentă cu utilizarea clauzei `CASE` (într-o comandă SQL). O formă a acestei clauze este:

<code>CASE expr WHEN expr_1 THEN     valoare_1 [WHEN expr_2 THEN     valoare_2 ... WHEN expr_n THEN     valoare_n ] [ELSE valoare] END</code>	În funcție de valoarea expresiei <code>expr</code> returnează <code>valoare_i</code> corespunzătoare primei clauze <code>WHEN .. THEN</code> pentru care <code>expr = expresie_i</code> ; dacă nu corespunde cu nici o clauză <code>WHEN</code> atunci returnează valoarea din <code>ELSE</code> . Nu se poate specifica <code>NULL</code> pentru toate valorile de returnat. Toate valorile trebuie să aibă același tip de date.
---	---

**2. Funcțiile multiple-row (agregat)** pot fi utilizate pentru a returna informația corespunzătoare fiecărui dinte grupurile obținute în urma divizării liniilor tabelului cu ajutorul clauzei `GROUP BY`. Ele pot apărea în clauzele `SELECT`, `ORDER BY` și `HAVING`. Server-ul `Oracle` aplică aceste funcții fiecărui grup de linii și returnează un singur rezultat pentru fiecare mulțime.

Dintre funcțiile grup definite în sistemul `Oracle`, se pot enumera: `AVG`, `SUM`, `MAX`, `MIN`, `COUNT`, `STDDEV`, `VARIANCE` etc. Tipurile de date ale argumentelor funcțiilor grup pot fi `CHAR`, `VARCHAR2`, `NUMBER` sau `DATE`. Funcțiile `AVG`, `SUM`, `STDDEV` și `VARIANCE` operează numai asupra valorilor numerice. Funcțiile `MAX` și `MIN` pot opera asupra valorilor numerice, caracter sau dată calendaristică.

Toate funcțiile grup, cu excepția lui `COUNT(*)`, ignoră valorile `null`. `COUNT(expresie)` returnează numărul de linii pentru care expresia dată nu are valoarea `null`. Funcția `COUNT` returnează un număr mai mare sau egal cu zero și nu întoarce niciodată valoarea `null`.

Când este utilizată clauza `GROUP BY`, server-ul sortează implicit mulțimea rezultată în ordinea crescătoare a valorilor coloanelor după care se realizează gruparea.

## II. [Join]

**Join**-ul este operația de regăsire a datelor din două sau mai multe tabele, pe baza valorilor comune ale unor coloane. De obicei, aceste coloane reprezintă cheia primară, respectiv cheia externă a tabelelor.

Condiția de *join* se poate scrie în clauza `WHERE` a instrucțiunii `SELECT`. Într-o instrucțiune `SELECT` care unește tabele prin operația de *join*, se recomandă ca numele coloanelor să fie precedate de numele sau alias-urile tabelelor pentru claritate și pentru îmbunătățirea timpului de acces la baza de date. Dacă același nume de coloană apare în mai mult de două tabele, atunci numele coloanei se prefixează **obiigatoriu** cu numele sau

alias-ul tabelului corespunzător. Pentru a realiza un *join* între ***n* tabele**, va fi nevoie de cel puțin ***n – 1* condiții de *join***.

**Inner join (equijoin, join simplu)** – corespunde situației în care valorile de pe coloanele ce apar în condiția de *join* trebuie să fie egale.

Operația va fi reluată și completată în cadrul laboratorului 3.

### III. [Exerciții]

#### [Funcții pe șiruri de caractere]

1. Scrieți o cerere care are următorul rezultat pentru fiecare angajat:

<prenume angajat> <nume angajat> castiga <salariu> lunar dar doreste <salariu de 3 ori mai mare>. Etichetati coloana “Salariu ideal”. Pentru concatenare, utilizați atât funcția **CONCAT** cât și operatorul “||”.

```
SELECT CONCAT(CONCAT(...)) || ' castiga ' || salary
    || ... "Salariu ideal"
FROM employees;
```

2. Scrieți o cerere prin care să se afișeze prenumele salariatului cu prima literă majusculă și toate celelalte litere minuscule, numele acestuia cu majuscule și lungimea numelui, pentru angajații al căror nume începe cu J sau M sau care au a treia literă din nume A. Rezultatul va fi ordonat descrescător după lungimea numelui. Se vor eticheta coloanele corespunzătoare. Se cer 2 soluții (cu operatorul **LIKE** și funcția **SUBSTR**).
3. Să se afișeze, pentru angajații cu prenumele „Steven”, codul și numele acestora, precum și codul departamentului în care lucrează. Căutarea trebuie să nu fie case-sensitive, iar eventualele *blank*-uri care preced sau urmează numelui trebuie ignorate.
4. Să se afișeze pentru toți angajații al căror nume se termină cu litera 'e', codul, numele, lungimea numelui și poziția din nume în care apare prima data litera 'a'. Utilizați **alias-uri** corespunzătoare pentru coloane.

#### [Funcții aritmetice]

5. Să se afișeze detalii despre salariații care au lucrat un număr întreg de săptămâni până la data curentă.

**Obs:** Soluția necesită rotunjirea diferenței celor două date calendaristice. De ce este necesar acest lucru?

6. Să se afișeze codul salariatului, numele, salariul, salariul mărit cu 15%, exprimat cu două zecimale și numărul de sute al salariului nou rotunit la 2 zecimale. Etichetați ultimele două coloane “Salariu nou”, respectiv “Numar sute”. Se vor lua în considerare salariații al căror salariu nu este divizibil cu 1000.
7. Să se listeze numele și data angajării salariaților care câștigă comision. Să se eticheteze coloanele „Nume angajat”, „Data angajării”. Utilizați funcția **RPAD** pentru a determina ca data angajării să aibă lungimea de 20 de caractere.

#### [Funcții și operații cu date calendaristice]

8. Să se afișeze data (numele lunii, ziua, anul, ora, minutul și secunda) de peste 30 zile.
9. Să se afișeze numărul de zile rămase până la sfârșitul anului.
10. a) Să se afișeze data de peste 12 ore.  
b) Să se afișeze data de peste 5 minute

**Obs:** Cât reprezintă 5 minute dintr-o zi?

11. Să se afișeze numele și prenumele angajatului (într-o singură coloană), data angajării și data negocierii salarizării, care este prima zi de luni după 6 luni de serviciu. Etichetați această coloană "Negociere".

12. Pentru fiecare angajat să se afișeze numele și numărul de luni de la data angajării. Etichetați coloana "Luni lucrate". Să se ordoneze rezultatul după numărul de luni lucrate. Se va rotunji numărul de luni la cel mai apropiat număr întreg.

**Obs:** În clauza *ORDER BY*, precizarea criteriului de ordonare se poate realiza și prin indicarea *alias-urilor* coloanelor sau a pozițiilor acestora în clauza *SELECT*.

13. Să se afișeze numele, data angajării și ziua săptămânii în care a început lucrul fiecare salariat. Etichetați coloana "Zi". Ordonați rezultatul după ziua săptămânii, începând cu Luni.

### [Functii diverse]

14. Să se afișeze numele angajaților și comisionul. Dacă un angajat nu câștigă comision, să se scrie "Fara comision". Etichetați coloana "Comision".

*SELECT \_\_, NVL(\_\_, \_\_) \_\_  
FROM \_\_;*

15. Să se listeze numele, salariul și comisionul tuturor angajaților al căror venit lunar (salariu + valoare comision) depășește 10000.

### [Instrucțiunea CASE, comanda DECODE]

16. Să se afișeze numele, codul job-ului, salariul și o coloană care să arate salariul după mărire. Se presupune că pentru IT\_PROG are loc o mărire de 20%, pentru SA\_REP creșterea este de 25%, iar pentru SA\_MAN are loc o mărire de 35%. Pentru ceilalți angajați nu se acordă mărire. Să se denumească coloana "Salariu renegociat".

### [Join]

17. Să se afișeze numele salariatului, codul și numele departamentului pentru toți angajații.

*SELECT \_\_, employees.department\_id, \_\_  
FROM employees, departments  
WHERE employees.department\_id=departments.department\_id;  
sau  
SELECT \_\_, e.department\_id, \_\_  
FROM employees e, departments d  
WHERE e.department\_id=d.department\_id;*

**Obs:** Am realizat operația de join între tabelele *employees* și *departments*, pe baza coloanei comune *department\_id*. Observați utilizarea *alias-urilor*. Ce se întâmplă dacă eliminăm condiția de *join*?

**Obs:** Numele sau *alias-urile* tabelelor sunt obligatorii în dreptul coloanelor care au același nume în mai multe tabele. Altfel, nu sunt necesare dar este recomandată utilizarea lor pentru o mai bună claritate a cererii.

18. Să se listeze codurile și denumirile job-urilor care există în departamentul 30.

19. Să se afișeze numele angajatului, numele departamentului și orașul pentru toți angajații care câștigă comision.

20. Să se afișeze numele salariatului și numele departamentului pentru toți salariații care au litera A inclusă în nume.

21. Să se afișeze numele, titlul job-ului și denumirea departamentului pentru toți angajații care lucrează în Oxford.
22. Să se afișeze codul angajatului și numele acestuia, împreună cu numele și codul șefului său direct. Se vor eticheta coloanele Ang#, Angajat, Mgr#, Manager.  
**Obs:** Realizăm operația de self-join (inner join al tabelului cu el însuși).
23. Să se modifice cererea anterioară pentru a afișa toți salariații, inclusiv cei care nu au șef.  
**Obs:** Realizăm operația de outer-join, indicată în SQL prin “(+)” plasat la dreapta coloanei deficitare în informație.
24. Scrieți o cerere care afișează numele angajatului, codul departamentului în care acesta lucrează și numele colegilor săi de departament. Se vor eticheta coloanele corespunzător.
25. Creați o cerere prin care să se afișeze numele, codul job-ului, titlul job-ului, numele departamentului și salariul angajaților. Se vor include și angajații al căror departament nu este cunoscut.
26. Să se afișeze numele și data angajării pentru salariații care au fost angajați după Gates.
27. Să se afișeze numele salariatului și data angajării împreună cu numele și data angajării șefului direct pentru salariații care au fost angajați înaintea șefilor lor. Se vor eticheta coloanele Angajat, Data\_ang, Manager și Data\_mgr.

# Baze de date - Anul 1

## Laborator 3

---

### Interogări multi-relație: Operația de join. Operatori pe mulțimi. Funcții (completare).

#### I. [Obiective]

În acest laborator vom continua lucrul cu interogări **multi-relație** (acestea sunt cele care regăsesc date din mai multe tabele).

În laboratorul precedent am introdus deja diferite tipuri de **join**. Vom relua această operație, vom analiza și o altă metodă de implementare a ei și, de asemenea, vom utiliza **operatori pe mulțimi**.

Foarte utile în rezolvarea exercițiilor propuse vor fi **funcțiile SQL**, prezentate în laboratorul 2. Vom completa lista de funcții utilizate anterior cu alte cîteva exemple, la finalul laboratorului.

#### II. [Join]

Am implementat deja operația de **join** (compunere a tabelelor) în cadrul unor exemple relative la modelul utilizat în exemple și exerciții (HR).

**Join-ul** este operația de regăsire a datelor din două sau mai multe tabele, pe baza **valorilor comune ale unor coloane**. De obicei, aceste coloane reprezintă **cheia primară**, respectiv **cheia externă a tabelelor**.

Reamintim că, pentru a realiza un **join între n tabele**, va fi nevoie de **cel puțin n – 1 condiții de join**. Dacă una dintre condițiile de join nu este specificată, va fi generat produsul cartezian al tabelelor respective.

##### Tipuri de join :

- **Inner join (equijoin, join simplu)** – corespunde situației în care valorile de pe coloanele ce apar în condiția de *join* trebuie să fie egale.
- **Nonequijoin** – condiția de *join* conține alți operatori decât operatorul de egalitate.
- **Left / Right Outer join** – un *outer join* este utilizat pentru a obține în rezultat și înregistrările care nu satisfac condiția de *join*. O operație de *join* implementată cu ajutorul unei condiții specificate în clauza *WHERE* devine *outer-join* adăugând semnul plus inclus între paranteze **(+)** în acea parte a condiției de *join* care este **deficitară în informație**. Efectul acestui operator este de a uni liniile tabelului care nu este deficitar în informație, cărora nu le corespunde nici o linie în celălalt tabel,

cu o linie cu valori *null*. Operatorul (+) poate fi plasat în orice parte a condiției de *join*, dar nu în ambele părți.

**Obs:** O condiție care presupune un *outer join* nu poate utiliza operatorul *IN* și nu poate fi legată de altă condiție prin operatorul *OR*.

- **Full outer join** – left outer join + right outer join

**Self join** – un caz particular al operației de join, ce apare atunci când este realizat *join*-ul unui tabel cu el însuși. În ce situație concretă (relativ la modelul nostru) apărea această operație?

#### Join introdus în standardul SQL3 (SQL:1999):

Pentru *join*, sistemul *Oracle* oferă și o sintaxă specifică, în conformitate cu standardul *SQL3 (SQL:1999)*. Această sintaxă nu aduce beneficii, în privința performanței, față de *join*-urile care folosesc sintaxa utilizată anterior.

Tipurile de *join* conforme cu *SQL3* sunt definite prin cuvintele cheie **CROSS JOIN** (pentru produs cartezian), **NATURAL JOIN**, **JOIN**, **LEFT | RIGHT | FULL OUTER JOIN**, clauzele **USING** și **ON**.

Sintaxa corespunzătoare acestor tipuri de *join* (în standardul *SQL3*) este următoarea:

```
SELECT tabel_1.nume_colonă, tabel_2.nume_colonă
FROM tabel_1
[CROSS JOIN tabel_2]
/ [NATURAL JOIN tabel_2]
/ [JOIN tabel_2 USING (nume_colonă) ]
/ [JOIN tabel_2 ON (conditie) ]
/ [{LEFT | RIGHT | FULL} [OUTER] JOIN tabel_2
  { USING (nume_colonă) | ON (tabel_1.nume_colonă =
    tabel_2.nume_colonă) }];
```

- **NATURAL JOIN** presupune existența unor coloane având același nume în ambele tabele. Clauza determină selectarea liniilor din cele două tabele, care au valori egale în aceste coloane. Dacă tipurile de date ale coloanelor cu nume identice sunt diferite, va fi returnată o eroare.

Coloanele având același nume în cele două tabele trebuie să nu fie precedate de numele sau *alias*-ul tabelului corespunzător.

- **JOIN tabel\_2 USING (nume\_colonă)** efectuează un *equijoin* pe baza coloanei cu numele specificat în sintaxă. Această clauză este utilă dacă există mai multe coloane având același nume, dar operația de *join* nu trebuie realizată după toate aceste coloane. Coloanele referite în clauza *USING* trebuie să nu conțină

calificatori (să nu fie precedate de nume de tabele sau *alias-uri*) în nici o apariție a lor în instrucțiunea SQL. Clauzele *NATURAL JOIN* și *USING* nu pot coexista în aceeași instrucțiune SQL.

- ***JOIN tabel\_2 ON (conditie)*** efectuează un *join* pe baza condiției exprimate în clauza *ON*. Această clauză permite specificarea separată a condițiilor de *join*, respectiv a celor de căutare sau filtrare (din clauza *WHERE*).

În cazul operației *equijoin*, *conditie* are forma următoare :

*tabel\_1.nume\_colonă = tabel\_2.nume\_colonă*

- **{LEFT | RIGHT | FULL} [OUTER] JOIN tabel\_2 {USING (nume\_colonă) | ON (tabel\_1.nume\_colonă = tabel\_2.nume\_colonă)}** efectuează *outer join* la stânga, dreapta, respectiv în ambele părți pe baza egalității coloanei specificate în clauza *USING*, respectiv a condiției exprimate în clauza *ON*.

Un *join* care returnează rezultatele unui *inner join*, dar și cele ale *outer join-urilor* la stânga și la dreapta se numește *full outer join*.

### III. [Operatori pe mulțimi]

Operatorii pe mulțimi combină rezultatele obținute din două sau mai multe interogări. Cererile care conțin operatori pe mulțimi se numesc *cereri compuse*. Există patru operatori pe mulțimi: ***UNION***, ***UNION ALL***, ***INTERSECT*** și ***MINUS***.

Toți operatorii pe mulțimi au aceeași precedență. Dacă o instrucțiune SQL conține mai mulți operatori pe mulțimi, server-ul *Oracle* evaluează cererea de la stânga la dreapta (sau de sus în jos). Pentru a schimba această ordine de evaluare, se pot utiliza paranteze.

- Operatorul ***UNION*** returnează toate liniile selectate de două cereri, eliminând duplicatele. Acest operator nu ignoră valorile *null*.
- Operatorul ***UNION ALL*** returnează toate liniile selectate de două cereri, fără a elmina duplicatele. Precizările făcute asupra operatorului *UNION* sunt valabile și în cazul operatorului *UNION ALL*. În cererile asupra cărora se aplică *UNION ALL* nu poate fi utilizat cuvântul cheie *DISTINCT*.
- Operatorul ***INTERSECT*** returnează toate liniile comune cererilor asupra cărora se aplică. Acest operator nu ignoră valorile *null*.
- Operatorul ***MINUS*** determină liniile returnate de prima cerere care nu apar în rezultatul celei de-a doua cereri. Pentru ca operatorul *MINUS* să funcționeze, este necesar ca toate coloanele din clauza *WHERE* să se afle și în clauza *SELECT*.

#### *Observații:*

- În mod implicit, pentru toți operatorii cu excepția lui *UNION ALL*, rezultatul este ordonat crescător după valorile primei coloane din clauza *SELECT*.

- Pentru o cerere care utilizează operatori pe mulțimi, cu excepția lui *UNION ALL*, server-ul *Oracle* elimină liniile dupicate.
- În instrucțiunile *SELECT* asupra căror se aplică operatori pe mulțimi, coloanele selectate trebuie să corespundă ca număr și tip de date. Nu este necesar ca numele coloanelor să fie identice. Numele coloanelor din rezultat sunt determinate de numele care apar în clauza *SELECT* a primei cereri.

#### IV. [Functii]

Completări: utilizarea alternativă a funcției DECODE și a structurii CASE; din nou NVL și NVL2; COALESCE; NULLIF

Reamintim:

- *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 și forma:

*CASE*

```
WHEN expr_logica_1 THEN val_2
[WHEN expr_logica_3 THEN val_4
...]
[ELSE default]
```

*END*

#### V. [Exerciții - join]

1. Scrieți o cerere prin care se afișează numele, luna (în litere) și anul angajării pentru toți salariații din același departament cu Gates, al căror nume conține litera "a". Se va exclude Gates. Se vor da 2 soluții pentru determinarea apariției literei "a" în nume. De

asemenea, pentru una dintre metode se va da și varianta *join*-ului conform standardului SQL3.

**Soluție:** Operația cerută nu se bazează pe o relație directă (care ar fi generat o egalitate între o cheie externă și cheia primară corespunzătoare). În schimb, relația este una indirectă, ce ar putea fi exprimată ca “ANGAJAT este coleg cu ANGAJAT”. Reamintim, din cursul de proiectare a bazelor de date, că în modelele de date nu se includ astfel de relații, ce derivă din altele. În cazul nostru, faptul că doi angajați lucrează în același departament determină relația de colegialitate dintre ei.

```
SELECT e1.last_name, TO_CHAR(e1.hire_date, 'MONTH') AS "Luna",
EXTRACT(YEAR FROM e1.hire_date) as "An"
FROM employees e1,employees e2
WHERE e1.department_id = e2.department_id
    AND LOWER(e1.last_name) LIKE '%a%'
    AND LOWER(e1.last_name) != 'gates'
    AND LOWER(e2.last_name) = 'gates';
```

```
SELECT e1.last_name, TO_CHAR(e1.hire_date, 'MONTH') AS "Luna",
EXTRACT(YEAR FROM e1.hire_date) as "An"
FROM employees e1
JOIN employees e2
ON (e1.department_id = e2.department_id)
WHERE LOWER(e1.last_name) LIKE '%a%'
    AND LOWER(e1.last_name) != 'gates'
    AND LOWER(e2.last_name) = 'gates';
```

```
SELECT e1.last_name, TO_CHAR(e1.hire_date, 'MONTH') AS "Luna",
EXTRACT(YEAR FROM e1.hire_date) as "An"
FROM employees e1
JOIN employees e2
ON (e1.department_id = e2.department_id)
WHERE INSTR(LOWER(e1.last_name), 'a') != 0
    AND LOWER(e1.last_name) != 'gates'
    AND LOWER(e2.last_name) = 'gates';
```

2. Să se afișeze codul și numele angajaților care lucrează în același departament cu cel puțin un angajat al cărui nume conține litera “t”. Se vor afișa, de asemenea, codul și numele departamentului respectiv. Rezultatul va fi ordonat alfabetic după nume.
- Dați și soluția care utilizează sintaxa specifică Oracle (anterioară SQL3) pentru join.

**Soluție:**

```
SELECT DISTINCT e1.employee_id, e1.last_name, e1.department_id,
d.department_name
FROM employees e1, employees e2, departments d
WHERE e1.department_id = e2.department_id
    AND e1.department_id = d.department_id
    AND LOWER(e2.last_name) LIKE '%t%'
ORDER BY e1.last_name;
```

```
SELECT UNIQUE e1.employee_id, e1.last_name, e1.department_id,
d.department_name
```

```

FROM employees e1
JOIN employees e2 ON (e1.department_id = e2.department_id)
JOIN departments d ON (e1.department_id = d.department_id)
WHERE LOWER(e2.last_name) LIKE '%t%'
ORDER BY e1.last_name;

```

- 3.** Să se afișeze numele, salariul, titlul job-ului, orașul și țara în care lucrează angajații conduși direct de King.

Dați două metode de rezolvare a acestui exercițiu.

**Soluție:** Tratăm inclusiv cazul în care există subalterni direcți ai lui King care nu sunt asignați unui departament (coloana *department\_id* este *null*). Un astfel de angajat nu va apărea în rezultat, deoarece îl “pierdem” atunci când se realizează *join-ul* cu tabelul DEPARTMENTS.

Asfel, această variantă de cerere:

```

SELECT e.last_name, e.salary, j.job_title, l.city,
       c.country_name
FROM employees e
JOIN employees m ON (e.manager_id = m.employee_id)
JOIN jobs j ON (e.job_id = j.job_id)
JOIN departments d ON (e.department_id = d.department_id)
JOIN locations l ON (l.location_id = d.location_id)
JOIN countries c ON (c.country_id = l.country_id)
WHERE m.last_name = 'King';

```

va putea returna un rezultat diferit de următoarea (pe care o considerăm **soluția corectă** a acestui exercițiu):

```

SELECT e.last_name, e.salary, j.job_title, l.city,
       c.country_name
FROM employees e
JOIN employees m ON (e.manager_id = m.employee_id)
JOIN jobs j ON (e.job_id = j.job_id)
LEFT JOIN departments d ON (e.department_id = d.department_id)
LEFT JOIN locations l ON (l.location_id = d.location_id)
LEFT JOIN countries c ON (c.country_id = l.country_id)
WHERE m.last_name = 'King';

```

#### Observații:

- Nu am pus problema pierderii de rezultate la *join-ul* cu tabelul JOBS, deoarece atributul *job\_id* are constrângerea NOT NULL. Ne putem convinge de acest lucru cu ajutorul comenzi “describe jobs;”. Prin urmare, coloana *job\_id* din EMPLOYEES nu va fi niciodată *null*, deci nu se va pierde nicio înregistrare în acel *join*.
- De ce sunt necesare ultimele două operații LEFT JOIN de mai sus ? Pentru a păstra linia “câștigată” la primul LEFT JOIN, operația se propagă la toate *join-urile* care îi urmează. Altfel, doar cu primul LEFT JOIN în cerere, rezultatul va fi același cu cel de la varianta anterioară (cea incorectă).

**Temă:** A doua metodă (transformarea cererii astfel încât operațiile de join să fie specificate în clauza WHERE)

4. Să se afișeze codul departamentului, numele departamentului, numele și job-ul tuturor angajaților din departamentele al căror nume conține sirul ‘ti’. De asemenea, se va lista salariul angajaților, în formatul “\$99,999.00”. Rezultatul se va ordona alfabetic după numele departamentului, și în cadrul acestuia, după numele angajaților.

**Soluție:**

```
SELECT d.department_id, d.department_name, e.last_name,
       j.job_title, TO_CHAR(e.salary, '$99,999.00') Salariu
FROM departments d
JOIN employees e ON (e.department_id = d.department_id)
JOIN jobs j ON (e.job_id = j.job_id)
WHERE LOWER(d.department_name) LIKE '%ti%'
ORDER BY d.department_name, e.last_name;
```

De ce nu mai folosim OUTER JOIN? Fiindcă avem o condiție (LIKE) legată de coloana *department\_name*, nu ar avea sens obținerea înregistrărilor pentru care *department\_id* este *null* (în tabelul EMPLOYEES). Aceste linii ar conține valoarea *null* și pentru *department\_name*, deci cu siguranță nu ar îndeplini condiția cerută.

5. Să se afișeze numele angajaților, numărul departamentului, numele departamentului, orașul și job-ul tuturor salariaților al căror departament este localizat în Oxford.

**Soluție:**

```
SELECT e.last_name, department_id, department_name, city ,
       job_title
FROM employees e
JOIN departments d USING(department_id)
JOIN locations l USING(location_id)
JOIN jobs j USING(job_id)
WHERE LOWER(l.city) = 'oxford';
```

Motivul pentru care nu folosim OUTER JOIN nici la această problemă este similar cu cel de la problema anterioară: deoarece specificăm o condiție de egalitate pentru coloana *city* (din tabelul LOCATIONS), nu are sens să regăsim în rezultat linii pentru care *department\_id* este *null*, deoarece acestea ar conține valori *null* și pe coloanele *location\_id* și *city*.

6. Să se modifice cererea de la problema 2 astfel încât să afișeze codul, numele și salariul tuturor angajaților care câștigă mai mult decât salariul mediu pentru job-ul corespunzător și lucrează într-un departament cu cel puțin unul dintre angajații al căror nume conține litera “t”. Vom considera salariul mediu al unui job ca fiind egal cu media aritmetică a limitelor sale admise (specificate în coloanele *min\_salary*, *max\_salary* din tabelul JOBS).

**Soluție:**

```
SELECT UNIQUE e1.employee_id, e1.last_name, e1.salary
FROM employees e1
JOIN employees e2 ON (e2.department_id = e1.department_id)
```

```

JOIN jobs j ON (j.job_id = e1.job_id)
WHERE LOWER(e2.last_name) LIKE '%t%' --INSTR(LOWER(e2.last_name), 't') != 0
AND e1.salary > (j.min_salary + j.max_salary) / 2
ORDER BY e1.last_name;

```

7. Să se afișeze numele salariaților și numele departamentelor în care lucrează. Se vor afișa și salariații care nu au asociat un departament. (outer join, 2 variante).

**Soluție:**

```

SELECT e.last_name, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id (+);

```

```

SELECT e.last_name, d.department_name
FROM employees e
LEFT OUTER JOIN departments d ON (e.department_id =
d.department_id);

```

8. Să se afișeze numele departamentelor și numele salariaților care lucrează în ele. Se vor afișa și departamentele care nu au salariați. (outer join, 2 variante)

```

SELECT e.last_name, d.department_name
FROM employees e
RIGHT OUTER JOIN departments d ON (e.department_id =
d.department_id);

```

9. Cum se poate implementa *full outer join*?

**Observație:** *Full outer join* se poate realiza fie prin reuniunea rezultatelor lui *right outer join* și *left outer join*, fie utilizând sintaxa specifică standardului SQL3.

**Soluție:**

```

SELECT e.last_name, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id (+)
UNION
SELECT e.last_name, d.department_name
FROM employees e, departments d
WHERE e.department_id (+)= d.department_id;

```

```

SELECT e.last_name, d.department_name
FROM employees e
FULL OUTER JOIN departments d ON (e.department_id =
d.department_id);

```

Ce observați, comparând rezultatele celor două metode de mai sus? Din cauză că există perechi egale de (*last\_name, department\_name*) și deoarece operatorul UNION elimină duplicatele, rezultatul primei cereri conține (în mod eronat) mai puține linii. Pentru ca doi angajați având același nume și departament să fie considerați diferiți, în prima metodă de

mai sus introducem o coloană care să le asigure unicitatea (chiar cheia primara – *employee\_id*) :

```
SELECT e.employee_id, e.last_name, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id (+)
UNION
SELECT e.employee_id, e.last_name, d.department_name
FROM employees e, departments d
WHERE e.department_id (+)= d.department_id;
```

## VI. [Exerciții - operatori pe mulțimi]

- 10.** Se cer codurile departamentelor al căror nume conține sirul “re” sau în care lucrează angajați având codul job-ului “SA\_REP”.

Cum este ordonat rezultatul?

Ce se întâmplă dacă înlocuim *UNION* cu *UNION ALL* în comanda precedentă?

**Soluție:**

```
SELECT D.DEPARTMENT_ID
FROM DEPARTMENTS D
WHERE LOWER(D.DEPARTMENT_NAME) LIKE '%re%'
UNION
SELECT DEPARTMENT_ID
FROM EMPLOYEES E
WHERE E.JOB_ID='SA_REP';
```

- 11.** Să se obțină codurile departamentelor în care nu lucrează nimeni (nu este introdus nici un salariat în tabelul *employees*). Se cer două soluții (*MINUS*, *NOT IN*).

**Observație:** Operatorii pe mulțimi pot fi utilizati în subcereri. Coloanele care apar în clauza *WHERE* a interogării trebuie să corespundă, ca număr și tip de date, celor din clauza *SELECT* a subcererii.

Comentați necesitatea tratării valorilor *null* în varianta utilizării operatorului *NOT IN*.

**Soluție:**

```
SELECT d.department_id
FROM departments d
MINUS
SELECT UNIQUE department_id
FROM employees;
```

```
SELECT department_id
FROM departments
WHERE department_id NOT IN
(
    SELECT d.department_id
    FROM departments d
    JOIN employees e ON(d.department_id=e.department_id)
);
```

A doua varianta utilizeaza o subcerere necorelata. Aceste subcereri vor fi prezentate in continuarea laboratorului.

- 12.** Se cer codurile departamentelor al căror nume conține sirul “re” și în care lucrează angajați având codul job-ului “HR\_REP”.

**Soluție:**

```
SELECT d.department_id
FROM departments d
WHERE LOWER(d.department_name) LIKE '%re%'
INTERSECT
SELECT e.department_id
FROM employees e
WHERE e.job_id = 'HR_REP';
```

- 13.** Să se determine codul angajaților, codul job-urilor și numele celor al căror salariu este mai mare decât 3000 sau este egal cu media dintre salariul minim și cel maxim pentru job-ul respectiv.

**Soluție:**

```
SELECT e.employee_id, e.job_id, e.last_name
FROM employees e
WHERE (e.salary > 3000)
UNION
SELECT e.employee_id, e.job_id, e.last_name
FROM employees e
JOIN jobs j ON (j.job_id = e.job_id)
WHERE e.salary = (j.min_salary + j.max_salary) / 2;
```

Cererea anterioară utilizează un operator pe mulțimi. Evident, există și această variantă mai simplă:

```
SELECT e.employee_id, e.job_id, e.last_name
FROM employees e
JOIN jobs j ON (e.job_id = j.job_id)
WHERE e.salary > 3000 OR e.salary = (j.min_salary + j.max_salary)
/ 2;
```

## VII. [Exerciții - funcții]

- 14.** Să se afișeze informații despre departamente, în formatul următor: „Departamentul *<department\_name>* este condus de {*<manager\_id>* | *nimeni*} și {are salariați | nu are salariați}”. (Se vor utiliza *NVL*, *NVL2*, *outer join*, *CASE*). (**Temă**)
- 15.** Să se afișeze numele, prenumele angajaților și lungimea numelui pentru înregistrările în care aceasta este diferită de lungimea prenumelui. (Se va utiliza *NULLIF*). (**Temă**)
- 16.** 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. (*DECODE* și cele 2 forme ale lui *CASE*). (**Temă**)

# Baze de date - Anul 1

## Laborator 4

---

### Subcereri nesincronizate (necorelate). Gruparea datelor (1)

#### I. [Obiective]

Prezentarea conceptului de subcerere. Unde pot fi folosite subcererile? Clasificarea subcererilor.

Introducere în gruparea datelor.

#### II. [Subcereri]

O **subcerere** este o comandă *SELECT* încapsulată într-o clauză a altrei instrucțiuni *SQL*, numită instrucțiune „părinte”. Utilizând subcereri, se pot construi interogări complexe pe baza unor instrucțiuni simple. Subcererile mai sunt numite instrucțiuni *SELECT* imbicate sau interioare.

Subcererea returnează o valoare care este utilizată de către instrucțiunea „părinte”. Utilizarea unei subcereri este echivalentă cu efectuarea a două cereri secvențiale și utilizarea rezultatului cererii interne ca valoare de căutare în cererea externă (principală).

În general, subcererile pot apărea în clauzele *SELECT*, *FROM*, *WHERE*, *HAVING* ale comenzi *SELECT*.

Subcererile sunt de 2 tipuri :

➤ **Necorelate (nesincronizate)**, de forma :

```
SELECT lista_select  
FROM nume_tabel  
WHERE expresie operator (SELECT lista_select  
                         FROM nume_tabel);
```

- cererea internă este executată prima și determină o valoare (sau o mulțime de valori);
- cererea externă se execută o singură dată, utilizând valorile returnate de cererea internă.

➤ **Corelate (sincronizate)**, de forma :

```
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);
```

- 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.

**Observații:** operator poate fi:

- *single-row operator* ( $>$ ,  $=$ ,  $\geq$ ,  $<$ ,  $\neq$ ,  $\leq$ ), 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.

Operatorul *NOT* poate fi utilizat în combinație cu *IN*, *ANY* și *ALL*.

### III. [Funcții grup și clauza GROUP BY]

Clauza *GROUP BY* este utilizată pentru **a diviza liniile unui tabel în grupuri**. Pentru a returna informația corespunzătoare fiecărui astfel de grup, pot fi utilizate funcțiile agregat. Aceste funcții pot apărea în clauzele:

- *SELECT*
- *ORDER BY*
- *HAVING*.

Server-ul *Oracle* aplică aceste funcții fiecărui grup de linii și returnează **un singur rezultat** pentru fiecare mulțime.

Dintre funcțiile grup definite în sistemul *Oracle*, se pot enumera: *AVG*, *SUM*, *MAX*, *MIN*, *COUNT*, *STDDEV*, *VARIANCE* etc. Tipurile de date ale argumentelor funcțiilor grup pot fi *CHAR*, *VARCHAR2*, *NUMBER* sau *DATE*.

- Funcțiile *AVG*, *SUM*, *STDDEV* și *VARIANCE* operează numai asupra valorilor numerice.
- Funcțiile *MAX* și *MIN* pot opera asupra valorilor numerice, caracter sau dată calendaristică.
- Absența clauzei *GROUP BY* conduce la aplicarea funcției grup pe mulțimea tuturor liniilor tabelului.
- Toate funcțiile grup, cu excepția lui *COUNT(\*)*, ignoră valorile *null*. *COUNT(expresie)* returnează numărul de linii pentru care expresia dată nu are valoarea *null*. Funcția *COUNT* returnează un număr mai mare sau egal cu zero și nu întoarce niciodată valoarea *null*.

Expresiile din clauza *SELECT* a unei cereri care conține opțiunea *GROUP BY* trebuie să reprezinte **o proprietate unică de grup**, adică fie un atribut de grupare, fie o funcție de agregare aplicată tuplurilor unui grup, fie o expresie formată pe baza primelor două. Toate expresiile din clauza *SELECT*, cu excepția funcțiilor de agregare, se trec în clauza *GROUP BY* (unde pot apărea cel mult 255 expresii).

#### IV. [Clauza HAVING]

- Clauza *HAVING* a comenzi *SELECT* permite restricționarea grupurilor de linii returnate, la cele care îndeplinesc o anumită condiție.
- Dacă această clauză este folosită în absența unei clauze *GROUP BY*, aceasta presupune că **gruparea se aplică întregului tabel**, deci este returnată o singură linie, care este reținută în rezultat doar dacă este îndeplinită condiția din clauza *HAVING*.

#### V. [Exercitii - subcereri necorelate]

1. Folosind subcereri, să se afișeze numele și data angajării pentru salariații care au fost angajați după Gates.

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date > (SELECT hire_date
                     FROM employees
                     WHERE INITCAP(last_name)='Gates');
```

2. Folosind subcereri, scrieți o cerere pentru a afișa numele și salariul pentru toți colegii (din același departament) lui Gates. Se va exclude Gates.

```
SELECT last_name, salary
FROM employees
WHERE department_id IN (SELECT department_id
                         FROM employees
                         WHERE LOWER(last_name)='gates')
AND LOWER(last_name) <> 'gates';
```

În care caz nu se poate înlocui "=" cu "IN"??

3. Folosind subcereri, să se afișeze numele și salariul angajaților conduși direct de președintele companiei (acesta este considerat angajatul care nu are manager).
4. Scrieți o cerere pentru a afișa numele, codul departamentului și salariul angajaților al căror cod de departament și salariu coincid cu codul departamentului și salariul unui angajat care câștigă comision.

**Soluție:**

```
SELECT last_name, department_id, salary
```

```

FROM employees
WHERE (department_id, salary) IN
(
    SELECT department_id, salary
    FROM employees
    WHERE commission_pct IS NOT NULL
);

```

5. Rezolvați problema 6 din laboratorul precedent utilizând subcereri: să se afișeze codul, numele și salariul tuturor angajaților care câștigă mai mult decât salariul mediu pentru job-ul corespunzător și lucrează într-un departament cu cel puțin unul dintre angajații al căror nume conține litera "t". Vom considera salariul mediu al unui job ca fiind egal cu media aritmetică a limitelor sale admise (specificate în coloanele *min\_salary*, *max\_salary* din tabelul JOBS).

**Soluție:**

```

SELECT e.employee_id, e.last_name, e.salary
FROM employees e
WHERE e.salary >
(
    SELECT (j.min_salary+j.max_salary)/2
    FROM jobs j
    WHERE j.job_id=e.job_id
)
AND e.job_id IN
(
    SELECT job_id
    FROM employees m
    WHERE e.department_id=m.department_id
    AND LOWER(m.last_name) LIKE '%t%'
);

```

6. Scrieți o cerere pentru a afișa angajații care câștigă mai mult decât oricare funcționar (job-ul conține sirul "CLERK"). Sortați rezultatele după salariu, în ordine descrescătoare.

Ce rezultat este returnat dacă se înlocuiește "ALL" cu "ANY"?

**Soluție:**

```

SELECT *
FROM employees e
WHERE salary > ALL
(
    SELECT salary
    FROM employees
    WHERE job_id LIKE '%CLERK'
);

```

7. Scrieți o cerere pentru a afișa numele, numele departamentului și salariul angajaților care nu câștigă comision, dar al căror șef direct câștigă comision.
8. Să se afișeze numele, departamentul, salariul și job-ul tuturor angajaților al căror salariu și comision coincid cu salariul și comisionul unui angajat din Oxford.

9. Să se afișeze numele angajaților, codul departamentului și codul job-ului salariaților al căror departament se află în Toronto.

## VI. [Exercitii – gruparea datelor]

10. a) Functiile grup includ valorile *NULL* in calcule?  
b) Care este deosebirea dintre clauzele *WHERE* și *HAVING*?
11. Să se afișeze cel mai mare salariu, cel mai mic salariu, suma și media salariilor tuturor angajaților. Etichetați coloanele Maxim, Minim, Suma, respectiv Media. Să se rotunjească rezultatele.
12. Să se afișeze minimul, maximul, suma și media salariilor pentru fiecare job.
13. Să se afișeze numărul de angajați pentru fiecare job.
14. Să se determine numărul de angajați care sunt şefi. Etichetați coloana “Nr. manageri”.  
**Observație:** Este necesar cuvântul cheie *DISTINCT*. Ce obținem dacă îl omitem?
15. Să se afișeze diferența dintre cel mai mare și cel mai mic salariu mediu pe departamente. Etichetați coloana “Diferenta”.
16. Scrieți o cerere pentru a se afișa numele departamentului, locația, numărul de angajați și salariul mediu pentru angajații din acel departament. Coloanele vor fi etichetate corespunzător.  
**Observație:** În clauza *GROUP BY* se trec obligatoriu toate coloanele prezente în clauza *SELECT*, care nu sunt argument al funcțiilor grup (a se vedea ultima observație de la punctul I).
17. Să se afișeze codul și numele angajaților care câștigă mai mult decât salariul mediu din firmă. Se va sorta rezultatul în ordine descrescătoare a salariilor.
18. Pentru fiecare șef, să se afișeze codul său și salariul celui mai puțin platit subordonat al său. Se vor exclude cei pentru care codul managerului nu este cunoscut. De asemenea, se vor exclude grupurile în care salariul minim este mai mic de 1000\$. Sortați rezultatul în ordine descrescătoare a salariilor.
19. Pentru departamentele în care salariul maxim depășește 3000\$, să se obțină codul, numele acestor departamente și salariul maxim pe departament.
20. Care este salariul mediu minim al job-urilor existente? Salariul mediu al unui job va fi considerat drept media aritmetică a salariilor celor care îl practică.
21. Să se afișeze codul, numele departamentului și suma salariilor pe departamente.
22. Să se afișeze maximul salariilor medii pe departamente.

**23.** Să se obțină codul, titlul și salariul mediu al job-ului pentru care salariul mediu este minim.

**24.** Să se afișeze salariul mediu din firmă doar dacă acesta este mai mare decât 2500.  
(clauza *HAVING* fără *GROUP BY*)

# Baze de date - Anul 1

## Laborator 5

---

**Gruparea datelor (2). Funcția *DECODE* și expresia *CASE*. Subcereri nesincronizate în clauza *FROM*.**

### I. [Obiective]

- Continuarea exercițiilor referitoare la gruparea datelor.
- Utilizarea *DECODE/CASE* în cadrul exercițiilor de tip cerere cu rezultat bidimensional.
- În laboratorul anterior am lucrat cu subcereri nesincronizate în clauza *WHERE*, însă am amintit faptul că acestea pot apărea și în alte clauze ale comenzi *SELECT (SELECT, FROM, HAVING)*. În acest laborator, vom utiliza subcerile nesincronizate în clauza *FROM*.

### II. [Exerciții – gruparea datelor]

1. Să se afișeze codurile departamentelor, codurile job-urilor și o coloană reprezentând suma salariilor pe departamente și, în cadrul acestora, pe job-uri.
2. Modificați cererea anterioară astfel încât rezultatul să includă numele departamentelor și titlurile job-urilor.
3. Să se afișeze numele departamentului și cel mai mic salariu din departamentul având cel mai mare salariu mediu.
4. Să se afișeze codul, numele departamentului și numărul de angajați care lucrează în acel departament pentru:
  - a) departamentele în care lucrează mai puțin de 4 angajați;
  - b) departamentul care are numărul maxim de angajați.
5. Să se afișeze salariații care au fost angajați în aceeași zi a lunii (ca număr al zilei în lună) în care cei mai mulți dintre salariați au fost angajați.
6. Să se obțină numărul departamentelor care au cel puțin 15 angajați.
7. Să se obțină codul departamentelor și suma salariilor angajaților care lucrează în acestea, în ordine crescătoare. Se consideră departamentele care au mai mult de 10 angajați și al căror cod este diferit de 30.

- 8.** Să se afișeze codul, numele departamentului, numărul de angajați și salariul mediu din departamentul respectiv, împreună cu numele, salariul și jobul angajaților din acel departament. Se vor afișa și departamentele fără angajați.

**Observație:** Cerința implică alăturarea valorilor la nivel de linie și a celor la nivel de grup. Soluția cerută aici are doar join-uri și grupări de date. Exercițiul va fi reluat ulterior, pentru a fi rezolvat cu alte metode.

- 9.** Să se obțină, pentru departamentele având codul > 80, salariul total pentru fiecare job din cadrul departamentului. Se vor afișa orașul, numele departamentului, jobul și suma salariilor. Se vor eticheta coloanele corespunzător.

- 10.** Care sunt angajații (cod, nume) care au mai avut cel puțin două joburi?

- 11.** Să se calculeze comisionul mediu din firmă, luând în considerare toate liniile din tabel.

**Observație:** Funcțiile grup ignoră valorile *null*. Prin urmare, instrucțiunea:

```
SELECT AVG(commission_pct)
FROM      employees;
```

va returna media valorilor pe baza liniilor din tabel pentru care există o valoare diferită de *null*. Astfel, reiese că suma valorilor se împarte la numărul de valori diferite de *null*. Calculul mediei pe baza tuturor liniilor din tabel se poate realiza utilizând funcțiile *NVL*, *NVL2* sau *COALESCE*:

```
SELECT AVG(NVL(commission_pct, 0))
FROM      employees;
```

O altă variantă este dată de o cerere de forma:

```
SELECT SUM(commission_pct) /COUNT(*)
FROM employees;
```

### III. [Exerciții – DECODE / CASE]

- 12.** Să se afișeze denumirea job-ului, salariul total pentru job-ul respectiv în toate departamentele și salariul total pentru job-ul respectiv în fiecare dintre departamentele 30, 50, 80. Se vor eticheta coloanele corespunzător. Rezultatul va apărea sub forma de mai jos:

Job	Dep30	Dep50	Dep80	Total
.....				
.....				

- 13.** Să se afișeze numărul total de angajați și, din acest total, numărul celor care au fost angajați în 1997, 1998, 1999 și 2000. Denumiți capetele de tabel în mod corespunzător.

#### IV. [Exerciții – subcereri nesincronizate în clauza FROM]

Subcererile pot apărea în clauzele *SELECT*, *WHERE*, *FROM*, *HAVING* ale unei cereri. **O subcerere care apare în clauza *FROM* se mai numește view (vizualizare) in-line.**

Ce obiecte au apărut până acum în clauza *FROM*? – Tabelele = obiectele care stochează efectiv datele. Pe lângă acestea, putem specifica în *FROM* tabelele virtuale (vizualizările).

- 14.** Să se afișeze codul, numele departamentului și suma salariilor pe departamente.

```
SELECT d.department_id, department_name, a.suma
FROM departments d, (SELECT department_id ,SUM(salary) suma
                      FROM      employees
                      GROUP BY department_id) a
WHERE d.department_id = a.department_id;
```

- 15.** Utilizând subcereri, să se afișeze titlul job-ului, salariul mediu corespunzător și diferența dintre media limitelor (*min\_salary*, *max\_salary*) și media reală.

- 16.** Modificați cererea anterioară, pentru a determina și listarea numărului de angajați corespunzători fiecărui job.

- 17.** Pentru fiecare departament, să se afișeze denumirea acestuia, precum și numele și salariul celor mai slab plătiți angajați din cadrul său.

# Baze de date - Anul 1

## Laborator 6

---

### Subcereri sincronizate. Clauza *WITH*. Analiza *top-n*.

#### I. [Obiective]

- Lucrul cu subcereri sincronizate (corelate).
- Definirea de blocuri de cerere înaintea instrucțiunii *SELECT* propriu-zise, cu scopul simplificării acesteia.
- Aflarea primelor *n* înregistrări ordonate după un anumit criteriu.

#### II. [Subcereri corelate (sincronizate)]

O subcerere (cerere imbricată sau încubărită) corelată poate fi regăsită în clauza *WHERE*, având 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_2 = extern.expresie_1);
```

Modul de execuție :

- 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.

**Observație:** 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.

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

- *SELECT* (vezi mai jos)
- *FROM* (vezi laboratorul 5)
- *WHERE*
- *HAVING* (vezi laboratorul 4)

## Operatorul *EXISTS*

- În instrucțiunile *SELECT* imbricate, este permisă utilizarea oricărui operator logic.
- Pentru a testa dacă valoarea recuperată de cererea externă există în multimea 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.

### III. [Exerciții – subcereri sincronizate]

1. a) Să se afișeze informații despre angajații al căror salariu depășește 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, afișându-se și numele departamentului, media salariilor acestuia și numărul de angajați. Se cer 2 soluții (cu subcerere nesincronizată în clauza *FROM* și cu subcerere sincronizată în clauza *SELECT*).

2. Să se afișeze numele și salariul angajaților al căror salariu este mai mare decât salariile medii din toate departamentele. Se cer 2 variante de rezolvare: cu operatorul *ALL* și cu funcția *MAX*.
3. Să se afișeze numele și salariul celor mai puțin plătiți angajați din fiecare departament (3 soluții: cu și fără sincronizare, subcerere în clauza *FROM*).
4. Pentru fiecare departament, să se obțină denumirea acestuia și numele salariatului având cea mai mare vechime din departament. Să se ordoneze rezultatul după numele departamentului.
5. Să se obțină numele salariaților care lucrează într-un departament în care există cel puțin un 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) );
```

**Observație:** Deoarece nu este necesar ca instrucțiunea *SELECT* interioară (subcererea) să returneze o anumită valoare, se poate selecta o constantă ('x', 1 etc.). De altfel, din punct de vedere al performanței, selectarea unei constante este mai eficientă decât selectarea unei coloane, nemaifiind necesară accesarea datei respective.

- Să se obțină numele primilor 3 angajați având cele mai mari salarii. Rezultatul se va afișa în ordine crescătoare a salariilor.

**Soluția 1:** subcerere sincronizată

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

- Să se afișeze codul, numele și prenumele angajaților care au cel puțin doi subalterni.
- Să se determine locațiile în care se află cel puțin un departament.

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

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

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

#### 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 același 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.
- 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 angajatilor.

```
WITH val_dep AS (...),
val_medie AS (...)

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

**11.** Să se afișeze codul, prenumele și numele (pe aceeași coloană), codul *job*-ului și data angajării, ale subalternilor subordonaților direcți ai lui Steven King care au cea mai mare vechime.

#### V. [Analiza *top-n*]

Pentru aflarea primelor *n* rezultate ale unei cereri, este utilă funcția *ROWNUM*. Aceasta returnează numărul de ordine al unei linii în rezultat. Condiția ce utilizează această funcție trebuie aplicată asupra unei mulțimi ordonate de înregistrări. Cum obținem acea mulțime?

**12.** Să se determine primii 10 cei mai bine plătiți angajați.

**13.** Să se determine cele mai slab plătite 3 *job*-uri, din punct de vedere al mediei salariilor acestora.

#### Exercițiu suplimentar

**14.** Să se afișeze:

- suma salariilor, pentru *job*-urile al căror cod începe cu litera S;
- media generală a salariilor, pentru *job*-ul având salariul maxim;
- salariul minim pe *job*, pentru fiecare dintre celealte *job*-uri.

# Baze de date - Anul 1

## Laborator 7

---

### Operatorul *DIVISION. SQL\*Plus*

#### I. [Obiective]

- Implementarea operatorului *DIVISION* prin diferite metode.
- Prezentarea succintă a câtorva comenzi utile din mediul *SQL\*Plus*, prezente și în *SQL Developer*.
- Exerciții recapitulative.

#### II. [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 simulață 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

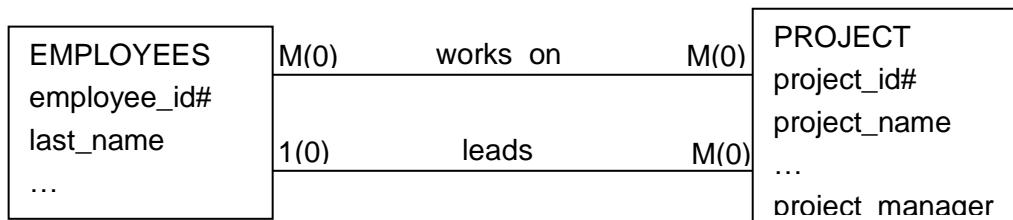
- *delivery\_date* este data la care proiectul este livrat efectiv
- *project\_manager* reprezintă codul managerului de proiect și este cheie externă. Pe cine referă această coloană ? Ce relație implementează această cheie externă?

2) *WORKS\_ON(project\_id#, employee\_id#, start\_date, end\_date)*

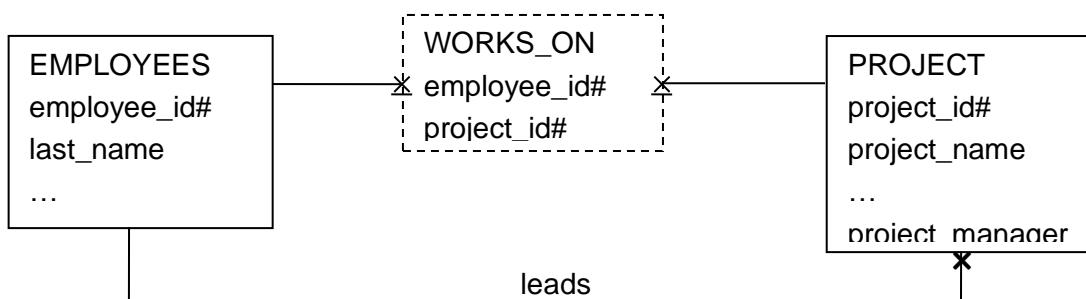
- cheia primară a relației este compusă din atributele *employee\_id* și *project\_id*.

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  a.employee_id#=b.employee_id#
            AND   a.project_id#=b.project_id#
            AND   b.budget=10000)
      )
  
```

```

        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 include B  $\Leftrightarrow$  B-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));

```

### III. [Exerciții – DIVISION + alte cereri]

1. Să se listeze informații despre angajații care au lucrat în toate proiectele demarate în primele 6 luni ale anului 2016. Implementați toate variantele.
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. Presupunem că pot exista valori *null* în coloana *job\_id* din tabelul *EMPLOYEES*.
4. Pentru fiecare țară, să se afișeze numele său și numărul de angajați din cadrul acesteia.
5. Să se listeze angajații (codul și numele acestora) care au lucrat pe cel puțin două proiecte nelivrate la termen.
6. Să se listeze codurile angajaților și codurile proiectelor pe care au lucrat. Listarea va cuprinde și angajații care nu au lucrat pe niciun proiect.
7. Să se afișeze angajații care lucrează în același departament cu cel puțin un manager de proiect.
8. Să se afișeze angajații care nu lucrează în același departament cu niciun manager de proiect.
9. Să se determine departamentele având media salariilor mai mare decât un număr dat.

**Observație:** 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;*

10. Se cer informații (nume, prenume, salariu, număr proiecte) despre managerii de proiect care au condus 2 proiecte.
11. Să se afișeze lista angajaților care au lucrat numai pe proiecte conduse de managerul de proiect având codul 102.
12. a) Să se obțină numele angajaților care au lucrat **cel puțin** pe aceleași proiecte ca angajatul având codul 200.

**Observație:** Incluziunea dintre 2 mulțimi se testează cu ajutorul proprietății „A inclus în B  $\Leftrightarrow A-B = \emptyset$ ”. 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 minus lista proiectelor pe care au lucrat acei angajați.

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

13. Să se obțină angajații care au lucrat pe aceleași proiecte ca angajatul având codul 200.

**Obs:** Egalitatea între două mulțimi se testează cu ajutorul proprietății „ $A=B \Leftrightarrow A-B=\emptyset$  și  $B-A=\emptyset$ ”.

**14.** Modelul HR conține un tabel numit *JOB\_GRADES*, care stocă grilele de salarizare ale companiei.

- a) Afisați structura și conținutul acestui tabel.
- b) Pentru fiecare angajat, afisați numele, prenumele, salariul și grila de salarizare corespunzătoare. Ce operație are loc între tabelele din interogare?

## IV. [SQL\*Plus]

### Variabile de substituție

- Variabilele de substituție sunt utile în crearea de comenzi/script-uri dinamice (care depind de niște valori pe care utilizatorul le furnizează la momentul rulării).
- Variabilele de substituție se pot folosi pentru stocarea temporară de valori, transmiterea de valori între comenzi SQL etc. Ele pot fi create prin:
  - comanda *DEFINE*.( *DEFINE variabila = valoare* )
  - prefixarea cu & (indică existența unei variabile într-o comandă SQL; dacă variabila nu există, atunci SQL\*Plus o creează).
  - prefixarea cu && (indică existența unei variabile într-o comandă SQL; dacă variabila nu există, atunci SQL\*Plus o creează). Deosebirea față de & este că, dacă se folosește &&, atunci referirea ulterioară cu & sau && nu mai necesită ca utilizatorul să introducă de fiecare dată valoarea variabilei. Este folosită valoarea dată la prima referire.
- Variabilele de substituție pot fi eliminate cu ajutorul comenzi *UNDEF[INE]*

### Comanda *DEFINE*

Forma comenzi	Descriere
DEFINE variabila = valoare	Creează o variabilă utilizator cu valoarea de tip sir de caractere precizată.
DEFINE variabila	Afișează variabila, valoarea ei și tipul de date al acesteia.
DEFINE	Afișează toate variabilele existente în sesiunea curentă, împreună cu valorile și tipurile lor de date.

***Observații:***

- Variabilele de tip *DATE* sau *CHAR* trebuie să fie incluse între apostrofuri în comanda *SELECT*.
- Dupa cum numele sugerează, variabilele de sustitutie înlocuiesc/substituie în cadrul comenzi *SQL* variabila respectivă cu sirul de caractere introdus de utilizator.
- Variabilele de sustitutie pot fi utilizate pentru a înlocui la momentul rulării:
  - condiții *WHERE*;
  - clauza *ORDER BY*;
  - expresii din lista *SELECT*;
  - nume de tabel;
  - o intreagă comandă *SQL*;
- Odată definită, o variabilă rămâne până la eliminarea ei cu o comanda *UNDEF* sau până la terminarea sesiunii *SQL\*Plus* respective.
- Comanda *SET VERIFY ON | OFF* permite afișarea sau nu a formei comenzi înainte și după înlocuirea variabilei de substituție.

**Comenzi interactive în *SQL\*Plus***

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

**Fisiere script**

De obicei, un fișier script constă în comenzi *SQL\*Plus* și cel puțin o instrucțiune *SELECT*. În mediul *SQL\*Plus*, fișierul se execută prin comenziile @ sau *START*. În *SQL Developer*, se încarcă fișierul și se acționează butonul *Run Script*.

## V. [Exerciții – SQL\*Plus]

**15.** Ce comenzi SQL\*Plus ați utilizat în laboratoarele precedente?

**16.** Care sunt setările actuale pentru dimensiunea paginii și a liniei în interfața SQL\*Plus?

Setați 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
```

**17.** Să se afișeze codul, numele, salariul și codul departamentului din care face parte un angajat al cărui cod este introdus de utilizator de la tastatură. Analizați diferențele dintre cele 4 posibilități 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;
```

**18.** Să se afișeze numele, codul departamentului și salariul anual pentru toți angajații care au un anumit job.

**19.** Să se afișeze numele, codul departamentului și salariul anual pentru toate persoanele care au fost angajate după o anumită dată calendaristică.

**20.** Să se afișeze o coloană aleasă de utilizator, dintr-un tabel ales de utilizator, ordonând după aceeași coloană care se afișează. De asemenea, este obligatorie precizarea unei condiții în clauza WHERE.

```
SELECT    &&p_coloana  
FROM      &p_tabel  
WHERE     &p_where  
ORDER BY  &p_coloana;
```

**21.** Să se realizeze un script (fișier de comenzi) 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 calendaristică MM/DD/YY.

**22.** Să se realizeze un script pentru a afișa numele angajatului, codul job-ului, salariul și numele departamentului pentru salariații care lucrează într-o locație dată de utilizator. Va fi permisă căutarea case-insensitive.

# Baze de date - Anul 1

## Laborator 8

---

### Limbajul de manipulare a datelor (LMD)

### Limbajul de control al datelor (LCD)

#### I. [Obiective]

- Operații asupra conținutului tabelelor bazei de date (inserare, actualizare, ștergere)
- Conceptul de tranzacție
- Controlul tranzacțiilor (permanentizare, anulare)

#### II. [LMD și LCD]

- Comenzile SQL care alcătuiesc **LMD** permit:
  - regăsirea datelor (*SELECT*);
  - adăugarea de noi înregistrări (*INSERT*);
  - modificarea valorilor coloanelor din înregistrările existente (*UPDATE*);
  - adăugarea sau modificarea condiționată de înregistrări (*MERGE*);
  - suprimarea de înregistrări (*DELETE*).
- **Tranzacția** este o unitate logică de lucru, constituită dintr-o secvență de comenzi care trebuie să se execute **atomic** (ca un întreg) pentru a menține **consistența** bazei de date.
- Server-ul *Oracle* asigură coerența datelor pe baza tranzacțiilor, inclusiv în eventualitatea unei anomalii a unui proces sau a sistemului. Tranzacțiile oferă mai multă flexibilitate și control în modificarea datelor.
- Comenzile SQL care alcătuiesc **LCD** sunt:
  - *ROLLBACK* – pentru a renunța la modificările aflate în așteptare se utilizează instrucțiunea *ROLLBACK*. În urma execuției acesteia, se încheie tranzacția, se anulează modificările asupra datelor, se restaurează starea lor precedentă și se eliberează blocările asupra liniilor.
  - *COMMIT* – determină încheierea tranzacției curente și permanentizarea modificărilor care au intervenit pe parcursul acesteia. Instrucțiunea suprimă toate punctele intermediare definite în tranzacție și eliberează blocările tranzacției.

**Observație:** O comandă LDD (*CREATE, ALTER, DROP*) determină un **COMMIT implicit**.

- *SAVEPOINT* – instrucțiunea *SAVEPOINT* marchează un punct intermediar în procesarea tranzacției. În acest mod este posibilă împărțirea tranzacției în subtranzacții. Această instrucțiune nu face parte din standardul *ANSI* al limbajului *SQL*.

### III. [Comanda *INSERT*]

#### 1. Inserări mono-tabel

Comanda *INSERT* are următoarea sintaxă simplificată:

```
INSERT INTO obiect [AS alias] [ (nume_colonă [, nume_colonă ...] ) ]
{VALUES ( {expr | DEFAULT} [, {expr | DEFAULT} ...] )
| subcerere}
```

Subcererea specificată în comanda *INSERT* returnează linii care vor fi adăugate în tabel.

Dacă în tabel se introduc linii prin intermediul unei subcereri, coloanele din lista *SELECT* trebuie să corespundă, ca număr și tip, celor precizate în clauza *INTO*. În absența unei liste de coloane în clauza *INTO*, subcererea trebuie să furnizeze valori pentru fiecare atribut al obiectului destinație, respectând ordinea în care acestea au fost definite.

#### Observații (tipuri de date):

- Pentru claritate, este recomandată utilizarea unei liste de coloane în clauza *INSERT*.
- În clauza *VALUES*, valorile de tip caracter și dată calendaristică trebuie incluse între apostrofuri. Nu se recomandă includerea între apostrofuri a valorilor numerice, întrucât aceasta ar determina conversii隐含的 la tipul *NUMBER*.
- Pentru introducerea de valori speciale în tabel, pot fi utilizate funcții.
- Adăugarea unei linii care va conține valori *null* se poate realiza în mod:
  - Implicit, prin omiterea numelui coloanei din lista de coloane;
  - Explicit, prin specificarea în lista de valori a cuvântului cheie *null*
  - În cazul sirurilor de caractere sau al datelor calendaristice se poate preciza sirul vid ("").

#### Observații (erori):

Server-ul *Oracle* aplică automat toate tipurile de date, domeniile de valori și constrângerile de integritate. La introducerea sau actualizarea de înregistrări, pot apărea erori în următoarele situații:

- Nu a fost specificată o valoare pentru o coloană *NOT NULL*;

- Există valori dupicate care încalcă o constrângere de unicitate;
- A fost încălcată constrângerea de cheie externă sau o constrângere de tip *CHECK*;
- Există o incompatibilitate în privința tipurilor de date;
- S-a încercat inserarea unei valori având o dimensiune mai mare decât a coloanei corespunzătoare.

## 2. Inserări multi-tabel

O inserare multi-tabel presupune introducerea de linii calculate pe baza rezultatelor unei subcereri, într-unul sau mai multe tabele. Acest tip de inserare este util în mediul *data warehouse*.

În absența acestui tip de inserare, ar fi necesare  $n$  operații independente *INSERT INTO...SELECT...*, unde  $n$  reprezintă numărul tabelelor destinație. Aceasta presupune  $n$  procesări ale aceleiași surse de date și, prin urmare, creșterea de  $n$  ori a timpului necesar procesului.

Sintaxa comenzi *INSERT* în acest caz poate fi:

➤ Pentru inserări necondiționate:

***INSERT ALL INTO... [INTO...]***

*subcerere;*

➤ Pentru inserări condiționate:

***INSERT [ALL | FIRST]***

***WHEN condiție THEN INTO...***

***[WHEN condiție THEN INTO...***

***[ELSE INTO ...]]***

*subcerere;*

- *ALL* determină evaluarea tuturor condițiilor din clauzele *WHEN*. Pentru cele a căror valoare este *TRUE*, se inserează înregistrarea specificată în opțiunea *INTO* corespunzătoare.

- *FIRST* determină inserarea corespunzătoare primei clauze *WHEN* a cărei condiție este evaluată *TRUE*. Toate celelalte clauze *WHEN* sunt ignorate.

## IV. [Exerciții – *INSERT*]

1. Să se creeze tabelele *EMP\_pnu*, *DEPT\_pnu* (în sirul de caractere “pnu”, *p* reprezintă prima literă a prenumelui, iar *nu* reprezintă primele două litere ale numelui dumneavoastră), prin copierea structurii și conținutului tabelelor *EMPLOYEES*, respectiv *DEPARTMENTS*.

```
CREATE TABLE EMP_pnu AS SELECT * FROM employees;
CREATE TABLE DEPT_pnu AS SELECT * FROM departments;
```

2. Listați structura tabelelor sursă și a celor create anterior. Ce se observă?
3. Listați conținutul tabelelor create anterior.
4. Pentru introducerea constrângerilor de integritate, executați instrucțiunile LDD indicate în continuare. Prezentarea detaliată a LDD se va face în cadrul laboratorului 4.

```
ALTER TABLE emp_pnu
ADD CONSTRAINT pk_emp_pnu PRIMARY KEY(employee_id);
ALTER TABLE dept_pnu
ADD CONSTRAINT pk_dept_pnu PRIMARY KEY(department_id);
ALTER TABLE emp_pnu
ADD CONSTRAINT fk_emp_dept_pnu
    FOREIGN KEY(department_id) REFERENCES dept_pnu(department_id);
```

**Observație:** Ce constrângere nu am implementat?

5. Să se insereze departamentul 300, cu numele *Programare* în *DEPT\_pnu*.

Analizați cazurile, precizând care este soluția corectă și explicând erorile celorlalte variante. Pentru a anula efectul instrucțiunii(ilor) corecte, utilizați comanda *ROLLBACK*.

a) *INSERT INTO DEPT\_pnu*

*VALUES (300, 'Programare');*

b) *INSERT INTO DEPT\_pnu (department\_id, department\_name)*

*VALUES (300, 'Programare');*

c) *INSERT INTO DEPT\_pnu (department\_name, department\_id)*

*VALUES (300, 'Programare');*

d) *INSERT INTO DEPT\_pnu (department\_id, department\_name, location\_id)*

*VALUES (300, 'Programare', null);*

e) *INSERT INTO DEPT\_pnu (department\_name, location\_id)*

*VALUES ('Programare', null);*

Execuția variantei care a fost corectă de două ori. Ce se obține și de ce?

6. Să se insereze un angajat corespunzător departamentului introdus anterior în tabelul *EMP\_pnu*, precizând valoarea *NULL* pentru coloanele a căror valoare nu este cunoscută la inserare (metoda implicită de inserare). Determinați ca efectele instrucțiunii să devină permanente.

Atenție la constrângerile *NOT NULL* asupra coloanelor tabelului!

7. Să se mai introducă un angajat corespunzător departamentului 300, precizând după numele tabelului lista coloanelor în care se introduc valori (metoda explicită de inserare). Se presupune că data angajării acestuia este cea curentă (*SYSDATE*). Salvați înregistrarea.

- 8.** Este posibilă introducerea de înregistrări prin intermediul subcererilor (specificate în locul tabelului). Ce reprezintă, de fapt, aceste subcereri? Să se analizeze următoarele comenzi *INSERT*:

```

INSERT INTO emp_pnu (employee_id, last_name, email, hire_date,
                      job_id, salary, commission_pct)
VALUES (252, 'Nume252', 'nume252@emp.com', SYSDATE, 'SA_REP', 5000, NULL);

SELECT employee_id, last_name, email, hire_date, job_id, salary, commission_pct
FROM emp_pnu
WHERE employee_id=252;

ROLLBACK;

INSERT INTO
(SELECT employee_id, last_name, email, hire_date, job_id, salary, commission_pct
FROM emp_pnu)
VALUES (252, 'Nume252', 'nume252@emp.com', SYSDATE, 'SA_REP', 5000, NULL);

SELECT employee_id, last_name, email, hire_date, job_id, salary, commission_pct
FROM emp_pnu
WHERE employee_id=252;

ROLLBACK;

```

Introduceți un angajat precizând pentru valoarea *employee\_id* o subcerere care returnează (codul maxim +1).

- 9.** Se poate utiliza clauza WITH pentru rezolvarea problemei anterioare?

- 10.** Creați un nou tabel, numit *EMP1\_PNU*, care va avea aceeași structură ca și *EMPLOYEES*, dar nicio înregistrare. Copiați în tabelul *EMP1\_PNU* salariații (din tabelul *EMPLOYEES*) al căror comision depășește 25% din salariu.

```

CREATE TABLE emp1_pnu AS SELECT * FROM employees WHERE 1=0;
--DELETE FROM emp1_pnu; --necesar daca nu aveam clauza WHERE de mai sus
INSERT INTO emp1_pnu
    SELECT *
        FROM employees
        WHERE commission_pct > 0.25;
SELECT employee_id, last_name, salary, commission_pct
FROM emp_pnu;
ROLLBACK;

```

Ce va conține tabelul *EMP1\_PNU* în urma acestei succesiuni de comenzi?

- 11.** Inserați o nouă înregistrare în tabelul *EMP\_PNU* care să totalizeze salariile, să calculeze media comisioanelor, iar câmpurile de tip dată să conțină data curentă și câmpurile de tip caracter să conțină textul 'TOTAL'. Numele și prenumele angajatului vor corespunde utilizatorului curent (*USER*). Pentru câmpul *employee\_id* se va introduce valoarea 0, iar *manager\_id* și *department\_id* vor avea valoarea null.
- 12.** Să se creeze un fișier (*script file*) care să permită introducerea de înregistrări în tabelul *EMP\_PNU* în mod interactiv. Se vor cere utilizatorului: codul, numele, prenumele și salariul angajatului. Câmpul *email* se va completa automat prin concatenarea primei litere din prenume și a primelor 7 litere din nume.

*REM setari*

*REM comenzi ACCEPT*

*INSERT INTO emp\_pnu*

*VALUES (&...);*

*REM suprimarea variabilelor utilizate*

*REM anularea setarilor, prin stabilirea acestora la valorile implicate*

Execuți script-ul pentru a introduce 2 înregistrări în tabel.

- 13.** Creați 2 tabele *emp2\_pnu* și *emp3\_pnu* cu aceeași structură ca tabelul *EMPLOYEES*, dar fără înregistrări (acceptăm omiterea constrângerilor de integritate). Prin intermediul unei singure comenzi, copiați din tabelul *EMPLOYEES*:
- în tabelul *EMP1\_PNU* salariații care au salariul mai mic decât 5000;
  - în tabelul *EMP2\_PNU* salariații care au salariul cuprins între 5000 și 10000;
  - în tabelul *EMP3\_PNU* salariații care au salariul mai mare decât 10000.

Verificați rezultatele, apoi ștergeți toate înregistrările din aceste tabele.

*INSERT ALL*

*WHEN ... THEN*

*INTO ...*

*...*

*ELSE*

*INTO ...*

*SELECT \* FROM employees;*

*SELECT \* FROM emp1\_pnu;*

*SELECT \* FROM emp2\_pnu;*

*SELECT \* FROM emp3\_pnu;*

*DELETE FROM emp1\_pnu;*

*DELETE FROM emp2\_pnu;*

*DELETE FROM emp3\_pnu;*

14. Să se creeze tabelul *EMPO\_PNU* cu aceeași structură ca tabelul *EMPLOYEES* (fără constrângeri), dar fără nici o înregistrare. Copiați din tabelul *EMPLOYEES*:

- în tabelul *EMPO\_PNU* salariații care lucrează în departamentul 80;
- în tabelul *EMP1\_PNU* salariații care au salariul mai mic decât 5000;
- în tabelul *EMP2\_PNU* salariații care au salariul cuprins între 5000 și 10000;
- în tabelul *EMP3\_PNU* salariații care au salariul mai mare decât 10000.

Dacă un salariat se încadrează în tabelul *EMPO\_PNU* atunci acesta nu va mai fi inserat și în alt tabel (tabelul corespunzător salariului său).

*INSERT FIRST*

*WHEN ... THEN*

*INTO ...*

...

*ELSE*

*INTO ...*

*SELECT \* FROM employees;*

*SELECT \* FROM emp\*\_pnu;*

## V. [Comanda *UPDATE*]

Sintaxa simplificată a comenzi ***UPDATE*** este:

*UPDATE nume\_tabel [alias]  
SET col1 = expr1[, col2=expr2]  
[WHERE conditie];*  
sau  
*UPDATE nume\_tabel [alias]  
SET (col1,col2,...) = (subcerere)  
[WHERE conditie];*

### Observații:

- în general, pentru identificarea unei linii se folosește o condiție ce implică cheia primară;
- dacă nu apare clauza *WHERE* atunci sunt afectate toate liniile tabelului specificat;
- cazurile în care instrucțiunea *UPDATE* nu poate fi executată sunt similare celor în care eșuează instrucțiunea *INSERT*. Acestea au fost menționate anterior.

## VI. [Exerciții – *UPDATE*]

15. Măriți salariul tuturor angajaților din tabelul *EMP\_PNU* cu 5%. Vizualizați, iar apoi anulați modificările.
16. Schimbați jobul tuturor salariaților din departamentul 80 care au comision în 'SA\_REP'. Anulați modificările.
17. Să se promoveze Douglas Grant la funcția de manager în departamentul 20, având o creștere de salariu de 1000. Se poate realiza modificarea prin intermediul unei singure comenzi?
18. Schimbați salariul și comisionul celui mai prost plătit salariat din firmă, astfel încât să fie egale cu salariul și comisionul șefului său.
19. Să se modifice adresa de e-mail pentru angajații care câștigă cel mai mult în departamentul în care lucrează astfel încât acesta să devină inițiala numelui concatenată cu prenumele. Dacă nu are prenume atunci în loc de acesta apare caracterul ‘.’. Anulați modificările.
20. Pentru fiecare departament să se mărească salariul celor care au fost angajați primii astfel încât să devină media salariilor din companie. Țineți cont de liniile introduse anterior.
21. Să se modifice jobul și departamentul angajatului având codul 114, astfel încât să fie la fel cu cele ale angajatului având codul 205.
22. Creați un script prin intermediul căruia să fie posibilă actualizarea în mod interactiv de înregistrări ale tabelului *dept\_pnu*. Se va cere codul departamentului care urmează a fi actualizat, se va afișa linia respectivă, iar apoi se vor cere valori pentru celelalte câmpuri.

## VII. [Comanda *DELETE*]

Sintaxa simplificată a comenzi ***DELETE*** este:

*DELETE FROM nume\_tabel  
[WHERE conditie];*

Dacă nu se specifică nicio condiție, atunci vor fi șterse toate liniile din tabel.

## VIII. [Exerciții – *DELETE*]

23. Ștergeți toate înregistrările din tabelul *DEPT\_PNU*. Ce înregistrări se pot șterge? Anulați modificările.

24. Stergeți angajații care nu au comision. Anulați modificările.
25. Suprimați departamentele care nu au nici un angajat. Anulați modificările.
26. Să se creeze un fișier *script* prin care se cere un cod de angajat din tabelul *EMP\_PNU*.  
Se va lista înregistrarea corespunzătoare acestuia, iar apoi linia va fi suprimată din tabel.

#### IX. [Exerciții – LMD, LCD]

27. Să se steargă un angajat din tabelul *EMP\_PNU* prin intermediul script-ului creat la problema 26. Modificările vor deveni permanente.
28. Să se mai introducă o linie în tabel, rulând încă o dată fișierul creat la exercițiul 12.
29. Să se marcheze un punct intermediar în procesarea tranzacției.

SAVEPOINT p

30. Să se steargă tot conținutul tabelului. Listați conținutul tabelului.
31. Să se renunțe la cea mai recentă operație de stergere, fără a renunța la operația precedentă de introducere.

ROLLBACK TO p

32. Listați conținutul tabelului. Determinați ca modificările să devină permanente.

# Baze de date - Anul 1

## Laborator 9

---

### Limbajul de definire a datelor (LDD) (partea I)

#### I. [Obiective]

- Operații de definire (creare, modificare, suprimare) a tabelelor bazei de date

#### II. [LDD]

- În general, instrucțiunile *LDD* sunt utilizate pentru definirea structurii corespunzătoare obiectelor unei scheme: tabele, vizualizări, vizualizări materializate, indecsi, sinonime, clustere, proceduri și funcții stocate, declanșatori, pachete stocate etc.
- Aceste instrucțiuni permit:
  - crearea, modificarea și suprimarea obiectelor unei scheme și a altor obiecte ale bazei de date, inclusiv baza însăși și utilizatorii acesteia (*CREATE, ALTER, DROP*);
  - modificarea numelor obiectelor unei scheme (*RENAME*);
  - ștergerea datelor din obiectele unei scheme, fără suprimarea structurii obiectelor respective (*TRUNCATE*).
- Implicit, o instrucțiune *LDD* permanentizează (*COMMIT*) efectul tuturor instrucțiunilor precedente și marchează începutul unei noi tranzacții.
- Instrucțiunile *LDD* au efect imediat asupra bazei de date și înregistrează informația în dicționarul datelor.
- Definirea unui obiect presupune: crearea (*CREATE*), modificarea (*ALTER*) și suprimarea sa (*DROP*).

#### Reguli de numire a obiectelor bazei de date

- Identifierii obiectelor trebuie să înceapă cu o literă și să aibă maximum 30 de caractere, cu excepția numelui bazei de date care este limitat la 8 caractere și celui al legăturii unei baze de date, a cărui lungime poate atinge 128 de caractere.
- Numele poate conține caracterele A-Z, a-z, 0-9, \_, \$ și #.
- Două obiecte ale aceluiași utilizator al *server-ului Oracle* nu pot avea același nume.
- Identifierii nu pot fi cuvinte rezervate ale *server-ului Oracle*.

- Identifierii obiectelor nu sunt *case-sensitive*.

### III. [Definirea tabelelor]

#### 1. Crearea tabelelor

- Formele simplificate ale comenzi de creare a tabelelor sunt:

```
CREATE TABLE nume_tabel (
    coloana_1 tip_date [DEFAULT valoare]
        [constrangere_nivel_coloana [constrangere_nivel_coloana]...],
    .....
    coloana_n tip_date [DEFAULT valoare]
        [constrangere_nivel_coloana [constrangere_nivel_coloana]...],
    [constrangeri_nivel_tabel]
);
sau
CREATE TABLE nume_tabel [(coloana_1,..., coloana_n)]
AS subcerere;
```

- Constraințele **definite** **aupra unui tabel** pot fi de următoarele tipuri:

- *NOT NULL* – coloana nu poate conține valoarea *Null*; (*NOT NULL*)
- *UNIQUE* – pentru coloane sau combinații de coloane care trebuie să aibă valori unice în cadrul tabelului; (*UNIQUE (col1, col2, ...)*)
- *PRIMARY KEY* – identifică în mod unic orice înregistrare din tabel. Implică *NOT NULL + UNIQUE*; (*PRIMARY KEY (col1, col2, ...)*)
- *FOREIGN KEY* – stabilește o relație de cheie externă între o coloană a tabelului și o coloană dintr-un tabel specificat.

[*FOREIGN KEY nume\_col*]  
*REFERENCES nume\_tabel(nume\_coloana)*  
[*ON DELETE {CASCADE/ SET NULL}*]  
- *FOREIGN KEY* este utilizat într-o constrângere la nivel de tabel pentru a defini coloana din tabelul „copil”;  
- *REFERENCES* identifică tabelul „părinte” și coloana corespunzătoare din acest tabel;  
- *ON DELETE CASCADE* determină ca, odată cu ștergerea unei linii din tabelul „părinte”, să fie șterse și liniile dependente din tabelul „copil”;

- *ON DELETE SET NULL* determină modificarea automată a valorilor cheii externe la valoarea *null*, atunci când se șterge valoarea „părinte“.
- *CHECK* – specifică o condiție care trebuie să fie adevărată la nivel de coloană sau linie (*CHECK (conditie)*).

#### **Observații:**

- Constrângerile pot fi create odată cu tabelul sau adăugate ulterior cu o comandă *ALTER TABLE*.
- Constrângerile se pot implementa la nivel de coloană doar dacă nu referă o altă coloană a tabelului.
- În cazul în care o constrângere referă mai multe coloane, ea poate fi definită doar la nivel de tabel. De exemplu, dacă cheia primară (sau o cheie unică) este compusă, ea nu poate fi definită la nivel de coloane, ci doar la nivelul întregului tabel.
- Constrângerea de tip *NOT NULL* se poate declara doar la nivel de coloană.

- Principalele **tipuri de date** pentru coloanele tabelelor sunt următoarele:

Tip de date	Descriere
VARCHAR2(n) [BYTE   CHAR]	Definește un sir de caractere de dimensiune variabilă, având lungimea maximă de <i>n</i> octeți sau caractere. Valoarea maximă a lui <i>n</i> corespunde la 4000 octeți, iar cea minimă este de un octet sau un caracter.
CHAR(n) [BYTE   CHAR]	Reprezintă un sir de caractere de lungime fixă având <i>n</i> octeți sau caractere. Valoarea maximă a lui <i>n</i> corespunde la 2000 octeți. Valoarea implicită și minimă este de un octet.
NUMBER(p, s)	Reprezintă un număr având <i>p</i> cifre, dintre care <i>s</i> cifre formează partea zecimală
LONG	Conține siruri de caractere având lungime variabilă, care nu pot ocupa mai mult de 2GB.
DATE	Reprezintă date calendaristice valide, între 1 ianuarie 4712 i.Hr. și 31 decembrie 9999 d.Hr.

## 2. Modificarea (structurii) tabelelor

- **Modificarea structurii unui tabel** se face cu ajutorul comenzi ***ALTER TABLE***. Forma comenzi depinde de tipul modificării aduse:
  - Adăugarea unei noi coloane (nu se poate specifica poziția unei coloane noi în structura tabelului; o coloană nouă devine automat ultima în cadrul structurii tabelului)

**ALTER TABLE nume\_tabel**

**ADD (coloana tip\_de\_date [DEFAULT expr][, ...]);**

- Modificarea unei coloane (schimbarea tipului de date, a dimensiunii sau a valorii implicite a acesteia; schimbarea valorii implicite afectează numai inserările care succed modificării)

**ALTER TABLE nume\_tabel**

**MODIFY (coloana tip\_de\_date [DEFAULT expr][, ...]);**

- Eliminarea unei coloane din structura tabelului:

**ALTER TABLE nume\_tabel**

**DROP COLUMN coloana;**

**Observații:**

- Dimensiunea unei coloane numerice sau de tip caracter poate fi mărită, dar nu poate fi micșorată decât dacă acea coloană conține numai valori *null* sau dacă tabelul nu conține nici o linie.
- Tipul de date al unei coloane poate fi modificat doar dacă valorile coloanei respective sunt *null*.
- O coloană *CHAR* poate fi convertită la tipul de date *VARCHAR2* sau invers, numai dacă valorile coloanei sunt *null* sau dacă nu se micșorează dimensiunea coloanei.

- Comanda *ALTER* permite **adăugarea unei constrângeri într-un tabel existent, eliminarea, activarea sau dezactivarea constrângerilor.**

- Pentru adăugare de constrângeri, comanda are forma:

**ALTER TABLE nume\_tabel**

**ADD [CONSTRAINT nume\_constr] tip\_constr (coloana);**

- Pentru eliminare de constrângeri:

**ALTER TABLE nume\_tabel**

**DROP CONSTRAINT nume\_constr;**

- Pentru activare/dezactivare constrângere:

**ALTER TABLE nume\_tabel**

**MODIFY CONSTRAINT nume\_constr ENABLE|DISABLE;**

sau

**ALTER TABLE nume\_tabel**

**ENABLE|DISABLE CONSTRAINT nume\_constr;**

### 3. Suprimarea tabelelor

- Ștergerea fizică a unui tabel, inclusiv a înregistrărilor acestuia, se realizează prin comanda:

**DROP TABLE** *nume\_tabel*;

- Pentru ștergerea conținutului unui tabel și păstrarea structurii acestuia se poate utiliza comanda:

**TRUNCATE TABLE** *nume\_tabel*;

**Observație:** Fiind operație *LDD*, comanda *TRUNCATE* are efect definitiv (spre deosebire de *DELETE* care, fiind o comandă *LMD*, poate fi anulată).

### 4. Redenumirea tabelelor

- Comanda **RENAME** permite redenumirea unui tabel, vizualizare sau secvență.

**RENAME** *nume1\_object TO nume2\_object*;

**Observații:**

- În urma redenumirii sunt transferate automat constrângerile de integritate, indecșii și privilegiile asupra vechilor obiecte.
- Sunt validate toate obiectele ce depind de obiectul redenumit, cum ar fi vizualizări, sinonime sau proceduri și funcții stocate.

### 5. Consultarea dicționarului datelor

- Informații despre tabelele create se găsesc în vizualizările din dicționarul datelor:
  - *USER\_TABLES* – informații complete despre tabelele utilizatorului.
  - *TAB* – informații de bază despre tabelele existente în schema utilizatorului.
- Informații despre constrângerile găsim în *USER\_CONSTRAINTS*, iar despre coloanele implicate în constrângerile în *USER\_CONS\_COLUMNS*.

## IV. [Exerciții]

1. Să se creeze tabelul *ANGAJATI\_pnu* (*pnu* se alcatuiește din prima literă din prenume și primele două din numele studentului) corespunzător schemei relaționale:

*ANGAJATI\_pnu(cod\_ang number(4), nume varchar2(20), prenume varchar2(20), email char(15), data\_ang date, job varchar2(10), cod\_sef number(4), salariu number(8, 2), cod\_dep number(2))*

în următoarele moduri:

- a) fără precizarea vreunei chei sau constrângeri;
- b) cu precizarea cheilor primare la nivel de coloană și a constrângерilor *NOT NULL* pentru coloanele nume și salariu;
- c) cu precizarea cheii primare la nivel de tabel și a constrângерilor *NOT NULL* pentru coloanele *nume* și *salariu*.

Se presupune că valoarea implicită a coloanei *data\_ang* este *SYSDATE*.

**Observație:** Nu pot exista două tabele cu același nume în cadrul unei scheme, deci recrearea unui tabel va fi precedată de suprimarea sa prin comanda:

*DROP TABLE ANGAJATI\_pnu;*

**2.** Adăugați următoarele înregistrări în tabelul *ANGAJATI\_pnu*:

Cod_ang	Nume	Prenume	Email	Data_ang	Job	Cod_sef	Salariu	Cod_dep
100	Nume1	Prenume1	Null	Null	Director	null	20000	10
101	Nume2	Prenume2	Nume2	02-02-2014	Inginer	100	10000	10
102	Nume3	Prenume3	Nume3	05-06-2010	Programator	101	5000	20
103	Nume4	Prenume4	Null	Null	Inginer	100	9000	20
104	Nume5	Prenume5	Nume5	Null	Programator	101	3000	30

Prima și a patra înregistrare vor fi introduse specificând coloanele pentru care introduceti date efectiv, iar celelalte vor fi inserate fără precizarea coloanelor în comanda *INSERT*. Salvați comenziile de inserare.

- 3.** Creați tabelul *ANGAJATI10\_pnu*, prin copierea angajaților din departamentul 10 din tabelul *ANGAJATI\_pnu*. Listați structura noului tabel. Ce se observă?
- 4.** Introduceti coloana *comision* în tabelul *ANGAJATI\_pnu*. Coloana va avea tipul de date *NUMBER(4,2)*.
- 5.** Este posibilă modificarea tipului coloanei *salariu* în *NUMBER(6,2)*?
- 6.** Setați o valoare *DEFAULT* pentru coloana salariu.
- 7.** Modificați tipul coloanei *comision* în *NUMBER(2, 2)* și al coloanei *salariu* în *NUMBER(10,2)*, în cadrul aceleiași instrucțiuni.
- 8.** Actualizați valoarea coloanei *comision*, setând-o la valoarea 0.1 pentru salariații al căror job începe cu litera I. (*UPDATE*)
- 9.** Modificați tipul de date al coloanei *email* în *VARCHAR2*.
- 10.** Adăugați coloana *nr\_telefon* în tabelul *ANGAJATI\_pnu*, setându-i o valoare implicită.
- 11.** Vizualizați înregistrările existente. Suprimați coloana *nr\_telefon*.

Ce efect ar avea o comandă *ROLLBACK* în acest moment?

12. Redenumiți tabelul *ANGAJATI\_pnu* în *ANGAJATI3\_pnu*.
13. Consultați vizualizarea TAB din dicționarul datelor. Redenumiți *angajati3\_pnu* în *angajati\_pnu*.
14. Suprimați conținutul tabelului *angajati10\_pnu*, fără a suprima structura acestuia.

15. Creați tabelul *DEPARTAMENTE\_pnu*, corespunzător schemei relaționale:

*DEPARTAMENTE\_pnu (cod\_dep# number(2), nume varchar2(15), cod\_director number(4))*  
specificând doar constrângerea *NOT NULL* pentru nume (nu precizați deocamdată constrângerea de cheie primară).

```
CREATE TABLE departamente_pnu ( ... );
DESC departamente_pnu
```

16. Introduceți următoarele înregistrări în tabelul *DEPARTAMENTE\_pnu*:

Cod_dep	Nume	Cod_director
10	Administrativ	100
20	Proiectare	101
30	Programare	Null

17. Introduceți constângerea de cheie primară asupra coloanei *cod\_dep*, fără suprimarea și recrearea tabelului (comanda *ALTER*).

**Observație:**

- Introducerea unei constrângerii după crearea tabelului presupune că toate liniile existente în tabel la momentul respectiv satisfac noua constrângere.
- Specificarea constrângerilor permite numirea acestora.
- În situația în care constrângerile sunt precizate la nivel de coloană sau tabel (în *CREATE TABLE*) ele vor primi implicit nume atribuite de sistem, dacă nu se specifică vreun alt nume într-o clauză *CONSTRAINT*.

**Exemplu :** CREATE TABLE alfa (

```
    X NUMBER CONSTRAINT nn_x NOT NULL,
    Y VARCHAR2 (10) NOT NULL
```

```
);
```

18. Să se precizeze constrângerea de cheie externă pentru coloana *cod\_dep* din *ANGAJATI\_pnu*:
  - a) fără suprimarea tabelului (*ALTER TABLE*);
  - b) prin suprimarea și recrearea tabelului, cu precizarea noii constrângerii la nivel de coloană (*{DROP, CREATE} TABLE*). De asemenea, se vor mai preciza constrângerile (la nivel de coloană, în măsura în care este posibil):
    - *PRIMARY KEY* pentru *cod\_ang*;

- FOREIGN KEY pentru *cod\_sef*;
- UNIQUE pentru combinația *nume + prenume*;
- UNIQUE pentru *email*;
- NOT NULL pentru *nume*;
- verificarea *cod\_dep > 0*;
- verificarea ca salariul să fie mai mare decât comisionul\*100.

**19.** Suprimați și recreați tabelul, specificând toate constrângerile la nivel de tabel (în măsura în care este posibil).

**20.** Reintroduceți date în tabel, utilizând (și modificând, dacă este necesar) comenzi salvate anterior.

**21.** Ce se întâmplă dacă se încearcă suprimarea tabelului *departamente\_pnu*?

**22.** Analizați structura vizualizărilor *USER\_TABLES*, *TAB*, *USER\_CONSTRAINTS*.

**Observație:** Pentru a afla informații despre tabelele din schema curentă, sunt utile cererile:

SELECT \* FROM tab;

sau

SELECT table\_name FROM user\_tables;

**23. a)** Listați informațiile relevante (cel puțin nume, tip și tabel) despre constrângerile asupra tabelelor *angajati\_pnu* și *departamente\_pnu*.

```
SELECT constraint_name, constraint_type, table_name
FROM     user_constraints
WHERE    lower(table_name) IN ('angajati_pnu', 'departamente_pnu');
```

**Observație:** Tipul constrângerilor este marcat prin:

- P – pentru cheie primară
- R – pentru constrângerea de integritate referențială (cheie externă);
- U – pentru constrângerea de unicitate (UNIQUE);
- C – pentru constrângerile de tip CHECK.

b) Aflați care sunt coloanele la care se referă constrângerile asupra tabelelor *angajati\_pnu* și *departamente\_pnu*.

```
SELECT table_name, constraint_name, column_name
FROM     user_cons_columns
WHERE   LOWER(table_name) IN ('angajati_pnu ', 'departamente_pnu '');
```

**24.** Introduceți constrângerea NOT NULL asupra coloanei *email*.

**25.** (Încercați să) adăugați o nouă înregistrare în tabelul *ANGAJATI\_pnu*, care să corespundă codului de departament 50. Se poate?

**26.** Adăugați un nou departament, cu numele Testare, codul 60 și directorul null în *DEPARTAMENTE\_pnu*. *COMMIT*.

**27.** (Încercați să) ștergeți departamentul 20 din tabelul *DEPARTAMENTE\_pnu*. Comentați.

**28.** Ștergeți departamentul 60 din *DEPARTAMENTE\_pnu*. *ROLLBACK*.

**29.** (Încercați să) introduceți un nou angajat, specificând valoarea 114 pentru *cod\_sef*. Ce se obține?

**30.** Adăugați un nou angajat, având codul 114. Încercați din nou introducerea înregistrării de la exercițiul 29.

**Ce concluzii reies din exercițiile precedente? Care este ordinea de inserare, atunci când avem constrângeri de cheie externă?**

**31.** Se dorește ștergerea automată a angajaților dintr-un departament, odată cu suprimarea departamentului. Pentru aceasta, este necesară introducerea clauzei *ON DELETE CASCADE* în definirea constrângerii de cheie externă. Suprimați constrângerea de cheie externă asupra tabelului *ANGAJATI\_pnu* și reintroduceți această constrângere, specificând clauza *ON DELETE CASCADE*.

**32.** Ștergeți departamentul 20 din *DEPARTAMENTE\_pnu*. Ce se întâmplă? *Rollback*.

**33.** Introduceți constrângerea de cheie externă asupra coloanei *cod\_director* a tabelului *DEPARTAMENTE\_pnu*. Se dorește ca ștergerea unui angajat care este director de departament să implice setarea automată a valorii coloanei *cod\_director* la *null*.

**34.** Actualizați tabelul *DEPARTAMENTE\_PNU*, astfel încât angajatul având codul 102 să devină directorul departamentului 30. Ștergeți angajatul având codul 102 din tabelul *ANGAJATI\_pnu*. Analizați efectele comenzi. *Rollback*.

Este posibilă suprimarea angajatului având codul 101? Comentați.

**35.** Adăugați o constrângere de tip *check* asupra coloanei *salariu*, astfel încât acesta să nu poată depăși 30000.

**36.** Încercați actualizarea salariului angajatului 100 la valoarea 35000.

**37.** Dezactivați constrângerea creată anterior și reîncercați actualizarea. Ce se întâmplă dacă încercăm reactivarea constrângerii?

# Baze de date - Anul 1

## Laborator 10

---

### Limbajul de definire a datelor (LDD) (partea II)

#### I. [Obiective]

- Operații de definire a altor tipuri de obiecte ale bazei de date: vizualizări, secvențe

#### II. [Definirea vizualizărilor (*view*)]

- Vizualizările sunt tabele virtuale construite pe baza unor tabele sau a altor vizualizări, denumite tabele de bază.
- Vizualizările nu conțin date, dar reflectă datele din tabelele de bază.
- Vizualizările sunt definite de o cerere SQL, motiv pentru care mai sunt denumite cereri stocate.
- Avantajele utilizării vizualizărilor:
  - restricționarea accesului la date;
  - simplificarea unor cereri complexe;
  - asigurarea independenței datelor de programele de aplicații;
  - prezentarea de diferite imagini asupra datelor.
- Crearea vizualizărilor se realizează prin comanda *CREATE VIEW*, a cărei sintaxă simplificată este:

***CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW***

*nume\_vizualizare [[alias, alias, ..]]*

*AS subcerere*

***[WITH CHECK OPTION [CONSTRAINT nume\_constrangere]]***

***[WITH READ ONLY [CONSTRAINT nume\_constrangere]];***

- *OR REPLACE* se utilizează pentru a schimba definiția unei vizualizări fără a mai reacorda eventualele privilegii.
- Opțiunea *FORCE* permite crearea vizualizării înainte de definirea tabelelor, ignorând erorile la crearea vizualizării.
- Subcererea poate fi oricât de complexă dar nu poate conține clauza *ORDER BY*. Dacă se dorează ordonare se utilizează *ORDER BY* la interogarea vizualizării.
- *WITH CHECK OPTION* permite inserarea și modificarea prin intermediul vizualizării numai a liniilor ce sunt accesibile vizualizării. Dacă lipsește numele constrângerii atunci sistemul asociază un nume implicit de tip *SYS\_Cn* acestei

constrangeri ( $n$  este un număr generat astfel încât numele constrângerii să fie unic).

- *WITH READ ONLY* asigură că prin intermediul vizualizării nu se pot executa operații LMD.
- Modificarea vizualizărilor se realizează prin recrearea acestora cu ajutorul opțiunii *OR REPLACE*. Totuși, începând cu *Oracle9i*, a devenit posibilă utilizarea comenzi *ALTER VIEW* pentru adăugare de constrângerii vizualizării.
- Suprimarea vizualizărilor se face cu comanda *DROP VIEW* :

***DROP VIEW nume\_vizualizare;***

- Informații despre vizualizări se pot găsi în dicționarul datelor interogând vizualizările: *USER\_VIEWS*, *ALL\_VIEWS*. Pentru aflarea informațiilor despre coloanele actualizabile, este utilă vizualizarea *USER\_UPDATABLE\_COLUMNS*.
- Subcererile însoțite de un alias care apar în comenziile *SELECT*, *INSERT*, *UPDATE*, *DELETE*, *MERGE* se numesc vizualizări *inline*. Spre deosebire de vizualizările propriu zise, acestea nu sunt considerate obiecte ale schemei ci sunt entități temporare (valabile doar pe perioada execuției instrucțiunii LMD respective).

## Operații LMD asupra vizualizărilor

Vizualizările se pot clasifica în simple și complexe. Această clasificare este importantă pentru că asupra vizualizărilor simple se pot realiza operații LMD, dar în cazul celor complexe acest lucru nu este posibil întotdeauna (decât prin definirea de triggeri de tip *INSTEAD OF*).

- Vizualizările simple sunt definite pe baza unui singur tabel și nu conțin funcții sau grupări de date.
- Vizualizările compuse sunt definite pe baza mai multor tabele sau conțin funcții sau grupări de date.
- Nu se pot realiza operații LMD în vizualizări ce conțin:
  - funcții grup,
  - clauzele *GROUP BY*, *HAVING*, *START WITH*, *CONNECT BY*,
  - cuvântul cheie *DISTINCT*,
  - pseudocoloana *ROWNUM*,
  - operatori pe mulțimi.
- Nu se pot actualiza:

- coloane ale căror valori rezultă prin calcul sau definite cu ajutorul funcției *DECODE*,
- coloane care nu respectă constrângerile din tabelele de bază.
- Pentru vizualizările bazate pe mai multe tabele, orice operație *INSERT*, *UPDATE* sau *DELETE* poate modifica datele doar din unul din tabelele de bază. Acest tabel este cel protejat prin cheie (*key preserved*). În cadrul unei astfel de vizualizări, un tabel de bază se numește *key-preserved* dacă are proprietatea că fiecare valoare a cheii sale primare sau a unei coloane având constrângerea de unicitate, este unică și în vizualizare.
- Prima condiție ca o vizualizare a cărei cerere conține un join să fie modificabilă este ca instrucțiunea LMD să afecteze un singur tabel din operația de join.
- Reactualizarea tabelelor implică reactualizarea corespunzătoare a vizualizărilor!!!

Reactualizarea vizualizărilor implică reactualizarea tabelelor de bază? Nu întotdeauna! Există restricții care trebuie respectate, însă atunci când reactualizarea poate avea loc, datele modificate sunt cele din tabelele de bază!

### III. [Exerciții - vizualizări]

1. Pe baza tabelului EMP\_PNU, să se creeze o vizualizare VIZ\_EMP30\_PNU, care conține codul, numele, email-ul și salariul angajaților din departamentul 30. Să se analizeze structura și conținutul vizualizării. Ce se observă referitor la constrângerile? Ce se obține de fapt la interogarea conținutului vizualizării? Inserați o linie prin intermediul acestei vizualizări; comentați.
2. Modificați VIZ\_EMP30\_PNU astfel încât să fie posibilă inserarea/modificarea conținutului tabelului de bază prin intermediul ei. Inserați și actualizați o linie (cu valoarea 300 pentru codul angajatului) prin intermediul acestei vizualizări.

**Observație:** Trebuie introduse neapărat în vizualizare coloanele care au constrângerea *NOT NULL* în tabelul de bază (altfel, chiar dacă tipul vizualizării permite operații LMD, acestea nu vor fi posibile din cauza nerespectării constrângerilor *NOT NULL*).

Unde a fost introdusă linia? Mai apare ea la interogarea vizualizării?

Ce efect are următoarea operație de actualizare?

*UPDATE viz\_emp30\_pnu*

*SET hire\_date=hire\_date-15*

*WHERE employee\_id=300;*

Comentați efectul următoarelor instrucțiuni, analizând și efectul asupra tabelului de bază:

*UPDATE emp\_pnu*

```
SET department_id=30  
WHERE employee_id=300;  
  
UPDATE viz_emp30_pnu  
SET hire_date=hire_date-15  
WHERE employee_id=300;
```

Ștergeți angajatul având codul 300 prin intermediul vizualizării. Analizați efectul asupra tabelului de bază.

3. Să se creeze o vizualizare, VIZ\_EMPSAL50\_PNU, care contine coloanele cod\_angajat, nume, email, functie, data\_angajare si sal\_anual corespunzătoare angajaților din departamentul 50. Analizați structura și conținutul vizualizării.
4. a) Inserați o linie prin intermediul vizualizării precedente. Comentați.  
b) Care sunt coloanele actualizabile ale acestei vizualizări? Verificați răspunsul în dicționarul datelor (*USER\_UPDATABLE\_COLUMNS*).  
c) Inserați o linie specificând valori doar pentru coloanele actualizabile.  
d) Analizați conținutul vizualizării VIZ\_EMPSAL50\_PNU și al tabelului EMP\_PNU.
5. a) Să se creeze vizualizarea VIZ\_EMP\_DEP30\_PNU, astfel încât aceasta să includă coloanele vizualizării VIZ\_EMP\_30\_PNU, precum și numele și codul departamentului. Să se introducă aliasuri pentru coloanele vizualizării.

**Observație:** Asigurați-vă că există constrângerea de cheie externă între tabelele de bază ale acestei vizualizări.

- b) Inserați o linie prin intermediul acestei vizualizări.  
c) Care sunt coloanele actualizabile ale acestei vizualizări? Ce fel de tabel este cel ale cărui coloane sunt actualizabile? Inserați o linie, completând doar valorile corespunzătoare.  
d) Ce efect are o operație de ștergere prin intermediul vizualizării VIZ\_EMP\_DEP30\_PNU? Comentați.
6. Să se creeze vizualizarea VIZ\_DEPT\_SUM\_PNU, care conține codul departamentului și pentru fiecare departament salariul minim, maxim și media salariilor. Ce fel de vizualizare se obține (complexă sau simplă)? Se poate actualiza vreo coloană prin intermediul acestei vizualizări?
7. Modificați vizualizarea VIZ\_EMP30\_PNU astfel încât să nu permită modificarea sau inserarea de linii ce nu sunt accesibile ei. Vizualizarea va selecta și coloana

department\_id. Dați un nume constrângerii și regăsiți-o în vizualizarea *USER\_CONSTRAINTS* din dicționarul datelor. Încercați să modificați și să inserați linii ce nu îndeplinesc condiția department\_id = 30.

8. a) Definiți o vizualizare, VIZ\_EMP\_S\_PNU, care să conțină detalii despre angajații corespunzători departamentelor care încep cu litera S. Se pot inseră/actualiza linii prin intermediul acestei vizualizări? În care dintre tabele? Ce se întâmplă la stergerea prin intermediul vizualizării?
- b) Recreați vizualizarea astfel încât să nu se permită nici o operație asupra tabelelor de bază prin intermediul ei. Încercați să introduceți sau să actualizați înregistrări prin intermediul acestei vizualizări.
9. Să se consulte informații despre vizualizările utilizatorului curent. Folosiți vizualizarea dicționarului datelor *USER\_VIEWS* (coloanele *VIEW\_NAME* și *TEXT*).

```
SELECT view_name, text
FROM user_views
WHERE view_name LIKE '%PNU';
```

10. Să se selecteze numele, salariul, codul departamentului și salariul maxim din departamentul din care face parte, pentru fiecare angajat. Este necesară o vizualizare inline?
11. Să se creeze o vizualizare VIZ\_SAL\_PNU, ce conține numele angajaților, numele departamentelor, salariile și locațiile (orașele) pentru toți angajații. Etichetați sugestiv coloanele. Considerați ca tabele de bază tabelele originale din schema HR. Care sunt coloanele actualizabile?
12. a) Să se creeze vizualizarea V\_EMP\_PNU asupra tabelului EMP\_PNU care conține codul, numele, prenumele, email-ul și numărul de telefon ale angajaților companiei. Se va impune unicitatea valorilor coloanei email și constrângerea de cheie primară pentru coloana corespunzătoare codului angajatului.

**Observație:** Constrângerile asupra vizualizărilor pot fi definite numai în modul *DISABLE NOVALIDATE*. Aceste cuvinte cheie trebuie specificate la declararea constrângerii, nefiind permisă precizarea altor stări.

```
CREATE VIEW viz_emp_pnu (employee_id, first_name, last_name,
                           email          UNIQUE
DISABLE NOVALIDATE, phone_number,
CONSTRAINT pk_viz_emp_pnu PRIMARY KEY (employee_id) DISABLE
NOVALIDATE)
AS SELECT employee_id, first_name, last_name, email, phone_number
FROM emp_pnu;
```

- b) Să se adauge o constrângere de cheie primară asupra vizualizării VIZ\_EMP\_S\_PNU.

- 13.** Să se implementeze în două moduri constrângerea ca numele angajaților nu pot începe cu sirul de caractere „Wx”.

Metoda 1:

```
ALTER TABLE emp_pnu
ADD CONSTRAINT ck_name_emp_pnu
CHECK (UPPER(last_name) NOT LIKE 'WX%');
```

Metoda 2:

```
CREATE OR REPLACE VIEW viz_emp_wx_pnu
AS SELECT *
  FROM      emp_pnu
  WHERE      UPPER(last_name) NOT LIKE 'WX%'
WITH CHECK OPTION CONSTRAINT ck_name_emp_pnu2;
UPDATE    viz_emp_wx_pnu
SET        nume = 'Wxyz'
WHERE      employee_id = 150;
```

#### IV. [Definirea secvențelor]

- Secvența este un obiect al bazei de date ce permite generarea de întregi unici pentru a fi folosiți ca valori pentru cheia primară sau coloane numerice unice. Secvențele sunt independente de tabele, așa că aceeași secvență poate fi folosită pentru mai multe tabele.
- Crearea secvențelor se realizează prin comanda *CREATE SEQUENCE*, a cărei sintaxă este:

***CREATE SEQUENCE nume\_secv***

***[INCREMENT BY n]***

***[START WITH n]***

***[{MAXVALUE n / NOMAXVALUE}]***

***[{MINVALUE n / NOMINVALUE}]***

***[{CYCLE / NOCYCLE}]***

***[{CACHE n / NOCACHE}]***

- La definirea unei secvențe se pot specifica:

- numele secvenței
- diferența dintre 2 numere generate succesiv, implicit fiind 1 (*INCREMENT BY*);
- numărul inițial, implicit fiind 1 (*START WITH*);
- valoarea maximă, implicit fiind  $10^{27}$  pentru o secvență ascendentă și -1 pentru una descendentă;

- valoarea minimă, implicit fiind 1 pentru o secvență ascendentă și  $-10^{27}$  pentru o secvență descendente;
- dacă secvența ciclează după ce atinge limita; (*CYCLE*);
- câte numere să încarce în *cache*, implicit fiind încărcate 20 de numere (*CACHE*).
- Informații despre secvențe găsim în dicționarul datelor. Pentru secvențele utilizatorului curent, interogăm *USER\_SEQUENCES*. Alte vizualizări utile sunt *ALL\_SEQUENCES* și *DBA\_SEQUENCES*.
- Pseudocoloanele *NEXTVAL* și *CURRVAL* permit lucrul efectiv cu secvențele.
  - *Nume\_secv.NEXTVAL* – returnează următoarea valoare a secvenței, o valoare unică la fiecare referire. Trebuie aplicată cel puțin o dată înainte de a folosi *CURRVAL*;
  - *Nume\_secv.CURRVAL* – obține valoarea curentă a secvenței.

**Observație:** Pseudocoloanele se pot utiliza în:

- lista *SELECT* a comenziilor ce nu fac parte din subcereri;
- lista *SELECT* a unei cereri ce apare într-un *INSERT*;
- clauza *VALUES* a comenzi *INSERT*;
- clauza *SET* a comenzi *UPDATE*.

**Observație:** Pseudocoloanele nu se pot utiliza:

- în lista *SELECT* a unei vizualizări;
- într-o comandă *SELECT* ce conține *DISTINCT*, *GROUP BY*, *HAVING* sau *ORDER BY*;
- într-o subcerere în comenziile *SELECT*, *UPDATE*, *DELETE*
- în clauza *DEFAULT* a comenziilor *CREATE TABLE* sau *ALTER TABLE*.
- Stergerea secvențelor se face cu ajutorul comenzi *DROP SEQUENCE*.

***DROP SEQUENCE nume\_secventa;***

## V. [Exerciții – secvențe]

14. Creați o secvență pentru generarea codurilor de departamente, *SEQ\_DEPT\_PNU*. Secvența va începe de la 400, va crește cu 10 de fiecare dată și va avea valoarea maximă 10000, nu va cicla și nu va încărca nici un număr înainte de cerere.
15. Să se selecteze informații despre secvențele utilizatorului curent (nume, valoare minimă, maximă, de incrementare, ultimul număr generat).
16. Creați o secvență pentru generarea codurilor de angajați, *SEQ\_EMP\_PNU*.

- 17.** Să se modifice toate liniile din EMP\_PNU (dacă nu mai există, îl recreeați), regenerând codul angajaților astfel încât să utilizeze secvența SEQ\_EMP\_PNU și să avem continuitate în codurile angajaților.
- 18.** Să se insereze câte o înregistrare nouă în EMP\_PNU și DEPT\_PNU utilizând cele 2 secvențe create.
- 19.** Să se selecteze valorile curente ale celor 2 secvențe.

```
SELECT seq_emp_pnu.currval  
FROM dual ;
```

- 20.** Ștergeți secvența SEQ\_DEPT\_PNU.