Baze de date-Anul 2
Laborator 1 SQL

I. Introducere. Comanda SELECT. Cereri mono-relație

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

Un sistem de gestiune a bazei de date (SGBD) este un produs software care asigură interacțiunea cu o bază de date, permițând definirea, consultarea şi actualizarea datelor din baza de date.

2. Ce este SQL?

SQL (*Structured Query Language*) este un limbaj neprocedural pentru interogarea şi prelucrarea informațiilor din baza de date.

Compilatorul limbajului *SQL* generează automat o procedură care accesează baza de date şi execută comanda dorită.

SQL permite atât definirea, prelucrarea şi interogarea datelor, cît şi controlul accesului la acestea. Comenzile *SQL* pot fi integrate în programe scrise în alte limbaje, de exemplu *Cobol*, *C*, *C++*, *Java* etc.

3. 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 comenzile componente, unele dintre aceste categorii sunt evidențiate ca limbaje în cadrul *SQL*, şi anume:

- limbajul de definire a datelor (*LDD*) – comenzile CREATE, ALTER, DROP;
- limbajul de prelucrare a datelor (*LMD*) – comenzile INSERT, UPDATE, DELETE, SELECT;
- limbajul de control al datelor (*LCD*) – comenzile COMMIT, ROLLBACK.

Pe lângă comenzile care alcătuiesc aceste limbaje, *SQL* cuprinde:

- instrucțiuni pentru controlul sesiunii;
- instrucțiuni pentru controlul sistemului;
- instrucțiuni *SQL* încapsulate.

4. Analizați sintaxa simplificată a comenzii SELECT:

SELECT { [{***DISTINCT*** | ***UNIQUE***} | ***ALL***] *lista_campuri* | *}
FROM [*nume_schemă*.]*nume_obiect*]
[, [*nume_schemă*.]*nume_obiect* …]
[***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_obiect*.]*nume_coloană*
[, [[*nume_schemă*.]*nume_obiect*.]*nume_coloană*] …]
[***NOWAIT*** | ***WAIT*** *număr_întreg*]];

Un element din *lista_campuri* are forma: *expresie [AS] alias*.

Care dintre clauze sunt obligatorii?

5. Care sunt regulile de scriere a comenzilor SQL (acceptă abrevieri, e nevoie de caracter de terminare)? In instructiunea urmatoare sunt 3 erori. Care sunt acestea?

SQL> SELECT employee_id, last_name
2 salary x 12 ANNUAL SALARY
3 FROM employees;

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

Dacă un alias conține *blank*-uri, el va fi scris obligatoriu între ghilimele. Altfel, ghilimelele pot fi omise. *Alias*-ul apare în rezultat, ca şi cap de coloană pentru expresia respectivă. Doar cele specificate între ghilimele sunt *case-sensitive*, celelalte fiind scrise implicit cu majuscule.

II. Exerciții

1. a) Consultați diagrama exemplu HR (Human Resources) pentru lucrul în cadrul laboratoarelor SQL.

b) Identificați cheile primare şi cele externe ale tabelelor existente în schemă, precum şi tipul relațiilor dintre aceste tabele.

2. Să se inițieze o sesiune SQL*Plus folosind *user ID*-ul şi parola 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, afişând valorile tuturor câmpurilor.

Obs: *SELECT * FROM nume_tabel;*

5. Să se afişeze codul angajatului, numele, codul job-ului, data angajarii. Ce fel de operație este aceasta (selecție sau proiecție)?.

6. Să se listeze, cu şi fără duplicate, codurile job-urilor din tabelul *EMPLOYEES*.

SELECT job_id FROM employees;
SELECT DISTINCT job_id FROM employees;

7. Să se afişeze numele concatenat cu job_id-ul, separate prin virgula si spatiu, si etichetati coloana “Angajat si titlu”.

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

SQL> SELECT last_name|| ‘, ‘ || job_id “Angajat si titlu”
FROM employees;

8. Creati o cerere prin care sa se afiseze toate datele din tabelul *EMPLOYEES*. Separați fiecare coloană printr-o virgulă. Etichetati coloana ”Informatii complete”.

9. Sa se listeze numele si salariul angajaților care câştigă mai mult de 2850 $.

SELECT last_name, salary
FROM employees
WHERE salary > 2850;

10. Să se creeze o cerere pentru a afişa numele angajatului şi numărul departamentului pentru angajatul nr. 104.

11. Să se afişeze numele şi salariul pentru toți angajații al căror salariu nu se află în domeniul 1500-2850$.

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

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

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

13. Să se afişeze numele salariaților şi codul departamentelor pentru toti angajații din departamentele 10 şi 30 în ordine alfabetică a numelor.

SELECT __, __
FROM ___
____ department_id IN (10, 30)
_____ _____;

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

expresie IN (valoare_1, valoare_2, …, valoare_n)

14. Să se afişeze numele şi salariile angajatilor care câştigă mai mult de 3500 $ şi lucrează în departamentul 10 sau 30. Se vor eticheta coloanele drept *Angajat* si *Salariu lunar*.

15. Care este data curentă? Afişați diferite formate ale acesteia.

Obs:

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

SELECT SYSDATE
FROM dual;

- Datele calendaristice pot fi formatate 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; …)
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, YYY, YYYY	Ultimele 2, 3, respectiv 4 cifre din an.
YEAR	Anul, scris în litere (ex: *two thousand four*).
HH12, HH24	Orele din zi, între 0-12, respectiv 0-24.
MI	Minutele din oră.
SS	Secundele din minut.
SSSSS	Secundele trecute de la miezul nopții.

16. Sa se afiseze numele şi data angajării pentru fiecare salariat care a fost angajat in 1987. Se cer 2 soluții: una în care se lucrează cu formatul implicit al datei şi alta prin care se formatează data.

Varianta1:

```
SELECT first_name, last_name, hire_date
FROM employees
WHERE hire_date LIKE ('%87%');
```

Varianta 2:

```
SELECT first_name, last_name, hire_date
FROM employees
WHERE  TO_CHAR(hire_date, 'YYYY')='1987';
```

Sunt obligatorii ghilimelele de la şirul '1987'? Ce observați?

17. Să se afişeze numele şi job-ul pentru toți angajații care nu au manager.

18. Sa se afiseze numele, salariul si comisionul pentru toti salariatii care castiga comisioane. Sa se sorteze datele in ordine descrescatoare a salariilor si comisioanelor.

19. Eliminați clauza WHERE din cererea anterioară. Unde sunt plasate valorile NULL în ordinea descrescătoare?

20. Să se listeze numele tuturor angajatilor care au a treia literă din nume 'A'.

Obs: Pentru compararea şirurilor de caractere, împreună cu operatorul LIKE se utilizează caracterele *wildcard*:

- % - reprezentând orice şir de caractere, inclusiv şirul vid;
- _ (*underscore*) – reprezentând un singur caracter şi numai unul.

21. Să se listeze numele tuturor angajatilor care au 2 litere 'L' in nume şi lucrează în departamentul 30 sau managerul lor este 101.

22. Să se afiseze numele, job-ul si salariul pentru toti salariatii al caror job conține şirul "clerk" sau "rep" si salariul nu este egal cu 1000, 2000 sau 3000 $. (operatorul NOT IN)

23. Să se afiseze numele, salariul si comisionul pentru toti angajatii al caror salariu este mai mare decat comisionul (*salary*commission_pct*) marit de 5 ori.

Baze de date-Anul 2
Laborator 2 SQL

Funcții *SQL.* Cereri multi-relație (introducere)

I. [Funcții SQL]

Funcțiile *SQL* sunt predefinite în sistemul *Oracle* şi pot fi utilizate în instrucțiuni *SQL*. Ele nu trebuie confundate cu funcțiile definite de utilizator, scrise în *PL*/*SQL*.

Dacă o funcție *SQL* este apelată cu un argument având un alt tip de date decât cel aşteptat, sistemul converteşte implicit argumentul înainte să evalueze funcția.

Dacă o funcție *SQL* este apelată cu un argument *null*, ea returnează automat valoarea *null*. Singurele funcții care nu urmează această regulă sunt *CONCAT*, *NVL* şi *REPLACE*.

Principalele funcții *SQL* pot fi clasificate în următoarele categorii:

- Funcții *single-row*
- Funcții *multiple-row* (funcții agregat)

1. Funcțiile *single row* returnează câte o singură linie rezultat pentru fiecare linie a tabelului sau vizualizării interogate. Aceste funcții pot apărea în listele *SELECT*, clauzele *WHERE*, *START WITH*, *CONNECT BY* şi *HAVING*. În ceea ce priveşte tipul argumentelor asupra cărora operează şi al rezultatelor furnizate, funcțiile *single row* pot fi clasificate în clase corespunzătoare.

- **Funcțiile de conversie** cele mai importante sunt:

Funcție	Descriere	Exemplu conversie
TO_CHAR	converteşte (sau formatează) un număr sau o dată calendaristică în şir de caractere	*TO_CHAR(7) = ' 7'* *TO_CHAR(-7) = '-7'* *TO_CHAR(SYSDATE, 'DD/MM/YYYY') = ' 02/12/2005'*
TO_DATE	converteşte (sau formatează) un număr sau un şir de caractere în dată calendaristică	*TO_DATE('02-DEC-2005','dd-mon-yyyy')*
TO_NUMBER	converteşte (sau formatează) un şir de caractere în număr	*TO_NUMBER ('-25789', 'S99,999') = -25,789*

Obs: Există două tipuri de conversii:

- ***implicite***, realizate de sistem atunci când este necesar;
- ***explicite***, indicate de utilizator prin intermediul funcțiilor 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*.

- **Funcțiile pentru prelucrarea caracterelor** sunt prezentate în următorul tabel:

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

	fiind transformate în minuscule	
INSTR(string, 'chars' [,start [,n]])	caută în *string*, începând de de la poziția *start*, a *n*-a apariție a secvenței *chars* şi întoarce poziția respectivă; dacă *start* nu este specificat, căutarea se face de la începutul şirului; dacă *n* nu este specificat, se caută prima apariție a secvenței *chars*;	*INSTR (LOWER('AbC aBcDe'), 'ab', 5, 2) = 0* *INSTR (LOWER('AbCdE aBcDe'), 'ab', 5) = 7*
ASCII(char)	furnizează codul *ASCII* al primului caracter al unui şir	*ASCII ('alfa') = ASCII ('a') = 97*
CHR(num)	întoarce caracterul corespunzător codului *ASCII* specificat	*CHR(97)= 'a'*
CONCAT(string1, string2)	realizează concatenarea a două şiruri de caractere	*CONCAT ('In', 'fo') = 'Info'*
TRANSLATE(string, source, destination)	fiecare caracter care apare în şirurile de caractere *string* şi *source* este transformat în caracterul corespunzător (aflat pe aceeaşi poziție ca şi în *source*) din şirul de caractere *destination*	*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* şi vom da numai apelul funcției şi rezultatul returnat.

- **Funcțiile aritmetice** *single-row* pot opera asupra:
  - unei singure valori, şi 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* şi *GREATEST*, care întorc cea mai mică, respectiv cea mai mare valoare a unei liste de expresii.

- **Funcțiile pentru prelucrarea datelor calendaristice** sunt:

Funcție	Descriere	Exemplu
SYSDATE	întoarce data şi timpul curent	*SELECT SYSDATE FROM dual;* (de revăzut utilizarea acestei funcții împreună cu *TO_CHAR* în cadrul laboratorului 1)
ADD_MONTHS(expr_date, nr_luni)	întoarce data care este după *nr_luni* luni de la data *expr_date*;	*ADD_MONTHS('02-DEC-2005', 3)* = '02-MAR-2006'.
NEXT_DAY(expr_date, day)	întoarce următoarea dată după data *expr_date*, a cărei zi a săptămânii este cea specificată	*NEXT_DAY('02-DEC-2005', 'Monday') = '05-DEC-2005'*

	prin şirul de caractere *day*	
LAST_DAY(expr_date)	întoarce data corespunzătoare ultimei zile a lunii din care data *expr_date* face parte	*LAST_DAY('02-DEC-2005') = '31-DEC-2005'*
MONTHS_BETWEEN(expr_date2, expr_date1)	întoarce numărul de luni dintre cele două date calendaristice specificate. Data cea mai recentă trebuie specificată în primul argument, altfel rezultatul este negativ.	*MONTHS_BETWEEN('02-DEC-2005', '10-OCT-2002') = 37.7419355* *MONTHS_BETWEEN('10-OCT-2002', '02-DEC-2005') = -37.7419355*
TRUNC(expr_date)	întoarce data *expr_date*, dar cu timpul setat la ora 12:00 AM (miezul nopții)	*TO_CHAR(TRUNC(SYSDATE), 'dd/mm/yy HH24:MI') = '02/12/05 00:00'*
ROUND(expr_date)	dacă data *expr_date* este înainte de miezul zilei, întoarce data *d* cu timpul setat la ora 12:00 AM; altfel, este returnată data corespunzătoare zilei următoare, cu timpul setat la ora 12:00 AM	TO_CHAR(ROUND(SYSDATE*), 'dd/mm/yy hh24:mi am') =* '*03/12/05 00:00 AM'*
LEAST(d1, d2, …, dn), GREATEST(d1, d2, …, dn)	dintr-o listă de date calendaristice, funcțiile întorc prima, respectiv ultima dată în ordine cronologică	*LEAST(SYSDATE, SYSDATE + 3, SYSDATE – 5) = SYSDATE-5* *GREATEST(SYSDATE, SYSDATE + 3, SYSDATE – 5) = SYSDATE + 3*

Operațiile care se pot efectua asupra datelor calendaristice sunt următoarele:

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

- **Funcții diverse:**

Funcție	**Descriere**	**Exemplu**
DECODE(value, if1, then1, if2, then2, … , ifN, thenN, else)	returnează *then1* dacă *value* este egală cu *if1*, *then2* dacă value este egală cu *if2* etc.; dacă *value* nu este egală cu nici una din valorile *if*, atunci funcția întoarce valoarea	*DECODE ('a', 'a', 'b', 'c') = 'b'* *DECODE ('b', 'a', 'b', 'c') = 'c'* *DECODE ('c', 'a', 'b', 'c') =*

	else;	*'c'*
NVL(expr_1, expr_2)	dacă *expr_1* este *NULL,* întoarce *expr_2*; altfel, întoarce *expr_1*. Tipurile celor două expresii trebuie să fie compatibile sau *expr_2* să poată fi convertit implicit la *expr_1*	*NVL(NULL, 1) = 1* *NVL(2, 1) = 2* *NVL('a', 1) = 'a' -- conversie implicită* *NVL(1, 'a') -- eroare --nu are loc conversia implicită*
NVL2(expr_1, expr_2, expr_3)	dacă *expr_1* este *NOT NULL*, întoarce *expr_2*, altfel întoarce *expr_3*	*NVL2(1, 2, 3) = 2* *NVL2 (NULL, 1, 2) = 2*
NULLIF (expr_1, expr_2)	Daca *expr_1 = expr_2* atunci funcția returnează *NULL*, altfel returnează expresia *expr_1*. Echivalent cu *CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END*	*NULLIF (1, 2) = 1* *NULLIF (1,1) = NULL*
COALESCE (expr_1, expr_2, ... , expr_n)	Returnează prima expresie NOT NULL din lista de argumente.	*COALESCE (NULL, NULL, 1, 2, NULL) = 1*
UID, *USER*	întorc *ID*-ul, respectiv *username*-ul utilizatorului *ORACLE* curent	*SELECT USER* *FROM dual;*
VSIZE(expr)	întoarce numărul de octeți ai unei expresii de tip *DATE*, *NUMBER* sau *VARCHAR2*	*SELECT VSIZE(salary)* *FROM employees* *WHERE employee_id=200;*

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

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

2. Funcțiile multiple-row (agregat) pot fi utilizate pentru a returna informația corespunzătoare fiecăruia dinte grupurile obținute în urma divizării liniilor tabelului cu ajutorul clauzei GROUP BY. Ele pot apărea în clauzele *SELECT*, *ORDER BY* şi *HAVING*. *Server*-ul *Oracle* aplică aceste funcții fiecărui grup de linii şi 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* şi *VARIANCE* operează numai asupra valorilor numerice. Funcțiile *MAX* şi *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 şi 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 scrie în clauza *WHERE* a instrucțiunii *SELECT*. Într-o instrucțiune *SELECT* care uneşte tabele prin operația de *join*, se recomandă ca numele coloanelor să fie precedate de numele sau alias-urile tabelelor pentru claritate şi pentru îmbunătățirea timpului de acces la baza de date. Dacă acelaşi nume de coloană apare în mai mult de două tabele, atunci numele coloanei se prefixează **obligatoriu** 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 şi completată în cadrul laboratorului 3.

III. [Exerciții]

[Funcții pe şiruri 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 şi operatorul “||”.
2. Scrieți o cerere prin care să se afişeze prenumele salariatului cu prima litera majusculă şi toate celelalte litere mici, numele acestuia cu majuscule şi 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 corespunzător. Se cer 2 soluții (cu operatorul *LIKE* şi funcția *SUBSTR*).
3. Să se afişeze pentru angajații cu prenumele „Steven”, codul, numele şi 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 afişeze pentru toți angajații al căror nume se termină cu litera 'e', codul, numele, lungimea numelui şi poziția din nume în care apare prima data litera 'a'. Utilizați *alias*-uri corespunzătoare pentru coloane.

[Funcții aritmetice]

5. Să se afişeze detalii despre salariații care au lucrat un număr întreg de săptămâni până la data curentă.
6. Să se afişeze codul salariatului, numele, salariul, salariul mărit cu 15%, exprimat cu două zecimale şi numărul de sute al salariului nou rotunjit 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 şi data angajării salariaților care câştigă comision. Să se eticheteze coloanele „Nume angajat”, „Data angajarii”. Pentru a nu obține *alias*-ul datei angajării trunchiat, utilizați funcția *RPAD*.

[Funcții şi operații cu date calendaristice]

8. Să se afişeze data (numele lunii, ziua, anul, ora, minutul si secunda) de peste 30 zile.
9. Să se afişeze numărul de zile rămase până la sfârşitul anului.
10. a) Să se afişeze data de peste 12 ore.

    b) Să se afişeze data de peste 5 minute.
11. Să se afişeze numele şi prenumele angajatului (într-o singură coloană), data angajării şi data negocierii salariului, care este prima zi de Luni după 6 luni de serviciu. Etichetați această coloană “Negociere”.
12. Pentru fiecare angajat să se afişeze numele şi 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.
13. Să se afişeze numele, data angajării şi ziua săptămânii în care a început lucrul fiecare salariat. Etichetați coloana “Zi”. Ordonați rezultatul după ziua săptămânii, începând cu Luni.

[Funcții diverse]

14. Să se afişeze numele angajaților şi comisionul. Dacă un angajat nu câştigă comision, să se scrie “Fara comision”. Etichetați coloana “Comision”.
15. Să se listeze numele, salariul şi comisionul tuturor angajaților al căror venit lunar depăşeşte 10000$.

[Instrucțiunea CASE, comanda DECODE]

16. Să se afişeze numele, codul job-ului, salariul şi o coloană care să arate salariul după mărire. Se presupune că pentru IT_PROG are loc o mărire de 20%, pentru SA_REP creşterea 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 afişeze numele salariatului, codul şi numele departamentului pentru toți angajații.

Obs: Numele sau alias-urile tabelelor sunt obligatorii în dreptul coloanelor care au acelaşi 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 titlurile job-urile care există în departamentul 30.
19. Să se afişeze numele angajatului, numele departamentului şi locatia pentru toți angajații care câştigă comision.

20. Să se afişeze numele salariatului şi numele departamentului pentru toți salariații care au litera A inclusă în nume.
21. Să se afişeze numele, job-ul, codul şi numele departamentului pentru toți angajații care lucrează în Oxford.
22. Să se afişeze codul angajatului şi numele acestuia, împreună cu numele şi codul şefului său direct. Se vor eticheta coloanele Ang#, Angajat, Mgr#, Manager.
23. Să se modifice cererea precedenta pentru a afişa toți salariații, inclusiv cei care nu au şef.
24. Creați o cerere care să afişeze numele angajatului, codul departamentului şi toți salariații care lucrează în acelaşi departament cu el. Se vor eticheta coloanele corespunzător.
25. Să se listeze structura tabelului JOBS. Creați o cerere prin care să se afişeze numele, codul job-ului, titlul job-ului, numele departamentului şi salariul angajaților.
26. Să se afişeze numele şi data angajării pentru salariații care au fost angajați după *Gates*.
27. Să se afişeze numele salariatului şi data angajării împreună cu numele şi data angajării şefului direct pentru salariații care au fost angajați înaintea şefilor lor. Se vor eticheta coloanele Angajat, Data_ang, Manager si Data_mgr.

Baze de date-Anul 2
Laborator 3 SQL

Interogări multi-relație. Operația de join. Operatori pe mulțimi. Subcereri necorelate.

I. [Obiective]

În acest laborator vom continua lucrul cu interogări ***multi-relație*** (acestea sunt cele care regăsesc date din mai multe tabele). Am introdus deja diferite tipuri de ***join***. Vom relua aceste operații, vom analiza şi o altă metodă de implementare a lor şi de asemenea vom utiliza ***operatori pe mulțimi*** şi ***subcereri necorelate*** (fără sincronizare).

Foarte utile în rezolvarea exercițiilor propuse vor fi ***funcțiile SQL***, prezentate în laboratorul 2.

II. [Join]

Am implementat deja operația de ***join*** (compunere a tabelelor) în cadrul unor exemple relative la modelul luat în considerare (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***.

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 egalitate.
- ***Left | Right | Full Outer join*** – un *outer join* este utilizat pentru a obține în rezultat şi înregistrările care nu satisfac condiția de *join*. Operatorul pentru *outer join* este semnul plus inclus între paranteze **(+)**, care se plasează 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 şi cărora nu le corespunde nicio 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.

  Full outer join – left outer join + right outer join.

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

Obs: Un caz special al operației de join este ***self join*** – join-ul unui tabel cu el însuşi. În ce situație concretă (relativ la modelul nostru) apare această operație?

Obs : *Alias*-urile pot fi utilizate oriunde, ca o notație mai scurtă, în locul denumirii tabelului. Ele pot avea lungimea de maxim 30 de caractere, dar este recomandat să fie scurte şi sugestive. Dacă este atribuit un alias unui tabel din clauza FROM, atunci el trebuie să înlocuiască aparițiile numelui tabelului în instrucțiunea SELECT.

Un alias poate fi utilizat pentru a califica denumirea unei coloane. Calificarea unei coloane cu numele sau alias-ul tabelului se poate face :

- opțional, pentru claritate şi pentru îmbunătățirea timpului de acces la baza de date;
- obligatoriu, ori de câte ori există o ambiguitate privind sursa coloanei. Ambiguitatea constă, de obicei, în existența unor coloane cu acelaşi nume în mai

mult de două tabele.

Join* în standardul SQL3 (SQL:1999)*:

Pentru *join*, sistemul *Oracle* oferă şi o sintaxă specifică, introdusă de către standardul *SQL3* (*SQL*:*1999*). Această sintaxă nu aduce beneficii în privința performanței față de *join*-urile care se specifică în clauza *WHERE*. Tipurile de *join* conforme cu *SQL3* sunt definite prin cuvintele cheie *CROSS JOIN* (pentru produs cartezian), *NATURAL JOIN*, *FULL OUTER JOIN*, clauzele *USING* şi *ON*.

Sintaxa corespunzătoare standardului *SQL3* este următoarea:

SELECT *tabel_1.nume_coloană*, *tabel_2.nume_coloană*
FROM *tabel_1*
[***CROSS JOIN*** *tabel_2*]
| [***NATURAL JOIN*** *tabel_2*]
| [***JOIN*** *tabel_2* ***USING*** (*nume_coloană*)]
| [***JOIN*** *tabel_2* ***ON*** (*tabel_1.nume_coloană = tabel_2.nume_coloană*)]
| [***LEFT*** | ***RIGHT*** | ***FULL OUTER JOIN*** *tabel_2*
ON (*tabel_1.nume_coloană = tabel_2.nume_coloană*)];

- *NATURAL JOIN* presupune existența unor coloane având acelaşi 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 acelaşi nume în cele două tabele trebuie să nu fie precedate de numele sau *alias*-ul tabelului corespunzător.

- *JOIN tabel_2 USING nume_coloană* efectuează un *equijoin* pe baza coloanei cu numele specificat în sintaxă. Această clauză este utilă dacă există coloane având acelaşi nume, dar tipuri de date diferite. Coloanele referite în clauza *USING* ***trebuie să nu conțină calificatori*** (să nu fie precedate de nume de tabele sau *alias*-uri) în nicio apariție a lor în instrucțiunea *SQL*. Clauzele *NATURAL JOIN* şi *USING* nu pot coexista în aceeaşi instrucțiune *SQL*.

- *JOIN tabel_2 ON tabel_1.nume_coloană = tabel_2.nume_coloană* efectuează un *equijoin* 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*).

- *LEFT*, *RIGHT* şi *FULL OUTER JOIN tabel_2 ON* (*tabel_1.nume_coloană = tabel_2.nume_coloană*) efectuează *outer join* la stânga, dreapta, respectiv în ambele părți pe baza condiției exprimate în clauza *ON*.

Un *join* care returnează rezultatele unui *inner join*, dar şi cele ale *outer join*-urilor la stânga şi la dreapta se numeşte *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* şi *MINUS*.

Toți operatorii pe mulțimi au aceeaşi 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* şi are precedență mai mică decât operatorul *IN*.
- Operatorul ***UNION ALL*** returnează toate liniile selectate de două cereri, fără a elimina duplicatele. Precizările făcute asupra operatorului *UNION* sunt valabile şi 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 şi 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 duplicat.
- În instrucțiunile *SELECT* asupra cărora se aplică operatori pe mulțimi, coloanele selectate trebuie să corespundă ca număr şi 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. [Subcereri]

Prin intermediul subcererilor se pot construi interogări complexe pe baza unor instrucțiuni simple.

O subcerere (subinterogare) este o comandă *SELECT* integrată într-o clauză a altei instrucțiuni *SQL*, numită instrucțiune „părinte“ sau instrucțiune exterioară. Subcererile mai sunt numite instrucțiuni *SELECT* imbricate sau interioare.

Rezultatele subcererii sunt utilizate în cadrul cererii exterioare, pentru a determina rezultatul final. În funcție de modul de evaluare a subcererii în raport cu cererea exterioară, subcererile pot fi:

- nesincronizate (necorelate) sau
- sincronizate (corelate).

Prima clasă de subcereri este evaluată dinspre interior către exterior, adică interogarea externă acționează pe baza rezultatului cererii interne. Al doilea tip de subcerere este evaluat invers, adică interogarea externă furnizează valori cererii interne, iar rezultatele subcererii sunt transferate cererii externe.

- Subcererile nesincronizate care apar în clauza *WHERE* a unei interogări sunt de forma următoare:

```
SELECT   expresie1, expresie2, ...
FROM     nume_tabel1
WHERE    expresie_condiție operator (SELECT   expresie
                                     FROM     nume_tabel2);
```

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

➢ Subcererile sincronizate care apar în clauza *WHERE* a unei interogări au următoarea formă generală:

SELECT *expresie_ext_1*[, *expresie_ext_2 …*]
FROM *nume_tabel_1 extern*
WHERE *expresie_condiție operator*
(***SELECT*** *expresie*
FROM *nume_tabel_2*
WHERE *expresie = extern.expresie_ext*);

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

Obs: *operator* poate fi:

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

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

Cuvintele cheie *ANY* şi *ALL* pot fi utilizate cu subcererile care produc o singură coloană de valori. Dacă subcererea este precedată de către cuvântul cheie *ALL*, atunci condiția va fi adevărată numai dacă este satisfăcută de către toate valorile produse de subcerere. Astfel, *<ALL* are semnificația „mai mic decât minimul“, iar *>ALL* este echivalent cu „mai mare decât maximul“. Dacă subcererea este precedată de către cuvântul cheie *ANY*, condiția va fi adevărată dacă este satisfăcută de către oricare (una sau mai multe) dintre valorile produse de subcerere. În comparații, *<ANY* are semnificația „mai mic decât maximul“; *>ANY* înseamnă „mai mare decât minimul“; *=ANY* este echivalent cu operatorul *IN*.

Dacă subcererea returnează mulțimea vidă, atunci condiția *ALL* va returna valoarea *true*, iar condiția *ANY* va returna valoarea *false*. Standardul *ISO* permite utilizarea cuvântului cheie *SOME*, în locul lui *ANY*.

V. [Exerciții - join]

1. Scrieți o cerere pentru a se afişa numele, luna (în litere) şi anul angajării pentru toți salariații din acelaşi 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 din metode se va da şi varianta join-ului conform standardului *SQL99*.

2. Să se afişeze codul şi numele angajaților care lucrează în acelasi departament cu cel puțin un angajat al cărui nume conține litera “t”. Se vor afişa, de asemenea, codul şi

numele departamentului respectiv. Rezultatul va fi ordonat alfabetic după nume. Se vor da 2 soluții pentru join (condiție în clauza WHERE şi sintaxa introdusă de standardul SQL3).

3. Sa se afiseze numele, salariul, titlul job-ului, oraşul şi țara în care lucrează angajatii condusi direct de King.

4. Sa se afiseze codul departamentului, numele departamentului, numele si job-ul tuturor angajatilor din departamentele al căror nume conține şirul 'ti'. De asemenea, se va lista salariul angajaților, în formatul "$99,999.00". Rezultatul se va ordona alfabetic după numele departamentului, şi în cadrul acestuia, după numele angajaților.

5. Sa se afiseze numele angajatilor, numarul departamentului, numele departamentului, oraşul si job-ul tuturor salariatilor al caror departament este localizat in Oxford.

6. Afisati codul, numele si salariul tuturor angajatilor care castiga mai mult decat salariul mediu pentru job-ul corespunzător si lucreaza intr-un departament cu cel putin unul din angajatii al caror nume contine litera "t".

Obs: Salariul mediu pentru un job se va considera drept media aritmetică a valorilor minime şi maxime admise pentru acesta (media valorilor coloanelor min_salary şi max_salary).

7. Să se afişeze numele salariaților şi numele departamentelor în care lucrează. Se vor afişa şi salariații care nu au asociat un departament. (right outer join, 2 variante).

8. Să se afişeze numele departamentelor şi numele salariaților care lucrează în ele. Se vor afişa şi departamentele care nu au salariați. (left outer join, 2 variante)

9. Cum se poate implementa full outer join?

Obs: Full outer join se poate realiza fie prin reuniunea rezultatelor lui right outer join şi left outer join, fie utilizând sintaxa introdusă de standardul SQL99.

VI. [Exerciții - operatori pe mulțimi]

10. Se cer codurile departamentelor al căror nume conține şirul "re" sau în care lucrează angajați având codul job-ului "SA_REP". Cum este ordonat rezultatul?

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

12. Sa se obtina codurile departamentelor in care nu lucreaza nimeni (nu este introdus niciun salariat in tabelul employees). Se cer două soluții.

13. Se cer codurile departamentelor al căror nume conține şirul "re" şi în care lucrează angajați având codul job-ului "HR_REP".

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

VII. [Exercitii - subcereri necorelate (nesincronizate)]

15. Folosind subcereri, să se afişeze numele şi data angajării pentru salariații care au fost angajați după Gates.

16. Folosind subcereri, scrieți o cerere pentru a afişa numele şi salariul pentru toți colegii (din acelaşi departament) lui Gates. Se va exclude Gates.

? Se putea pune ”=” în loc de ”IN”? In care caz nu se poate face această înlocuire?

17. Folosind subcereri, să se afişeze numele şi salariul angajaților conduşi direct de preşedintele companiei (acesta este considerat angajatul care nu are manager).

18. Scrieti o cerere pentru a afişa numele, codul departamentului si salariul angajatilor al caror număr de departament si salariu coincid cu numarul departamentului si salariul unui angajat care castiga comision.

19. Rezolvați problema 6 utilizând subcereri.

20. Scrieti o cerere pentru a afisa angajatii care castiga mai mult decat oricare functionar (job-ul conține şirul “CLERK”). Sortati rezultatele dupa salariu, in ordine descrescatoare. (ALL). Ce rezultat este returnat dacă se înlocuieşte “ALL” cu “ANY”?

21. Scrieți o cerere pentru a afişa numele, numele departamentului şi salariul angajaților care nu câştigă comision, dar al căror şef direct coincide cu şeful unui angajat care câştigă comision.

22. Sa se afiseze numele, departamentul, salariul şi job-ul tuturor angajatilor al caror salariu si comision coincid cu salariul si comisionul unui angajat din Oxford.

23. Să se afişeze numele angajaților, codul departamentului şi codul job-ului salariaților al căror departament se află în Toronto.

Baze de date-Anul 2
Laborator 4 SQL

Funcții grup şi clauzele GROUP BY, HAVING. Operatorii ROLLUP şi CUBE.

I. [Funcții grup şi 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. Ele pot apărea în clauzele:
    - *SELECT*
    - *ORDER BY*
    - *HAVING*.

Server-ul *Oracle* aplică aceste funcții fiecărui grup de linii şi 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* şi *VARIANCE* operează numai asupra valorilor numerice.
    - Funcțiile *MAX* şi *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 şi 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.
- În clauza *GROUP BY* a unei cereri se pot utiliza operatorii *ROLLUP* şi *CUBE*. Aceşti operatori sunt disponibili începând cu versiunea *Oracle8i*.
- 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).

II. [Clauza HAVING]

Opțiunea *HAVING* 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*.

III. [Operatorul ROLLUP]

Acest operator furnizează valori agregat şi superagregat corespunzătoare expresiilor din clauza *GROUP BY*. Operatorul *ROLLUP* poate fi folosit pentru extragerea de statistici şi informații totalizatoare din mulțimile rezultate. Acest operator poate fi util la generarea de rapoarte, diagrame şi grafice.

Operatorul *ROLLUP* creează grupări prin deplasarea într-o singură direcție, de la dreapta la stânga, de-a lungul listei de coloane specificate în clauza *GROUP BY*. Apoi, se aplică funcția agregat acestor grupări. Dacă sunt specificate *n* expresii în operatorul *ROLLUP*, numărul de grupări generate va fi *n* + 1. Liniile care se bazează pe valoarea primelor *n* expresii se numesc linii obişnuite, iar celelalte se numesc linii superagregat.

Dacă în clauza *GROUP B Y* sunt specificate *n* coloane, pentru a produce subtotaluri fără operatorul *ROLLUP* ar fi necesare *n* + 1 instrucțiuni *SELECT* conectate prin *UNION ALL* . Aceasta ar face execuția cererii ineficientă pentru că fiecare instrucțiune *SELECT* determină accesarea tabelului. Operatorul *ROLLUP* determină rezultatele efectuând un singur acces la tabel şi este util atunci când sunt implicate multe coloane în producerea subtotalurilor.

Ilustrăm aplicarea acestui operator prin urmatorul exemplu.

Exemplu:

Pentru departamentele având codul mai mic decât 50, să se afişeze:

- pentru fiecare departament şi pentru fiecare an al angajării (corespunzător departamentului respectiv), valoarea totală a salariilor angajaților în acel an;
- valoarea totală a salariilor pe departamente (indiferent de anul angajării);
- valoarea totală a salariilor (indiferent de anul angajării şi de departament).

```
SELECT   department_id, TO_CHAR(hire_date, 'yyyy'), SUM(salary)
FROM    employees
WHERE   department_id < 50
GROUP BY ROLLUP(department_id, TO_CHAR(hire_date, 'yyyy'));
```

Instrucțiunea precedentă va avea un rezultat de forma:

DEPARTMENT_ID	TO_CHAR(hire_date,'yyyy')	SUM(SALARY)
10	1987	4400
10		4400
20	1996	13000
20	1997	6000
20		19000
30	1994	11000
30	1995	3100
30	1997	5700
30	1998	2600
30	1999	2500
30		24900
40	1994	6500
40		6500
		54800

În rezultatul prezentat anterior se pot distinge 3 tipuri de linii.

1) Prima linie reprezintă suma salariilor angajaților în 1987 din departamentul care are codul 10. În mod similar se interpretează liniile din rezultat care au toate coloanele completate.

2) Linia a doua conține valoarea totală a salariilor din departamentul al cărui cod este 10. La fel se interpretează toate liniile care se disting prin faptul că valoarea coloanei *TO_CHAR(hire_date, 'dd')* este *null*.

3) Ultima linie conține suma salariilor tuturor angajatiilor din departamentele al căror cod este mai mic decat 50. Întrucât această linie corespunde totalului general, ea conține valoarea *null* pe toate coloanele, cu excepția câmpului *SUM*(*salary*)*.*

IV. [Operatorul CUBE]

Operatorul *CUBE* grupează liniile selectate pe baza valorilor tuturor combinațiilor posibile ale expresiilor specificate şi returnează câte o linie totalizatoare pentru fiecare grup. Acest operator este folosit pentru a produce mulțimi de rezultate care sunt utilizate în rapoarte. În vreme ce *ROLLUP* produce subtotalurile doar pentru o parte dintre combinațiile posibile, operatorul *CUBE* produce subtotaluri pentru toate combinațiile posibile de grupări specificate în clauza *GROUP BY*, precum şi un total general.

Dacă există *n* coloane sau expresii în clauza *GROUP BY*, vor exista 2^n combinații posibile superagregat. Din punct de vedere matematic, aceste combinații formează un cub *n*-dimensional, de aici provenind numele operatorului. Pentru producerea de subtotaluri fără ajutorul operatorului *CUBE* ar fi necesare 2^n instrucțiuni *SELECT* conectate prin *UNION ALL*.

Exemplu:

Pentru departamentele având codul mai mic decât 50 să se afişeze:

- valoarea totală a salariilor corespunzătoare fiecărui an de angajare, din cadrul fiecărui departament;
- valoarea totală a salariilor din fiecare departament (indiferent de anul angajarii);
- valoarea totală a salariilor corespunzatoare fiecărui an de angajare (indiferent de departament);
- valoarea totală a salariilor (indiferent de departament si de anul angajarii).

```
SELECT   department_id, TO_CHAR(hire_date, 'yyyy'), SUM(salary)
FROM    employees
WHERE   department_id < 50
GROUP BY CUBE(department_id, TO_CHAR(hire_date, 'yyyy'));
```

În plus față de rezultatul corespunzător operației *ROLLUP*, operatorul *CUBE* va produce linii care reprezintă suma salariilor pentru fiecare an de angajare corespunzător unui departament având codul mai mic decât 50. Aceste linii se disting prin faptul că valoarea coloanei *department_id* este *null*.

V. [Exerciții – funcții grup şi clauzele GROUP BY, HAVING]

1. a) Functiile grup includ valorile *NULL* in calcule?
   b) Care este deosebirea dintre clauzele *WHERE* şi *HAVING*?
2. Să se afişeze cel mai mare salariu, cel mai mic salariu, suma şi media salariilor tuturor angajaților. Etichetați coloanele Maxim, Minim, Suma, respectiv Media. Sa se rotunjeasca rezultatele.
3. Să se afişeze minimul, maximul, suma şi media salariilor pentru fiecare job.
4. Să se afişeze numărul de angajați pentru fiecare job.
5. Să se determine numărul de angajați care sunt şefi. Etichetati coloana "Nr. manageri".
6. Să se afişeze diferența dintre cel mai mare si cel mai mic salariu. Etichetati coloana.

7. Scrieți o cerere pentru a se afişa numele departamentului, locația, numărul de angajați şi salariul mediu pentru angajații din acel departament. Coloanele vor fi etichetate corespunzător.

!!!Obs: În clauza *GROUP BY* se trec obligatoriu toate coloanele prezente în clauza *SELECT*, care nu sunt argument al funcțiilor grup.

8. Să se afişeze codul şi numele angajaților care câstiga mai mult decât salariul mediu din firmă. Se va sorta rezultatul în ordine descrescătoare a salariilor.

9. Pentru fiecare şef, să se afişeze codul său şi salariul celui mai prost platit subordonat. 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.

10. Pentru departamentele in care salariul maxim depăşeşte 3000$, să se obțină codul, numele acestor departamente şi salariul maxim pe departament.
11. Care este salariul mediu minim al job-urilor existente? Salariul mediu al unui job va fi considerat drept media arirmetică a salariilor celor care îl practică.

!!!Obs: Într-o imbricare de funcții agregat, criteriul de grupare specificat în clauza GROUP BY se referă doar la funcția agregat cea mai interioară. Astfel, într-o clauză SELECT în care există funcții agregat imbricate **nu mai pot apărea alte expresii**.

12. Să se afişeze codul, numele departamentului şi suma salariilor pe departamente.

13. Să se afişeze maximul salariilor medii pe departamente.

14. Să se obtina codul, titlul şi salariul mediu al job-ului pentru care salariul mediu este minim.

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

16. Să se afişeze suma salariilor pe departamente şi, în cadrul acestora, pe job-uri.

17. Să se afişeze numele departamentului si cel mai mic salariu din departamentul avand cel mai mare salariu mediu.

18. Sa se afiseze codul, numele departamentului si numarul de angajati care lucreaza in acel departament pentru:

    a) departamentele in care lucreaza mai putin de 4 angajati;
    b) departamentul care are numarul maxim de angajati.

19. Sa se afiseze salariatii care au fost angajati în aceeaşi zi a lunii în care cei mai multi dintre salariati au fost angajati.

20. Să se obțină numărul departamentelor care au cel puțin 15 angajați.

21. Să se obțină codul departamentelor şi suma salariilor angajaților care lucrează în acestea, în ordine crescătoare. Se consideră departamentele care au mai mult de 10 angajați şi al căror cod este diferit de 30.

22. Sa se afiseze codul, numele departamentului, numarul de angajati si salariul mediu din departamentul respectiv, impreuna cu numele, salariul si jobul angajatilor din acel departament. Se vor afişa şi departamentele fără angajați (outer join).

23. Scrieti o cerere pentru a afisa, pentru departamentele avand codul > 80, salariul total pentru fiecare job din cadrul departamentului. Se vor afisa orasul, numele departamentului, jobul si suma salariilor. Se vor eticheta coloanele corespunzator.

Obs: Plasați condiția *department_id* > 80, pe rând, în clauzele *WHERE* şi *HAVING*. Testați în fiecare caz. Ce se observă? Care este diferența dintre cele două abordări?

24. Care sunt angajatii care au mai avut cel putin doua joburi?

25. Să se calculeze comisionul mediu din firmă, luând în considerare toate liniile din tabel.

VI. [Exerciții – ROLLUP şi CUBE]

26. Analizați cele 2 exemple prezentate mai sus (III – IV), referitor la operatorii *ROLLUP* şi *CUBE*.

VII. [Exerciții – DECODE, subcereri în clauza SELECT]

27. Scrieți o cerere pentru a afişa job-ul, salariul total pentru job-ul respectiv pe departamente si salariul total pentru job-ul respectiv pe departamentele 30, 50, 80.
    Se vor eticheta coloanele corespunzător. Rezultatul va apărea sub forma de mai jos:

    ```
    Job          Dep30       Dep50   Dep80     Total
    ---------------------------------------------------------------------------
    ……
    ```

28. Să se creeze o cerere prin care să se afişeze numărul total de angajați şi, din acest total, numărul celor care au fost angajați în 1997, 1998, 1999 si 2000. Denumiti capetele de tabel in mod corespunzator.

29. Rezolvați problema 22 cu ajutorul subcererilor specificate în clauza SELECT.

VIII. [Exerciții – subcereri în clauza FROM]

Obs: Subcererile pot apărea în clauza SELECT, WHERE sau FROM a unei cereri.
O subcerere care apare în clauza FROM se mai numeşte view in-line.

30. Să se afişeze codul, numele departamentului şi suma salariilor pe departamente.

31. Să se afişeze numele, salariul, codul departamentului si salariul mediu din departamentul respectiv.

32. Modificați cererea anterioară, pentru a determina şi listarea numărului de angajați din departamente.

33. Pentru fiecare departament, să se afişeze numele acestuia, numele şi salariul celor mai prost plătiți angajați din cadrul său.

34. Rezolvați problema 22 cu ajutorul subcererilor specificate în clauza FROM.

Baze de date-Anul 2
Laborator 5 SQL

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

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

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

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

Obs:

→ Lista de expresii care urmează operatorului ROLLUP este parcursă de la dreapta la stânga, suprimându-se câte o expresie .

→ O cerere în care apare un astfel de operator este echivalentă cu reuniunea (UNION ALL) a n+1 cereri.

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

- Pentru determinarea modului în care a fost obținută o valoare totalizatoare cu ROLLUP sau CUBE, se utilizează funcția:
  - ***GROUPING***(expresie)

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

- Dacă se doreşte obținerea numai a anumitor grupări superagregat, acestea pot fi precizate prin intermediul clauzei :
  - ***GROUPING SETS*** ((expr_11, expr_12, …, expr_1n), (expr_21, expr_22, …expr_2m), …)

Exerciții:

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

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

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

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

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

4. Să se afişeze salariul maxim al angajatilor doar daca acesta este mai mare decât 15000.

II. [Subcereri corelate (sincronizate)]

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

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

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

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

Obs: O subcerere (corelată sau necorelată) poate apărea în clauzele:
- SELECT
- FROM (vezi laboratorul 4)
- WHERE
- HAVING (vezi laboratorul 4)
- START WITH (vezi mai jos – la cereri ierarhice)

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

Exerciții:

5. a) Să se afişeze informații despre angajații al căror salariu depăşeşte valoarea medie a salariilor colegilor săi de departament.
   b) Analog cu cererea precedentă, afişându-se şi numele departamentului şi media salariilor acestuia şi numărul de angajați (2 solutii: subcerere necorelată în clauza FROM, subcerere corelată în clauza SELECT).

6. Să se afişeze numele şi salariul angajaților al căror salariu este mai mare decât salariile medii din toate departamentele. Se cer 2 variante de rezolvare: cu operatorul ALL sau cu funcția MAX.

7. Sa se afiseze numele si salariul celor mai prost platiti angajati din fiecare departament (se cer 3 solutii: subcerere sincronizata, subcerere nesincronizata si subcerere în clauza FROM).

8. Pentru fiecare departament, să se obtina numele salariatului avand cea mai mare vechime din departament. Să se ordoneze rezultatul după numele departamentului.

9. Sa se obtina numele salariatilor care lucreaza intr-un departament in care exista cel putin un angajat cu salariul egal cu salariul maxim din departamentul 30 (operatorul exists).

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

10. Sa se obtina numele primilor 3 angajati avand salariul maxim. Rezultatul se va afişa în ordine crescătoare a salariilor.

11. Să se afişeze codul, numele şi prenumele angajaților care au cel puțin doi subalterni.

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

Obs: Ca alternativă a lui EXISTS, poate fi utilizat operatorul IN. Scrieți şi această solutie.

13. Să se determine departamentele în care nu există nici un angajat (operatorul exists; cererea a mai fost rezolvata si printr-o cerere necorelata).

III. [Subcereri ierarhice]

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

• *START WITH* specifică o condiție care identifică liniile ce urmează să fie considerate ca rădăcini ale cererii ierarhice respective. Dacă se omite această clauză, sistemul Oracle utilizează toate liniile din tabel drept linii rădăcină.

• *CONNECT BY* specifică o condiție care identifică relația dintre liniile „părinte" şi „copil" ale ierarhiei. Condiția trebuie să conțină operatorul *PRIOR* pentru a face referință la linia „părinte".

• Operatorul *PRIOR* face referință la linia „părinte". Plasarea acestui operator determină direcția interogării, dinspre „părinte" spre „copil" (*top-down*) sau invers (*bottom-up*). Traversarea *top-down*, respectiv *bottom-up* a arborelui se realizează prin specificări de forma următoare:

Top-down: ***CONNECT BY PRIOR*** *cheie_parinte = cheie_copil;*
Bottom-up: ***CONNECT BY PRIOR*** *cheie_copil = cheie_parinte;*

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

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

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

Exerciții:

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

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

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

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

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

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

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

IV. [Clauza WITH]

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

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

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

V . [Analiza top-n]

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

21. Să se detemine primii 10 cei mai bine plătiți angajați.

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

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

Obs:

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

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

CASE poate avea si forma:

```
CASE
     WHEN expr_logica_1 THEN val_2
     WHEN expr_logica_3 THEN val_4
     …
     ELSE default
END
```

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

24. Să se afişeze numele, prenumele angajaților şi lungimea numelui pentru înregistrările în care aceasta este diferită de lungimea prenumelui. (*NULLIF*)

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

26. Să se afişeze:

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

Se poate folosi DECODE?

Baze de date-Anul 2
Laborator 6 SQL

I. Operatorul *DIVISION*.
II. Variabile de substitutie

I. Implementarea operatorului DIVISION în *SQL*

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

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

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

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

Extindem diagrama *HR* cu o nouă entitate, *PROJECT*, şi o nouă asociere: „angajat lucreaza în cadrul unui proiect”, între entitățile *EMPLOYEES* şi *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* şi *PROJECT* este „angajat conduce proiect”. Aceasta este o relație *one-to-many*.

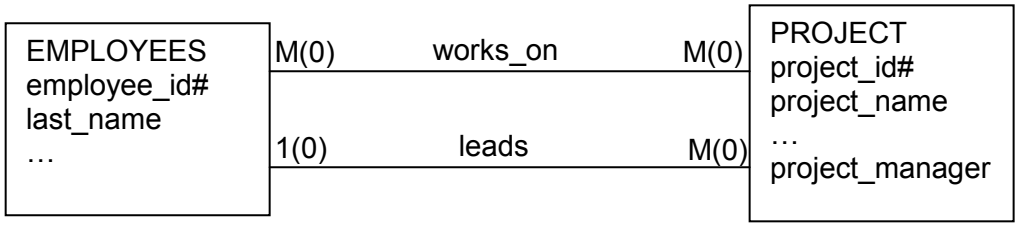
Noile tabele au următoarele scheme relaționale:

1) *PROJECT*(project_id#, project_name, budget, start_date, deadline, delivery_date, project_manager)
- project_id reprezintă codul proiectului şi 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 şi este cheie externă. Pe cine referă această coloană ? Ce relație implementează această cheie externă?

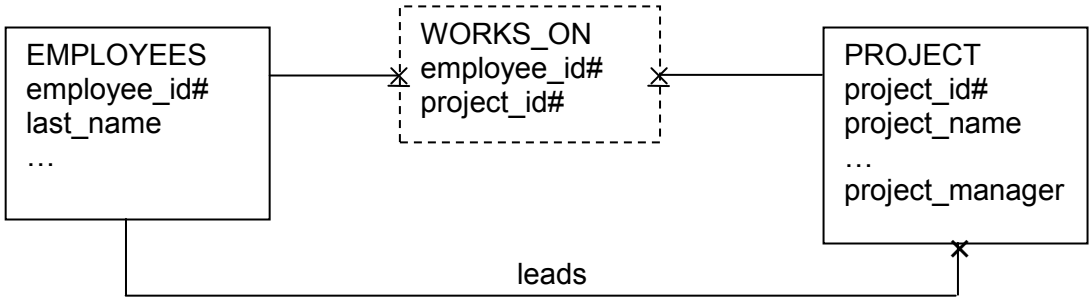
2) *WORKS_ON*(project_id#, employee_id#, start_date, end_date)
- cheia primară a relației este compusă din atributele employee_id şi project_id.

Scriptul pentru crearea noilor tabele şi 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 ataşați tuturor proiectelor pentru care s-a alocat un buget egal cu 10000.

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

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

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

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

Metoda 3 (operatorul *MINUS*):

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

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

Metoda 4 (A include B => B\A = Ø):

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

Exerciții (*DIVISION* + alte cereri):

1. Să se listeze informații despre angajații care au lucrat în toate proiectele demarate în primele 6 luni ale anului 2006. 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 şi job-ul curent.
4. Pentru fiecare țară, să se afişeze numărul de angajați din cadrul acesteia.
5. Să se listeze angajații (codul şi numele acestora) care au lucrat pe cel puțin două proiecte nelivrate la termen.
6. Să se listeze codurile angajaților şi codurile proiectelor pe care au lucrat. Listarea va cuprinde şi angajații care nu au lucrat pe nici un proiect.
7. Să se afişeze angajații care lucrează în acelaşi departament cu cel puțin un manager de proiect.
8. Să se afişeze angajații care nu lucrează în acelaşi departament cu nici un manager de proiect.
9. Să se determine departamentele având media salariilor mai mare decît un număr dat.

Obs : Este necesară o variabilă de substituție. Apariția acesteia este indicată prin caracterul „&”. O prezentare a variabilelor de substituție va fi făcută în a doua parte a acestui laborator.
HAVING AVG(salary) > &p;

10. Se cer informații (nume, prenume, salariu, număr proiecte) despre managerii de proiect care au condus 2 proiecte.
11. Să se afişeze 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 aceleaşi proiecte ca şi angajatul având codul 200.
    b) Să se obțină numele angajaților care au lucrat **cel mult** pe aceleaşi proiecte ca şi angajatul având codul 200.

Obs: Incluziunea dintre 2 mulțimi se testează cu ajutorul proprietății „A inclus în B => A-B =Ø".

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

Obs: Egalitatea între două mulțimi se testează cu ajutorul proprietății „A=B => A-B=Ø şi B-A=Ø".

14. Modelul HR conține un tabel numit JOB_GRADES, care conține grilele de salarizare ale companiei.
    a) Afişați structura şi conținutul acestui tabel.
    b) Pentru fiecare angajat, afişați numele, prenumele, salariul şi grila de salarizare corespunzătoare. Ce operație are loc între tabelele din interogare?

II. Variabile de substitutie

- Variabilele de substitutie sunt utile in crearea de comenzi/script-uri dinamice (care depind de unele valori pe care utilizatorul le furnizeaza la momentul rularii).
- Variabilele de substitutie se pot folosi pentru stocarea temporara de valori, transmiterea de valori intre comenzi *SQL* etc. Ele pot fi create prin:
    - comanda *DEFINE*.(*DEFINE* variabila = valoare;)
    - Prefixarea cu & (indica existenta unei variabile intr-o comanda *SQL*, daca variabila nu exista, atunci ea este creata).
    - Prefixarea cu && (indica existenta unei variabile intr-o comanda *SQL*, daca variabila nu exista, atunci *ea este creata*). Deosebirea fata de & este ca, daca se foloseste &&, atunci referirea ulterioara cu & sau && nu mai cere ca utilizatorul sa introduca de fiecare data valoarea variabilei. Este folosita valoarea data la prima referire.

Variabilele de substitutie pot fi eliminate cu ajutorul comenzii *UNDEF*[*INE*]

Comanda *DEFINE*

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

Observatii:

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

Comenzi interactive in scripturi

Comanda	Descriere
ACC[*EPT*] *variabila* [*tip*] [*PROMPT text*]	Citeşte o linie de intrare şi o stochează într-o variabilă utilizator.
PAU[*SE*] [*text*]	Afişează o linie vidă, urmată de o linie conținând text, apoi aşteaptă ca utilizatorul să apese tasta *return*. De asemenea, această comandă poate lista două linii vide, urmate de aşteptarea răspunsului din partea utilizatorului.
PROMPT [*text*]	Afişează mesajul specificat sau o linie vidă pe ecranul utilizatorului.

Exercitii (*SQL*Plus*)

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

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;
```

16. Sa se afiseze numele, codul departamentului si salariul anual pentru toti angajatii care au un anumit job.

17. Sa se afiseze numele, codul departamentului si salariul anual pentru toti angajatii care au fost angajati dupa o anumita data calendaristica.

18. Sa se afiseze o coloana aleasa de utilizator, dintr-un tabel ales de utilizator, ordonand dupa aceeasi coloana care se afiseaza. De asemenea, este obligatorie precizarea unei conditii WHERE.

19. Să se realizeze un script prin care să se afişeze numele, job-ul şi data angajării salariatilor care au început lucrul între 2 date calendaristice introduse de utilizator. Să se concateneze numele si job-ul, separate prin spatiu si virgulă, si să se eticheteze coloana "Angajati".
Se vor folosi comanda ACCEPT si formatul pentru data calendaristica MM/DD/YY.

20. Sa se realizeze un script pentru a afisa numele angajatului, codul job-ului, salariul si numele departamentului pentru salariatii care lucreaza intr-o locatie data de utilizator. Va fi permisa cautarea case-insensitive.

21. a)Să se citească două date calendaristice de la tastatură si să se afiseze zilele dintre aceste două date.

b)Modificati cererea anterioară astfel încât să afiseze doar zilele lucrătoare dintre cele două date calendaristice introduse.

Baze de date-Anul 2
Laborator 7 SQL

Limbajul de manipulare a datelor (LMD)
Limbajul de control al datelor (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 consistența datelor pe baza tranzacțiilor, inclusiv în eventualitatea unei anomalii a unui proces sau a sistemului. Tranzacțiile oferă mai multă flexibilitate şi control în modificarea datelor.
- Comenzile SQL care alcătuiesc **LCD** sunt:
  - ROLLBACK – pentru a renunța la modificările aflate în aşteptare se utilizează instrucțiunea *ROLLBACK*. În urma execuției acesteia, se încheie tranzacția, se anulează modificările asupra datelor, se restaurează starea lor precedentă şi se eliberează blocările asupra liniilor.
  - COMMIT - determină încheierea tranzacției curente şi permanentizarea modificărilor care au intervenit pe parcursul acesteia. Instrucțiunea suprimă toate punctele intermediare definite în tranzacție şi eliberează blocările tranzacției.

    Obs: 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*.

I. Comanda *INSERT*

1. Inserări mono-tabel

Comanda INSERT are următoarea sintaxă simplificată:

INSERT INTO *obiect* [***AS*** *alias*] [(*nume_coloană* [, *nume_coloană …*])]
{***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 şi 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 şi dată calendaristică trebuie incluse între apostrofuri. Nu se recomandă includerea între apostrofuri a valorilor numerice, întrucât aceasta ar determina conversii implicite 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 şirurilor de caractere sau al datelor calendaristice se poate preciza şirul vid ('').

Observații (erori):

Server-ul *Oracle* aplică automat toate tipurile de date, domeniile de valori şi 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 duplicat 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. Inserari 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, introdus de *Oracle9i*, este util în mediul *data warehouse*.

Pentru o astfel de inserare, în versiunile anterioare lui *Oracle9i* erau necesare *n* operații independente *INSERT INTO…SELECT…*, unde *n* reprezintă numărul tabelelor destinație. Aceasta presupunea *n* procesări ale aceleiaşi surse de date şi, prin urmare, creşterea de *n* ori a timpului necesar procesului.

Sintaxa comenzii *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.

Exerciții [I]

1. Să se creeze tabelele *EMP_pnu*, *DEPT_pnu* (în şirul de caractere “pnu”, *p* reprezintă prima literă a prenumelui, iar *nu* reprezintă primele două litere ale numelui dumneavoastră), prin copierea structurii şi 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 şi 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);

Obs: Ce constrângere nu am implementat?

5. Să se insereze departamentul 300, cu numele *Programare* în *DEPT_pnu*.
   Analizați cazurile, precizând care este soluția corectă şi 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);

Executați varianta care a fost corectă de două ori. Ce se obține şi 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). Efectele instrucțiunii să devină permanente.

7. Să se mai introducă un angajat corespunzător departamentului 300, precizând după numele tabelului lista coloanelor în care se introduc valori (metoda explicita 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? (view) Încercați dacă este posibilă introducerea unui angajat, precizând pentru valoarea employee_id o subcerere care returnează (codul maxim +1).

9. Creați un nou tabel, numit *EMP1_PNU*, care va avea aceeaşi structură ca şi *EMPLOYEES*, dar nici o înregistrare. Copiați în tabelul *EMP1_PNU* salariații (din tabelul *EMPLOYEES*) al căror comision depăşeşte 25% din salariu.

10. Inserați o nouă înregistrare în tabelul *EMP_PNU* care să totalizeze salariile, să facă media comisioanelor, iar câmpurile de tip dată să conțină data curentă şi câmpurile de tip caracter să conțină textul 'TOTAL'. Numele şi prenumele angajatului să corespundă utilizatorului curent (*USER*). Pentru câmpul *employee_id* se va introduce valoarea 0, iar pentru *manager_id* şi *department_id* se va da valoarea null.

11. Să se creeze un fişier (script file) care să permită introducerea de înregistrări în tabelul EMP_PNU în mod interactiv. Se vor cere utilizatorului: codul, numele, prenumele si salariul angajatului. Câmpul email se va completa automat prin concatenarea primei litere din prenume şi a primelor 7 litere din nume. Executati script-ul pentru a introduce 2 inregistrari in tabel.

12. Creați 2 tabele emp2_pnu şi emp3_pnu cu aceeaşi 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 şi 10000;
    - în tabelul EMP3_PNU salariații care au salariul mai mare decât 10000.

    Verificați rezultatele, apoi ştergeți toate înregistrările din aceste tabele.

13. Să se creeze tabelul EMP0_PNU cu aceeaşi structură ca tabelul EMPLOYEES (fără constrângeri), dar fără nici o înregistrare. Copiați din tabelul EMPLOYEES:
    - în tabelul EMP0_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 şi 10000;
    - în tabelul EMP3_PNU salariații care au salariul mai mare decât 10000.

    Dacă un salariat se încadrează în tabelul emp0_pnu atunci acesta nu va mai fi inserat şi în alt tabel (tabelul corespunzător salariului său).

II. Comanda UPDATE

Sintaxa simplificată a comenzii *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:

- de obicei pentru identificarea unei linii se foloseşte 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 eşuează instrucțiunea INSERT. Acestea au fost menționate anterior.

Exerciții [II]

14. Măriți salariul tuturor angajaților din tabelul *EMP_PNU* cu 5%. Vizualizati, iar apoi anulați modificările.

15. Schimbați jobul tuturor salariaților din departamentul 80 care au comision în 'SA_REP'. Anulați modificările.

16. Să se promoveze Douglas Grant la manager în departamentul 20, având o creştere de salariu cu 1000$. Se poate realiza modificarea prin intermediul unei singure comenzi?

17. Schimbați salariul şi comisionul celui mai prost plătit salariat din firmă, astfel încât să fie egale cu salariul si comisionul şefului său.

18. Să se modifice adresa de e-mail pentru angajații care câştigă 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.

19. 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.

20. Să se modifice jobul şi departamentul angajatului având codul 114, astfel încât să fie la fel cu cele ale angajatului având codul 205.

21. Creați un script prin intermediul caruia sa fie posibilă actualizarea în mod interactiv de înregistrări ale tabelului dept_pnu. Se va cere codul departamentului care urmează a fi actualizat, se va afişa linia respectivă, iar apoi se vor cere valori pentru celelalte câmpuri.

III. Comanda DELETE

Sintaxa simplificată a comenzii ***DELETE*** este:

```
DELETE FROM nume_tabel
WHERE conditie];
```

Daca nu se specifica nici o conditie, vor fi şterse toate liniile din tabel.

Exerciții [III]

22. Ştergeți toate înregistrările din tabelul DEPT_PNU. Ce înregistrări se pot şterge?

23. Ştergeți angajații care nu au comision. Anulați modificările.

24. Suprimați departamentele care un au nici un angajat. Anulați modificările.

25. Să se creeze un fişier script prin care se cere utilizatorului un cod de angajat din tabelul EMP_PNU. Se va lista inregistrarea corespunzatoare acestuia, iar apoi linia va fi suprimată din tabel.

26. Să se mai introducă o linie in tabel, rulând încă o dată fişierul creat la exercițiul 11.

27. Să se marcheze un punct intermediar in procesarea tranzacției.

28. Să se şteargă tot conținutul tabelului. Listați conținutul tabelului.

29. Să se renunțe la cea mai recentă operație de ştergere, fără a renunța la operația precedentă de introducere.

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

IV. Comanda MERGE

Instrucțiunea *MERGE* permite inserarea sau actualizarea condiționată a datelor dintr-un tabel al bazei de date. Sintaxa ei simplificată este următoarea:

MERGE INTO nume_tabel [*alias*]
USING {*tabel* | *vizualizare* | *subcerere*} [*alias*]
ON (*condiție*)
WHEN MATCHED THEN
    UPDATE SET
        coloana_1 = {*expr_u1* | *DEFAULT*},…,
        coloana_n = {*expr_un* | *DEFAULT*}
WHEN NOT MATCHED THEN
INSERT (coloana_1,…, coloana_n)
VALUES (expr_i1,…, expr_in);

Instrucțiunea efectuează:

- *UPDATE* dacă înregistrarea există deja în tabel
- *INSERT* dacă înregistrarea este nouă.

În acest fel, se pot evita instrucțiunile *UPDATE* multiple.

Exerciții [IV]

31. Să se şteargă din tabelul EMP_PNU toți angajații care câştigă comision.
    Să se introducă sau să se actualizeze datele din tabelul EMP_PNU pe baza tabelului employees.

Baze de date - Anul 2
Laborator 8

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

- În general, instrucțiunile *LDD* sunt utilizate pentru definirea structurii corespunzătoare obiectelor unei scheme: tabele, vizualizări, vizualizări materializate, indecşi, sinonime, clustere, proceduri şi funcții stocate, declanşatori, pachete stocate etc.
- Aceste instrucțiuni permit:
  - crearea, modificarea şi suprimarea obiectelor unei scheme şi a altor obiecte ale bazei de date, inclusiv baza însăşi şi utilizatorii acesteia (*CREATE*, *ALTER*, *DROP*);
  - modificarea numelor obiectelor unei scheme (*RENAME*);
  - ştergerea datelor din obiectele unei scheme, fără suprimarea structurii obiectelor respective (*TRUNCATE*).
- Implicit, o instrucțiune *LDD* permanentizează (*COMMIT*) efectul tuturor instrucțiunilor precedente şi marchează începutul unei noi tranzacții.
- Instrucțiunile *LDD* au efect imediat asupra bazei de date şi înregistrează informația în dicționarul datelor.
- Definirea unui obiect presupune: crearea (CREATE), modificarea (ALTER) şi suprimarea sa (DROP).
- **Reguli de numire a obiectelor bazei de date**
  - Identificatorii obiectelor trebuie să înceapă cu o literă şi să aibă maximum 30 de caractere, cu excepția numelui bazei de date care este limitat la 8 caractere şi 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, _, *$* şi #.
  - Două obiecte ale aceluiaşi utilizator al *server*-ului *Oracle* nu pot avea acelaşi nume.
  - Identificatorii nu pot fi cuvinte rezervate ale *server*-ului *Oracle*.
  - Identificatorii obiectelor nu sunt *case-sensitive*.

Definirea tabelelor

1. Crearea tabelelor

- Formele simplificate ale comenzii 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;*

- **Constrângerile 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 - stabileşte o relație de cheie externă între o coloană a tabelului şi 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“ şi coloana corespunzătoare din acest tabel;
    - *ON DELETE CASCADE* determină ca, odată cu ştergerea unei linii din tabelul „părinte“, să fie şterse şi liniile dependente din tabelul „copil“;
    - *ON DELETE SET NULL* determină modificarea automată a valorilor cheii externe la valoarea *null*, atunci când se şterge valoarea „părinte“.
  - CHECK- o condiție care să fie adevărată la nivel de coloană sau linie (*CHECK* (*conditie*)).

Obs:
- Constrângerile pot fi create o dată cu tabelul sau adăugate ulterior cu o comandă *ALTER TABLE*.
- Constrângerile de tip CHECK se pot implementa la nivel de coloană doar dacă nu referă o altă coloană a tabelului.
- În cazul în care cheia primară (sau o cheie unică) este compusă, ea nu poate fi definită la nivel de coloane, ci doar la nivel de 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]	Defineşte un şir de caractere de dimensiune variabilă, având lungimea maximă de *n* octeți sau caractere. Valoarea maximă a lui *n* corespunde la 4000 octeți, iar cea minimă este de un octet sau un caracter.
CHAR(n) [BYTE | CHAR]	Reprezintă un şir de caractere de lungime fixă având *n* octeți sau caractere. Valoarea maximă a lui *n* corespunde la 2000 octeți. Valoarea implicită şi minimă este de un octet.
NUMBER(p, s)	Reprezintă un număr având *p* cifre, dintre care *s* cifre formează partea zecimală
LONG	Conține şiruri de caractere având lungime variabilă, care nu pot ocupa mai mult de 2GB.
DATE	Reprezintă date calendaristice valide, între 1 ianuarie 4712 i.Hr. şi 31 decembrie 9999 d.Hr.

2. Modificarea (structurii) tabelelor

- **Modificarea structurii unui tabel** se face cu ajutorul comenzii ***ALTER TABLE***. Forma comenzii 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****;***

Obs:

- dimensiunea unei coloane numerice sau de tip caracter poate fi mărită, dar nu poate fi micşorată 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 modifică 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 PRIMARY KEY | UNIQUE(*col1, col2, ...*) | ***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

- Ştergerea fizică a unui tabel, inclusiv a înregistrărilor acestuia, se realizează prin comanda:
  DROP TABLE *nume_tabel;*

- Pentru ştergerea conținutului unui tabel şi păstrarea structurii acestuia se poate utiliza comanda:
  TRUNCATE TABLE *nume_tabel;*

!!!Obs: Fiind operație *LDD*, comanda *TRUNCATE* are efect definitiv.

4. Redenumirea tabelelor

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

RENAME *nume1_obiect* ***TO*** *nume2_obiect;*

Obs:

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

5. Consultarea dicționarului datelor

Informații despre tabelele create se găsesc în vizualizările:

- USER_TABLES –informații complete despre tabelele utilizatorului.
- TAB – informații de bază despre tabelele existente în schema utilizatorului.

Informații despre constângeri găsim în USER_CONSTRAINTS, iar despre coloanele implicate în constrângeri în USER_CONS_COLUMNS.

Exerciții

1. Să se creeze tabelul ANGAJATI_pnu (pnu se alcatuieşte din prima literă din prenume şi 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 si a constrangerilor NOT NULL pentru coloanele nume şi salariu;
c) cu precizarea cheii primare la nivel de tabel si a constrângerilor NOT NULL pentru coloanele nume şi salariu.

Se presupune că valoarea implicită a coloanei data_ang este SYSDATE.

Obs: Nu pot exista două tabele cu acelaşi 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	02-02-2004	Inginer	100	10000	10
102	Nume3	Prenume3	Nume3	05-06-2000	Analist	101	5000	20
103	Nume4	Prenume4	Null	Null	Inginer	100	9000	20
104	Nume5	Prenume5	Nume5	Null	Analist	101	3000	30

Prima si a patra înregistrare vor fi introduse specificând coloanele pentru care introduceți date efectiv, iar celelalte vor fi inserate fără precizarea coloanelor în comanda INSERT.
Salvați comenzile de inserare într-un fişier l8p2.sql.

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 in 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) şi al coloanei salariu la NUMBER(10,2), în cadrul aceleiaşi instrucțiuni.
8. Actualizati valoarea coloanei comision, setând-o la valoarea 0.1 pentru salariații al căror job începe cu litera A. (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 şi 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. Se va preciza apoi cheia primara cod_dep, fără suprimarea şi recreerea tabelului (comanda ALTER).

Obs:

- Introducerea unei constrângeri după crearea tabelului, presupune ca toate liniile existente în tabel la momentul respective să satisfacă noua constrângere.
- Acest mod de specificare a constrângerilor permite numirea acestora.
- In situația in 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 şi recrearea tabelului, cu precizarea noii constrângeri la nivel de coloană ({DROP, CREATE} TABLE). De asemenea, se vor mai preciza constrângerile (la nivel de coloană, dacă 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 sa fie mai mare decat comisionul*100.
19. Suprimați şi recreați tabelul, specificând toate constrângerile la nivel de tabel (în măsura în care este posibil).
20. Reintroduceți date în tabel, utilizând (şi modificând, dacă este necesar fişierul l8p2.sql.
21. Ce se întâmplă dacă se încearcă suprimarea tabelului departamente_pnu?
22. Analizați structura vizualizărilor USER_TABLES, TAB, USER_CONSTRAINTS.

Obs: 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 şi tabel) despre constrângerile asupra tabelelor angajati_pnu şi departamente_pnu.

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

Obs: 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* şi *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. (Incercați 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 Analiza, codul 60 şi directorul null în DEPARTAMENTE_pnu. COMMIT.
27. (Incercați să) ştergeți departamentul 20 din tabelul DEPARTAMENTE_pnu. Comentați.

28. Ştergeți departamentul 60 din DEPARTAMENTE_pnu. ROLLBACK.
29. (Incercați să) introduceți un nou angajat, specificând valoarea 114 pentru cod_sef. Ce se obține?
30. Adăugați un nou angajat, având codul 114. Incercați 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 doreşte ştergerea 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 şi reintroduceți această constrângere, specificând clauza ON DELETE CASCADE.
32. Ştergeți departamentul 20 din DEPARTAMENTE_pnu. Ce se întâmplă? Rollback.
33. Introduceți constrângerea de cheie externă asupra coloanei *cod_director* a tabelului DEPARTAMENTE_pnu. Se doreşte ca ştergerea unui angajat care este director de departament să atragă după sine 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. Ştergeți angajatul având codul 102 din tabelul ANGAJATI_pnu. Analizați efectele comenzii. 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 depăşi 30000.
36. Incercați actualizarea salariului angajatului 100 la valoarea 35000.
37. Dezactivați constrângerea creată anterior şi reîncercați actualizarea. Ce se întâmplă dacă încercăm reactivarea constrângerii?

Baze de date-Anul 2
Laborator 9 SQL

Limbajul de definire a datelor (LDD) - II :

Definirea tabelelor temporare, a vizualizărilor, secvențelor, indecşilor, clusterelor, sinonimelor, vizualizărilor materializate

I. Definirea tabelelor temporare

Pentru crearea tabelelor temporare, se utilizează următoarea formă a comenzii *CREATE TABLE*:

CREATE GLOBAL TEMPORARY TABLE [*schema.*]*nume_tabel*
({*nume_coloană_1 tip_date* [***DEFAULT*** *expresie*]
[*constr_coloană* [*constr_coloană*] …]
| *constr_tabel_sau_view* }
[, {*nume_coloană_2 …* }])
[***ON COMMIT*** {***DELETE*** | ***PRESERVE***} ***ROWS***;

- Un tabel temporar stochează date numai pe durata unei tranzacții sau a întregii sesiuni.
- Definiția unui tabel temporar este accesibilă tuturor sesiunilor, dar informațiile dintr-un astfel de tabel sunt vizibile numai sesiunii care inserează linii în acesta.
- Tabelelor temporare nu li se alocă spațiu la creare decât dacă s-a folosit clauza "*AS subcerere*"; altfel, spațiul este alocat la prima instrucțiune "*INSERT*" care a introdus linii în el. De aceea, dacă o instrucțiune *DML*, inclusiv "*SELECT*", este executată asupra tabelului înaintea primului "*INSERT*", ea vede tabelul ca fiind vid.
- Precizarea opțiunii *ON COMMIT* determină dacă datele din tabelul temporar persistă pe durata unei tranzacții sau a unei sesiuni :
  - Clauza *DELETE ROWS* se utilizează pentru definirea unui tabel temporar specific unei tranzacții, caz în care sistemul trunchiază tabelul, ştergând toate liniile acestuia după fiecare operație de permanentizare (*COMMIT*).
  - Clauza *PRESERVE ROWS* se specifică pentru a defini un tabel temporar specific unei sesiuni, caz în care sistemul trunchiază tabelul la terminarea sesiunii.
- O sesiune este ataşată unui tabel temporar dacă efectuează o operație *INSERT* asupra acestuia. Detaşarea sesiunii de un tabel temporar are loc :
  - în urma execuției unei comenzi *TRUNCATE*,
  - la terminarea sesiunii sau
  - prin efectuarea unei operații *COMMIT*, respectiv *ROLLBACK* asupra unui tabel temporar specific tranzacției.
- Comenzile *LDD* pot fi efectuate asupra unui tabel temporar doar dacă nu există nici o sesiune ataşată acestuia.

Exerciții [I]

1. Creați un tabel temporar *TEMP_TRANZ_PNU*, cu datele persistente doar pe durata unei tranzacții. Acest tabel va conține o singură coloană *x*, de tip *NUMBER*. Introduceți o înregistrare în tabel. Listați conținutul tabelului. Permanentizați tranzacția şi listați din nou conținutul tabelului.
2. Creați un tabel temporar *TEMP_SESIUNE_PNU*, cu datele persistente pe durata sesiunii. Cerințele sunt cele de la punctul 1.
3. Inițiați încă o sesiune. Listați structura şi conținutul tabelelor create anterior. Introduceți încă o linie în fiecare din cele două tabele.
4. Ştergeți tabelele create anterior. Cum se poate realiza acest lucru?

5. Să se creeze un tabel temporar *angajati_azi_pnu*. Sesiunea fiecărui utilizator care se ocupă de angajări va permite stocarea în acest tabel a angajaților pe care i-a recrutat la data curentă. La sfârşitul sesiunii, aceste date vor fi şterse. Se alocă spațiu acestui tabel la creare ?

6. Inserați o nouă înregistrare în tabelul *angajati_azi_pnu*. Incercați actualizarea tipului de date al coloanei *last_name* a tabelului *angajati_azi_pnu*.

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 inainte de definirea tabelelor, ignorând erorile la crearea vizualizării.
- Subcererea poate fi oricât de complexă dar nu poate conține clauza *ORDER BY*. Dacă se doreşte ordonare se utilizează *ORDER BY* la interogarea vizualizării.
- *WITH CHECK OPTION* permite inserarea şi modificarea prin intermediul vizualizării numai a liniilor ce sunt accesibile vizualizării. Daca lipseşte numele constrângerii atunci sistemul asociază un nume implicit de tip SYS_C*n* acestei constrangeri (*n* este un număr astfel încât numele constrângerii să fie unic).
- *WITH READ ONLY* asigură că prin intermediul vizualizarii nu se pot executa operatii *LMD*.

➢ **Modificarea vizualizărilor** se realizează prin recrearea acestora cu ajutorul opțiunii *OR REPLACE*. Totuşi, din începând cu *Oracle9i*, este posibilă utilizarea comenzii *ALTER VIEW* pentru adăugare de constrângeri 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 comenzile *SELECT, INSERT. UPDATE, DELETE, MERGE* se numesc vizualizări *inline*. Spre deosebire de vizualizările propriu zise acestea nu sunt considerate obiecte ale schemei ci sunt niste entități temporare (valabile doar pe perioada execuției instrucțiunii LMD respective).

- **Operații LMD asupra vizualizărilor**
  - Vizualizările se pot împărți în simple şi 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 intotdeauna (decât prin definirea de *triggeri* de tip *INSTEAD OF*).
    - Vizualizările ***simple*** sunt definite pe baza unui singur tabel şi nu conțin funcții sau grupări de date.
    - Vizualizările ***compuse*** sunt definite pe baza unuia sau a 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*).

- Reactualizarea tabelelor implică reactualizarea corespunzătoare a vizualizărilor!!!
  Reactualizarea vizualizărilor implică reactualizarea tabelelor de bază? NU! Există restricții!!!

Exerciții [II]

7. Să se creeze o vizualizare *VIZ_EMP30_PNU*, care conține codul, numele, email-ul şi salariul angajaților din departamentul 30. Să se analizeze structura şi conținutul vizualizării. Ce se observă referitor la constrângeri? Ce se obține de fapt la interogarea conținutului vizualizării ? Inserați o linie prin intermediul acestei vizualizări; comentați.
8. Modificați *VIZ_EMP30_PNU* astfel încât să fie posibilă inserarea/modificarea conținutului tabelului de bază prin intermediul ei. Inserați şi actualizați o linie prin intermediul acestei vizualizări.

Obs : 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*).

9. 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 şi conținutul vizualizării.
10. a) Inserați o linie prin intermediul vizualizării precedente. Comentați.

    b) Care sunt coloanele actualizabile ale acestei vizualizări?

    c) Inserați o linie specificând valori doar pentru coloanele actualizabile.

    d) Analizați conținutul vizualizării *viz_empsal50_pnu* şi al tabelului *emp_pnu*.

11. a) Să se creeze vizualizarea *VIZ_EMP_DEP30_PNU*, astfel încât aceasta să includă coloanele vizualizării *VIZ_EMP_30_PNU*, precum şi numele şi codul departamentului. Să se introducă aliasuri pentru coloanele vizualizării. 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 ştergere prin intermediul vizualizării *viz_emp_dep30_pnu*? Comentați.

12. Să se creeze vizualizarea *VIZ_DEPT_SUM_PNU*, care conține codul departamentului şi pentru fiecare departament salariul minim, maxim si media salariilor. Ce fel de vizualizare se obține (complexa sau simpla)? Se poate actualiza vreo coloană prin intermediul acestei vizualizări?

13. Modificați vizualizarea *VIZ_EMP30_PNU* astfel încât să nu permită modificarea sau inserarea de linii ce nu sunt accesibile ei. Vizualizarea va selecta şi coloana *department_id*. Dați un nume constrângerii şi regăsiți-o în vizualizarea *USER_CONSTRAINTS* din dicționarul datelor. Încercați să modificați şi să inserați linii ce nu îndeplinesc condiția *department_id* = 30.

14. a) Definiți o vizualizare, *VIZ_EMP_S_PNU*, care să conțină detalii despre angajații corespunzători departamentelor care încep cu litera S. Se pot insera/actualiza linii prin intermediul acestei vizualizări? În care dintre tabele? Ce se întâmplă la ştergerea 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.

15. Să se consulte informații despre vizualizările utilizatorului curent. Folosiți vizualizarea dicționarului datelor *USER_VIEWS* (coloanele *VIEW_NAME* şi *TEXT*).
Obs*:* Coloana *TEXT* este de tip *LONG* aşa că trebuie utilizată comanda *SET LONG* n (din *SQL*Plus*) pentru a seta numărul de caractere afişate în cazul selectării unei coloane de tip *LONG*.

16. Să se selecteze numele, salariul, codul departamentului şi salariul maxim din departamentul din care face parte, pentru fiecare angajat. Este necesară o vizualizare *inline*?

17. Să se creeze o vizualizare *VIZ_SAL_PNU*, ce conține numele angajaților, numele departamentelor, salariile şi locațiile (oraşele) pentru toți angajații. Etichetați sugestiv coloanele. Considerați ca tabele de bază tabelele originale din schema HR. Care sunt coloanele actualizabile?

18. Să se creeze vizualizarea *V_EMP_PNU* asupra tabelului *EMP_PNU* care conține codul, numele, prenumele, email-ul şi numărul de telefon ale angajaților companiei. Se va impune unicitatea valorilor coloanei email şi constrângerea de cheie primară pentru coloana corespunzătoare codului angajatului.
    Obs*:* 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;
```

19. Să se adauge o constrângere de cheie primară asupra vizualizării viz_emp_s_pnu (*alter view*).

III. 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, aşa că aceeaşi 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 initial, implicit fiind 1 (START WITH);
- valoarea maximă, implicit fiind 10^{27} pentru o secvență ascendentă şi –1 pentru una descendentă;
- valoarea minimă;
- dacă secvența ciclează după ce atinge limita; (CYCLE)
- câte numere să încarce în *cache server*, 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* şi *DBA_SEQUENCES*.

- **Pseudocoloanele *NEXTVAL* şi *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.

Obs: Pseudocoloanele se pot utiliza în:
- lista *SELECT* a comenzilor ce nu fac parte din subcereri;
- lista *SELECT* a unei cereri ce apare într un *INSERT*;
- clauza *VALUES* a comenzii *INSERT*;
- clauza *SET* a comenzii *UPDATE*.

Obs : Pseudocoloanele nu se pot utiliza:
- în lista *SELECT* a unei vizualizări;
- într-o comanda *SELECT* ce conține *DISTINCT*, *GROUP BY*, *HAVING* sau *ORDER BY*;
- într-o subcerere în comenzile *SELECT*, *UPDATE*, *DELETE*
- în clauza *DEFAULT* a comenzilor *CREATE TABLE* sau *ALTER TABLE*.

- **Ştergerea secvențelor** se face cu ajutorul comenzii *DROP SEQUENCE*.
  DROP SEQUENCE *nume_secventa;*

Exerciții [III]

20. Creați o secvență pentru generarea codurilor de departamente, *SEQ_DEPT_PNU*. Secvența va începe de la 200, va creşte cu 10 de fiecare dată şi va avea valoarea maximă 10000, nu va cicla şi nu va încărca nici un număr înainte de cerere.
21. Să se selecteze informații despre secvențele utilizatorului curent (nume, valoare minimă, maximă, de incrementare, ultimul număr generat).
22. Creați o secvență pentru generarea codurilor de angajați, *SEQ_EMP_PNU*.
23. Să se modifice toate liniile din *EMP_PNU* (dacă nu mai există, îl re-creeați), regenerând codul angajaților astfel încât să utilizeze secvența *SEQ_EMP_PNU* şi să avem continuitate în codurile angajaților.
24. Să se insereze câte o inregistrare nouă în *EMP_PNU* şi *DEPT_PNU* utilizând cele 2 secvențe create.
25. Să se selecteze valorile curente ale celor 2 secvențe.
26. Ştergeți secvența SEQ_DEPT_PNU.

IV. Definirea indecşilor

- Un index este un obiect al unei scheme utilizator care este utilizat de *server*-ul *Oracle* pentru a mări performanțele unui anumit tip de cereri asupra unui tabel.

- Indecşii :
  - evită scanarea completă a unui tabel la efectuarea unei cereri;
  - reduc operațiile de citire/scriere de pe disc utilizând o cale mai rapidă de acces la date şi anume pointeri la liniile tabelului care corespund unor anumite valori ale unei chei (coloane);
  - sunt independenți de tabelele pe care le indexează, în sensul că dacă sunt şterşi nu afectează conținutul tabelelor sau comportamentul altor indecşi;
  - sunt menținuți şi utilizați automat de către *server*-ul *Oracle*;
  - la ştergerea unui tabel, sunt şterşi şi indecşii asociați acestuia.

- Tipuri de indecşi:
  - indecşi normali (indecsi ce folosesc B-arbori);
  - indecşi *bitmap*, care stochează identificatorii de linie (*ROWID*) asociați cu o valoare cheie sub forma unui *bitmap* – sunt de obicei folosiți pentru coloane care nu au un domeniu mare de valori în contextul unei concurențe limitate, de exemplu în *data warehouse*;
  - indecşi partiționați, care constau din partiții corespunzătoare valorilor ce apar în coloanele indexate ale tabelului;
  - indecşi bazați pe funcții (pe expresii). Aceştia permit construcția cererilor care evaluează valoarea returnată de o expresie, expresie ce poate conține funcții predefinite sau definite de utilizator.

- Indecşii pot fi creați :
  - automat: odată cu definirea unei constrangeri *PRIMARY KEY* sau *UNIQUE*;
  - manual: cu ajutorul comenzii *CREATE INDEX*;

- Se creează un index atunci când:
  - O coloană conține un domeniu larg de valori;
  - O coloană conține nu număr mare de valori null;
  - Una sau mai multe coloane sunt folosite des în clauza *WHERE* sau în condiții de join în programele de aplicații
  - Tabelul este mare şi de obicei cererile obțin mai puțin de 2%-4% din liniile tabelului.

- Nu se creează un index atunci când:
  - Tabelul este mic;
  - Coloanele nu sunt folosite des în clauza *WHERE* sau în condițiile de join ale cererilor;
  - Majoritatea cererilor obțin peste 2%-4% din conținutul tabelului;
  - Tabelul este modificat frecvent;
  - Coloanele indexate sunt referite des în expresii;

- Informații despre indecşi şi despre coloanele implicate în indecşi se pot găsi în vizualizările dicționarului datelor *USER_INDEXES, USER_IND_COLUMNS, ALL_INDEXES, ALL_IND_COLUMNS*.

- **Crearea unui index** se face prin comanda:

  ***CREATE** {**UNIQUE** | **BITMAP**} **INDEX** nume_index*
  ***ON** tabel (coloana1 [, coloana2…]) (**source1**, **source2**) ;*

- **Modificarea unui index** se face prin comada ***ALTER INDEX (source1, source2)***.

- **Eliminarea unui index** se face prin comanda: ***DROP INDEX** nume_index;*

Exerciții [IV]

27. Să se creeze un index (normal, neunic) *IDX_EMP_LAST_NAME_PNU*, asupra coloanei *last_name* din tabelul *emp_pnu*.
28. Să se creeze indecşi unici asupra codului angajatului (*employee_id*) şi asupra combinației *last_name*, *first_name*, *hire_date* prin două metode (automat şi manual).

Obs : Pentru metoda automată impuneți constrângeri de cheie primară asupra codului angajatului şi constrângere de unicitate asupra celor 3 coloane. Este recomandabilă această metodă.

29. Creați un index neunic asupra coloanei *department_id* din *EMP_PNU* pentru a eficientiza *join*-urile dintre acest tabel si *DEPT_PNU*.
30. Prespupunând că se fac foarte des căutari *case insensitive* asupra numelui departamentului şi asupra numelui angajatului, definiți doi indecşi bazați pe expresiile *UPPER(department_name)*, respectiv *LOWER(last_name)*.
31. Să se selecteze din dicționarul datelor numele indexului, numele coloanei, poziția din lista de coloane a indexului şi proprietatea de unicitate a tuturor indecşilor definiți pe tabelele *EMP_PNU* şi *DEPT_PNU*.
32. Eliminați indexul de la exercițiul 27.

Alte exemple la aceasta **adresa**.

V. Clustere

- Un *cluster* reprezintă o grupare fizică a două sau mai multe tabele, relativ la una sau mai multe coloane, cu scopul măririi performanțelor.

- Coloanele comune definesc cheia *cluster*-ului.

- Crearea unui *cluster* presupune:
  - crearea structurii *cluster*-ului;
  - crearea indexului *cluster*-ului;
  - crearea tabelelor care vor compune *cluster*-ul.

- Crearea unui cluster se realizează prin comanda:

```
CREATE CLUSTER nume_cluster
(nume_coloana tip_data [,nume_coloana tip_data] …) [SIZE n]
```

- Ştergerea unui cluster se realizează prin comanda:

```
DROP CLUSTER nume_cluster;
```

- Ştergerea unui cluster, a tuturor tabelelor definite relativ la acel cluster si constrângerile lor de integritate se realizează prin comanda:

```
DROP CLUSTER nume_cluster
INCLUDING TABLES
CASCADE CONSTRAINTS;
```

- Modificarea unui cluster permite redefinirea condițiilor, modificarea parametrilor de stocare şi a caracteristicilor de stare (ALTER CLUSTER).

Exerciții [V]

33. Creați un cluster denumit *angajati_pnu* având cheia denumită *angajat* şi dimensiunea 512 bytes. Extensia inițială alocată *cluster*-ului va avea dimensiunea 100 bytes, iar următoarele extensii alocate vor avea dimensiunea de 50 bytes. Pentru a specifica dimensiunile în kilobytes sau megabytes se foloseste K, respectiv M (de exemplu, 100K sau 100M).

```
CREATE CLUSTER angajati_pnu
(angajat NUMBER(6))
SIZE 512
STORAGE (initial 100 next 50);
```

34. Definiți un index pe cheia *cluster*-ului.

```
CREATE INDEX idx_angajati_pnu ON CLUSTER angajati_pnu;
```

35. Adăugați *cluster*-ului următoarele trei tabele:
    - tabelul ang_1_pnu care va conține angajații având salariul mai mic decât 5000;
    - tabelul ang_2_pnu care va conține angajații având salariul intre 5000 şi 10000;
    - tabelul ang_3_pnu care va conține angajații având salariul mai mare decât 10000.

CREATE TABLE ang_1_pnu
CLUSTER angajati_pnu(employee_id)
*AS SELECT * FROM employees WHERE salary < 5000;*

36. Afişaţi informaţii despre cluster-ele create de utilizatorul current (USER_CLUSTERS).

37. Afişaţi numele cluster-ului din care face parte tabelul ang_3_pnu (USER_TABLES).

38. Eliminaţi tabelul ang_3_pnu din cluster.

39. Consultând vizualizarea USER_TABLES, verificaţi dacă tabelul ang_3_pnu mai face parte dintr-un cluster.

40. Stergeţi tabelul ang_2_pnu. Consultaţi vizualizarea USER_TABLES afişaţi numele tabelelor care fac parte din cluster-ul definit.

41. Stergeţi cluster-ul eliminând si tabelele asociate.

DROP CLUSTER angajati_pnu
INCLUDING TABLES CASCADE CONSTRAINTS;

Alte informaţii la aceaste adrese: **link1** şi **link2**.

VI. Definirea sinonimelor

- Pentru a simplifica accesul la obiecte, acestora li se pot asocia sinonime. Crearea unui sinonim este utilă pentru a evita referirea unui obiect ce aparţine altui utilizator prefixându-l cu numele utilizatorului şi pentru a scurta numele unor obiecte cu numele prea lung.
- Informaţii despre sinonime se găsesc în vizualizarea din dicţionarul datelor *USER_SYNONYMS*.

- **Crearea sinonimelor** se realizează prin comanda:
  CREATE [PUBLIC] SYNONYM *nume_sinonim*
  FOR *obiect;*

- **Eliminarea sinonimelor** se face prin comanda :
  DROP SYNONYM *nume_sinonim;*

Exerciţii [VI]

42. Creaţi un sinonim public *EMP_PUBLIC_PNU* pentru tabelul *EMP_PNU*.
43. Creaţi un sinonim *V30_PNU* pentru vizualizarea *VIZ_EMP30_PNU*.
44. Creaţi un sinonim pentru *DEPT_PNU*. Utilizaţi sinonimul pentru accesarea datelor din tabel. Redenumiţi tabelul (*RENAME* …TO ..). Încercaţi din nou să utilizaţi sinonimul pentru a accesa datele din tabel. Ce se obţine?

45. Eliminați sinonimele create anterior prin intermediul unui script care să selecteze numele sinonimelor din *USER_SYNONYMS* care au terminația "*pnu*" şi să genereze un fişier cu comenzile de ştergere corespunzătoare.

```
SET FEEDBACK OFF
SET HEADING OFF
SET TERMOUT  OFF
SPOOL  "H:\...\delSynonym.sql"
SELECT 'DROP SYNONYM ' || synonym_name ||'; '
FROM user_synonyms
WHERE lower(synonym_name) LIKE '%pnu'
/
SPOOL OFF
SET FEEDBACK ON
SET HEADING ON
SET TERMOUT  ON
```

Obs:
SET TERMOUT OFF – suprimă afişarea rezultatelor pe ecran (acestea vor fi regăsite numai în fişierul specificat în comanda SPOOL).
SPOOL nume_fisier determină înregistrarea tuturor comenzilor care urmează şi a rezultatelor acestora în fişierul specificat.
SPOOL OFF determină oprirea înregistrării.

VII. Definirea vizualizărilor materializate

- O vizualizare materializată, cunoscută în versiunile anterioare sub numele de clişeu (*snapshot*), este un obiect al schemei ce stochează rezultatele unei cereri şi care este folosit pentru a rezuma, calcula, replica şi distribui date. Vizualizările materializate sunt utile în domenii precum *data warehouse*, suportul deciziilor şi calcul distribuit sau mobil.
- Clauza *FROM* a cererii poate referi tabele, vizualizări sau alte vizualizări materializate. Luate în ansamblu, aceste obiecte sunt referite prin tabele *master* (în temeni de replicare) sau prin tabele detaliu (în termeni de *data warehouse*).
- Optimizorul pe bază de costuri (cel folosit de *Oracle9i*) poate utiliza vizualizările materializate pentru a îmbunătăți execuția cererilor. Acesta recunoaşte automat situațiile în care o astfel de vizualizare poate şi trebuie să fie utilizată pentru rezolvarea unei cereri. În urma unui asemenea demers, optimizorul rescrie cererea utilizând vizualizarea materializată.
- Din câteva puncte de vedere, vizualizările materializate sunt similare indecşilor:
  - consumă spațiu de stocare;
  - trebuie reactualizate când datele din tabelele de bază sunt modificate;
  - îmbunătățesc performanța execuției instrucțiunilor *SQL* dacă sunt folosite pentru rescrierea cererilor;
  - sunt transparente aplicațiilor *SQL* şi utilizatorilor.
- Spre deosebire de indecşi, vizualizările materializate pot fi accesate utilizând instrucțiuni *SELECT* şi pot fi actualizate prin instrucțiunile *INSERT*, *UPDATE*, *DELETE*.
- Asupra unei vizualizări materializate se pot defini unul sau mai mulți indecşi.
- Similar vizualizărilor obişnuite, asupra celor materializate se pot defini constrângerile *PRIMARY KEY*, *UNIQUE* şi *FOREIGN KEY*. Singura stare validă a unei constrângeri este *DISABLE NOVALIDATE*.

Pentru compatibilitate cu versiunile anterioare, cuvintele cheie *SNAPSHOT* şi *MATERIALIZED VIEW* sunt echivalente.

- **Crearea vizualizărilor materializate** se realizează prin comanda CREATE MATERIALIZED VIEW, a cărei sintaxă simplificată este:

CREATE MATERIALIZED VIEW [*schema*.]*nume_viz_materializată*
[{*proprietăți_vm* | ***ON PREBUILT TABLE***
[{***WITH*** | ***WITHOUT***} ***REDUCED PRECISION***] }]
[*refresh_vm*] [***FOR UPDATE***]
[{***DISABLE*** | ***ENABLE***} ***QUERY REWRITE***] ***AS*** *subcerere*;

Opțiunea *ON PREBUILT TABLE* permite considerarea unui tabel existent ca fiind o vizualizare materializată predefinită.

Clauza *WITH REDUCED PRECISION* permite ca precizia coloanelor tabelului sau vizualizării materializate să nu coincidă cu precizia coloanelor returnate de *subcerere*.

Printre *proprietăți_vm* poate fi menționată opțiunea *BUILD IMMEDIATE* | *DEFERRED* care determină introducerea de linii în vizualizarea materializată imediat, respectiv la prima operație de reactualizare (*refresh*). În acest ultim caz, până la prima operație de actualizare, vizualizarea nu va putea fi utilizată în rescrierea cererilor. Opțiunea *IMMEDIATE* este implicită.

Prin *refresh_vm* se specifică metodele, modurile şi momentele la care sistemul va reactualiza vizualizarea materializată. Sintaxa simplificată a clauzei este următoarea:

{***REFRESH***
[{***FAST*** | ***COMPLETE*** | ***FORCE***}] [***ON*** {***DEMAND*** | ***COMMIT***}]
[***START WITH*** *data*] [***NEXT*** *data*]
[***WITH*** {***PRIMARY KEY*** | ***ROWID***}]
| ***NEVER REFRESH***}

Opțiunea *FAST* indică metoda de reactualizare incrementală, care se efectuează corespunzător modificărilor survenite în tabelele *master*. Modificările sunt stocate într-un fişier *log* asociat tabelului *master*. Clauza *COMPLETE* implică reactualizarea completă, care se realizează prin reexecutarea completă a cererii din definiția vizualizării materializate. Clauza *FORCE* este implicită şi presupune reactualizarea de tip *FAST*, dacă este posibil. În caz contrar, reactualizarea va fi de tip *COMPLETE*.

Clauza *ON COMMIT* indică declanşarea unei operații de reactualizare de tip *FAST* ori de câte ori sistemul permanentizează o tranzacție care operează asupra unui tabel *master* al vizualizării materializate. Aceasta ar putea duce la creşterea timpului pentru completarea operației *COMMIT*, întrucât reactualizarea va face parte din acest proces. Clauza nu este permisă pentru vizualizările materializate ce conțin tipuri obiect.

Clauza *ON DEMAND* este implicită şi indică efectuarea reactualizării vizualizării materializate la cererea utilizatorului, prin intermediul procedurilor specifice din pachetul *DBMS_MVIEW* (*REFRESH*, *REFRESH_ALL_MVIEWS*, *REFRESH_DEPENDENT*).

Opțiunile *START WITH* şi *NEXT* nu pot fi specificate dacă s-au precizat *ON COMMIT* sau *ON DEMAND*. Expresiile de tip dată calendaristică indicate în cadrul acestor opțiuni specifică momentul primei reactualizări automate şi determină intervalul dintre două reactualizări automate consecutive.

Clauza *WITH PRIMARY KEY* este implicită şi permite ca tabelele *master* să fie reorganizate fără a afecta eligibilitatea vizualizării materializate pentru reactualizarea de tip *FAST*. Tabelul *master* trebuie să conțină o constrângere *PRIMARY KEY*. Clauza nu poate fi specificată pentru vizualizări materializate obiect.

Opțiunea *WITH ROWID* asigură compatibilitatea cu tabelele *master* din versiunile precedente lui *Oracle8*.

Clauza *NEVER REFRESH* previne reactualizarea vizualizării materializate prin mecanisme *Oracle* sau prin proceduri. Pentru a permite reactualizarea, trebuie efectuată o operație *ALTER MATERIALIZED VIEW…REFRESH*.

Clauza *FOR UPDATE* permite actualizarea unei vizualizări materializate. *QUERY REWRITE* permite specificarea faptului că vizualizarea materializată este eligibilă pentru operația de rescriere a cererilor.

Opțiunea *AS* specifică cererea care defineşte vizualizarea materializată. Vizualizările materializate nu pot conține coloane de tip *LONG*. Dacă în clauza *FROM* a cererii din definiția vizualizării materializate se face referință la o altă vizualizare materializată, atunci aceasta va trebui reactualizată întotdeauna înaintea celei create în instrucțiunea curentă.

- ***Modificarea vizualizărilor materializate***

O sintaxă simplificată a comenzii *ALTER MATERIALIZED VIEW* este următoarea:

ALTER MATERIALIZED VIEW *nume_viz_materializată*
[*alter_vm_refresh*]
[{***ENABLE*** | ***DISABLE***} ***QUERY REWRITE***
| ***COMPILE*** | ***CONSIDER FRESH***];

Clauza *alter_vm_refresh* permite modificarea metodelor, modurilor şi timpului implicit de reactualizare automată. Clauza *QUERY REWRITE*, prin opțiunile *ENABLE* şi *DISABLE*, determină ca vizualizarea materializată să fie, sau nu, eligibilă pentru rescrierea cererilor.

Clauza *COMPILE* permite revalidarea explicită a vizualizării materializate.

Opțiunea *CONSIDER FRESH* indică sistemului să considere vizualizarea materializată ca fiind reactualizată şi deci eligibilă pentru rescrierea cererilor.

- **Suprimarea vizualizărilor materializate**

Pentru ştergerea unei vizualizări materializate existente în baza de date se utilizează instrucțiunea:

DROP MATERIALIZED VIEW *nume_viz_materializată*;

Obs: La ştergerea unui tabel *master*, sistemul nu va suprima vizualizările materializate bazate pe acesta. Atunci când se încearcă reactualizarea unei astfel de vizualizări materializate, va fi generată o eroare.

Exerciții [V]

46. Să se creeze şi să se completeze cu înregistrări o vizualizare materializată care va conține numele joburilor, numele departamentelor şi suma salariilor pentru un job, în cadrul unui departament. Reactualizările ulterioare ale acestei vizualizări se vor realiza prin reexecutarea cererii din definiție. Vizualizarea creată va putea fi aleasă pentru rescrierea cererilor.

```
CREATE MATERIALIZED VIEW job_dep_sal_pnu
  BUILD IMMEDIATE
  REFRESH COMPLETE
  ENABLE QUERY REWRITE
  AS SELECT   d.department_name, j.job_title, SUM(salary) suma_salarii
    FROM      employees e, departments d, jobs j
    WHERE     e.department_id = d. department_id
    AND       e.job_id = j.job_id
    GROUP BY d.department_name, j.job_title;
```

47. Să se creeze tabelul *job_dep_pnu*. Acesta va fi utilizat ca tabel sumar preexistent în crearea unei vizualizări materializate ce va permite diferențe de precizie şi rescrierea cererilor.

```
CREATE TABLE job_dep_pnu (
 job VARCHAR2(10),
 dep NUMBER(4),
 suma_salarii NUMBER(9,2));

CREATE MATERIALIZED VIEW vm_job_dep_pnu
  ON PREBUILT TABLE WITH REDUCED PRECISION
  ENABLE QUERY REWRITE
  AS SELECT   d.department_name, j.job_title, SUM(salary) suma_salarii
    FROM      employees e, departments d, jobs j
    WHERE     e.department_id = d. department_id
    AND       e.job_id = j.job_id
    GROUP BY d.department_name, j.job_title;
```

Să se adauge o linie nouă în această vizualizare.

48. Să se creeze o vizualizare materializată care conține informațiile din tabelul *dep_pnu*, permite reorganizarea acestuia şi este reactualizată la momentul creării, iar apoi la fiecare 5 minute.

```
CREATE MATERIALIZED VIEW dep_vm_pnu
  REFRESH FAST START WITH SYSDATE NEXT SYSDATE + 1/288
  WITH PRIMARY KEY
  AS SELECT * FROM dep_pnu;
```

Pentru reactualizarea de tip *FAST*, este necesar un fişier *log* în care să fie stocate modificările. Instrucțiunea precedentă generează eroarea „*ORA-23413*: *table … does not have a materialized view log*“. Pentru remedierea acestei situații, înainte de crearea vizualizării, se va lansa următoarea comandă:

```
CREATE MATERIALIZED VIEW LOG ON dep_pnu;
```

49. Să se modifice vizualizarea materializată *job_dep_sal_pnu* creată anterior, astfel încât metoda de reactualizare implicită să fie de tip *FAST*, iar intervalul de timp la care se realizează reactualizarea să fie de 7 zile. Nu va fi permisă utilizarea acestei vizualizări pentru rescrierea cererilor.

```
ALTER MATERIALIZED VIEW job_dep_sal_pnu
  REFRESH FAST NEXT SYSDATE + 7 DISABLE QUERY REWRITE;
```

Pentru că nu se specifică valoarea corespunzătoare opțiunii *START WITH* în clauza *REFRESH*, următoarea reactualizare va avea loc la momentul stabilit prin comanda de creare a vizualizării materializate sau prin ultima comandă de modificare a acesteia. Sistemul va reactualiza vizualizarea evaluând expresia din clauza *NEXT*, iar apoi va executa această operație o dată pe săptămână.

41. Să se şteargă vizualizările materializate create anterior.

RECAPITULARE 1

Se dă schema:

TURIST(#id_turist, nume, prenume, data_nastere)
ACHIZITIONEAZA(#cod_turist, #cod_excursie, #data_start, data_end, data_achizitie, discount)
EXCURSIE(#id_excursie, denumire, pret, destinatie, durata, cod_agentie, nr_locuri)
AGENTIE(#id_agentie, denumire, oras)

1. Să se afişeze denumirea primei excursii achiziționate.
2. Afişați de câte ori a fost achiziționată fiecare excursie.
3. Să se afişeze pentru fiecare agenție, denumirea, oraşul, numărul de excursii oferite, media prețurilor excursiilor oferite.
4. a. Să se obțină numele şi prenumele turiştilor care au achiziționat cel puțin 2 excursii.
   b. Să se obțină numărul turiştilor care au achiziționat cel putin 2 excursii.
5. Afişați informații despre turiştii care nu au achiziționat excursii cu destinația Paris.
6. Afişați codul şi numele turiştilor care au achiziționat excursii spre cel puțin două destinații diferite.
7. Să se afişeze pentru fiecare agenție, denumirea şi profitul obținut. (Profitul obținut din vânzarea unei excursii este **pret – pret * discount** Dacă discountul este necunoscut profitul este prețul excursiei).
8. Să se afişeze denumirea şi oraşul pentru agențiile care au cel puțin 3 excusii oferite la un preț mai mic decăt 2000 euro.
9. Să se afişeze excursiile care nu au fost achiziționate de către nici un turist.
10. Afişați informații despre excursii, inclusiv denumirea agenției. Pentru excursiile pentru care nu este cunoscută agenția se va afişa textul “agentie necunoscuta”.
11. Să se afişeze informații despre excursiile care au prețul mai mare decăt excursia cu denumirea “Orasul luminilor” existentă în oferta agenției cu codul 10.
12. Să se obțină lista turiştilor care au achiziționat excursii cu o durata mai mare de 10 zile. (se va lua în considerare data_start şi data_end)
13. Să se obțină excusiile achiziționate de cel puțin un turist vârsta mai mică de 30 de ani.
14. Să se obțină turiştii care nu au achiziționat nicio excursie oferită de agenții din Bucuresti.
15. Să se obțină numele turiştilor care au achiziționat excursii care conțin în denumire “1 mai” de la o agenție din Bucureşti.
16. Să se obțina numele, prenume turiştilor şi excursiile oferite de agenția “Smart Tour” achiziționate de către aceştia.
17. Să se afişeze excursiile pentru care nu mai există locuri pentru data de plecare 14 -aug-2011.
18. Să se obțină codurile turiştilor şi codul ultimei excursii achiziționate de către aceştia.
19. Afişați topul celor mai scumpe excursii (primele 5).
20. Afişați numele turiştilor care au achiziționat excursii cu data de plecare în aceeaşi luna cu luna în care îşi serbează ziua de naştere.

21. Să se afişeze informaţii despre turiştii care au achiziţionat excusii de 2 persoane de la agenţii din Constanţa.
22. În funcţie de durata excursiei afişaţi în ordine excursiile cu durata mică (durată de maxim 5 zile), medie (durata între 6 şi 19 de zile), lunga (durata peste 20 de zile).
23. Afişaţi numărul excursiilor, câte excursii sunt oferite de agenţii din Constanţa, câte sunt oferite de agenţii din Bucureşti.

    Numar excursii Nr. ex Constanta Nr. ex Bucuresti
    ---
24. Afişaţi excursiile care au fost achiziţionate de toţi turiştii în vârsta de 24 ani.
25. Afişaţi valoarea totală a excursiilor:
    -Pentru fiecare agenţie şi în cadrul agenţiei pentru fiecare destinaţie.
    -Pentru fiecare agenţi (indiferent de destinaţie)
    -Întreg tabelul.
    Afişaţi o coloană care se indice intervenţia celorlalte coloane în obţinerea rezultatului.
26. Să se obţină pentru fiecare agenţie media preţurilor excursiilor oferite de agenţiile concurente (situate în acelaşi oraş).

RECAPITULARE 2

27. Afisati valoarea totala a excursiilor achizitionate:
-Pentru fiecare agentie si in cadrul agentiei pentru fiecare destinatie.
-Pentru fiecare agentie (indiferent de destinatie).
-Pentru fiecare destinatie (indiferent de agentie).
-Intreg tabelul.

28. Afisati valoarea totala a excursiilor achizitionate
-Pentru fiecare agentie in fiecare an in care a vandut excursii.
-Pentru intreg tabelul.

29. Sa se afiseze denumirea excursiilor cu codul agentiei necunoscut care nu au fost achizitionate de niciun turist nascut in anul 1984.

30. Creati copii pentru toate tabelele din schema. Adaugati cheile primare si externe. Cheile externe vor fi create astfel incat la stergerea unei inregistrari sa se stearga si inregistrarile corelate. Pentru urmatoarele exercitii se vor utiliza tabelele copie.

31. Modificati discount-ul obtinut la achizitioanrea excursiilor care au un pret peste medie astfel incat sa devina egal cu maximul discount-urilor oferite pana acum. Anulati modificarile.

32. Stergeti toate excursiile care au pretul mai mic decat media preturilor din agentie care le pune la dispozitie. Anulati modificarile.

33. Stergeti constrangerea de cheie externa dintre tabelul excursie si agentie. Adaugati doua excursii cu coduri inexistente de agentii. Modificati codul agentie pentru excursiile care nu corespund unei agnetii existente astfel incat sa fie cod necunoscut (null). Anulati modificarile.

34. Creati o vizualizare v_excursie asupra tabelului excursie_*** care sa permita inserarea in acesta doar a excursiilor oferite de agentie 10.
Testati inserarea unei noi inregistrari in tabelul excursie_*** prin intermediul vizualizarii v_excusie_***.
Permanentizati modificarea.

35. Stergeti toate inregistrarile din tabelul achizitioneaza_***.
Creati un SAVEPOINT cu denumirea a.

36. Populati tabelul achizitioneaza_*** astfel incat sa contina toate excursiile achizitionate in 2010 (din tabelul achizitioneaza) dar cu data_start si data_end decalate cu 1 luna.

37. Mariti cu 10% discount-ul pentru excursiile oferite de agentia 10.

38. Stergeti achizitiile turistilor pentru care data nasterii nu este cunoscuta.

39. Folosind comanda MERGE actualizati informatiile din tabelul achizitioneaza_*** astfel incat sa coincida cu informatiile din tabelul achizitioneaza. Anulati modificarile pentru exe 36 - 39. Anulati toate midificarile.

39. Scadeti cu 10% pretul excursiilor care au fost achizitioante de cele mai multe ori.

40. Adaugati tabelului turist_*** constrangerile
-Numele sa nu fie NULL
-Numele si prenumele sa fie unice.

41. Adaugati tabelului achizitioneaza constrangerile
-data_start < data_end
-data_achizitie sa fie implicit data sistemului.

42. Blocati inregistrarea care contine cea mai recenta achizitie cu data_start > data curenta.
SELECT
FROM
WHERE
FOR UPDATE

Modificati achizitia selectata atribuind coloanei data_achizitie valoare default.
UPDATE
SET data_achizitie = DEFAULT
WHERE

Deblocati inregistrarea prin permanentizarea modificarilor.

43. Sa se afiseze numele si prenumele turistilor care au achizitionat cel putin aceleasi excursii ca si turistul Stanciu.

44. Sa se afiseze numele si prenumele turistilor care au achizitionat cel mult aceleasi excursii ca si turistul Stanciu.

45. Sa se afiseze numele si prenumele turistilor care au achizitionat exact aceleasi excursii ca si turistul Stanciu.

46. Sa se insereze un nou turist cu prenume necunoscut. Codul va fi retinut intr-o variabila de legatura. Sa se afiseze valoarea variabilei de legatura. Codul si numele vor fi citite de la tastatura. Pentru cod folositi valoarea 100.

47. Modificati prenumele turistului cu codul 100. Se va afisa numele si prenumele turistului pentru care s-a facut modificarea.

48. Stergeti turistul cu codul 100. Afisati numele si prenumele turistului sters.