Baze de date-Anul 2
Laborator 1 SQL

1. Introducere

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

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

2. Ce este SQL?

SQL (*Structured Query Language*) este un limbaj neprocedural standard pentru interogarea şi prelucrarea informaţiilor din bazele de date **relationale**.

3. Care sunt instrucţiunile SQL?

În funcţie de tipul acţiunii pe care o realizează, instrucţiunile *SQL* se împart în mai multe categorii. Datorită importanţei pe care o au comenzile componente, unele dintre aceste categorii sunt evidenţiate 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:

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

4. Analizaţi sintaxa ***simplificată*** a comenzii SELECT:

5.

SELECT { [{***DISTINCT*** | ***UNIQUE***} | ***ALL***] *lista_campuri* | *}
FROM [*nume_schemă*.]*nume_obiect*]
[, [*nume_schemă*.]*nume_obiect* …]
[***WHERE*** *condiţie_clauza_where*]
[***START WITH*** *condiţie_clauza_start_with*
CONNECT BY *condiţie_clauza_connect_by*]
[***GROUP BY*** *expresie* [*, expresie* …]
[***HAVING*** *condiţie_clauza_having*]]
[***ORDER BY*** {*expresie* | *poziţie*} [, {*expresie* | *poziţie*} …]];

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

Care dintre clauze sunt obligatorii?

2. Exerciţii

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

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

2. Să se iniţieze o sesiune SQLDeveloper folosind *user ID*-ul şi parola comunicate.

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 **conţinutul** 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 pentru fiecare angajat

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

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

8. Sa se listeze numele si salariul angajaţilor care câştigă mai mult de 2850 $.

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

10. Să se afişeze numele şi salariul pentru toţi angajaţii al căror salariu nu se află în domeniul 1500-2850$.

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

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

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

12. Să se afişeze numele salariaţilor şi codul departamentelor pentru toti angajaţii din departamentele 10 şi 30 în ordine alfabetică a numelor.

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

Obs: Apartenenţa la o mulţime 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)

13. Să se listeze numele şi salariile angajatilor care câştigă mai mult de 1500 $ şi lucrează în departamentul 10 sau 30. Se vor eticheta coloanele drept *Angajat* si *Salariu lunar*.

14. Care este data curentă? Afişaţi diferite formate ale acesteia.

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

SQL> SELECT SYSDATE
FROM dual;

- Datele calendaristice pot fi formatate cu ajutorul funcţiei TO_CHAR(data, format), unde formatul poate fi alcătuit dintr-o combinaţie a următoarelor elemente:

Element	***Semnificaţie***
D	Numărul zilei din săptămâna (duminica=1; luni=2; …sâmbătă=6)
DD	Numărul zilei din lună.
DDD	Numărul zilei din an.
DY	Numele zilei din săptămână, printr-o abreviere de 3 litere (MON, THU etc.)
DAY	Numele zilei din săptămână, scris în întregime.
MM	Numărul lunii din an.
MON	Numele lunii din an, printr-o abreviere de 3 litere (JAN, FEB etc.)
MONTH	Numele lunii din an, scris în întregime.
Y	Ultima cifră din an
YY, 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 nopţii.

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

Varianta1:

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

Varianta 2:

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

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

16. Să se afişeze numele şi job-ul pentru toţi angajaţii care nu au manager.

17. Sa se afiseze numele, salariul si comisionul pentru toti salariatii care castiga comisioane. Sa se sorteze datele in ordine descrescatoare a salariilor si comisioanelor. Eliminaţi clauza WHERE din cererea anterioară. Unde sunt plasate valorile NULL în ordinea descrescătoare?

18. 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.

19. Să se listeze numele tuturor angajatilor care au 2 litere 'L' in nume şi lucrează în departamentul 50 sau managerul lor este 102.

20. Sa 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

Funcţii *SQL*

I. [Funcţii SQL]

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

Dacă o funcţie *SQL* este apelată cu un argument având un alt tip de date decât cel aşteptat, sistemul încearcă să convertească implicit argumentul înainte să evalueze funcţia.

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

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

- Funcţii *single-row*
- Funcţii *multiple-row* (funcţii agregat)

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

Obs: Testarea funcţiilor prezentate se face de maniera : *SELECT apel_functie FROM dual;* astfel că vom omite comanda *SELECT* şi vom da numai apelul funcţiei şi rezultatul returnat.

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

Funcţie	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') = ' 18/04/2007'*
TO_DATE	converteşte (sau formatează) un număr sau un şir de caractere în dată calendaristică	*TO_DATE('18-APR-2007','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 funcţiilor 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*.

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

Funcţie	Descriere	Exemplu
LENGTH(string)	întoarce lungimea şirului de caractere *string*	*LENGTH('Informatica')=11*
SUBSTR(string, start [,n])	întoarce subşirul lui *string* care începe pe poziţia *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) = 'atica'* (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 spaţiile libere din stânga lui *string*;	*LTRIM (' info') = 'info'*
RTRIM(string [,'chars'])	este similar funcţiei *LTRIM*, cu excepţia 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 părţi, 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 spaţii libere la stânga lui *string*;	*LPAD (LOWER('iNfO'),6) = ' info'*
RPAD(string, length [,'chars'])	este similar funcţiei *LPAD*, dar adăugarea de caractere se face la dreapta şirului;	*RPAD (LOWER('InfO'), 6, 'X') = 'infoXX'*
REPLACE(string1, string2 [,string3])	întoarce *string1* cu toate apariţiile lui *string2* înlocuite prin *string3*; dacă *string3* nu este specificat, atunci toate apariţiile 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 fiind transformate în minuscule	*INITCAP ('iNfO') = 'Info'*

INSTR(string, 'chars' [,start [,n]])	caută în *string*, începând de de la poziţia *start*, a *n*-a apariţie a secvenţei *chars* şi întoarce poziţia respectivă; dacă *start* nu este specificat, căutarea se face de la începutul şirului; dacă *n* nu este specificat, se caută prima apariţie a secvenţei *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 poziţie ca şi în *source*) din şirul de caractere *destination*	*TRANSLATE('$a$aa','$','b') = 'babaa'* *TRANSLATE('$a$aaa','$a','bc') = 'bcbccc'*

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

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

Funcţie	Descriere	Exemplu
SYSDATE	întoarce data şi timpul curent	*SELECT SYSDATE FROM dual;* (de revăzut utilizarea acestei funcţii 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-APR-2007', 3)* = '02-JUL-2007'.
NEXT_DAY(expr_date, day)	întoarce următoarea dată după data *expr_date*, a cărei zi a săptămânii este cea specificată prin şirul de caractere *day*	*NEXT_DAY('18-APR-2007', 'Monday') = '23-APR-2007'*
LAST_DAY(expr_date)	întoarce data corespunzătoare ultimei zile a lunii din care data *expr_date* face parte	*LAST_DAY('02-DEC-2007') = '31-DEC-2007'*

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 nopţii)	*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, funcţiile întorc prima, respectiv ultima dată în ordine cronologică	*LEAST(SYSDATE, SYSDATE + 3, SYSDATE – 5) = SYSDATE-5* *GREATEST(SYSDATE, SYSDATE + 3, SYSDATE – 5) = SYSDATE + 3*

Operaţiile care se pot efectua asupra datelor calendaristice sunt următoarele:

Operaţie	**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.

- **Funcţii diverse:**

Funcţie	**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 funcţia întoarce valoarea *else*;	*DECODE ('a', 'a', 'b', 'c') = 'b'* *DECODE ('b', 'a', 'b', 'c') = 'c'* *DECODE ('c', 'a', 'b', 'c') = '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 funcţia 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 octeţi ai unei expresii de tip *DATE*, *NUMBER* sau *VARCHAR2*	*SELECT VSIZE(salary)* *FROM employees* *WHERE employee_id=200;*

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

CASE *WHEN cond_1 THEN* *valoare_1* *[WHEN cond_2 THEN* *valoare_2* *...* *WHEN cond_n THEN* *valoare_n]* *[ELSE valoare]* *END*	În funcţie de valoarea *cond_i* returnează *valoare_i* corespunzătoare primei clauze *WHEN .. THEN* pentru care *cond_i* este adevarata; 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. Funcţiile multiple-row (agregat) pot fi utilizate pentru a returna informaţia corespunzătoare fiecăruia dinte grupurile obţinute în urma divizării liniilor tabelului cu ajutorul clauzei GROUP BY. Ele vor for fi tratate intr-un laborator dedicat.

II. [Exerciţii]

[Funcţii pe şiruri de caractere]

1. Scrieţi 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, utilizaţi atât funcţia *CONCAT* cât şi operatorul “||”.

2. Scrieţi o cerere prin care să se afişeze prenumele salariatului cu prima litera majusculă şi toate celelalte litere minuscule, numele acestuia cu majuscule şi lungimea numelui, pentru

angajaţii 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 soluţii (cu operatorul *LIKE* şi funcţia *SUBSTR*).

3. Să se afişeze pentru angajaţii 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 toţi angajaţii al căror nume se termină cu litera 'e', codul, numele, lungimea numelui şi poziţia din nume în care apare prima data litera 'a'. Utilizaţi *alias*-uri corespunzătoare pentru coloane.

[Funcţii aritmetice]

5. Să se afişeze detalii despre salariaţii care au lucrat un număr întreg de săptămâni până la data curentă. Este necesară rotunjirea diferentei celor două date calendaristice?
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. Etichetaţi ultimele două coloane “Salariu nou”, respectiv “Numar sute”. Se vor lua în considerare salariaţii al căror salariu nu este divizibil cu 1000.
7. Să se listeze numele, salariul şi o coloana care sa reprezinte nivelul venitului (pentru fiecare 1000 sa fie folosit cate un simbol $). **Ex**: 6750 -> ‘$$$$$$’

[Funcţii şi operaţii 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. Etichetaţi această coloană “Negociere”.
12. Pentru fiecare angajat să se afişeze numele şi numărul de luni de la data angajării. Etichetaţi 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. Etichetaţi coloana “Zi”. Ordonaţi rezultatul după ziua săptămânii, începând cu Luni.

[Funcţii diverse]

14. Să se afişeze numele angajaţilor şi comisionul. Dacă un angajat nu câştigă comision, să se scrie “Fara comision”. Etichetaţi coloana “Comision”.
15. 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 ceilalţi angajaţi nu se acordă mărire. Să se denumească coloana "Salariu renegociat" (ambele solutii: CASE si DECODE).

Baze de date-Anul 2
Laborator 3 SQL

Cereri multi-relaţie

I. [Join]

Join-ul este operaţia 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.

Condiţia de *join* se scrie în clauza *WHERE* a instrucţiunii *SELECT*. Într-o instrucţiune *SELECT* care uneşte tabele prin operaţia de *join*, se recomandă ca numele coloanelor să fie precedate de numele sau alias-urile tabelelor pentru claritate şi pentru îmbunătăţirea 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 puţin ***n* – 1 condiţii de *join***.

Tipuri de join :

- ***Inner join (equijoin, join simplu)*** – corespunde situaţiei în care valorile de pe coloanele ce apar în condiţia de *join* trebuie să fie egale.
- ***Nonequijoin -*** condiţia de *join* conţine alţi operatori decât operatorul egalitate.
- ***Left* | *Right* | *Full Outer join*** – un *outer join* este utilizat pentru a obţine în rezultat şi înregistrările care nu satisfac condiţia de *join*. Operatorul pentru *outer join* este semnul plus inclus între paranteze **(+)**, care se plasează în acea parte a condiţiei de *join* care este **deficitară în informaţie**. Efectul acestui operator este de a uni liniile tabelului care nu este deficient în informaţie şi cărora nu le corespunde nici o linie în celălalt tabel cu o linie cu valori *null*. Operatorul (+) poate fi plasat în orice parte a condiţiei de *join*, dar nu în ambele părţi.

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

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

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

Obs : *Alias*-urile pot fi utilizate oriunde, ca o notaţie 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 apariţiile numelui tabelului în instrucţiunea SELECT.

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

- opţional, pentru claritate şi pentru îmbunătăţirea timpului de acces la baza de date;
- obligatoriu, ori de câte ori există o ambiguitate privind sursa coloanei. Ambiguitatea constă, de obicei, în existenţa 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 privinţa performanţei faţă 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 existenţa 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 conţină calificatori (să nu fie precedate de nume de tabele sau *alias*-uri) în nici o apariţie a lor în instrucţiunea *SQL*. Clauzele *NATURAL JOIN* şi *USING* **nu pot coexista** în aceeaşi instrucţiune *SQL*.
- ***JOIN tabel_2 ON tabel_1.nume_coloană = tabel_2.nume_coloană*** efectuează un *equijoin* pe baza condiţiei exprimate în clauza *ON*. Această clauză permite specificarea separată a condiţiilor 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 părţi pe baza condiţiei 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*.

II. [Exerciţii]

[Join - non standard] – urmatoarele probleme vor fi rezolvate scriind conditia de join in clauza where; pentru fiecare problema va fi discutat daca se impune o operatie de **join sau outer join**.

1. Să se listeze job-urile care există în departamentul 30.
2. Să se afişeze numele angajatului, numele departamentului şi locatia pentru toţi angajaţii care câştigă comision.
3. Să se afişeze numele salariatului şi numele departamentului pentru toţi salariaţii care au litera A inclusă în nume.
4. Să se afişeze numele, job-ul, codul şi numele departamentului pentru toţi angajaţii care lucrează în Oxford.
5. Să se afişeze codul angajatului şi numele acestuia, împreună cu numele şi codul şefului său direct. Se vor eticheta coloanele Ang#, Angajat, Mgr#, Manager. (operaţia de **self-join**)

6. Creaţi o cerere prin care să se afişeze numele, codul job-ului, titlul job-ului, numele departamentului şi salariul angajaţilor.
7. Să se afişeze numele şi data angajării pentru salariaţii care au fost angajaţi după *Gates*.

[Join - standard] – urmatoarele probleme vor fi rezolvate scriind conditia de join in clauza from; pentru fiecare problema va fi discutat daca se impune o operatie de **join sau outer join**

8. Sa se afiseze codul şi numele angajaţilor care lucrează în acelasi departament cu cel puţin un angajat al cărui nume conţine litera “t”. Se vor afişa, de asemenea, codul şi numele departamentului respectiv. Rezultatul va fi ordonat alfabetic după nume.
9. Sa se afiseze numele, salariul, titlul job-ului, oraşul şi ţara în care lucrează angajatii condusi direct de **King**.
10. Sa se afiseze codul departamentului, numele departamentului, numele si job-ul tuturor angajatilor din departamentele al căror nume conţine şirul ‘ti’. De asemenea, se va lista salariul angajaţilor, în formatul “$99,999.00”. Rezultatul se va ordona alfabetic după numele departamentului, şi în cadrul acestuia, după numele angajaţilor.
11. Sa se afiseze departamentele fara angajati.
12. Afisati angajatii care au ocupat de mai multe ori aceeasi functie. (**doar** join).
13. Scrieti o cerere care afiseaza codurile angajatilor care au ocupat 3 joburi distincte in firma (**doar** join).
14. Se considera tabelul numere(nr number). Sunt inserate urmatoarele valori (1), (2), (3), (4), (5), (6). Sa se genereze toate permutarile (720 linii). (`create table numere(nr number); insert into numere values(1); etc; commit;`)

Baze de date-Anul 2
Laborator 4 SQL

Operatori pe mulţimi. Subcereri.

I. [Operatori pe mulţimi]

Operatorii pe mulţimi combină rezultatele obţinute din două sau mai multe interogări. Cererile care conţin operatori pe mulţimi se numesc ***cereri compuse***. Există patru operatori pe mulţimi: *UNION*, *UNION ALL*, *INTERSECT* şi *MINUS*.

Toţi operatorii pe mulţimi au aceeaşi precedenţă. Dacă o instrucţiune *SQL* conţine mai mulţi operatori pe mulţimi, *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 precedenţă 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.

Observaţii:

- În mod implicit, pentru toţi operatorii cu excepţia lui *UNION ALL*, **rezultatul este ordonat crescător** după valorile primei coloane din clauza *SELECT*.
- Pentru o cerere care utilizează operatori pe mulţimi, cu excepţia lui *UNION ALL*, *server*-ul *Oracle* elimină liniile duplicat.
- În instrucţiunile *SELECT* asupra cărora se aplică operatori pe mulţimi, 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.

II. [Subcereri]

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

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

Rezultatele subcererii sunt utilizate în cadrul cererii exterioare, pentru a determina rezultatul final. În funcţie 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 acţionează 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_condiţie operator (SELECT   expresie
                                     FROM     nume_tabel2);
```

  - cererea internă este executată prima şi determină o valoare (sau o mulţime 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_condiţie 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 precedenţi se repetă până când nu mai există linii candidat.

Obs: *operator* poate fi:

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

Operatorul *NOT* poate fi utilizat în combinaţie 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 condiţia va fi adevărată numai dacă este satisfăcută de către toate valorile produse de subcerere. Astfel, *<ALL* are semnificaţia „mai mic decât minimul“, iar *>ALL* este echivalent cu „mai mare decât maximul“. Dacă subcererea este precedată de către cuvântul cheie *ANY*, condiţia va fi adevărată dacă este satisfăcută de către oricare (una sau mai multe) dintre valorile produse de subcerere. În comparaţii, *<ANY* are semnificaţia „mai mic decât maximul“; *>ANY* înseamnă „mai mare decât minimul“; *=ANY* este echivalent cu operatorul *IN*.

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

III. [Exerciţii - operatori pe mulţimi]

1. Se cer codurile departamentelor al căror nume conţine şirul “re” sau în care lucrează angajaţi având codul job-ului “SA_REP”. Cum este ordonat rezultatul? Ce se întâmplă dacă înlocuim *UNION* cu *UNION ALL*?

2. Sa se obtina codurile departamentelor in care nu lucreaza nimeni (nu este introdus nici un salariat in tabelul *employees*). Se cer două soluţii.

```
SELECT department_id
FROM departments
WHERE department_id NOT IN (SELECT department_id
                            FROM employees);
```

? Este corecta aceasta varianta? De ce ?

3. Se cer codurile departamentelor al căror nume conţine şirul “re” şi în care lucrează angajaţi având codul job-ului “HR_REP”.

IV. [Exercitii - subcereri necorelate]

4. Folosind subcereri, să se afişeze numele şi data angajării pentru salariaţii care au fost angajaţi după Gates.

5. Folosind subcereri, scrieţi o cerere pentru a afişa numele şi salariul pentru toţi 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?

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

7. 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.

8. Scrieti o cerere pentru a afisa angajatii care castiga mai mult decat oricare functionar (job-ul conţine şirul “CLERK”). Sortati rezultatele dupa salariu, in ordine descrescatoare. (*ALL*)

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

10. 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.

IV. [Exercitii - subcereri corelate]

11. Utilizati o cerere corelata pentru a afisa pentru fiecare angajat codul, numele, salariul, precum si numele sefului direct.

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

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

13. Sa se obtina numele primilor 7 angajati avand salariul maxim. Rezultatul se va afişa în ordine crescătoare a salariilor (implementati 2 solutii: subcerere corelata si necorelata).

14. Sa se obtina numele angajatilor care castiga unul dintre cele mai mari 7 salarii. Rezultatul se va afişa în ordine crescătoare a salariilor (implementati 2 solutii: subcerere corelata si necorelata).

15. Afisati informatii despre angajatii care castiga unul dintre cel de al 7-lea, al 25-lea, al 29-lea, al 34-lea sau al 41-lea salariu.

Baze de date-Anul 2
Laborator 5 SQL

Funcţii grup şi clauzele GROUP BY, HAVING.

I. [Funcţii grup şi clauza GROUP BY]

- Clauza *GROUP BY* este utilizată pentru a diviza liniile unui tabel în **grupuri**. Pentru a returna informaţia corespunzătoare fiecărui astfel de grup, pot fi utilizate funcţiile agregat. Ele pot apărea în clauzele:
    - *SELECT*
    - *ORDER BY*
    - *HAVING*.

Server-ul *Oracle* aplică aceste funcţii fiecărui grup de linii şi returnează **un singur rezultat pentru fiecare mulţime**.

- Dintre funcţiile grup definite în sistemul *Oracle*, se pot enumera: *AVG*, *SUM*, *MAX*, *MIN*, *COUNT*, *STDDEV*, *VARIANCE* etc. Tipurile de date ale argumentelor funcţiilor grup pot fi *CHAR*, *VARCHAR2*, *NUMBER* sau *DATE*.
    - Funcţiile *AVG*, *SUM*, *STDDEV* şi *VARIANCE* operează numai asupra valorilor numerice.
    - Funcţiile *MAX* şi *MIN* pot opera asupra valorilor numerice, caracter sau dată calendaristică.
- Absenţa clauzei GROUP BY conduce la aplicarea funcţiei grup **pe mulţimea tuturor liniilor tabelului**.
- Toate funcţiile grup**, cu excepţia lui *COUNT*(*)**, ignoră valorile *null*. *COUNT*(*expresie*) returnează numărul de linii pentru care expresia dată nu are valoarea *null*. Funcţia *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 mulţimea rezultată în ordinea crescătoare a valorilor coloanelor după care se realizează gruparea.
- Expresiile din clauza *SELECT* a unei cereri care conţine opţiunea *GROUP BY* **trebuie să reprezinte o proprietate unică de grup**, adică fie un atribut de grupare, fie o funcţie de agregare aplicată tuplurilor unui grup, fie o expresie formată pe baza primelor două. **Toate expresiile din clauza *SELECT*, cu excepţia funcţiilor de agregare, se trec în clauza *GROUP BY*** (unde pot apărea cel mult 255 expresii).

II. [Clauza HAVING]

Opţiunea *HAVING* permite restricţionarea grupurilor de linii returnate, la cele care îndeplinesc o anumită condiţie.

Dacă această clauză este folosită în absenţa unei clauze *GROUP BY*, aceasta presupune că gruparea se aplică întregului tabel, deci este returnată o singură linie, care este reţinută în rezultat doar dacă este îndeplinită condiţia din clauza *HAVING*.

III. [Exerciţii – funcţii grup şi clauzele GROUP BY, HAVING]

1. Să se afişeze cel mai mare salariu, cel mai mic salariu, suma şi media salariilor tuturor angajaţilor.
2. Să se afişeze minimul, maximul, suma şi numarul de angajati **pentru fiecare job**.
3. Să se determine numărul de angajaţi care sunt şefi.

4. Scrieți o cerere pentru a se afişa numele departamentului, locaţia, numărul de angajaţi şi salariul mediu pentru angajaţii din acel departament. Coloanele vor fi etichetate corespunzător. Este necesara precizarea coloanei *department_id* in clauza *GROUP BY*?

!!!Obs: În clauza *GROUP BY* se trec obligatoriu toate coloanele prezente în clauza *SELECT*, care nu sunt argument al funcţiilor grup (a se vedea ultima observaţie de la punctul I).

5. Să se afişeze codul şi numele angajaţilor care câstiga mai mult decât salariul mediu din firmă.
6. 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 4000$. Sortaţi rezultatul în ordine descrescătoare a salariilor.
9. Care este salariul mediu minim al job-urilor existente?

   Obs*:* Într-o imbricare de funcţii agregat, criteriul de grupare specificat în clauza *GROUP BY* se referă doar la funcţia agregat cea mai interioară. Astfel, într-o clauză SELECT în care există funcţii agregat imbricate **nu mai pot apărea alte expresii**.
10. Să se afişeze maximul salariilor medii pe departamente.
11. Sa se obtina codul, titlul şi salariul mediu al job-ului pentru care salariul mediu este minim.
12. Să se afişeze salariul mediu din firmă doar dacă acesta este mai mare decât 2500. (clauza *HAVING* fără *GROUP BY*)
13. Să se afişeze numele departamentului si cel mai mic salariu din departamentul avand cel mai mare salariu mediu.
14. 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.

15. Sa se afiseze salariatii care au fost angajati în aceeaşi zi a lunii în care cei mai multi dintre salariati au fost angajati.
16. 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 angajaţi (implementați 3 soluții: subcerere from, subcerere select, doar join).
17. Scrieți 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: Plasaţi condiţia *department_id* > 80, pe rând, în clauzele *WHERE* şi *HAVING*. Testaţi în fiecare caz. Ce se observă? Care este diferenţa dintre cele două abordări?

18. Să se calculeze comisionul mediu din firmă, luând în considerare toate liniile din tabel.
19. Să se creeze o cerere prin care să se afişeze numărul total de angajaţi şi, din acest total, numărul celor care au fost angajaţi în 1997, 1998, 1999 si 2000. Denumiti capetele de tabel in mod corespunzător.

Baze de date-Anul 2
Laborator 6 SQL

Operatorii ROLLUP şi CUBE. Clauza GROUPING SETS. Funcţia GROUPING. Cereri ierarhice. Clauza WITH.

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

- **[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 informaţii totalizatoare din mulţimile rezultate. Acest operator poate fi util la generarea de rapoarte, diagrame şi grafice.

Operatorul *ROLLUP* creează grupări prin deplasarea într-o singură direcţie, de la dreapta la stânga, de-a lungul listei de coloane specificate în clauza *GROUP BY*. Apoi, se aplică funcţia 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 BY* sunt specificate *n* coloane, pentru a produce subtotaluri fără operatorul *ROLLUP* ar fi necesare *n* + 1 instrucţiuni *SELECT* conectate prin *UNION ALL*. Aceasta ar face execuţia cererii ineficientă pentru că fiecare instrucţiune *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 angajaţilor 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'));
```

Instrucţiunea 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.

- Prima linie reprezintă suma salariilor angajaţilor în 1987 din departamentul care are codul 10. În mod similar se interpretează liniile din rezultat care au toate coloanele completate.
- Linia a doua conţine 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*.
- Ultima linie conţine suma salariilor tuturor angajatiilor din departamentele al căror cod este mai mic decat 50. Întrucât această linie corespunde totalului general, ea conţine valoarea *null* pe toate coloanele, cu excepţia câmpului *SUM*(*salary*)**.**

- **[Operatorul CUBE]**

Operatorul *CUBE* grupează liniile selectate pe baza valorilor tuturor combinaţiilor posibile ale expresiilor specificate şi returnează câte o linie totalizatoare pentru fiecare grup. Acest operator este folosit pentru a produce mulţimi de rezultate care sunt utilizate în rapoarte. În vreme ce *ROLLUP* produce subtotalurile doar pentru o parte dintre combinaţiile posibile, operatorul *CUBE* produce subtotaluri pentru **toate combinaţiile 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 combinaţii** posibile superagregat. Din punct de vedere matematic, aceste combinaţii formează un cub *n*-dimensional, de aici provenind numele operatorului. Pentru producerea de subtotaluri fără ajutorul operatorului *CUBE* ar fi necesare 2^n instrucţiuni *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 faţă de rezultatul corespunzător operaţiei *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*.

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

- ***GROUPING***(expresie)
  Aceasta întoarce:
  - valoarea 0, dacă expresia a fost utilizată pentru calculul valorii agregat
  - valoarea 1, dacă expresia nu a fost utilizată.

➢ Dacă se doreşte obţinerea numai a anumitor grupări superagregat, acestea pot fi precizate prin intermediul clauzei :

- ***GROUPING SETS*** ((expr_11, expr_12, …, expr_1n), (expr_21, expr_22, …expr_2m), …)

- Operatorul ***EXISTS***

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

II. [Subcereri ierarhice]

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

- *START WITH* specifică o condiţie care identifică liniile ce urmează să fie considerate ca rădăcini ale cererii ierarhice respective. Dacă se omite această clauză, sistemul *Oracle* utilizează toate liniile din tabel drept linii rădăcină.
- *CONNECT BY* specifică o condiţie care identifică relaţia dintre liniile „părinte“ şi „copil“ ale ierarhiei. Condiţia trebuie să conţină operatorul *PRIOR* pentru a face referinţă la linia „părinte“.
- Operatorul *PRIOR* face referinţă la linia „părinte“. Plasarea acestui operator determină direcţia interogării, dinspre „părinte“ spre „copil“ (*top-down*) sau invers (*bottom-up*). Traversarea *top-down*, respectiv *bottom-up* a arborelui se realizează prin specificări de forma următoare:

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

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

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

III. [Clauza WITH]

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

Exerciţii:

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

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

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

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

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

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

5. Sa se obțină numele salariatilor care lucreaza intr-un departament in care exista cel putin un angajat cu salariul egal cu salariul maxim din departamentul 30.

6. Să se afişeze codul, numele şi prenumele angajaţilor care au cel puţin doi subalterni.

7. Să se determine locaţiile în care se află cel puţin un departament (care este cea mai simpla solutie?).

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

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

10. 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.

11. Pentru fiecare linie din tabelul EMPLOYEES se va afișa o structura arborescenta in care va apărea angajatul, managerul său, managerul managerului etc. Coloanele afişate vor fi: codul angajatului, codul managerului, nivelul în ierarhie (LEVEL) si numele angajatului. Se vor folosi indentari (cine este angajatul care apare de cele mai multe ori in rezultat? de cate ori apare?)

12. Pentru fiecare angajat să se determine numărul de subalterni (**nu doar cei direcți**).

13. 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 pe departamente.

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

15. Să se afişeze informaţii despre departamente, în formatul următor: „Departamentul <*department_name*> este condus de {<*manager_id*> | *nimeni*} şi {are numărul de salariaţi <*n*> | nu are salariati}".

16. 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.

Baze de date-Anul 2
Laborator 7 SQL

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

I. Implementarea operatorului DIVISION în *SQL*

Diviziunea este o operaţie binară care defineşte o relaţie ce conţine valorile atributelor dintr-o relaţie care apar **în toate** valorile atributelor din cealaltă relaţie.

Operatorul *DIVISION* este legat de cuantificatorul universal (∀) care nu există în *SQL*. Cuantificatorul universal poate fi însă simulat cu ajutorul cuantificatorului existenţial (∃) utilizând relaţia:

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

Prin urmare, operatorul *DIVISION* poate fi exprimat în *SQL* prin succesiunea a doi operatori *NOT EXISTS*. Alte modalităţi 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 entităţile *EMPLOYEES* şi *PROJECT*. Aceasta este o relaţie *many-to-many*, care va conduce la apariţia unui tabel asociativ, numit *WORKS_ON*.

O altă asociere între entităţile *EMPLOYEES* şi *PROJECT* este „angajat conduce proiect”. Aceasta este o relaţie *one-to-many*.

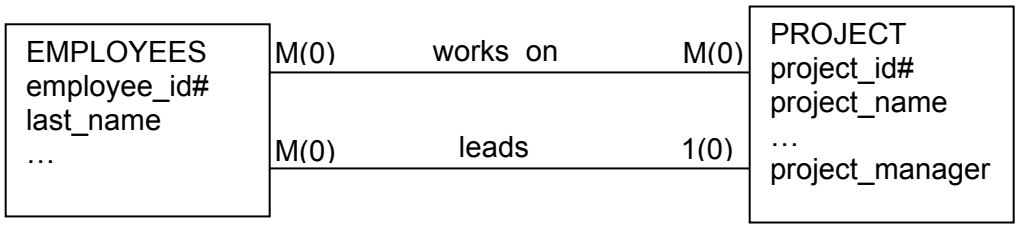
Noile tabele au următoarele scheme relaţionale:

1) *PROJECT*(project_id#, project_name, budget, start_date, deadline, delivery_date, project_manager)
- project_id reprezintă codul proiectului şi este cheia primară a relaţiei *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 relaţie implementează această cheie externă?

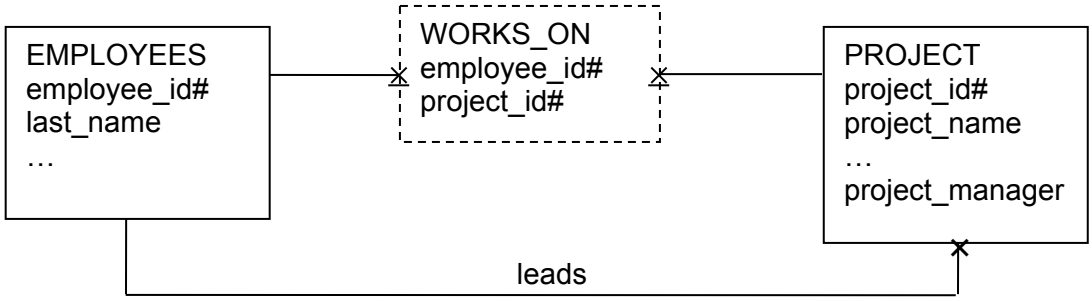
2) *WORKS_ON*(project_id#, employee_id#, start_date, end_date)
- cheia primară a relaţiei este compusă din atributele employee_id şi project_id.

Scriptul pentru crearea noilor tabele şi inserarea de date în acestea este *hr_project*.*sql*.

Diagrama entitate-relaţie 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 obţină codurile salariaţilor ataşaţi **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 funcţiei *COUNT*):

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

Metoda 3 (operatorul *MINUS*):

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

```
    SELECT employee_id, project_id FROM works_on
  ) t3;
```

Metoda 4 (A 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    project_id
            FROM      project p, works_on b
            WHERE     p.project_id=b.project_id
            AND       b.employee_id=a.employee_id));
```

II. Variabile de substituție

- Variabilele de substitutie sunt utile in crearea de comenzi/script-uri dinamice (care depind de niste valori pe care utilizatorul le furnizeaza la momentul rularii).
- Variabilele de substitutie se pot folosi pentru stocarea temporara de valori, transmiterea de valori intre comenzi *SQL* etc. Ele pot fi create prin:
    - **Prefixarea cu &** (indica existenta unei variabile intr-o comanda *SQL*, daca variabila nu exista, este creată).
    - **Prefixarea cu &&** (indica existenta unei variabile într-o comanda *SQL*, daca variabila nu exista, este creată). 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*]

Observatii:

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

Exerciţii:

1. Să se listeze informaţii despre angajaţii care au lucrat în toate proiectele demarate în primele 6 luni ale anului 2006. Implementaţi toate variantele.
2. a) Să se obţină numele angajaţilor care au lucrat **cel puţin** pe aceleaşi proiecte ca şi angajatul având codul 200.
   Obs: Incluziunea dintre 2 mulţimi se testează cu ajutorul proprietăţii „A inclus în B => A-B =Ø". Cum putem implementa acest lucru în SQL?
   Pentru rezolvarea exerciţiului, trebuie selectaţi angajaţii pentru care este vidă lista proiectelor pe care a lucrat angajatul 200 mai puţin lista proiectelor pe care au lucrat acei angajaţi.

   b) Să se obţină numele angajaţilor care au lucrat **cel mult** pe aceleaşi proiecte ca şi angajatul având codul 200.

   c) Să se obţină angajaţii care au lucrat pe aceleaşi proiecte ca şi angajatul având codul 200.

   Obs: Egalitatea între două mulţimi se testează cu ajutorul proprietăţii „A=B => A-B=Ø şi B-A=Ø".
3. **Pentru fiecare ţară**, să se afişeze numărul de angajaţi din cadrul acesteia.
4. Sa se afișeze codul, numele, salariul si codul departamentului din care face parte pentru un angajat al carui cod este introdus de utilizator de la tastatura.
5. Sa se afiseze numele, codul departamentului si salariul anual pentru toti angajatii care au un anumit job.
6. Sa se afiseze numele, codul departamentului si salariul anual pentru toti angajatii care au fost angajati dupa o anumita data calendaristica.
7. 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.
8. Să se citească două date calendaristice de la tastatură şi să se afişeze zilele dintre aceste două date.

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

01-apr-2008
02-apr-2008
...
14-apr-2008

Modificaţi cererea anterioară astfel încât să afişeze doar zilele lucrătoare dintre cele două date calendaristice introduse.

Baze de date-Anul 2
Laborator 8 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 condiţionată de înregistrări (*MERGE*);
  - suprimarea de înregistrări (*DELETE*).

- **Tranzacţia** este o unitate logică de lucru, constituită dintr-o secvenţă de comenzi care trebuie să se execute atomic (ca un întreg) pentru a menţine consistenţa bazei de date.
- *Server*-ul *Oracle* asigură consistenţa datelor pe baza tranzacţiilor, inclusiv în eventualitatea unei anomalii a unui proces sau a sistemului. Tranzacţiile oferă mai multă flexibilitate şi control în modificarea datelor.
- Comenzile SQL care alcătuiesc **LCD** sunt:
  - ROLLBACK – pentru a renunţa la modificările aflate în aşteptare se utilizează instrucţiunea *ROLLBACK*. În urma execuţiei acesteia, se încheie tranzacţia, se anulează modificările asupra datelor, se restaurează starea lor precedentă şi se eliberează blocările asupra liniilor.
  - COMMIT - determină încheierea tranzacţiei curente şi permanentizarea modificărilor care au intervenit pe parcursul acesteia. Instrucţiunea suprimă toate punctele intermediare definite în tranzacţie şi eliberează blocările tranzacţiei.

    Obs: O comandă LDD (*CREATE*, *ALTER*, *DROP*) determină un COMMIT implicit.
  - SAVEPOINT - Instrucţiunea *SAVEPOINT* marchează un punct intermediar în procesarea tranzacţiei. În acest mod este posibilă împărţirea tranzacţiei în subtranzacţii. Această instrucţiune 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 absenţa unei liste de coloane în clauza *INTO*, subcererea trebuie să furnizeze valori pentru fiecare atribut al obiectului destinaţie,

respectând ordinea în care acestea au fost definite.

Observaţii (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 funcţii.

Adăugarea unei linii care va conţine 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 ('').

Observaţii (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 situaţii:

- 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 privinţa 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* operaţii independente *INSERT INTO…SELECT…*, unde *n* reprezintă numărul tabelelor destinaţie. 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 necondiţionate:

INSERT ALL INTO… [***INTO***…]
subcerere;

- Pentru inserări condiţionate:

INSERT [***ALL*** | ***FIRST***]
WHEN *condiţie* ***THEN INTO***…
[***WHEN*** *condiţie* ***THEN INTO***…
[***ELSE INTO*** …]]
subcerere;

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

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

Exerciţii [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 conţinutului tabelelor *EMPLOYEES*, respectiv *DEPARTMENTS*.

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

2. Listaţi structura tabelelor sursă şi a celor create anterior. Ce se observă?
3. Listaţi conţinutul tabelelor create anterior.
4. Pentru introducerea constrângerilor de integritate, executaţi instrucţiunile 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);

5. Să se insereze departamentul 300, cu numele *Programare* în *DEPT_pnu*.
   Analizaţi cazurile, precizând care este soluţia corectă şi explicând erorile celorlalte variante. Pentru a anula efectul instrucţiunii(ilor) corecte, utilizaţi 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);

Executaţi varianta care a fost corectă de două ori. Ce se obţine 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). Determinaţi ca efectele instrucţiunii să devină permanente.

INSERT INTO EMP_pnu
VALUES (250, ‘Prenume’, ‘Nume’, null, null, …, 300);
COMMIT;

Atenţie la constrângerile *NOT NULL* asupra coloanelor tabelului!

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

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

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

ROLLBACK;

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

ROLLBACK;
```

Încercaţi dacă este posibilă introducerea unui angajat, precizând pentru valoarea *employee_id* o subcerere care returnează (codul maxim +1).

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

9. Să se creeze tabelele necesare cu aceeaşi structură ca a tabelului *EMPLOYEES* (fără constrângeri şi fără înregistrări). Copiaţi din tabelul *EMPLOYEES*:
   - în tabelul *EMP0_PNU* salariaţii care lucrează în departamentul 80;
   - în tabelul *EMP1_PNU* salariaţii care au salariul mai mic decât 5000;
   - în tabelul *EMP2_PNU* salariaţii care au salariul cuprins între 5000 şi 10000;
   - în tabelul *EMP3_PNU* salariaţii 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];
```

Observaţii:

- de obicei pentru identificarea unei linii se foloseşte o condiţie ce implică cheia primară;
- dacă nu apare clauza *WHERE* atunci sunt afectate toate liniile tabelului specificat;
- cazurile în care instrucţiunea *UPDATE* nu poate fi executată sunt similare celor în care eşuează instrucţiunea *INSERT*. Acestea au fost menţionate anterior.

Exerciţii [II]

10. Măriţi salariul tuturor angajaţilor din tabelul *EMP_PNU* cu 5%. Vizualizati, iar apoi anulaţi modificările.
11. 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?
12. Schimbaţi salariul şi comisionul celui mai prost plătit salariat din firmă, astfel încât să fie egale cu salariul si comisionul şefului său.
13. Să se modifice jobul şi departamentul angajatului având codul 114, astfel încât să fie la fel cu cele ale angajatului având codul 205.

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.

Exercitii [III]

14. Ştergeţi toate înregistrările din tabelul *DEPT_PNU*. Ce înregistrări se pot şterge? Anulaţi modificările.
15. Ştergeţi angajaţii care nu au comision. Anulaţi modificările.

IV. Exerciţii [LMD, LCD]

16. Să se şteargă un angajat din tabelul *EMP_PNU.*
17. Să se mai introducă o linie in tabelul *DEPT_PNU*.
18. Să se marcheze un punct intermediar in procesarea tranzacţiei. => *SAVEPOINT p*
19. Să se şteargă toate departamentele fară angajaţi. Listaţi conţinutul tabelului.
20. Să se renunţe la cea mai recentă operaţie de ştergere, fără a renunţa la operaţia precedentă de introducere.=> *ROLLBACK TO p*
21. Listaţi conţinutul tabelului *DEPT_PNU*. Determinaţi ca modificările să devină permanente.

Baze de date - Anul 2
Laboratorul 9

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

- În general, instrucţiunile *LDD* sunt utilizate pentru definirea structurii corespunzătoare obiectelor unei scheme: tabele, vizualizări, vizualizări materializate, indecşi, sinonime, clustere, proceduri şi funcţii stocate, declanşatori, pachete stocate etc.
- Aceste instrucţiuni 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 instrucţiune *LDD* permanentizează (*COMMIT*) efectul tuturor instrucţiunilor precedente şi marchează începutul unei noi tranzacţii.
- Instrucţiunile *LDD* au efect imediat asupra bazei de date şi înregistrează informaţia în dicţionarul 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 excepţia 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 conţine 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 conţine valoarea *Null*; (*NOT NULL*)
- **UNIQUE** – pentru coloane sau combinaţii 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 relaţie 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 condiţie 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* octeţi sau caractere. Valoarea maximă a lui *n* corespunde la 4000 octeţi, iar cea minimă este de un octet sau un caracter.
CHAR(n) [BYTE | CHAR]	Reprezintă un şir de caractere de lungime fixă având *n* octeţi sau caractere. Valoarea maximă a lui *n* corespunde la 2000 octeţi. 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	Conţine 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 poziţia 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 conţine numai valori *null* sau dacă tabelul nu conţine 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 conţinutului unui tabel şi păstrarea structurii acestuia se poate utiliza comanda:
TRUNCATE TABLE *nume_tabel;*

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

4. Redenumirea tabelelor

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

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 funcţii stocate.

5. Consultarea dicţionarului datelor

Informaţii despre tabelele create se găsesc în vizualizările:

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

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

Exerciţii

1. Să se creeze tabelul ANGAJATI_pnu (pnu se alcatuieşte din prima literă din prenume şi primele două din numele studentului) corespunzător schemei relaţionale:
   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))

cu precizarea cheii primare la nivel de coloana sau la nivel de tabel si a constrângerilor NOT NULL pentru coloanele nume și 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; recrearea unui tabel va fi precedată de suprimarea sa prin comanda:

DROP TABLE ANGAJATI_pnu;

2. Adăugaţi 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

3. Creaţi tabelul ANGAJATI10_pnu, prin copierea angajaţilor din departamentul 10 din tabelul ANGAJATI_pnu. Listaţi 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. Setaţi o valoare DEFAULT pentru coloana salariu.

7. Modificaţi tipul coloanei comision în NUMBER(2, 2) şi al coloanei salariu la NUMBER(10,2), în cadrul aceleiaşi instrucţiuni.
8. Actualizati valoarea coloanei comision, setând-o la valoarea 0.1 pentru salariaţii al căror job începe cu litera A. (UPDATE)
9. Modificaţi tipul de date al coloanei email în VARCHAR2.
10. Adăugaţi coloana nr_telefon în tabelul ANGAJATI_pnu, setându-i o valoare implicită.
11. Vizualizaţi înregistrările existente. Suprimaţi coloana nr_telefon. Ce efect ar avea o comandă ROLLBACK în acest moment?
12. Suprimați conținutul tabelului angajati10_pnu, fără a suprima structura acestuia.
13. Creaţi şi tabelul DEPARTAMENTE_pnu, corespunzător schemei relaţionale:
    DEPARTAMENTE_pnu (cod_dep# number(2), nume varchar2(15), cod_director number(4))
specificând doar constrângerea NOT NULL pentru nume (nu precizaţi deocamdată constrângerea de cheie primară).

    CREATE TABLE departamente_pnu (…);
    DESC departamente_pnu

14. Introduceţi următoarele înregistrări în tabelul DEPARTAMENTE_pnu:

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

15. 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 situaţia 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.

16. Să se precizeze constrângerea de cheie externă (fără suprimarea tabelului) pentru coloana cod_dep din ANGAJATI_pnu. Se vor mai adăuga constrângerile:
    - FOREIGN KEY pentru cod_sef;
    - UNIQUE pentru combinaţia nume + prenume;
    - UNIQUE pentru email;
    - NOT NULL pentru nume;
    - verificarea cod_dep >0;
    - verificarea ca salariul sa fie mai mare decat comisionul*100.
17. (Incercaţi să) adăugaţi o nouă înregistrare în tabelul ANGAJATI_pnu, care să corespundă codului de departament 50. Se poate?
18. Adăugaţi un nou departament, cu numele Analiza, codul 60 şi directorul null în DEPARTAMENTE_pnu. COMMIT.
19. (Incercaţi să) ştergeţi departamentul 20 din tabelul DEPARTAMENTE_pnu. Comentaţi.

20. Ştergeţi departamentul 60 din DEPARTAMENTE_pnu. ROLLBACK.

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

21. Se doreşte ştergerea automată a angajaţilor dintr-un departament, odată cu suprimarea departamentului. Pentru aceasta, este necesară introducerea clauzei ON DELETE CASCADE în definirea constrângerii de cheie externă. Suprimaţi constrângerea de cheie externă asupra tabelului ANGAJATI_pnu şi reintroduceţi această constrângere, specificând clauza ON DELETE CASCADE.
22. Ştergeţi departamentul 20 din DEPARTAMENTE_pnu. Ce se întâmplă? Rollback.
23. Adăugați o constrângere de tip *check* asupra coloanei salariu, astfel încât acesta să nu poată depăşi 30000.
24. Incercaţi actualizarea salariului angajatului 100 la valoarea 35000.
25. Dezactivaţi constrângerea creată anterior şi reîncercaţi actualizarea. Ce se întâmplă dacă încercăm reactivarea constrângerii?

Baze de date - Anul 2
Laborator 10

Limbajul de definire a datelor (LDD) - II :
Definirea vizualizărilor, secvenţelor

I. Definirea vizualizărilor (*view*)

- **Vizualizările sunt tabele virtuale** construite pe baza unor tabele sau a altor vizualizări, denumite tabele de bază.
- **Vizualizările nu conţin 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:
- restricţionarea accesului la date;
- simplificarea unor cereri complexe;
- asigurarea independenţei datelor de programele de aplicaţii;
- 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 definiţia unei vizualizări fără a mai reacorda eventualele privilegii.
- Opţiunea *FORCE* permite crearea vizualizării inainte de definirea tabelelor, ignorând erorile la crearea vizualizării.
- Subcererea poate fi oricât de complexă dar nu poate conţine 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 operaţii *LMD*.

➢ **Modificarea vizualizărilor** se realizează prin recrearea acestora cu ajutorul opţiunii *OR REPLACE*. Totuşi, 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;*

- Informaţii despre vizualizări se pot găsi în dicţionarul datelor interogând vizualizările: *USER_VIEWS*, *ALL_VIEWS* . Pentru aflarea informaţiilor despre coloanele actualizabile, este utilă vizualizarea *USER_UPDATABLE_COLUMNS*.

- Subcererile însoţite de un alias care apar în comenzile *SELECT, INSERT. UPDATE, DELETE* se numesc vizualizări *inline*. Spre deosebire de vizualizările propriu zise, acestea nu sunt considerate obiecte ale schemei ci sunt entităţi temporare (valabile doar pe perioada execuţiei instrucţiunii LMD respective).

- **Operaţii LMD asupra vizualizărilor**
  - Vizualizările se pot împărţi în **simple** şi **complexe**. Această clasificare este importantă pentru că asupra vizualizărilor simple se pot realiza operaţii *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 conţin funcţii** sau **grupări de date**.
    - **Vizualizările *compuse*** sunt definite pe baza mai multor tabele sau conţin funcţii sau grupări de date.
  - **Nu se pot realiza operaţii *LMD*** în vizualizări ce conţin:
    - funcţii grup,
    - clauzele *GROUP BY*, *HAVING, START WITH, CONNECT BY*,
    - cuvântul cheie *DISTINCT*,
    - pseudocoloana *ROWNUM*,
    - operatori pe mulţimi.
  - **Nu se pot actualiza**:
    - coloane ale căror valori rezultă prin calcul sau definite cu ajutorul funcţiei *DECODE*,
    - coloane care nu respectă constrângerile din tabelele de bază.
  - **Pentru vizualizările bazate pe mai multe tabele**, orice operaţie *INSERT*, *UPDATE* sau *DELETE* poate **modifica datele doar din unul din tabelele de bază**. Acest tabel este cel protejat prin cheie (***key preserved***). În cadrul unei astfel de vizualizări, un tabel de bază se numeşte *key-preserved* dacă are proprietatea că fiecare valoare a cheii sale primare sau a unei coloane având constrângerea de unicitate, este unică şi în vizualizare.
    Prima condiţie ca o vizualizare a cărei cerere conţine un *join* să fie modificabilă este ca instrucţiunea *LMD* să afecteze un singur tabel din operaţia de *join*.

- Reactualizarea tabelelor implică reactualizarea corespunzătoare a vizualizărilor!!!

Exerciţii [I]

1. Pe baza tabelului *EMP_PNU*, să se creeze o vizualizare *VIZ_EMP30_PNU*, care conţine codul, numele, email-ul şi salariul angajaţilor din departamentul 30. Să se analizeze structura şi conţinutul vizualizării. Ce se observă referitor la constrângeri? Ce se obţine de fapt la interogarea conţinutului vizualizării? Inseraţi o linie prin intermediul acestei vizualizări; comentaţi.
2. Modificaţi *VIZ_EMP30_PNU* astfel încât să fie posibilă inserarea/modificarea conţinutului tabelului de bază prin intermediul ei. Inseraţi şi actualizaţi o linie (cu valoarea 300 pentru codul angajatului) 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 operaţii *LMD*, acestea nu vor fi posibile din cauza nerespectării constrângerilor *NOT NULL*).

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

Ce efect are următoarea operaţie de actualizare?

UPDATE viz_emp30_pnu SET hire_date=hire_date-15
WHERE employee_id=300;

Comentaţi efectul următoarelor instrucţiuni, analizând şi efectul asupra tabelului de bază:
UPDATE emp_pnu SET department_id=30
WHERE employee_id=300;

UPDATE viz_emp30_pnu SET hire_date=hire_date-15
WHERE employee_id=300;

Ştergeţi angajatul având codul 300 prin intermediul vizualizării. Analizaţi efectul asupra tabelului de bază.

3. Să se creeze o vizualizare, *VIZ_EMPSAL50_PNU*, care contine coloanele *cod_angajat, nume, email, functie, data_angajare* si *sal_anual* corespunzătoare angajaţilor din departamentul 50. Analizaţi structura şi conţinutul vizualizării.

4. a) Inseraţi o linie prin intermediul vizualizării precedente. Comentaţi.
b) Care sunt coloanele actualizabile ale acestei vizualizări? Verificaţi răspunsul în dicţionarul datelor (*USER_UPDATABLE_COLUMNS*).

c) Inseraţi o linie specificând valori doar pentru coloanele actualizabile.
d) Analizaţi conţinutul vizualizării *viz_empsal50_pnu* şi al tabelului *emp_pnu*.

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

6. a) Definiţi o vizualizare, *VIZ_EMP_S_PNU*, care să conţină detalii despre angajaţii 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) Recreaţi vizualizarea astfel încât să nu se permită nici o operaţie asupra tabelelor de bază prin intermediul ei. Încercaţi să introduceţi sau să actualizaţi înregistrări prin intermediul acestei vizualizări.

7. Să se creeze o vizualizare *VIZ_SAL_PNU*, ce conţine numele angajaţilor, numele departamentelor, salariile şi locaţiile (oraşele) pentru toţi angajaţii. Etichetaţi sugestiv coloanele. Consideraţi ca tabele de bază tabelele originale din schema HR. Care sunt coloanele actualizabile?

8. Să se implementeze în două moduri constrângerea ca numele angajaţilor nu pot începe cu şirul de caractere „Wx”.

Metoda 1:

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

Metoda 2:

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

II. Definirea secvenţelor

- Secvenţa este un obiect al bazei de date ce permite generarea de întregi unici pentru a fi folosiţi ca valori pentru cheia primară sau coloane numerice unice. Secvenţele sunt independente de tabele, aşa că aceeaşi secvenţă poate fi folosită pentru mai multe tabele.

- **Crearea secvenţelor** 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 secvenţe se pot specifica:

- numele secvenţei
- diferenţa 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 secvenţă ascendentă şi –1 pentru una descendentă;
- valoarea minimă, implicit fiind 1 pentru o secvenţă ascendentă şi -10^{27} pentru o secvenţă descendentă;
- dacă secvenţa ciclează după ce atinge limita; (*CYCLE*)
- câte numere să încarce în *cache server*, implicit fiind încărcate 20 de numere (*CACHE*).

- Informaţii despre secvenţe găsim în dicţionarul datelor. Pentru secvenţele utilizatorului curent, interogăm *USER_SEQUENCES*. Alte vizualizări utile sunt *ALL_SEQUENCES* şi *DBA_SEQUENCES*.

- **Pseudocoloanele *NEXTVAL* şi *CURRVAL*** permit lucrul efectiv cu secvenţele.
  - *Nume_secv.NEXTVAL* - returnează următoarea valoare a secvenţei, o valoare unică la fiecare referire. Trebuie aplicată cel puţin o dată înainte de a folosi *CURRVAL*;
  - *Nume_secv.CURRVAL* – obţine valoarea curentă a secvenţei.

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 conţine *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 secvenţelor** se face cu ajutorul comenzii *DROP SEQUENCE*.
  DROP SEQUENCE *nume_secventa;*

Exerciţii [II]

9. Creaţi o secvenţă pentru generarea codurilor de departamente, *SEQ_DEPT_PNU*. Secvenţa va începe de la 400, 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.
10. Creaţi o secvenţă pentru generarea codurilor de angajaţi, *SEQ_EMP_PNU*. Să se modifice toate liniile din *EMP_PNU* (dacă nu mai există, îl recreeaţi), regenerând codul angajaţilor astfel încât să utilizeze secvenţa *SEQ_EMP_PNU* şi să avem continuitate în codurile angajaţilor.
11. Ștergeți secvența *SEQ_DEPT_PNU*.