LABORATOR 1 SQL - REZUMAT
CERERI MONOTABEL

1. Analizați sintaxa simplificată a comenzii SELECT. Care dintre clauze sunt obligatorii?

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

2. Să se inițieze o sesiune SQL*Plus folosind user ID-ul şi parola indicate.

3. a) Consultați diagrama exemplu HR (Human Resources) pentru lucrul în cadrul laboratoarelor SQL.
b) Identificați cheile primare şi cele externe ale tabelelor existente în schemă, precum şi tipul relațiilor dintre aceste tabele.

4. Să se listeze structura tabelului *employees*, observând tipurile de date ale coloanelor.

Obs: Se va utiliza comanda SQL*Plus

```
DESCRIBE nume_tabel
```

5. Să se listeze conținutul tabelului *departments*, afişând valorile tuturor câmpurilor.

Obs: Se va utiliza comanda SQL

```
SELECT * FROM nume_tabel;
```

6. Să se afişeze codul şi numele angajatului, adăugând coloanelor câte un alias.

7. Să se listeze fără duplicate codurile job-urilor din tabelul *employees*.

```
SELECT DISTINCT col FROM nume_tabel;
```

8. Să se listeze numele şi salariul angajaților care câştigă mai mult de 10000 $.

9. Să se modifice cererea anterioară astfel încât să afişeze numele şi salariul pentru toți angajații al căror salariu este cuprins între 5000$ şi10000$.

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

10. Să se afişeze numele şi salariul pentru toți angajații din departamentele 10 sau 30, în ordine alfabetică a numelor.

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

expresie IN (*valoare_1*, *valoare_2*, …, *valoare_n*)

```
SELECT  last_name, salary
FROM    employees
WHERE   department_id IN (10, 30)
ORDER BY  last_name;
```

11. Care este data curentă?

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

```
SELECT  SYSDATE
FROM    dual;
```

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

Element	**Semnificație**
D	Numărul zilei din săptămână (duminică=1; luni=2; …sâmbătă=6).
DD	Numărul zilei din lună.
DDD	Numărul zilei din an.
MM	Numărul lunii din an.
MON	Numele lunii din an, printr-o abreviere de 3 litere (JAN, FEB etc.).
MONTH	Numele complet al lunii din an.
YYYY	Anul în format numeric (4 cifre).
HH12, HH24	Orele din zi, între 0-12, respectiv 0-24.
AM sau PM	Indicator meridian: AM sau PM
MI	Minutele din oră.
SS	Secundele din minut.

12. Să se afişeze numele şi data angajării pentru fiecare salariat care a fost angajat în 1987.

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

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

```
SELECT last_name, job_id
FROM   employees
WHERE  manager_id IS NULL;
```

14. Să se listeze numele tuturor angajaților care au a treia litera din nume 'a'.

Obs: Pentru a forma măştile de caractere utilizate împreună cu operatorul LIKE cu scopul de a compara şirurile de caractere, se utilizează:

% - reprezentând orice şir de caractere, inclusiv şirul vid;

_ (underscore) – reprezentând un singur caracter.

```
SELECT DISTINCT last_name
FROM   employees
WHERE  last_name LIKE '__a%';
```

LABORATOR 2 - SQL - REZUMAT
FUNCȚII SQL (single-row)

1. Analizați următoarele **operații pe expresii de tip dată calendaristică**:

Operație	Tipul de date al rezultatului	Descriere
date -/+ number	Date	Scade/Adaugă un număr de zile dintr-o / la o dată.
date1 - date2	Number	Returnează numărul de zile dintre două date calendaristice.
date +/- number/24	Date	Scade/Adaugă un număr de ore la o / dintr-o dată calendaristică.

2. Să se afişeze data (luna, ziua, ora, minutul si secunda) de peste 10 zile.

```
SYSDATE+10
```

3. **a.** Să se afişeze data de peste 12 ore.

```
SYSDATE+12/24
```

b. Să se afişeze data de peste 5 minute.

```
SYSDATE+1/288
```

4. Analizați următoarele **funcții pentru prelucrarea datelor calendaristice**:

Funcție	Semnificație	Exemplu
SYSDATE	Întoarce data şi timpul curent	
MONTHS_BETWEEN (date1, date2)	Returnează numărul de luni dintre data *date1* şi data *date2*. Rezultatul poate fi pozitiv sau negativ după cum *date1* este mai recentă sau nu față de *date2*. Zecimalele reprezintă parți dintr-o luna!	ROUND(MONTHS_BETWEEN (SYSDATE + 31, SYSDATE)) = 1
ADD_MONTHS (date, n)	Adaugă *n* luni la o data specificată. Valoarea *n* trebuie să fie întreagă (pozitivă sau negativă).	MONTHS_BETWEEN (ADD_MONTHS(SYSDATE, 3), SYSDATE) = 3

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

```
SELECT   last_name,
         ROUND(MONTHS_BETWEEN(SYSDATE, hire_date)) "Luni lucrate"
FROM     employees
ORDER BY MONTHS_BETWEEN(SYSDATE, hire_date);

SELECT   last_name,
         ROUND(MONTHS_BETWEEN(SYSDATE, hire_date)) "Luni lucrate"
FROM     employees
ORDER BY "Luni lucrate";
```

```
SELECT last_name,
       ROUND(MONTHS_BETWEEN(SYSDATE, hire_date)) "Luni lucrate"
FROM   employees
       ORDER BY 2;
```

6. Analizați următoarea **funcție SQL**:

Funcție	Semnificație	Exemplu
NVL (expr1, expr2)	Returnează expr1 dacă aceasta nu este NULL, expr2 în caz contrar. Cele 2 expresii trebuie să aibă acelaşi tip sau expr2 să permită conversia implicită la tipul expresiei expr1.	NVL(NULL, 1) = 1 NVL(2, 1) = 2 NVL('c', 1) = 'c' -- face conversie NVL(1, 'c') -- eroare --nu face conversie

7. Să se afişeze numele angajaților şi comisionul. Dacă un angajat nu câştigă comision, să se scrie "Fara comision". Etichetați coloana "Comision".

```
SELECT last_name,
       NVL(TO_CHAR(commission_pct),'Fara comision') comision
FROM   employees;
```

8. Să se listeze numele, salariul şi comisionul tuturor angajaților al căror venit lunar depăşeşte 10000$.

```
SELECT last_name, salary, commission_pct,
       salary + salary * NVL(commission_pct, 0) venit_lunar
FROM   employees
WHERE  salary + salary * NVL(commission_pct, 0) > 10000
ORDER BY venit_lunar;
```

9. Analizați expresia CASE şi funcția DECODE:

Funcție/Expresie	Semnificație	Exemplu
CASE expr WHEN expr_bool1 THEN return_expr1 [WHEN expr_bool2 THEN return_expr2 ... WHEN expr_booln THEN return_exprn] [ELSE return_expr] END	În funcție de valoarea unei expresii returnează valoarea primei perechi WHEN .. THEN care se potriveşte sau dacă nu se potriveşte nici una expresia din ELSE. Nu se poate specifica NULL pentru toate expresiile de returnat. (return_expri). Toate expresiile trebuie sa aibă acelaşi tip de date	
DECODE (expr, expr_cautare1, expr_rezultat1, [expr_cautare2, expr_rezultat2, .. expr_cautaren, expr_rezultatn,] [rezultat_implicit])	Decodifică valoarea expresiei. Dacă valoarea expresiei este expr_cautarei atunci e returnată expr_rezultati. Dacă nu se potriveşte nici o expresie de căutare atunci e returnat rezultat_implicit.	DECODE (1, 1, 2, 3) = 2 DECODE (2, 1, 2, 3) = 3 DECODE (3, 1, 2, 3) = 3

10. Să se afişeze numele, codul funcției, salariul şi o coloana care să arate salariul după mărire. Se ştie că pentru IT_PROG are loc o mărire de 10%, pentru ST_CLERK 15%, iar pentru SA_REP o mărire de 20%. Pentru ceilalți angajați nu se acordă mărire. Să se denumească coloana "Salariu revizuit".

```
SELECT last_name, job_id, salary,
       DECODE(job_id,
                      'IT_PROG',  salary*1.1,
                      'ST_CLERK', salary*1.15,
                      'SA_REP', salary*1.2,
                       salary ) "salariu revizuit"
FROM employees;

SELECT last_name, job_id, salary,
     CASE job_id WHEN 'IT_PROG' THEN salary* 1.1
                 WHEN 'ST_CLERK' THEN salary*1.15
                 WHEN 'SA_REP' THEN salary*1.2
                 ELSE salary
     END "salariu revizuit"
FROM employees;
```

LABORATOR 3 - SQL -REZUMAT
CERERI MULTITABEL, SUBCERERI

Tipuri de *join*:

- **equijoin** (se mai numeşte *inner join* sau *simple join*) - compunerea a două tabele diferite după o condiție ce conține operatorul de egalitate.

```
SELECT last_name, department_name, location_id, e.department_id
FROM   employees e, departments d
WHERE  e.department_id = d.department_id;
```

  Obs: Numele sau alias-urile tabelelor sunt obligatorii în dreptul coloanelor care au acelaşi nume în mai multe tabele.

- **nonequijoin** - compunerea a două relații tabele după o condiție oarecare, ce NU conține operatorul de egalitate.

```
SELECT last_name, salary, grade_level
FROM   employees, job_grades
WHERE  salary BETWEEN lowest_sal AND highest_sal;
```

- **outerjoin** - compunerea externă a două tabele diferite completând una dintre relații cu valori NULL acolo unde nu există în aceasta nici un tuplu ce îndeplinește condiția de corelare. Relația completată cu valori NULL este cea în dreptul căreia apare "(+)". Operatorul (+) poate fi plasat în orice parte a condiției de join, dar nu în ambele părți. Full outer join = Left outer join UNION Right outer join.

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

- **selfjoin** - compunerea externă a unui tabel cu el însuşi după o condiție dată.

```
SELECT sef.last_name, angajat.last_name
FROM   employees sef, employees angajat
WHERE  sef.employee_id = angajat.manager_id
ORDER BY sef.last_name;
```

1. Să se afişeze numele, job-ul şi numele departamentului pentru toți angajații care lucrează în Seattle.

```
SELECT  last_name, job_id, department_name
FROM    employees e, departments d, locations s
WHERE   e.department_id = d.department_id
AND     d.location_id = s.location_id
AND     city = 'Seattle';
```

2. Să se afişeze numele salariaților şi numele departamentelor în care lucrează. Se vor afişa şi salariații care nu lucrează într-un departament.

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

3. Să se afişeze numele şi data angajării pentru salariații care au fost angajați după Fay.

```
SELECT last_name, hire_date
FROM   employees
```

```
WHERE  hire_date > (SELECT hire_date
                    FROM employees
                    WHERE last_name = 'Fay');
```

sau

```
SELECT a.last_name, a.hire_date
FROM   employees a, employees b
WHERE  UPPER(b.last_name)='FAY' AND a.hire_date>b.hire_date;
```

4. Să se afişeze numele şi job-ul tuturor angajaților din departamentul 'Sales'.

```
SELECT  last_name, job_id
FROM    employees
WHERE   department_id = (SELECT department_id
                         FROM   departments
                         WHERE  department_name ='Sales');
```

5. Rezolvați cererea anterioară utilizând joinuri.

LABORATOR 4 - SQL- REZUMAT
Funcții grup. Gruparea datelor.

Funcțiile multiple-row (grup sau agregat)

1. Să se afişeze cel mai mare salariu, cel mai mic salariu, suma şi media salariilor tuturor angajatilor. Etichetați coloanele Maxim, Minim, Suma, respectiv Media. Să se rotunjească rezultatele.

```
SELECT MIN(salary) min, MAX(salary) max, SUM(salary) suma,
       ROUND(AVG(salary)) media
FROM   employees;
```

2. Utilizând funcția grup COUNT să se determine:
a. numărul total de angajați;
b. numărul de angajați care au manager;
c. numărul de manageri.

3. Să se afişeze suma salariilor angajaților din departamentul 80.

4. Să se afişeze numărul de angajați pentru fiecare job.

```
SELECT   job_id, COUNT(employee_id) nr_angajati
FROM     employees
GROUP BY job_id;
```

5. Să se afişeze codul departamentului şi media salariilor pentru fiecare job din cadrul acestuia.

```
SELECT   department_id, job_id, AVG(salary)
FROM     employees
GROUP BY department_id, job_id;
```

6. Să se afişeze codul departamentelor pentru care salariul minim depăşeşte 5000$.

```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING MIN(salary)>5000;
```

7. Să se obțină codul departamentelor şi numărul de angajați al acestora pentru departamentele care au cel puțin 10 angajați.

8. Să se obțină numărul departamentelor care au cel puțin 10 angajați.

```
SELECT   COUNT(COUNT(employee_id))
FROM     employees
GROUP BY department_id
HAVING   COUNT(*) > =10;
```

9. Să se obțină job-ul pentru care salariul mediu este minim.

```
SELECT   job_id
FROM     employees
GROUP BY job_id
HAVING   AVG(salary) = (SELECT   MIN(AVG(salary))
                        FROM     employees
                        GROUP BY job_id);
```

10. 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 şi 2000. Datele vor fi afişate în forma următoare:

Total	1997	1998	1999	2000
50	10	5	25	1

```
SELECT COUNT(*) TOTAL,
    SUM(DECODE(TO_CHAR(hire_date,'yyyy'),1997,1,0)) "AN 1997",
    SUM(DECODE(TO_CHAR(hire_date,'yyyy'),1998,1,0)) "AN 1998",
    SUM(DECODE(TO_CHAR(hire_date,'yyyy'),1999,1,0)) "AN 1999",
    SUM(DECODE(TO_CHAR(hire_date,'yyyy'),2000,1,0)) "AN 2000"
FROM     employees;
```

Operatorii ROLLUP şi CUBE

11. Să se afişeze codurile departamentelor în care lucrează cel puţin un angajat, iar pentru fiecare dintre acestea şi pentru fiecare manager care lucrează în departamentul respectiv să se afişeze numărul de salariaţi. De asemenea, să se afişeze numărul de salariaţi pentru fiecare departament indiferent de manager şi numărul total de angajaţi din companie.

```
SELECT  department_id, manager_id, COUNT(employee_id)
FROM    employees
WHERE   manager_id IS NOT NULL AND department_id IS NOT NULL
GROUP BY ROLLUP (department_id, manager_id);
```

department_id	manager_id	COUNT(employee_id)
10	7782	1
10	7839	1
10		2
20	7566	2
20	7788	1
20	7839	1
20	7902	1
20		5
30	7698	5
30	7839	1
30		6
		13

12. Să se afişeze codurile departamentelor în care lucrează cel puţin un angajat, iar pentru fiecare dintre acestea şi pentru fiecare manager care lucrează în departamentul respectiv să se afişeze numărul de salariaţi. De asemenea, să se afişeze numărul de salariaţi pentru fiecare departament indiferent de manager, numărul de angajaţi subordonaţi unui manager indiferent de departament şi numărul total de angajaţi din companie.

```
SELECT  department_id, manager_id, COUNT(employee_id)
FROM    employees
WHERE   manager_id IS NOT NULL AND department_id IS NOT NULL
GROUP BY CUBE (department_id, manager_id);
```

department_id	manager_id	COUNT(employee_id)
10	7782	1
10	7839	1
10		2
20	7566	2
20	7788	1
20	7839	1
20	7902	1
20		5
30	7698	5
30	7839	1
30		6
	7566	2
	7698	5
	7782	1
	7788	1
	7839	3
	7902	1
		13

13. **Clauza GROUPING SETS.** Permite obținerea numai a anumitor grupări superagregat. Acestea pot fi precizate prin intermediul clauzei:

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

14. 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.

```
SELECT department_name, job_title, e.manager_id, MAX(salary) "Maxim",
 SUM(salary) "Suma"
FROM   employees e, departments d, jobs j
WHERE  e.department_id = d.department_id
AND     e.job_id = j.job_id
GROUP BY GROUPING SETS ((department_name, job_title),
                        (job_title, e.manager_id), ());
```

LABORATOR 5 - SQL - REZUMAT
Subcereri. Operatori. Cereri cu sincronizare (corelate).

Subcereri

1. Să se obțină numele primilor 5 angajați care au salariul cel mai mare. Rezultatul se va ordona descrescător după salariu.

```
SELECT last_name, job_id, salary
FROM employees e
WHERE 5>(SELECT COUNT(*)
         FROM employees
         WHERE salary > e.salary)
ORDER BY salary DESC;
```

sau

```
SELECT  *
FROM  (SELECT last_name, job_id, salary
       FROM  employees
       ORDER BY salary DESC)
WHERE ROWNUM<=5;
```

2. Să se afişeze numele, job-ul şi salariul celor mai prost plătiți angajați din fiecare departament.

Fără sincronizare

```
SELECT last_name, salary, job_id, department_id
FROM employees
WHERE (department_id, salary) IN (SELECT department_id, MIN(salary)
                                  FROM employees
                                  GROUP BY department_id);
```

Cu sincronizare

```
SELECT last_name, salary, job_id, department_id
FROM employees e
WHERE salary  = (SELECT MIN(salary)
                 FROM employees
                 WHERE department_id=e.department_id);
```

3. Să se obțină codurile şi numele departamentelor în care nu lucrează nimeni.
Rulați următoarea cerere SQL. Ce observați? Modificați cererea astfel încât să fie corectă.

Observație:

Dacă este utilizat operatorul *NOT NULL*, atunci subcererea nu trebuie să întoarcă valori *NULL*.

```
SELECT department_name, department_id
FROM departments
```

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

4. Să se afişeze numele şi salariul angajaților al căror salariu este mai mare decât salariile medii din toate departamentele.

```
SELECT last_name, salary
FROM employees
WHERE salary > ALL (SELECT AVG(salary)
                    FROM employees
                    GROUP BY department_id);
```

Operatori pe mulțimi

5. Să se creeze o cerere prin care să se afişeze numărul total de angajați şi, din acest total, numărul celor care au fost angajați în 1997.

```
SELECT COUNT(*)|| '  nr_total '  numar
FROM employees
  UNION
SELECT COUNT(*)|| '  nr_1980 '  numar_1997
FROM   employees
WHERE  TO_CHAR(hire_date,'YYYY')=1997;
```

6. Să se obțină, folosind operatorul *INTERSECT*, angajații care au salariul < 3000 şi al căror nume conține litera *a* pe poziția 3.

```
SELECT employee_id, last_name
FROM employees
WHERE salary<3000
  INTERSECT
SELECT employee_id, last_name
FROM employees
WHERE UPPER(last_name) LIKE '__A%';
```

7. Să se afişeze codurile departamentelor care nu au angajați, implementând operatorul MINUS.

```
SELECT department_id
FROM departments
   MINUS
SELECT DISTINCT department_id
FROM employees;
```

Operatorul boolean EXISTS

8. Să se obțină numele şi codul angajaților care au salariul mai mare decât angajatul cu codul 200.

```
SELECT employee_id, last_name
FROM employees e
WHERE EXISTS
       (SELECT  *
        FROM employees
        WHERE employee_id = 200
        AND e.salary >salary);
```

9. Să se determine codul şi numele departamentelor în care nu lucrează nimeni, folosind operatorul *EXISTS*.

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'x'
                  FROM employees
                  WHERE department_id =d.department_id);
```

LABORATOR 6 SQL - REZUMAT
SQL*PLUS

1. Să se afişeze numele, job-ul şi salariul angajaților care au salariul cuprins între 2 numere introduse de utilizator.

```
ACCEPT n PROMPT 'n='
ACCEPT m PROMPT 'm='

SELECT  last_name, job_id, salary
FROM    employees
WHERE   salary BETWEEN &n AND &m;
```

2. Să se mențină într-o variabilă de legătură numele salariatului având codul 100.

```
VARIABLE  v_nume VARCHAR2(20)
 BEGIN
       SELECT last_name
       INTO   :v_nume
       FROM   employees
       WHERE  employee_id = 100;
END;
/
PRINT  v_nume
```

LABORATOR 7 SQL - REZUMAT

Operatorul DIVISION

1. Să se afişeze codul şi numele proiectelor la care au lucrat toţi angajaţii din departamentul 20.

Varianta 1

```
SELECT  p.project_id, project_name
FROM    projects p, work w
WHERE   p.project_id=w.project_id
AND     employee_id IN (SELECT employee_id
                        FROM   employees
                        WHERE  department_id =20)
GROUP BY p.project_id, project_name
HAVING COUNT(*)=(SELECT COUNT(*)
                 FROM   employees
                 WHERE  department_id =20);
```

Varianta 2

```
SELECT  DISTINCT p.project_id, project_name
FROM    projects p, work w
WHERE   p.project_id=w.project_id
AND     NOT EXISTS (SELECT 'X'
                    FROM   employees e
                    WHERE  department_id=20
                    AND
                    NOT EXISTS (SELECT 'X'
                                FROM    work w1
                                WHERE   e.employee_id=w1.employee_id
                                AND     w.project_id=w1.project_id));
```

Limbajul de prelucrare a datelor (LMD)

1. Să se creeze tabele *emp_**** şi *dept_****, având aceeaşi structură şi date ca şi tabelele *employees*, respectiv *departments*.

```
CREATE TABLE emp_*** AS
SELECT * FROM employees WHERE 0=1;


CREATE TABLE dept_*** AS
SELECT * FROM departments WHERE 0=1;
```

2. Sintaxa simplificată a comenzii ***INSERT***

- pentru inserarea unei singure linii:

```
INSERT INTO nume_tabel [(col1,col2,...)]
VALUES (expresie1, expresie2, ...);
```

- pentru inserarea liniilor rezultat ale unei comenzi SELECT:

```
INSERT INTO nume_tabel [(col1,col2,...)]
comanda_SELECT;
```

3. Inserați în tabelul *emp_**** salariații (din tabelul *employees*) al căror comision depășește 25% din salariu. Salvați modificările.
4. Inserați o linie nouă în tabelul *dept_****, folosind valori introduse de la tastatură.
5. Inserați o linie nouă în tabelul *dept_****. Salvați într-o variabilă de legătură codul departamentului nou introdus. Afișați valoarea menținută în variabila respectivă. Anulați tranzacția.

```
VARIABLE v_cod NUMBER

INSERT INTO dept_*** (department_id, department_name,location_id)
VALUES (200, 'dept_nou', 2000)
RETURNING department_id INTO :v_cod;

PRINT v_cod

ROLLBACK;
```

6. Sintaxa simplificată a comenzii ***DELETE***

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

7. Ştergeți toate înregistrările din tabelele *emp_**** şi *dept_****. Inserați în aceste tabele toate înregistrările corespunzătoare din *employees*, respectiv *departments*. Permanentizați tranzacția.

```
DELETE FROM dept_***;
DELETE FROM emp_***;

INSERT INTO emp_***
SELECT * FROM employees;

INSERT INTO dept_***
SELECT * FROM departments;

COMMIT;
```

8. Eliminați departamentele care nu au nici un angajat. Anulați modificările.
9. Ştergeți un angajat al cărui cod este dat de la tastatură.
10. Să se şteargă angajatul având codul 100. Să se mențină numele acestuia într-o variabilă de legătură. Afișați valoarea acestei variabile.

```
VARIABLE t VARCHAR2(20)

DELETE FROM emp_***
WHERE  employee_id = 100
RETURNING first_name INTO :t;

PRINT t
```

```
ROLLBACK;
```

11. Sintaxa simplificată a comenzii ***UPDATE***:

```
UPDATE nume_tabel [alias]
SET    col1 = expr1[, col2=expr2]
[WHERE conditie];
```

sau

```
UPDATE nume_tabel [alias]
SET (col1,col2,...) = (subcerere)
[WHERE conditie];
```

12. Măriți cu 5% salariul tuturor angajaților . Anulați modificările.

```
UPDATE emp_***
SET salary = salary * 1.05;
ROLLBACK;
```

13. Măriți cu 5% salariul tuturor angajaților care lucrează în departamentul 50. Anulați modificările.

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

```
UPDATE emp_***
SET (job_id, department_id) = (SELECT job_id, department_id
                               FROM emp_***
                               WHERE employee_id = 205)
WHERE employee_id = 114;
ROLLBACK;
```

15. Să se modifice cererea de la exercițiul anterior astfel încât actualizarea coloanei *email* să fie realizată doar pentru angajatul având codul 200. Să se mențină numele şi emailul acestuia în două variabile de legătură. Să se anuleze tranzacția.

```
VARIABLE v_nume VARCHAR2(20)
VARIABLE v_email VARCHAR2(20)

UPDATE emp_***
SET   email = LOWER(SUBSTR(first_name,1,1)) || '_' ||LOWER(last_name)
WHERE employee_id = 200
RETURNING last_name, email INTO :v_nume, :v_email;

PRINT v_nume
PRINT v_email

ROLLBACK;
```

16. Măriți cu 1000 salariul unui angajat al cărui cod este introdus de la tastatură.

LABORATOR 8 SQL - REZUMAT
Limbajul de definire a datelor (CREATE, ALTER, DROP)

Crearea tabelelor

```
CREATE TABLE [schema.]nume_tabel (
     nume_coloana tip_de_date [DEFAULT expr], ...);

CREATE TABLE nume_tabel [(col1, col2...)]
      AS subcerere;
```

1. Creați tabelul *salariat_**** având următoarea structură:

Nume	Caracteristici	Tip
cod_ang	NOT NULL	NUMBER(4)
nume		VARCHAR2(25)
prenume		VARCHAR2(25)
functia		VARCHAR2(20)
sef		NUMBER(4)
data_angajarii	Valoare implicită data curentă	DATE
varsta		NUMBER(2)
email		CHAR(20)
salariu	Valoare implicită 0	NUMBER(9,2)

```
CREATE TABLE salariat_*** (
  cod_ang NUMBER(4) NOT NULL,
  nume    VARCHAR2(25),
  prenume VARCHAR2(25),
  functia VARCHAR2(20),
  sef     NUMBER(4),
  data_angajarii DATE DEFAULT SYSDATE,
  varsta  NUMBER(2),
  email   CHAR(20),
  salariu NUMBER(9,2) DEFAULT 0);
```

2. Se dau următoarele valori:

COD _ANG	NUME	PRENUME	FUNCTIA	SEF	DATA_ANG	VARSTA	EMAIL	SALARIU
1	.....	.....	director	null	.....	30	.....	5500
2	.....	.....	functionar	1	.....	25	.....	0

3. Inserați în tabelul salariat_*** prima înregistrare din tabelul de mai sus fără să precizați lista de coloane în comanda INSERT.

4. Inserați a doua înregistrare folosind o listă de coloane din care excludeți data_angajarii şi salariul care au valori implicite. Observați apoi rezultatul.

Modificarea tabelelor

5. Adăugați o nouă coloană tabelului *salariat_**** care să conțină data naşterii.

```
ALTER TABLE salariat_***
ADD (datan DATE);
```

6. Eliminați coloana *varsta* din tabelul salariat_***.

```
ALTER TABLE salariat_***
DROP COLUMN varsta;
```

Constrângeri

Tipuri de constrângeri:

- NOT NULL - coloane ce nu pot conține valoarea *Null*; (`NOT NULL`)
- UNIQUE - coloane sau combinații de coloane care trebuie să aibă valori unice în cadrul tabelului; (`UNIQUE (col1, col2, …)`)
- PRIMARY KEY - identifică în mod unic orice înregistrare din tabel. Echivalent cu NOT NULL + UNIQUE; (`PRIMARY KEY (col1, col2, … ))`
- FOREIGN KEY - stabileşte o relație de cheie externă - cheie primară între o coloană a tabelului şi o altă coloana dintr-un tabel specificat.

```
[FOREIGN KEY nume_col]
     REFERENCES nume_tabel(nume_coloana)
     [ ON DELETE {CASCADE| SET NULL}]
```

  - *FOREIGN KEY* este utilizat într-o constrângere la nivel de tabel pentru a defini coloana din tabelul „copil“;
  - *REFERENCES* identifică tabelul „părinte“ şi coloana corespunzătoare din acest tabel;
  - *ON DELETE CASCADE* determină ca, odată cu ştergerea unei linii din tabelul „părinte“, să fie şterse şi liniile dependente din tabelul „copil“;
  - *ON DELETE SET NULL* determină modificarea automată a valorilor cheii externe la valoarea *null*, atunci când se şterge valoarea „părinte“.

- CHECK - o condiție care să fie adevărată la nivel de coloană sau linie (`CHECK (conditie)`).

Constrângerile pot fi create cu tabelul sau adăugate ulterior cu o comandă ALTER TABLE.

Adăugarea constrângerilor la crearea tabelului (CREATE TABLE)

```
CREATE TABLE [schema.]nume_tabel (
      nume_coloana tip_de_date [DEFAULT expr]
           [constrangere_de_coloana], ...
      [constrangere la nivel de tabel])
```

7. Ştergeți şi apoi creați din nou tabelul *salariat_**** cu următoarea structură.

NUME	TIP	CONSTRÂNGERE
cod_ang	NUMBER(4)	Cheie primară
nume	VARCHAR2(25)	NOT NULL
prenume	VARCHAR2(25)	
data_nasterii	DATE	data_nasterii<data_angajarii
functia	VARCHAR2(9)	NOT NULL
sef	NUMBER(4)	Referă coloana cod_ang din acelaşi tabel
data_angajarii	DATE	

email	VARCHAR2(20)	unic
salariu	NUMBER(12,3)	> 0
cod_dept	NUMBER(4)	
		Combinația NUME şi PRENUME să fie unică

Observaţie:

Constrângerile de tip CHECK se pot implementa la nivel de coloană doar dacă nu referă o altă coloană a tabelului.

```
DROP TABLE salariat_***;
CREATE TABLE salariat_*** (
  cod_ang  NUMBER(4) PRIMARY KEY,
  nume     VARCHAR2(25) NOT NULL,
  prenume  VARCHAR2(25),
  data_nasterii DATE,
  functia  VARCHAR2(9) NOT NULL,
  sef      NUMBER(4) REFERENCES salariat_*** (cod_ang),
  data_angajarii DATE DEFAULT SYSDATE,
  email    VARCHAR2(20)UNIQUE,
  salariu  NUMBER(9,2) CONSTRAINT c_*** CHECK (salariu > 0),
  cod_dep  NUMBER(4),
CONSTRAINT const_c_*** CHECK (data_angajarii > data_nasterii),
CONSTRAINT const_u_*** UNIQUE (nume,prenume,data_nasterii));
```

8. Ştergeţi tabelul *salariat_****, iar apoi recreaţi-l implementând toate constrângerile la nivel de tabel.

Observaţie: Constrângerea de tip NOT NULL se poate declara doar la nivel de coloană.

```
DROP TABLE salariat_***;
CREATE TABLE salariat_*** (
cod_ang NUMBER(4),
nume    VARCHAR2(25) NOT NULL,
prenume VARCHAR2(25),
data_nasterii DATE,
functia VARCHAR2(9) NOT NULL,
sef     NUMBER(4),
data_angajarii DATE DEFAULT SYSDATE,
email   VARCHAR2(20),
salariu NUMBER(9,2),
cod_dep NUMBER(4),
CONSTRAINT ccp_*** PRIMARY KEY (cod_ang),
CONSTRAINT cce_*** FOREIGN KEY (sef) REFERENCES salariat_***(cod_ang),
CONSTRAINT cu1_*** UNIQUE (email),
CONSTRAINT cc1_*** CHECK (data_angajarii > data_nasterii),
CONSTRAINT cc2_*** CHECK (salariu > 0),
CONSTRAINT cu2_*** UNIQUE (nume,prenume,data_nasterii));
```

10. Creaţi tabelul *departament_**** care să aibă următoarea structură.

NUME	**TIP**	**CONSTRÂNGERI**
COD_DEP	NUMBER(4)	Cheie primară
NUME	VARCHAR2(20)	
ORAS	VARCHAR2(25)	Not null

b. Ulterior creării tabelului, adăugați constrângerea NOT NULL pe coloana *nume*.

```
ALTER TABLE departament_***
MODIFY nume NOT NULL;
```

c. Eliminați constrângerea NOT NULL definită pe coloana *oras*.

Adăugarea constrângerilor ulterior creării tabelului, eliminarea, activarea sau dezactivarea constrângerilor (ALTER TABLE – nu se aplică pentru NOT NULL)

- adaugă constrângeri

```
ALTER TABLE nume_tabel
ADD [CONSTRAINT nume_constr] tip_constr (coloana);
```

- elimină constrângeri

```
ALTER TABLE nume_tabel
DROP [CONSTRAINT nume_constr] tip_constr (coloana);
```

- activare/dezactivare constrângere

```
ALTER TABLE nume_tabel
MODIFY CONSTRAINT nume_constr ENABLE|DISABLE;
```

sau

```
ALTER TABLE nume_tabel
ENABLE| DISABLE nume_constr;
```

11. Inserați o nouă înregistrare în *salariat_*** de forma:

cod	nume	prenume	data_n	functia	sef	data_ang	email	salariu	cod_dep
2	N2	P2	11-JUN-1960	economist	1	Sysdate	E2	2000	10

Ce observați? Introduceți înregistrarea dar specificând valoarea NULL pentru coloana *sef*.

12. Încercați să adăugați o constrângere de cheie externă pe *cod_dep* din *salariat_****. Ce observați?

```
ALTER TABLE salariat_***
ADD CONSTRAINT cce2_*** FOREIGN KEY (cod_dep) REFERENCES
    departament_*** (cod_dep);
```

13. Inserați o nouă înregistrare în *departament_****. Apoi adăugați constrângerea de cheie externă definită anterior.

cod_dep	nume	loc
10	Economic	Bucuresti

14. Inserați noi înregistrări în *salariat_****, respectiv în *departament_****. Care trebuie să fie ordinea de inserare?

cod	nume	prenume	data_n	functia	sef	data_ang	email	salariu	cod_dep
3	N3	P3	11-JUN-1967	jurist	2	Sysdate	E3	2500	20

cod_dep	nume	loc
20	Juritic	Constanta

15. Ştergeți departamentul 20 din tabelul *departament_****. Ce observați?

LABORATOR 9 SQL- REZUMAT
Vizualizări

- **Definirea vizualizărilor**

1. Să se creeze vizualizarea *v_emp_**** care să conțină codul şi numele salariaților din tabelul *emp_****. Să se afişeze conținutul acesteia. Să se insereze o nouă înregistrare în această vizualizare. Ce observați? Să se şteargă vizualizarea v_emp_***.

```
CREATE VIEW v_emp_*** (cod, nume)
AS SELECT employee_id, last_name
   FROM   emp_***;

INSERT INTO v_emp_***
VALUES (400,'N1');

DROP VIEW v_emp_***;
```

2. Să se creeze vizualizarea *v_emp_**** care să conțină codul, numele, emailul, data angajării, salariul şi codul jobului salariaților din tabelul *emp_****. Să se analizeze structura şi conținutul vizualizării. Să se insereze o nouă înregistrare în această vizualizare. Să se verifice că noua înregistrare a fost inserată şi în tabelul de bază.

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

```
CREATE VIEW v_emp_***
AS SELECT employee_id, last_name, email, hire_date, salary,job_id
   FROM   emp_***;

DESC v_emp_***

SELECT * FROM v_emp_***;

INSERT INTO v_emp_***
VALUES (400,'N1','E1',SYSDATE,5000,'SA_REP');

SELECT employee_id, last_name, email, hire_date, salary, job_id
FROM   emp_***;
```

3. Să se şteargă angajatul având codul 400 din vizualizarea creată anterior. Ce efect va avea această acțiune asupra tabelului de bază?

```
DELETE FROM  v_emp_***
WHERE  employee_id = 400;

SELECT employee_id, last_name, salary
FROM   emp_***
WHERE  employee_id = 400;
```

5. a) Să se creeze vizualizarea *v_emp_dept_*** care să conțină *employee_id, last_name, hire_date, job_id, department_id* din tabelul *emp_**** şi coloana *department_name* din tabelul *dept_***.

```
CREATE VIEW  v_emp_dept_***
AS SELECT employee_id, last_name, email, hire_date, job_id,
          e.department_id, department_name
   FROM   emp_*** e, dept_*** d
   WHERE  e.department_id =d.department_id;
```

b) Să încerce inserarea înregistrării (500, 'N2', 'E2',SYSDATE,'SA_REP',30, 'Administrativ') în vizualizarea creată anterior.

```
INSERT INTO v_emp_dept_***
VALUES (500, 'N2', 'E2',SYSDATE,'SA_REP',30, 'Administrativ');
```

c) Care dintre coloanele vizualizării *v_emp_dept_*** sunt actualizabile?

```
SELECT  *
FROM    user_updatable_columns
WHERE   UPPER(table_name) = UPPER('v_emp_dept_***');
```

d) Adăugați tabelului *emp_**** constrângerea de cheie externă care referă tabelul *dept_***, apoi verificați ce coloane din vizualizarea *v_emp_dept_*** sunt actualizabile.

```
ALTER TABLE emp_***
ADD CONSTRAINT cp_emp_*** PRIMARY KEY (employee_id);

ALTER TABLE dept_***
ADD CONSTRAINT cp_dept1_*** PRIMARY KEY (department_id);

ALTER TABLE emp_***
ADD CONSTRAINT ce_emp1_*** FOREIGN KEY (department_id)
         REFERENCES dept_***(department_id);

SELECT  *
FROM    user_updatable_columns
WHERE   UPPER(table_name) = UPPER('v_emp_dept_***');
```

d) Recreați vizualizarea *v_emp_dept_***, apoi verificați ce coloane sunt actualizabile.

```
DROP VIEW v_emp_dept_***;
CREATE VIEW  v_emp_dept_***
AS SELECT employee_id, last_name, email, hire_date, job_id,
          e.department_id, department_name
   FROM   emp_*** e, dept_*** d
   WHERE  e.department_id =d.department_id;

SELECT column_name, updatable
FROM   user_updatable_columns
WHERE  UPPER(table_name) = UPPER('v_emp_dept_***');
```

f) Inserați o linie prin intermediul acestei vizualizări.
Obs. Tabelul ale cărui coloane sunt actualizabile este protejat prin cheie.

```
INSERT INTO v_emp_dept_*** (employee_id, last_name,email,hire_date,
                            job_id, department_id)
VALUES (500, 'N2', 'E2',SYSDATE,'SA_REP',30);
```

g) Ce efect are o operație de ștergere prin intermediul vizualizării *v_emp_dept_****? Comentați.

```
DELETE FROM v_emp_dept_***
WHERE employee_id = 500;

SELECT employee_id, last_name, hire_date, job_id, department_id
FROM   emp_***
WHERE employee_id = 500;

SELECT department_id, department_name
FROM   dept_***
WHERE department_id = 30;
```

4. a) Să se creeze vizualizarea *v_emp30_**** care să conțină numele, emailul, data angajării, salariul, codul jobului şi codul departamentului celor care lucrează în departamentul 30. În această vizualizare nu se va permite modificarea sau inserarea liniilor ce nu sunt accesibile ei. Dați un nume constrângerii.

```
CREATE VIEW v_emp30_***
  AS SELECT  employee_id, last_name, email, hire_date, salary, job_id,
             department_id
  FROM emp_***
  WHERE department_id=30
  WITH CHECK OPTION CONSTRAINT ck_option1_***;
```

b) Să se listeze structura şi conținutul vizualizării *v_emp30_****.

```
DESCRIBE v_emp30_***
SELECT * FROM v_emp30_***;
```

c) Să se încerce prin intermediul vizualizării inserarea unui angajat în departamentul 10 şi a unui angajat în departamentul 30.

```
INSERT INTO v_emp30_***
VALUES (111, 'N1', 'E1',SYSDATE,1000,'SA_REP',10);

INSERT INTO v_emp30_***
VALUES (11, 'N11', 'E11',SYSDATE,1000,'SA_REP',30);
```

d) Să se încerce prin intermediul vizualizării modificarea departamentului unui angajat.

```
UPDATE v_emp30_***
SET department_id =20
WHERE employee_id = 11;
```

5. Să se creeze vizualizarea *v_dept_**** asupra tabelului *dept_**** să nu permită efectuarea nici unei operații LMD. Testați operațiile de inserare, modificare şi ştergere asupra acestei vizualizări.

```
CREATE VIEW v_dept_***
AS SELECT *
FROM dept_***
WITH READ ONLY;
```