Recapitulare
SQL

In acest tutoriat am sa prezint pe scurt(sau cel putin asa sper) majoritatea clauzelor/functiilor/conceptelor ce au fost folosite in testele de laborator din anii trecuti.

1. **AS** – operator folosit pentru a schimba numele unei coloane. Ex:

```
1. --Afisati prenumele angajatului intr-o coloana cu numele "Prenume" si numele
2. --pe o coloana numita "Nume"
3. SELECT first_name AS Prenume, last_name AS Nume
4. FROM employees;
```

2. **|| (concatenare)** - operator folosit pentru a concatena mai multe date sau, date cu string-uri pe aceasi coloana. Ex:

```
1. --Afisati numele intreg al angajatului in aceasi coloana sub forma " Prenume, Nume"
2. SELECT first_name || ', ' || last_name
3. FROM employees;
```

3. **SYSDATE** – variabila ce contine date curenta. Ex:

```
1. --Afisati data curenta
2. SELECT SYSDATE
3. FROM DUAL;
```

4. **TO_CHAR(var, format)** – functie ce primeste un numar sau data calendaristica si returneaza un string al variabilei intr-un anumit format. Este folosit preponderent pentru a formata diferite date. Posibilele modalitati de a formata o data sunt:
    - Afiseaza un numar ce reprezinta diferite inforamtii daca 'format' =
        - 'D' – numarul zile din saptamana.
        - 'DD' – numarul zilei din luna.
        - 'DDD' – numarul zilei din an.
        - 'DY' – abrevierea cu 3 litere a numelui zilei din saptamana.
        - 'DAY' – numele intreg al zilei din saptamana.
        - 'MM' – numarul lunii din an.

- ’MON’ / ’MONTH’ – abrevierea cu 3 litere a numelui lunii / numele intreg al lunii.
- ’Y’ / ’YY’ / ’YYY’ / ’YYYY’ – ultima cifra din an/ ultimele 2 cifre din an/ ultimele 3 cifre din an/ ultimele 4 cifre din an.
- ’YEAR’ – anul scris cu litere.
- ’MI’ – minutele din ora .
- ’SS’ – secundele din minut.
- ’HH12’/’HH24’ – ora din zi in format de 12 sau 24 de ore.

Ex:

```
1.  --Afisati numele zilei, numarul lunii si anul in care
2.  --fiecare angajat a fost angajat
3.  SELECT TO_CHAR(hire_date,'DAY'), TO_CHAR(hire_date,'MM'), TO_CHAR(hire_date,'YYYY')
4.  FROM employees;
```

5. **NVL(A,B)** – functie ce daca A nu este NULL intoarce A, altfel intoarce B (nu functioneaza daca tipul lui B de date nu poate fi convertit in tipul lui A, adica daca A este numar si B string). Ex:

```
1.  --Afisati toate comisioanele angajatiilor,
2.  --daca comisionul este NULL sa se scrie 0
3.  SELECT NVL(commission_pct, 0)
4.  FROM employees;
```

6. **NVL2(A,B,C)** – functie ce daca A nu este NULL intoarce B, altfel intoarce C. Ex:

```
1.  --Sa se afiseze pt fiecare departament ce are manager "Departament permanent”,
2.  --daca un departament nu are manager se va afisa "Departmanet temporar"
3.  SELECT NVL2(manager_id,'Departament permanent','Departament temporar')
4.  FROM departments;
```

7. **IN** – operator cu care verific daca o valoare este intr-o lista de valori. Poate fi negat folsind NOT IN. Ex:

```
1.  1.  --Afisati toti angajatii ce poarta prenumele de
2.  2.  --'John' sau 'Michael' sau 'David' sau 'Jennifer'
3.  3.  SELECT *
4.  4.  FROM employees
5.  5.  WHERE first_name IN ('John', 'Michael', 'David', 'Jennifer')
```

8. **BETWEEN** – opearator cu care verific daca o valoare este cuprinsa intre alte 2 (functioneaza si daca vrem sa verificam prima litera dintr-un string). Poate fi negat folosind NOT BETWEEN. Ex:

```
--Afisati toti angajatii cu salariu cuprins intre 5000 si 9000
SELECT *
FROM employees
WHERE salary BETWEEN 5000 AND 9000;

--Afisati toti angajatii al caror initiala este intre "C" si "T"
SELECT *
FROM employees
WHERE last_name BETWEEN 'C' AND 'T';
```

9. **DISTINCT** – valorile de pe coloana repsectiva o sa fie distincte (orice valoare o sa apara numai o data). Ex:

```
--Afisati toate prenumele angajatilor o singura data
SELECT DISTINCT first_name
FROM employees;
```

10. **LIKE** – verifica daca un string este de un anumit format in functie de anumite caractere. Poate fi negat folosinf NOT LIKE:
    - '_' – semnifica un caracter oarecare. Ex:

```
--Afisati angajatii ce au numele de 5 litere si a 3-a este 'a'
SELECT *
FROM employees
WHERE last_name LIKE '__a__';
```

    - '%' – semnifica un numar de caractere mai mare sau egal cu 0. Ex:

```
--Afisati toti angajatii al caror nume se termina in 'n'
SELECT *
FROM employees
WHERE last_name LIKE '%n';

--Afisat toti angajatii al caror nume contine 'ab'
SELECT *
FROM employees
WHERE last_name LIKE '%ab%';
```

11. **LENGHT(string)** – functie ce intoarce lungimea unui string. Ex:

```
--Afisati lungimile prenumelor tututor angajatiilor
SELECT LENGTH(first_name)
FROM employees;
```

12. **UPPER(string)/LOWER(string)** – transforma toate literele dintr-un string in majuscule/litere mici. Ex:

```
--Afisati prenumele tuturor angajatilor cu majuscule
--si numele cu litere mici
SELECT UPPER(first_name), LOWER(last_name)
FROM employees;
```

13. **REPLACE(X, A, B)** – inlocuieste toate aparitiile string-ului A cu string-ul B in string-ul X. Ex:

```
--Inlocuiti 'x' cu 'ics'in toate prenumele angajatilor
SELECT REPLACE(first_name, 'x', 'cs')
FROM employees;
```

14. **SUM(nume_col), COUNT(nume_col), AVERAGE(nume_col)** – functii ce calculeaza suma, numara cate linii exista sau fac media unei coloane. Ex:

```
--Afisati suma tuturor salariilor
SELECT SUM(salary)
FROM employees;

--Afisati media tuturor salariilor
SELECT AVG(salary)
FROM employees;

--Afisati cati angajati sunt in firma
SELECT COUNT(employee_id)
FROM employees;
```

15. **JOIN** – este un concept ce leaga datele dintr-un tabel cu datele altui tabel intr-un mod logic. Pentru a face JOIN intre 2 tabele verificam ca valoarea unei coloane gasita in ambele tabele (de obicei cheia primara a unui tabel ce este cheie externa pentru celalalt) sa fie indetica. Liniile ce au valoarea pe aceasta coloana egala sunt unite, ele reprezentand un intreg. Ex:

```
--Sa se afiseze toti angajatii cu numele departamentului si al jobului lor
SELECT e.last_name, d.department_name, j.job_title
FROM employees e, departments d, jobs j HERE e.department_id = d.department_id
  AND e.job_id = j.job_id;
--OBS: daca punem un caracter/cuvant
--dupa numele tabelului acesta devine aliasul tabelului
```

16. **OUTER JOIN** – un join simplu uneste 2 linii din tabele diferite numai daca valoarea unei coloanei specificate de pe o linie dintr-un tabel are o linie cu aceasi valoare in celalalt tabel. Daca dorim totuis sa afisam TOATE liniile unui tabel intr-un join (inclusiv cele ce nu au corespondent) folosim OUTER JOIN. RIGHT OUTER JOIN daca tabelul pe care dorim sa-l afisam integral este in STANGA egalului(adica umplem tabelul din dreapta cu NULL daca nu exista corespondent) sau LEFT OUTER JOIN invers. Daca dorim sa afisam integral ambele tabele o sa facem FULL OUTER JOIN. Ex:

```
1.  --Sa se afiseze toti angajatii si departametele lor
2.  -- a) inclusiv angajatii ce nu au departament
3.  SELECT *
4.  FROM employees e, departments d
5.  WHERE e.department_id = d.department_id(+); -- RIGHT OUTER JOIN
6.
7.  -- b) inclusiv departamentele ce nu au angajati
8.  SELECT *
9.  FROM employees e, departments d
10. WHERE e.department_id(+) = d.department_id; -- LEFT OUTER JOIN
11.
12. – c)ambele tabele integral
13. SELECT *
14. FROM employees e, departments d
15. WHERE e.department_id(+) = d.department_id(+); -- FULL OUTER JOIN
16. -- Pentru a nu confunda LEFT si RIGHT JOIN ne putem gandi ca unde se afla (+)
17. – acel tabel este in OUTER JOIN
```

17. **CASE** – asemanator cu un switch el :
    - Primeste o valoarea si daca este egala cu una din valoarile scrise de utilizator intoarce ceva in functie de PRIMA ramura pe care intra. Daca nu intra pe nicio ramura intoarce o valoare default. Ex:

```
1.  --Sa se scrie numele intreg al tarii din fiecare locatie daca
2.  --aceasta este 'IT','JP','US' sau 'CA', daca nu sa se scrie 'TARA SECUNDARA'
3.  SELECT CASE country_id WHEN 'IT' THEN 'ITALY'
4.                         WHEN 'JP' THEN 'JAPAN'
5.                         WHEN 'US' THEN 'AMERICA'
6.                         WHEN 'CA' THEN 'CANADA'
7.                         ELSE 'TARA SECUNDARA' END
8.  FROM locations;
```

    - Primeste mai multe expresii si in functie de valorile lor de adevar intoarce PRIMA valoare unde expresia este adevarata. Daca nu intra pe nicio ramura intoarce o valoare default. Ex:

```
1.  --Sa se afiseze pentru toti angajatii: 'SALARIU MIC' daca
2.  --au un salariu mai mic de 5000, 'SALARIU MEDIU' daca este intre
3.  -- 5000 si 10000 si 'SALARIU MARE' altfel
4.  SELECT CASE WHEN salary < 5000 THEN 'SALARIU MIC'
5.              WHEN salary <=10000 THEN 'SALARIU MEDIU'
6.              ELSE 'SALARIU MARE' END
7.  FROM EMPLOYEES;
```

18. **DECODE(VAL, A1,B1, A2,B2, …,DEFAULT)** – are acelasi scop ca si CASE dar este o functie. Verifica daca VAL este egal cu un A si returneaza B-ul corespunzator, daca nu intoarce DEFAULT. Ex:

```
--Sa se scrie numele intreg al tarii din fiecare locatie daca
--aceasta este 'IT','JP','US' sau 'CA', daca nu sa se scrie 'TARA SECUNDARA'
SELECT DECODE(country_id, 'IT', 'ITALY', 'JP', 'JAPAN', 'US', 'AMERICA', 'CA', 'CANADA'
, 'TARA SECUNDARA' )
FROM locations;
```

19. **ORDER BY** – clauza ce ordoneaza rezultatul unei cereri in functie de una sau mai multe coloane. Daca in clauza sunt scrise mai multe coloane atunci sortarea se va face dupa prima coloana, daca 2 sau mai multe linii au valoarea primei coloane egale atunci o sa fie sortate dupa a doua coloana si tot asa. Dupa o coloana se poate sorta crescator (valoare default dar, optional, se poate trece ASC daca dorim sa se vada tipul sortari), sau descrescator (notata DESC). Ex:

```
--Sa se sorteze angajatii descrescator dupa salariu,
--daca 2 angajati au acelasi salariu sa se sorteze alfabetic dupa nume
SELECT *
FROM employees
ORDER BY salary DESC, last_name;
```

20. **SUBCERERI** – cereri ce pot fi facute in interiorul altor cereri si al caror rezultat poate fi folosit in clauza WHERE sau SELECT a unei cereri. Acestea sunt de 2 tipuri:
    - Nesincronizate – cand tabelele din cererea principala nu sunt folosite in subcerere.Ex:

```
--Afisati salariile tuturor managerilor
SELECT salary
FROM employees
WHERE employee_id IN (SELECT DISTINCT manager_id
                      FROM employees);
```

    - Sincronizate – cand tabelele din cererea principala sunt folosite in subcerere.Ex:

```
--Afisati toti angajatii impreuna cu suma salariilor
--tuturor angajatiilor din departamentul unde lucreaza
SELECT first_name , (SELECT SUM(salary)
                     FROM employees e2
                     WHERE e2.department_id = e1.department_id)
FROM employees e1;
```

21. **WITH** – este un mod de a stoca pe durata unei cereri rezultatele altor cereri si de a le putea trata ca tabele. De asemenea daca cream mai multe astfel de „tabele” cu WITH putem intr-un tabel sa accesam toate „tabelele” create anterior. Recomand ca pentru aceste „tabele” fiecare coloana din SELECT sa aiba un alias pentru a putea fi accesate usor. Ex:

```
--Afiati salariile tuturor managerilor
WITH manageri AS (SELECT DISTINCT manager_id AS manager
                  FROM employees)
SELECT e.salary
FROM employees e, manageri m
WHERE employee_id IN m.manager;
```

22. **Subcereri in FOR** – asemanator cu WITH numai ca cererile ce o sa poate fi folosite ca tabele sunt declarate in FOR. Obligatoriu pentru a putea folosi un astfel de tabel el are nevoie de un alias. Ex:

```
--Afiati salariile tuturor managerilor
SELECT e.salary
FROM employees e,(SELECT DISTINCT manager_id AS manager FROM employees) m
WHERE employee_id IN m.manager;
```

23. **ROWNUM** – este o valoare din tabelul ce contine date despre rezultatul cererii. Din ea putem sa micsoram numarul de linii cu care se lucreaza si care sunt afisate pe ecran. Este important de stiut ca aceasta micsorare a liniilor are loc in clauza WHERE deci TOATE operatiile is clauzele ce o sa aiba loc DUPA linia cu ROWNUM (ORDER BY de exemplu) o sa fie facute pe tabela micsorata. Acest lucru poate duce la afisari gresite daca nu este facuta cu atentie. Daca dorim sa facem operatii pe toate datele tabelei atunci este recomandata folosirea unei subcereri sau a lui WITH si apoi, in alta subcerere sau in cererea principala putem micsora numarul de linii, dupa ce ne-am asigurat ca pe acestea nu mai trebuie facuta nicio operatie. Ex:

```
--Afisati primii 5 cei mai bine platiti oameni
WITH angajati_ordonati AS(SELECT *
                          FROM employees
                          ORDER BY SALARY DESC)
SELECT *
FROM angajati_ordonati
WHERE ROWNUM <= 5;
```

24. **EXISTS** – verifica daca rezultatul unei subcereri este sau nu vid (are sau nu linii). Poate fi negat folosind NOT EXISTS. Ex:

```
--Afisati toti angajatii care au fost angajati in aceasi data cu altcineva
SELECT *
FROM employees e1
WHERE EXISTS ( SELECT *
               FROM employees e2
               WHERE e1.hire_date = e2.hire_date
                 AND e1.employee_id <> e2.employee_id);
```

25. **UNION** – permite reuniunea rezultatelor a 2 cereri daca acestea sunt compatibile(au coloanele de acelasi tip in aceasi ordine). Ex:

```
--Afisati numele departamentelor si id-ul job-urilor care incep cu 'IT'
SELECT department_name
FROM departments
WHERE department_name LIKE 'IT%'
UNION
SELECT job_id
FROM jobs
WHERE job_id LIKE 'IT%';
```

26. **INTERSECT/MINUS** – folosit exaxct ca UNION dar pentru a face intersectia/diferenta dintre rezultatele a 2 cereri.

27. **GROUP BY** – clauza folosita pentru a grupa mai multe linii in functie de valoarea de pe una sau mai multe coloane. Odata ce un tabel a fost grupat ne putem gandi la fiecare linie ca avand mai multe valori pentru coloanele ce nu sunt in GROUP BY. Datorita acestui lucru putem folosi functii de agregare pe ele (SUM, AVG, COUNT). Totusi aceste coloane cu mai multe valori nu pot fi afisate (acolo se afla mai multe valori, compilatorul nu stie pe care sa o aleaga). Tot ce putem afisa sunt coloanele dupa care am grupat. Ex:

```
--Afisati numarul de angajati al fiecarui manager
SELECT manager_id, COUNT(employee_id)
FROM employees
GROUP BY manager_id;
```

28. **HAVING** – clauza folosita dupa GROUP BY ce imi permite sa pun conditii cu functii agragate pentru coloanele cu mai multe valori. Ex:

```
--Afisati toti managerii ce au mai mult de 5 subordonati
SELECT manager_id, COUNT(employee_id)
FROM employees
GROUP BY manager_id
HAVING COUNT(employee_id) > 5;
```

29. **CITIRE DE LA TASTATURA** – pentru a citi de la tastatura exista 2 metode:
    - Citire fara memorare – punem & in fata unei valori si acesta va fi citita de la tastatura DE FIECARE DATA CAND APARE. Valoarea ei nu este niciodata memorata. Ex:

```
--Sa se afiseze toti angajati ce au prenumele identic
--cu un prenume cititi de la tastatura
SELECT *
FROM employees
WHERE first_name = &prenume;
--OBS: ce scriem in variabila o sa fie pus EXACT asa in cerinta
--de aceea cand citim string-uri trebuie trecute intre ''
--Ex: 'John'

--Sa se afiseze toti angajatii cu un salariu mai mare decat
--un numar citit de la tastatura si angajati intr-un an citit de la tastatura
SELECT *
FROM employees
WHERE salary > &variabile
  AND TO_CHAR(hire_date,'YYYY') = &variabila;
```

    - Citire cu memorare – daca dorim sa memoram variabila citita punem prima data cand cititm && in fata numelui, apoi &. Ex:

```
--Se citeste un string de la tastatura
--afisati toti angajati ce au acel string in nume sau prenume
SELECT *
FROM employees
WHERE first_name LIKE '%&&string%'
   OR last_name LIKE '%&string%';
--OBS: o variabila poate fi folosita si intr-un string
--si compilatorul o sa-si dea seama ca ea se afla acolo;
```

30. **CREATE TABLE** – creaza un tabel dupa specificatiile date. Primeste numele tabelului, numele fiecarei coloane, tipul coloanei si diversi modificatori ai coloanei (PRIMART KEY, FOREIGN KEY, NOT NULL, UNIQUE, DEFAULT). De asemenea poate crea un tabel si din rezultatul unei cereri. Ex:

```
--Sa cream un tabel cu numele ANIMAL ce contine:
--id_persoana(PK, NUMBER(4)), nume(NOTNULL,VARCHAR(50)) DEFAULT „John Doe”,
--rasa(NOT NULL,VARCHAR(50)),varsta(NOT NULL, NUMBER(2)) si greutate(NUMBER(3))
CREATE TABLE Animal(
id_animal NUMBER(4) PRIMARY KEY,
nume VARCHAR(50) DEFAULT 'John Doe' NOT NULL,
rasa VARCHAR(50) NOT NULL,
varsta NUMBER(2) NOT NULL,
greutate NUMBER(3)
);
--Sa cream un tabel cu numele emp, copie a employees
CREATE TABLE emp AS (SELECT * FROM employees);
```

31. **DROP TABEL** – sterge un tabel din baza de date:

```
--Stergeti tabelul Animal
DROP TABEL Animal
```

32. **INSERT INTO** – adauga date intr-un tabel. INSERT INTO primeste numele tabelului, numele coloanelor in care trebuie sa insereze(daca nu sunt specificate se va insera pe toate colonele) si apoi fie o lista de valori (una pentru fiecare coloana aleasa) , fie o cerere a carui rezultat o sa fie inserat. Ex:

```
--Inserati o linie in tabelul Animal
INSERT INTO Animal (nume, varsta, greutate, rasa, id_animale)
VALUES('Jesse',6,12, 'Caine',1 );

--Tabelul Persoana contine: persoana_id, nume si varsta
--Adaugati toti angajatii din employees in Persoana
INSERT INTO Persoana (SELECT employee_id, first_name ||' '|| last_name , null
                      FROM employees);
```

33. **DELETE** – sterge linii din tabel dupa o anumita conditie. Ex:

```
--Fie emp o copie a tabelului employees.
--Stergeti toti angajatii cu salariu mai mic de 5000
DELETE FROM emp WHERE salary < 5000;
```

34. **UPDATE** – updateaza una sau mai multe coloane dintr-un tabel alegand liniile dupa o anumita conditie (daca conditia nu este pusa toate liniile o sa fie updatate). Ex:

```
--Fie emp o copie a lui employees
--Mariti cu 2000 toate salariile mai mici de 5000
UPDATE emp
SET salary = salary + 2000
WHERE salary < 5000;
```

35. **ALTER TABLE** – Adauga sau sterge coloane/constrangeri intr-un tabel. Ex:

```
--adaug  o noua coloana
ALTER TABLE emp
ADD(CNP VARCHAR2(13) UNIQUE);

--sterg coloana "email"
ALTER TABLE emp
DROP COLUMN email;

--adaugare de constrangere noua ( fac email cheie primara)
ALTER TABLE emp
ADD CONSTRAINT Cheie_Primara PRIMARY KEY(email);
```

36. **VIEW** – un tabel creat cu o durata scurta de viata pentru a usura transferul de date. Se creaza asemanator cu un tabel normala ce primeste date dintr-o cerere. Ex:

```
--Creez un view care este o copie a tabelului employees
CREATE VIEW empView AS (SELECT * FROM employees);

--Sterg view-ul
DROP VIEW empView;
```

37. **ROLLUP(col1,col2,col3)** – Folosit in clauza GROUP BY, face reuniunea cererilor cand in GROUP BY este: GROUP BY col1,col2,col3 ∩ GROUP BY col1,col2 ∩ GROUP BY col1 la care se adauga cererea fara GROUP BY. Ex:

```
--Sa se afiseze toate salariile atat singure cat si cu toate persoanele ce le au
SELECT salary, last_name
FROM employees
GROUP BY ROLLUP(salary, last_name);
```

38. **CUBE(col1,col2,col3)** – Folosit in clauza GROUP BY, face reuniunea tuturor cererilor cand in GROUP BY se afla toate aranjamentele argumentelor din CUBE(rar intalnit).

Modele de subiect pot fi gasite urmand urmatoarele link-uri:

- Git Palcu :
  https://github.com/palcu/fmi/blob/master/bd.md

- Drive ANUL II (in folderul „Subiecte”):
  https://drive.google.com/drive/folders/1PXjn8-JKAEubLc16YYkx0_iiNM-LGN1-