

2

Structured Query Language SQL

Interogare SQL simplă

```
SELECT [DISTINCT] target-list  
FROM relation-list  
WHERE qualification
```

- *relation-list* - lista de nume de relații/tabele.
- *target-list* - listă de attribute ale relațiilor din
relation-list
- *qualification* - comparații logice (*Attr op const* sau *Attr1 op Attr2*, unde *op* is one of $<$, $>$, $=$, \leq , \geq , \neq) combinate cu AND, OR sau NOT.
- *DISTINCT* (optional) - indică faptul că rezultatul final nu conține duplicate.

Clauza WHERE

- Operatori care pot fi folosiți în clauza **WHERE**:

Operator	Descriere
=	Egalitate
<>, !=	Inegalitate
>	Mai mare
<	Mai mic
<=	Mai mic sau egal
>=	Mai mare sau egal
!<	Nu mai mic decât
!>	Nu mai mare decât

Clauza WHERE

■ Operatori care pot fi folosiți în clauza **WHERE**:

Operator	Descriere
IN	Într-o mulțime enumerată explicit
NOT IN	În afara unei mulțimi enumerate explicit
BETWEEN	Într-un interval închis
NOT BETWEEN	În afara unui interval închis
LIKE	Ca un șablon
NOT LIKE	Diferit de un șablon

Clauza WHERE

- Operatorul **LIKE** este folosit în clauza **WHERE** pentru a specifica un șablon de căutare într-o coloană

- Sintaxa:

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern;
```

- Exemple:

```
SELECT * FROM Persons WHERE City LIKE '%s';
SELECT * FROM Persons WHERE City LIKE 'S%';
SELECT * FROM Persons WHERE City NOT LIKE 'M%';
```

Clauza WHERE (SQL Server!)

- Putem folosi următoarele caractere pentru șablon:

Caracter	Descriere
_	Înlocuiește un singur caracter
%	Înlocuiește zero sau mai multe caractere
[charlist]	Orice caracter din listă
[^charlist]	Orice caracter care nu este în listă

Clauza WHERE

- Exemple de interogări care conțin clauza **WHERE**:

```
SELECT ContactName, CompanyName  
FROM Customers  
WHERE ContactName LIKE '%b';
```

```
SELECT ContactName, CompanyName  
FROM Customers  
WHERE ContactName BETWEEN 'g' AND 'p';
```

```
SELECT ContactName, Country  
FROM Customers  
WHERE Country IN ('Germany', 'Mexico');
```

Clauza WHERE

- Exemple de interogări care conțin clauza **WHERE**:

```
SELECT * FROM Customers WHERE City LIKE 'Ma%';  
SELECT * FROM Customers WHERE City LIKE '%ne%';  
SELECT * FROM Customers WHERE Country LIKE '_SA';  
SELECT * FROM Customers WHERE City LIKE '[bsp]';  
SELECT * FROM Customers WHERE City LIKE '[^bsp]';
```


Expresii și *string*-uri

- Obține triplete (cu vârsta studenților + alte două expresii) pentru studenții al căror nume începe și se termină cu B și conține cel puțin trei caractere.

```
SELECT S.age, age1=S.age-5, 2*S.age AS age2  
FROM Students S  
WHERE S.name LIKE 'B_ %B'
```

- **AS** și **=** sunt două moduri de redenumire a câmpurilor în rezultat.
- **LIKE** e folosit pentru comparații pe șiruri de caractere. **'_'** reprezintă orice caracter și **'%'** reprezintă 0 sau mai multe caractere arbitrare.

INNER JOIN

```
SELECT S.name, C.cname
FROM Students S,
Enrolled E, Courses C
WHERE S.sid = E.sid
AND E.cid = C.cid
```



```
SELECT S.name, C.cname
FROM Students S
INNER JOIN Enrolled E ON
S.sid = E.sid,
INNER JOIN Courses C ON
E.cid = C.cid
```

Students

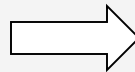
<i>sid</i>	<i>name</i>	<i>email</i>	<i>age</i>	<i>gr</i>
1234	John	j@cs.ro	21	331
1235	Smith	s@cs.ro	22	331
1236	Anne	a@cs.ro	21	332

Courses

<i>cid</i>	<i>cname</i>	<i>credits</i>
Alg1	Algorithms1	7
DB1	Databases1	6
DB2	Databases2	6

Enrolled

<i>sid</i>	<i>cid</i>	<i>grade</i>
1234	Alg1	9
1235	Alg1	10
1237	DB2	9



<i>name</i>	<i>cname</i>
John	Algorithms1
Smith	Algorithms1

LEFT OUTER JOIN

- Daca dorim sa regasim și studentii fără nici o notă la vreun curs:

```
SELECT S.name, C.cname
FROM Students S
LEFT OUTER JOIN Enrolled E
ON S.sid = E.sid,
LEFT OUTER JOIN Courses C
ON E.cid = C.cid
```

Students

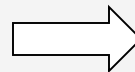
<i>sid</i>	<i>name</i>	<i>email</i>	<i>age</i>	<i>gr</i>
1234	John	j@cs.ro	21	331
1235	Smith	s@cs.ro	22	331
1236	Anne	a@cs.ro	21	332

Courses

<i>cid</i>	<i>cname</i>	<i>credits</i>
Alg1	Algorithms1	7
DB1	Databases1	6
DB2	Databases2	6

Enrolled

<i>sid</i>	<i>cid</i>	<i>grade</i>
1234	Alg1	9
1235	Alg1	10
1237	DB2	9



<i>name</i>	<i>cname</i>
John	Algorithms1
Smith	Algorithms1
Anne	NULL

RIGHT OUTER JOIN

- Pentru a gasi notele asiguate unor studenti inexistenti:

```
SELECT S.name, C.cname  
FROM Students S
```

```
RIGHT OUTER JOIN Enrolled E  
ON S.sid = E.sid,  
INNER JOIN Courses C ON  
E.cid = C.cid
```

Students

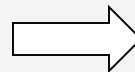
<i>sid</i>	<i>name</i>	<i>email</i>	<i>age</i>	<i>gr</i>
1234	John	j@cs.ro	21	331
1235	Smith	s@cs.ro	22	331
1236	Anne	a@cs.ro	21	332

Courses

<i>cid</i>	<i>cname</i>	<i>credits</i>
Alg1	Algorithms1	7
DB1	Databases1	6
DB2	Databases2	6

Enrolled

<i>sid</i>	<i>cid</i>	<i>grade</i>
1234	Alg1	9
1235	Alg1	10
1237	DB2	9



<i>name</i>	<i>cname</i>
John	Algorithms1
Smith	Algorithms1
NULL	Databases2

FULL OUTER JOIN

- LEFT+RIGHT OUTER JOIN
- In majoritatea SGBD OUTER e optional

```
SELECT S.name, C.cname
FROM Students S
FULL OUTER JOIN Enrolled E
ON S.sid = E.sid,
FULL OUTER JOIN Courses C
ON E.cid = C.cid
```

Students

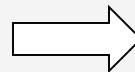
<i>sid</i>	<i>name</i>	<i>email</i>	<i>age</i>	<i>gr</i>
1234	John	j@cs.ro	21	331
1235	Smith	s@cs.ro	22	331
1236	Anne	a@cs.ro	21	332

Courses

<i>cid</i>	<i>cname</i>	<i>credits</i>
Alg1	Algorithms1	7
DB1	Databases1	6
DB2	Databases2	6

Enrolled

<i>sid</i>	<i>cid</i>	<i>grade</i>
1234	Alg1	9
1235	Alg1	10
1237	DB2	9



<i>name</i>	<i>cname</i>
John	Algorithms1
Smith	Algorithms1
NULL	Databases2
NULL	Databases1
Anne	NULL



A **SQL** query walks into a bar.

He approaches two **tables** and says:

*"Mind if I **join** you?"*

Valoarea **NULL**

- În anumite situații valorile particulare ale unor attribute (câmpuri) pot fi *necunoscute* sau *inaplicabile* temporar.
 - SQL permite utilizarea unei valori speciale null pentru astfel de situații.
- Prezența valorii *null* implică unele probleme suplimentare:
 - E necesară implementarea unei logici cu 3 valori: *true*, *false* și *null* (de exemplu o condiție de tipul *rating*>8 va fi întotdeauna evaluată cu *false* dacă valoarea câmpului *rating* este *null*)
 - E necesară adaugarea unui operator special IS NULL / IS NOT NULL.

Operatori de agregare

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

atribut

```
SELECT COUNT (*)  
FROM Students S
```

```
SELECT AVG (S.age)  
FROM Students S  
WHERE S.gr=921
```

```
SELECT COUNT (DISTINCT S.gr)  
FROM Students S  
WHERE S.name='Bob'
```

```
SELECT S.name  
FROM Students S  
WHERE S.age = ANY  
      (SELECT MAX(S2.age)  
       FROM Students S2)
```


GROUP BY / HAVING

For $i = 221, 222, 223, 224 \dots$:

```
SELECT MIN(S.age)
FROM   Students S
WHERE  S.gr =  $i$ 
```

GROUP BY / HAVING

```
SELECT [DISTINCT] target-list
FROM    relation-list
WHERE   qualification
GROUP BY grouping-list
HAVING  group-qualification
```

Numarul studentilor cu nota la cursurile cu 6 credite si media notelor acestora

```
SELECT  C.cid, COUNT (*) AS scount, AVG(grade)
FROM    Enrolled E, Courses C
WHERE   E.cid=C.cid AND C.credits=6
GROUP BY C.cid
```

Courses

<i>cid</i>	<i>cname</i>	<i>credits</i>
Alg1	Algorithms1	7
DB1	Databases1	6
DB2	Databases2	6

Students

<i>sid</i>	<i>name</i>	<i>email</i>	<i>age</i>	<i>gr</i>
1234	John	j@cs.ro	21	331
1235	Smith	s@cs.ro	22	331
1236	Anne	a@cs.ro	21	332

Enrolled

<i>sid</i>	<i>cid</i>	<i>grade</i>
1234	Alg1	9
1235	Alg1	10
1234	DB1	10
1234	DB2	9
1236	DB1	7

Enrolled

Courses

<i>sid</i>	<i>cid</i>	<i>grade</i>	<i>cid</i>	<i>cname</i>	<i>credits</i>
1234	Alg1	9	Alg1	Algorithms1	7
1234	Alg1	9	DB1	Databases1	6
1234	Alg1	9	DB2	Databases2	6
1235	Alg1	10	Alg1	Algorithms1	7
1235	Alg1	10	DB1	Databases1	6
1235	Alg1	10	DB2	Databases2	6
1234	DB1	10	Alg1	Algorithms1	7
1234	DB1	10	DB1	Databases1	6
1234	DB1	10	DB2	Databases2	6
1234	DB2	9	Alg1	Algorithms1	7
1234	DB2	9	DB1	Databases1	6
1234	DB2	9	DB2	Databases2	6
1236	DB1	7	Alg1	Algorithms1	7
1236	DB1	7	DB1	Databases1	6
1236	DB1	7	DB2	Databases2	6

```
SELECT C.cid,  
COUNT(*)AS scount,  
AVG(grade)AS average  
FROM Enrolled E,  
Courses C  
WHERE E.cid=C.cid  
AND C.credits=6  
GROUP BY C.cid
```

Enrolled

Courses

<i>sid</i>	<i>cid</i>	<i>grade</i>	<i>cid</i>	<i>cname</i>	<i>credits</i>
1234	Alg1	9	Alg1	Algoritmics 1	7
1235	Alg1	10	Alg1	Algoritmics 1	7
1234	DB1	10	DB1	Databases1	6
1234	DB2	9	DB2	Databases2	6
1236	DB1	7	DB1	Databases1	6

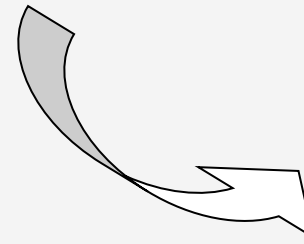
```
SELECT C.cid,  
COUNT(*)AS scount,  
AVG(grade)AS average  
FROM    Enrolled E,  
        Courses C  
WHERE   E.cid=C.cid  
        AND  
        C.credits=6  
GROUP BY C.cid
```

<i>sid</i>	<i>cid</i>	<i>grade</i>	<i>cid</i>	<i>cname</i>	<i>credits</i>
1234	DB1	10	DB1	Databases1	6
1234	DB2	9	DB2	Databases2	6
1236	DB1	7	DB1	Databases1	6

```

SELECT C.cid
COUNT(*) AS scount,
AVG(grade) AS average
FROM   Enrolled E,
       Courses C
WHERE  E.cid=C.cid
AND
      C.credits=6
GROUP BY C.cid
HAVING MAX(grade) = 10

```



<i>cid</i>	<i>scount</i>	<i>average</i>
DB1	2	8.5
DB2	1	9

Sortarea rezultatului interogarilor

- ORDER BY *column* [ASC | DESC] [, ...]

```
SELECT cname, sname, grade
FROM Courses C
      INNER JOIN Enrolled E ON C.cid = E.cid
      INNER JOIN Students S ON E.sid = S.sid
ORDER BY cname, grade DESC , sname
```

Sortarea rezultatului interogărilor

- Rezultatul e sortat după orice câmp din clauza SELECT, inclusiv expresii sau agregări:

```
SELECT gr, Count(*) as StudNo  
FROM Students C  
GROUP BY gr  
ORDER BY StudNo
```


Subinterogări

- O subinterogare este o interogare încorporată într-o altă interogare
- Se poate folosi o subinterogare în clauza WHERE a înlocui un JOIN

Exemplu:

- Dorim să găsim toți clienții care au plasat o comandă
- Varianta cu subinterogare:

```
SELECT CustomerID, AccountNumber  
FROM Sales.Customer  
WHERE CustomerID IN  
(SELECT CustomerID FROM Sales.SalesOrderHeader);
```

Subinterogări

■ Varianta cu JOIN:

```
SELECT DISTINCT C.CustomerID, C.AccountNumber  
FROM Sales.Customer C  
INNER JOIN Sales.SalesOrderHeader O  
ON C.CustomerID = O.CustomerID;
```

Subinterogări

- O subinterogare în clauza WHERE poate fi folosită și pentru a găsi înregistrările din primul tabel care nu au potriviri în cel de-al doilea tabel (în acest caz, se va folosi operatorul NOT IN sau NOT EXISTS)

Exemplu:

```
SELECT CustomerID, AccountNumber FROM Sales.Customer  
WHERE CustomerID NOT IN  
(SELECT CustomerID FROM Sales.SalesOrderHeader);
```

SAU

```
SELECT C.CustomerID, C.AccountNumber FROM Sales.Customer  
C WHERE NOT EXISTS (SELECT * FROM Sales.SalesOrderHeader  
O WHERE O.CustomerID = C.CustomerID);
```

ANY

- Dorim să afișăm toate produsele care au prețul mai mare decât prețul maxim de produs al cel puțin unei categorii oarecare de produse:

```
SELECT name, price
FROM Products
WHERE price > ANY
(SELECT MAX(price) FROM Products GROUP BY id_category);
```

- Dorim să afișăm toate produsele care au prețul egal cu prețul minim de produs al cel puțin unei categorii de produse:

```
SELECT name, price FROM Products WHERE price = ANY
(SELECT MIN(price) FROM Products GROUP BY id_category);
```

SAU

```
SELECT name, price FROM Products WHERE price IN (SELECT
MIN(price) FROM Products GROUP BY id_category);
```

ALL

- Dorim să afișăm toate produsele care au prețul mai mare decât prețul minim al tuturor categoriilor de produse:

```
SELECT name, price FROM Products WHERE price > ALL  
(SELECT MIN(price) FROM Products  
GROUP BY id_category);
```

- Dorim să afișăm toate produsele care au prețul mai mic decât prețul maxim al tuturor categoriilor de produse:

```
SELECT name, price FROM Products WHERE price < ALL  
(SELECT MAX(price) FROM Products  
GROUP BY id_category);
```

Reuniune, intersecție și diferență

- UNION (reuniune) se folosește pentru a îmbina rezultatele a două sau mai multe interogări într-un singur result-set
- Sintaxa:

```
SELECT <col1>, <col2>, <col3> FROM table1  
UNION [ALL]  
SELECT <col4>, <col5>, <col6> FROM table2;
```
- Fiecare interogare trebuie să conțină același număr de coloane, iar tipurile coloanelor trebuie să fie compatibile

Reuniune, intersecție și diferență

- UNION ALL va include înregistrări duplicate

- Exemplu (cu duplicate):

```
SELECT nume FROM Clienți  
UNION ALL  
SELECT nume FROM Angajați;
```

- Exemplu (fără duplicate):

```
SELECT nume FROM Clienți  
UNION  
SELECT nume FROM Angajați;
```

Reuniune, intersecție și diferență

- INTERSECT (intersecție) este folosit pentru a returna într-un singur result-set acele înregistrări care apar atât în result-set-ul interogării din partea dreaptă cât și în cel al interogării din partea stângă
- Sintaxa:

```
SELECT <col1>, <col2>, <col3>  
FROM table1  
INTERSECT  
SELECT <col4>, <col5>, <col6>  
FROM table2;
```


Reuniune, intersecție și diferență

■ Exemplu:

```
SELECT nume, prenume FROM Clienți
```

```
INTERSECT
```

```
SELECT nume, prenume FROM Angajați
```

```
INTERSECT
```

```
SELECT nume, prenume FROM Furnizori;
```

Reuniune, intersecție și diferență

- EXCEPT (diferență) este folosit pentru a returna acele înregistrări care apar în result-set-ul interogării din partea stângă dar nu apar în result-set-ul interogării din partea dreaptă
- Sintaxa:

```
SELECT <col1>, <col2>, <col3>  
FROM table1  
EXCEPT  
SELECT <col4>, <col5>, <col6>  
FROM table2;
```

Reuniune, intersecție și diferență

- Exemplu:

```
SELECT nume, prenume FROM Clienți  
EXCEPT  
SELECT nume, prenume FROM Angajați;
```

- Exemplu:

```
SELECT id_client FROM Clienți  
EXCEPT  
SELECT id_client FROM Comenzi;
```

Limbaajul SQL: DML

- DML = Data Manipulation Language
(Limbaaj de manipulare a datelor) - conține instrucțiuni pentru inserare, actualizare, ștergere și interogare a datelor stocate într-o bază de date relațională
- Cele mai folosite instrucțiuni DML sunt:
 - INSERT – inserează înregistrări noi
 - UPDATE – actualizează înregistrări
 - DELETE – șterge înregistrări
 - SELECT – extrage înregistrări

Limbajul SQL: DML

- Instrucțiunea **INSERT INTO** se folosește pentru a insera noi înregistrări într-un tabel
- Sintaxa:

```
INSERT INTO table_name
```

```
VALUES (value1, value2,...);
```

SAU

```
INSERT INTO table_name
```

```
(column_name1, column_name2, column_name3,...)
```

```
VALUES (value1, value2, value3, ...);
```

Limbajul SQL: DML

- Specificarea coloanelor după numele tabelului este opțională
- Prin specificarea coloanelor controlăm asocierile coloană-valoare, deci nu ne bazăm pe ordinea în care apar coloanele atunci când a fost creat tabelul sau când structura tabelului a fost modificată ultima dată
- Dacă nu specificăm o valoare pentru o coloană, SGBD-ul va verifica dacă există o valoare implicită pentru coloana respectivă iar dacă nu există și coloana nu permite NULL atunci inserarea nu va avea loc

Limbajul SQL: DML

- Exemplu de inserare a unei noi înregistrări în tabelul *Clienți*:

```
INSERT INTO Clienți  
(IDClient, Nume, Prenume, Localitate)  
VALUES (1, 'Pop', 'Anda', 'Sibiu');
```

SAU

```
INSERT INTO Clienți  
VALUES (1, 'Pop', 'Anda', 'Sibiu');
```

Limbajul SQL: DML

- Instrucțiunea **UPDATE** se folosește pentru a actualiza înregistrări într-un tabel

- Sintaxa:

```
UPDATE table_name
```

```
SET column1=value1, column2=value2,
```

```
...
```

```
WHERE some_column=some_value;
```

- Omiterea clauzei **WHERE** va rezulta în actualizarea tuturor înregistrărilor din tabel

Limbajul SQL: DML

- Exemplu de actualizare a unei înregistrări dintr-un tabel:

```
UPDATE Clienți  
SET Localitate='Cluj-Napoca'  
WHERE Nume='Pop' AND Prenume='Anda';
```

Limbajul SQL: DML

- Instrucțiunea **DELETE** se folosește pentru a șterge înregistrări dintr-un tabel
- Sintaxa:

```
DELETE FROM table_name  
WHERE some_column=some_value;
```

- Omiterea clauzei **WHERE** va rezulta în ștergerea tuturor înregistrărilor din tabel

Limbajul SQL: DML

- Exemplu de ștergere a tuturor înregistrărilor din tabelul *Clienți* pentru care coloana *Localitate* are valoarea 'Sibiu':

```
DELETE FROM Cliești  
WHERE Localitate='Sibiu';
```

- Exemplu de ștergere a tuturor înregistrărilor din tabelul *Clienți*:

```
DELETE FROM Cliești;
```

Limbajul SQL: DDL

- Instrucțiunea **CREATE TABLE** se folosește pentru a crea un tabel într-o bază de date
- Sintaxa:

```
CREATE TABLE table_name  
(  
    column_name1 data_type,  
    column_name2 data_type,  
    ...  
);
```

Limbajul SQL: DDL

- Dorim să creăm un tabel numit *Persoane* care conține câmpurile *id*, *nume*, *prenume*, *localitate*

```
CREATE TABLE Persoane  
(  
    id INT,  
        nume VARCHAR(30),  
    prenume VARCHAR(30),  
        localitate VARCHAR(30)  
);
```

Limbajul SQL: DDL

- Instrucțiunea **ALTER TABLE** se folosește pentru a modifica structura unui tabel
- Sintaxa instrucțiunii pentru adăugarea unei coloane într-un tabel:

```
ALTER TABLE table_name
```

```
ADD column_name datatype;
```

- Exemplu de adăugare a unei coloane într-un tabel:

```
ALTER TABLE Persoane
```

```
ADD data_nașterii DATE;
```

Limbajul SQL: DDL

- Sintaxa instrucțiunii pentru schimbarea tipului de date al unei coloane dintr-un tabel:

```
ALTER TABLE table_name
```

```
ALTER COLUMN column_name datatype;
```

- Exemplu de schimbare a tipului de date al unei coloane dintr-un tabel:

```
ALTER TABLE Persoane
```

```
ALTER COLUMN data_nașterii DATETIME;
```

Limbajul SQL: DDL

- Sintaxa instrucțiunii pentru ștergerea unei coloane dintr-un tabel:

```
ALTER TABLE table_name
```

```
DROP COLUMN column_name;
```

- Exemplu de ștergere a unei coloane dintr-un tabel:

```
ALTER TABLE Persoane
```

```
DROP COLUMN data_nașterii;
```


Limbajul SQL: DDL

- Instrucțiunea **DROP TABLE** se folosește pentru a șterge un tabel dintr-o bază de date

- Sintaxa:

```
DROP TABLE table_name;
```

- Exemplu:

```
DROP TABLE Persoane;
```

Limbajul SQL: DDL

- În limbajul SQL fiecare coloană, variabilă locală, expresie sau parametru are un tip de date
- Un tip de date este un atribut care specifică ce fel de valori pot fi stocate în obiectul respectiv
- Exemple:
int, tinyint, smallint, bigint, decimal, float, real, money, nchar, varchar, datetime, date, time

Limbajul SQL: DDL

- **Constrângerile** de integritate se pot specifica la crearea tabelului
(în instrucțiunea CREATE TABLE),
dar și după ce tabelul a fost creat
(cu ajutorul instrucțiunii ALTER TABLE)

Limbajul SQL: DDL

- Constrângeri:
 - NOT NULL
 - UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK
 - DEFAULT

Limbajul SQL: DDL

- În mod implicit un tabel permite inserarea de valori NULL
- Dacă nu dorim să permitem introducerea de valori NULL pentru o coloană, aplicăm **constrângerea NOT NULL** pe coloana respectivă
- Ca rezultat, nu vom putea insera sau actualiza înregistrări care nu specifică o valoare pentru coloana respectivă

Limbajul SQL: DDL

- Exemplu de definire a unei constrângeri NOT NULL la crearea unui tabel:

```
CREATE TABLE Studenți  
(  
    cod_s INT NOT NULL,  
        nume VARCHAR(50),  
    prenume VARCHAR(50),  
    oraș VARCHAR(50)  
);
```

Limbajul SQL: DDL

- **Constrângerea UNIQUE** se defineşte pe coloanele în care nu dorim să permitem valori duplicate
- Se pot defini mai multe constrângeri **UNIQUE** în acelaşi tabel
- Se poate defini pe una sau mai multe coloane
- În cazul în care o constrângere **UNIQUE** este definită pe mai multe coloane, combinaţia de valori din coloanele respective trebuie să fie unică la nivel de înregistrare

Limbajul SQL: DDL

- Exemplu de definire a unei constrângeri UNIQUE pe o coloană la crearea unui tabel:

```
CREATE TABLE Studenți  
(  
    cod_s INT UNIQUE,  
    nume VARCHAR(50),  
    prenume VARCHAR(50),  
    oraș VARCHAR(50)  
);
```


Limbajul SQL: DDL

- Exemplu de definire a unei constrângeri UNIQUE pe mai multe coloane la crearea unui tabel:

```
CREATE TABLE Studenți
(
    cod_s INT NOT NULL,
    nume VARCHAR(50),
    prenume VARCHAR(50),
    oraș VARCHAR(50),
    CONSTRAINT uc_StudentID UNIQUE
                                (cod_s, nume)
);
```

Limbajul SQL: DDL

- Definirea unei constrângeri UNIQUE după ce tabelul a fost creat se face cu ajutorul instrucțiunii ALTER TABLE
- Exemplu de definire a unei constrângeri UNIQUE pe o singură coloană:

```
ALTER TABLE Studenți  
ADD UNIQUE(cod_s);
```

- Exemplu de definire a unei constrângeri UNIQUE pe mai multe coloane:

```
ALTER TABLE Studenți  
ADD CONSTRAINT uc_StudentID  
UNIQUE(cod_s, nume);
```

Limbajul SQL: DDL

- O constrângere poate fi eliminată cu ajutorul instrucțiunii **DROP CONSTRAINT**

- Sintaxa:

```
ALTER TABLE table_name
```

```
DROP CONSTRAINT constraint_name;
```

- Exemplu:

```
ALTER TABLE Studenți
```

```
DROP CONSTRAINT uc_StudentID;
```

Limbajul SQL: DDL

- Exemplu de definire a unei constrângeri PRIMARY KEY la crearea unui tabel:

```
CREATE TABLE Studenți  
(  
    cod_s INT PRIMARY KEY,  
    nume VARCHAR(50),  
    prenume VARCHAR(50),  
    oraș VARCHAR(50)  
);
```

Limbajul SQL: DDL

- Exemplu de definire a unei constrângeri PRIMARY KEY pe mai multe coloane la crearea unui tabel:

```
CREATE TABLE Studenți  
(  
    cod_s INT,  
    nume VARCHAR(30),  
    prenume VARCHAR(50),  
    oraș VARCHAR(50),  
    CONSTRAINT pk_Student PRIMARY KEY (cod_s, nume)  
);
```

Limbajul SQL: DDL

- Pentru a putea crea o cheie primară după crearea tabelului, coloana sau coloanele pe care dorim să le includem în cheia primară trebuie să aibă definită o constrângere NOT NULL
- Exemplu de definire a unei constrângeri PRIMARY KEY după crearea tabelului:

```
ALTER TABLE Studenți  
ADD CONSTRAINT pk_Student PRIMARY KEY(cod_s,  
nume);
```

- Exemplu de eliminare a unei constrângeri PRIMARY KEY:

```
ALTER TABLE Studenți  
DROP CONSTRAINT pk_Student;
```

Limbajul SQL: DDL

- Un foreign key (cheie străină) pointează la un primary key (cheie primară) dintr-un alt tabel
- Tabelul Clienți

IDClient	Nume	Prenume	Localitate
1	Pop	Oana	Cluj-Napoca
2	Rus	Andrei	Sibiu

- Tabelul Comenzi

IDCom	NrCom	IDClient
1	3455	2
2	3456	1

Limbajul SQL: DDL

- Exemplu de definire a unei constrângeri FOREIGN KEY la crearea unui tabel:

```
CREATE TABLE Comenzi
(
IDCom INT PRIMARY KEY,
NrCom INT,
IDClient INT FOREIGN KEY
                REFERENCES Clienți(IDClient)
);
```


Limbajul SQL: DDL

- Exemplu de definire a unei constrângeri FOREIGN KEY cu numele `fk_Client` la crearea unui tabel:

```
CREATE TABLE Comenzi
(
IDCom INT PRIMARY KEY,
NrCom INT,
IDClient INT,
CONSTRAINT fk_Client FOREIGN KEY (IDClient)
REFERENCES Clienti(IDClient)
);
```

Limbajul SQL: DDL

- Exemplu de definire a unei constrângeri FOREIGN KEY după crearea tabelului:

```
ALTER TABLE Comenzi  
ADD FOREIGN KEY (IDClient)  
REFERENCES Clienți(IDClient);
```

SAU

```
ALTER TABLE Comenzi  
ADD CONSTRAINT fk_Client FOREIGN KEY (IDClient)  
REFERENCES Clienți(IDClient);
```

Limbajul SQL: DDL

- Se pot specifica acțiuni care vor fi efectuate în cazul în care un utilizator încearcă să șteargă sau să modifice un key spre care pointează un foreign key
- Următoarele acțiuni pot fi specificate în acest caz:

NO ACTION

CASCADE

SET NULL

SET DEFAULT

Limbajul SQL: DDL

- Exemplu de definire a unei constrângeri FOREIGN KEY cu acțiuni care au loc în caz de modificare sau ștergere:

```
CREATE TABLE Comenzi
(
  IDCom INT PRIMARY KEY,
  NrCOM INT,
  IDClient INT FOREIGN KEY REFERENCES
  Clienți(IDClient)
  ON DELETE CASCADE
  ON UPDATE CASCADE
) ;
```

Limbaajul SQL: DDL

- **Constrângerea CHECK** se foloseşte pentru a limita intervalul de valori ce se pot introduce pentru o anumită coloană
- Se poate defini pe o coloană, iar în acest caz limitează valorile ce pot fi introduse pentru coloana respectivă
- Se poate defini pe mai multe coloane

Limbajul SQL: DDL

- Exemplu de definire a unei constrângeri CHECK pe o coloană la crearea tabelului:

```
CREATE TABLE Clienți  
(  
  IDClient INT PRIMARY KEY CHECK(IDClient>0),  
  Nume VARCHAR(50) NOT NULL,  
  Prenume VARCHAR(50),  
  Localitate VARCHAR(50)  
);
```

Limbajul SQL: DDL

- Exemplu de constrângere CHECK definită pe mai multe coloane la crearea unui tabel:

```
CREATE TABLE Clienți
(
    IDClient INT PRIMARY KEY,
    Nume VARCHAR(50) NOT NULL,
    Prenume VARCHAR(50),
    Localitate VARCHAR(50),
    CONSTRAINT ck_IDClient CHECK(IDClient>0 AND
                                Localitate IN ('Cluj-Napoca', 'Sibiu'))
);
```

Limbajul SQL: DDL

- Exemplu de adăugare a unei constrângeri CHECK după crearea tabelului:

```
ALTER TABLE Clienți  
ADD CHECK (IDClient>0);
```

- Exemplu de adăugare și stabilire a unui nume pentru o constrângere CHECK după crearea tabelului:

```
ALTER TABLE Clienți  
ADD CONSTRAINT ck_Client  
CHECK (IDClient>0 AND Localitate IN ('Cluj-Napoca',  
'Sibiu'));
```


Limbajul SQL: DDL

- **Constrângerea DEFAULT** se folosește pentru a insera o valoare implicită într-o coloană
- Valoarea implicită va fi adăugată pentru toate înregistrările noi dacă nu se specifică o altă valoare
- Se poate folosi și pentru a insera valori sistem obținute prin apelul unor funcții
- Exemplu de definire a unei constrângeri DEFAULT după crearea unui tabel:

```
        ALTER TABLE Clienți  
        ADD CONSTRAINT d_Localitate DEFAULT 'Cluj-Napoca'  
FOR      Localitate;
```

- Eliminarea unei constrângeri DEFAULT
 ALTER TABLE Clienți
 DROP CONSTRAINT d_Localitate;

Limbajul SQL: DDL

- Exemplu de definire a unei constrângeri DEFAULT la crearea unui tabel:

```
CREATE TABLE Comenzi  
(  
  IDCom INT PRIMARY KEY,  
  NrCOM INT NOT NULL,  
  IDClient INT,  
  DataCom DATE DEFAULT GETDATE()  
);
```