



# SQL DML - Limbaj de manipulare a datelor

---

Seminar 2



# Limbajul SQL: DML

---

- DML = Data Manipulation Language (Limbaj 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, SQL Server 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: DML

---

- Instrucțiunea **SELECT** se folosește pentru a extrage date din baza de date
- Rezultatul este stocat într-un tabel rezultat numit **result-set**
- Sintaxa:

```
SELECT column_name(s) FROM table_name;
```

SAU

```
SELECT * FROM table_name;
```

- Exemple:

```
SELECT * FROM Clienți;
```

```
SELECT IDClient, Nume, Prenume FROM Clienți;
```



# Limbajul SQL: DML

---

- Într-un tabel, unele coloane pot conține valori duplicate
- Atunci când dorim să returnăm doar valorile distincte, folosim cuvântul cheie **DISTINCT**
- Sintaxa:

```
SELECT DISTINCT column_name(s)  
FROM table_name;
```

- Exemplu:

```
SELECT DISTINCT Localitate FROM Clienți;
```



# Limbajul SQL: DML

---

- Clauza **WHERE** se folosește cu scopul de a filtra înregistrări
- Sunt extrase doar înregistrările care îndeplinesc un anumit criteriu
- Sintaxa:

```
SELECT column_name(s) FROM table_name  
WHERE column_name operator value;
```

- Exemplu:

```
SELECT IDClient, Nume FROM Clienți  
WHERE IDClient=3;
```



# Limbajul SQL: DML

---

- SQL folosește apostrof pentru a delimita valorile de tip text/string
- Valorile numerice nu se delimitează cu apostrof
- Exemple:

```
SELECT IDClient, Nume, Prenume FROM Clienți WHERE  
Prenume='Anda' AND Nume='Pop';
```

```
SELECT Nume, Prenume, Localitate, Data_nașterii FROM  
Studenți WHERE An_înmatriculare=2015;
```



# Limbajul SQL: DML

---

- 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



# Limbajul SQL: DML

---

- 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





# Limbajul SQL: DML

---

- 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%';
```



# Limbajul SQL: DML

---

- 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



# Limbajul SQL: DML

---

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

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

```
SELECT * FROM Customers WHERE ContactName LIKE 'a%';
```

```
SELECT ContactName, CompanyName FROM Customers WHERE  
ContactName < 'b';
```

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

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



# Limbaajul SQL: DML

---

– 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]%';
```

```
SELECT name, price, price/2 AS [special price] FROM  
Products WHERE category_id=2;
```



# Limbajul SQL: DML

---

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

```
SELECT * FROM Persons WHERE BirthDate BETWEEN '2001-01-03 00:00:00' AND '2009-01-03 23:59:59';
```

```
SELECT * FROM Persons WHERE BirthDate IS NOT NULL;
```

```
SELECT * FROM Movies WHERE Year NOT BETWEEN 1980 AND 1988;
```

```
SELECT Name, Rating FROM Movies WHERE Rating=9 OR Rating=5;
```

```
SELECT Name, City FROM Persons WHERE Phone_number IS NULL;
```



# Alias

---

- Se poate da un alias unui tabel sau unei coloane
- Un alias este util atunci când numele unei coloane sau al unui tabel este lung sau complex
- Un alias este necesar atunci când numele unui tabel apare de două ori în clauza **FROM**
- Un alias este util când dorim să dăm un nume unei coloane cu valoare calculată
- Interogările care conțin alias-uri devin mai ușor de scris și de citit



# Alias

---

- Sintaxa pentru tabele:

```
SELECT column_name(s) FROM table_name AS alias;
```

- Exemplu cu alias:

```
SELECT C.ContactName, C.CustomerID, O.OrderID FROM  
Customers AS C, Orders AS O WHERE  
C.CustomerID=O.CustomerID;
```

- Exemplu fără alias:

```
SELECT Customers.ContactName, Customers.CustomerID,  
Orders.OrderID FROM Customers, Orders WHERE  
Customers.CustomerID=Orders.CustomerID;
```





# Alias

---

- Sintaxa pentru coloane:

```
SELECT column_name AS alias FROM table_name;
```

- Exemplu:

```
SELECT ContactName AS Name, City FROM Customers WHERE  
City IN ('Berlin', 'London');
```

- Alias-ul atribuit unei coloane înlocuiește numele original al acesteia în result-set



# Joins

---

- Folosim join-uri atunci când trebuie să extragem date din mai multe tabele pe baza unei relații între anumite coloane din aceste tabele într-un singur result-set
- Un join definește modul în care două tabele sunt legate într-o interogare prin:
  - specificarea coloanei din fiecare tabel care urmează a fi folosită pentru a realiza join-ul (de obicei un join specifică un foreign key dintr-un tabel și key-ul asociat din celălalt tabel)
  - specificarea unui operator logic (de exemplu: = sau <>) care va fi folosit pentru a compara valorile din coloane

# Joins

Tabelul Studenți

sid	nume	email	grupă
1234	Ana	ana@ymail.com	331
1235	Andrei	andrei@gmail.com	332
1236	Mihai	mh@yahoo.com	333

sid	cid	notă
1234	Alg1	9
1235	Alg1	10
1237	Db2	9

Tabelul Note

cid	nume	credite
Alg1	Algebră 1	7
Db1	Baze de date 1	6
Db2	Baze de date 2	6

Tabelul Cursuri



# Joins - Tipuri

---

- Inner Join – extrage înregistrări când este cel puțin o potrivire în cele două tabele
- Left Outer Join – extrage toate înregistrările din tabelul din partea stângă, inclusiv atunci când nu există potriviri în tabelul din partea dreaptă
- Right Outer Join – extrage toate înregistrările din tabelul din partea dreaptă, inclusiv atunci când nu există potriviri în tabelul din partea stângă
- Full Outer Join – combinație între Left Outer Join și Right Outer Join



# Inner Join

---

- Extrage înregistrări doar când există cel puțin o potrivire în ambele tabele
- Sintaxa:

```
SELECT column_name(s) FROM  
table_name1 INNER JOIN  
table_name2 ON table_name1.column_name =  
table_name2.column_name;
```

# Inner Join

---

– Exemplu:

```
SELECT S.num, C.num FROM Studenți S  
INNER JOIN Note N ON S.sid=N.sid  
INNER JOIN Cursuri C ON N.cid=C.cid;
```

– Rezultat:

num	num
Ana	Algebră 1
Andrei	Algebră 1



# Left Outer Join

---

- Se mai numește și Left Join
- Extrage toate înregistrările din tabelul din partea stângă, chiar și în cazul în care nu există potriviri în tabelul din partea dreaptă
- Sintaxa:

```
SELECT column_name(s) FROM table_name1
```

```
LEFT OUTER JOIN table_name2
```

```
ON table_name1.column_name = table_name2.column_name;
```



# Left Outer Join

---

– Exemplu:

– Dorim să returnăm toți studenții, indiferent dacă au note la un curs sau nu

```
SELECT S.num, C.num FROM Studenți S LEFT OUTER JOIN  
Note N ON S.sid = N.sid LEFT OUTER JOIN Cursuri C ON  
N.cid = C.cid;
```

– Rezultat:

nume	nume
Ana	Algebră 1
Andrei	Algebră 1
Mihai	NULL



# Right Outer Join

---

- Se mai numește și Right Join
- Extrage toate înregistrările din tabelul din partea dreaptă, indiferent dacă există sau nu potriviri în tabelul din partea stângă
- Sintaxa:

```
SELECT column_name(s) FROM table_name1
```

```
RIGHT OUTER JOIN table_name2
```

```
ON table_name1.column_name = table_name2.column_name;
```

# Right Outer Join

---

– Exemplu:

– Dorim să returnăm toate notele, inclusiv cele introduse din greșeală unor studenți inexistenți

```
SELECT S.num, N.nota, C.num FROM Studenți S RIGHT  
OUTER JOIN Note N ON S.sid = N.sid INNER JOIN Cursuri  
C ON N.cid = C.cid;
```

– Rezultat:

num	nota	num
Ana	9	Algebră 1
Andrei	10	Algebră 1
NULL	9	Baze de date 2



# Full Outer Join

---

- Se mai numește și Full Join
- Este o combinație între Left Outer Join și Right Outer Join
- Extrage toate înregistrările din tabelul din partea stângă și toate înregistrările din tabelul din partea dreaptă
- Sintaxa:

```
SELECT column_name(s) FROM table_name1  
FULL OUTER JOIN table_name2  
ON table_name1.column_name = table_name2.column_name;
```



# Full Outer Join

---

– Exemplu:

```
SELECT S.num, C.num FROM Studenți S FULL OUTER JOIN  
Note N ON S.sid = N.sid FULL OUTER JOIN Cursuri C ON  
N.cid = C.cid;
```

– Rezultat:

nume	nume
Ana	Algebră 1
Andrei	Algebră 1
Mihai	NULL
NULL	Baze de date 2
NULL	Baze de date 1



# Funcții de agregare

---

- Funcțiile de agregare realizează un calcul pe o mulțime de valori și returnează o singură valoare
- Exemple: COUNT, AVG, SUM, MIN, MAX, STRING\_AGG
- În afară de COUNT, toate funcțiile de agregare ignoră valorile NULL
- Funcțiile de agregare se folosesc de obicei împreună cu clauzele **GROUP BY** și **HAVING**
- Exemplu:

`SELECT COUNT(*) FROM Studenți;` – calculează numărul total de înregistrări din tabelul *Studenți*



# Funcții de agregare

---

– Exemplu:

`SELECT MAX(nr_matricol) FROM Studenți;` – returnează numărul matricol maxim din tabelul *Studenți*

– Exemplu:

`SELECT MIN(nr_matricol) FROM Studenți;` – returnează numărul matricol minim din tabelul *Studenți*

– Exemplu:

`SELECT SUM(preț) FROM Produse;` – calculează prețul total al produselor din tabelul *Produse*





# Clauza GROUP BY

---

- Se folosește pentru a grupa datele din una sau mai multe coloane
- Cel mai adesea funcțiile de agregare folosesc clauza **GROUP BY**
- Fiecare grup este reprezentat în rezultatul final al interogării de către o singură înregistrare
- Dacă o interogare conține **GROUP BY**, toate fazele interogării care se execută după **GROUP BY** (inclusiv SELECT, HAVING, ORDER BY) vor opera pe grupuri



# Clauza GROUP BY

---

– Sintaxa:

```
SELECT column_name1, aggregate_function(column_name2)
FROM table_name
WHERE column_name3 operator value
GROUP BY column_name1;
```



# Clauza GROUP BY

---

- Exemplu:
- Vrem să aflăm care este valoarea totală a comenzilor pentru fiecare client

```
SELECT CustomerID,  
SUM(Freight) AS TotalValue  
FROM Orders  
GROUP BY CustomerID;
```



# Clauza GROUP BY

---

- Exemplu:
- Dorim să aflăm numărul total de comenzi efectuate pentru fiecare client

```
SELECT CustomerID,  
COUNT(OrderID) AS [Number of Orders]  
FROM Orders  
GROUP BY CustomerID;
```



# Clauza GROUP BY

---

- Putem să grupăm și în funcție de mai multe coloane
- Exemplu:

```
SELECT CustomerID, OrderID  
FROM Orders  
GROUP BY CustomerID, OrderID;
```

- Coloanele care **nu apar** în clauza **GROUP BY** **nu pot apărea** în instrucțiunea **SELECT** decât într-o funcție de agregare (cum ar fi COUNT, SUM, AVG, MIN, MAX și altele)



# Clauza HAVING

---

- Se folosește pentru a filtra grupurile rezultate după procesarea clauzei **GROUP BY**
- Doar grupurile pentru care expresia specificată în clauza **HAVING** returnează TRUE vor fi returnate
- Clauza **HAVING** este procesată după ce sunt grupate înregistrările și poate fi folosită cu funcții de agregare



# Clauza HAVING

---

– Sintaxa:

```
SELECT column_name1,  
       aggregate_function(column_name2)  
FROM table_name  
WHERE column_name3 operator value  
GROUP BY column_name1  
HAVING aggregate_function(column_name2) operator value;
```





# Clauza HAVING

---

- Exemplu:
- Dorim să găsim clienții care au comenzi totale mai mici decât 300

```
SELECT CustomerID,  
SUM(Freight) AS TotalValue  
FROM Orders  
GROUP BY CustomerID  
HAVING SUM(Freight) < 300;
```



# Clauza HAVING

---

- Exemplu:
- Vrem să aflăm dacă totalul comenzilor unor anumiți clienți este mai mic decât 300

```
SELECT CustomerID,  
SUM(Freight) AS TotalValue  
FROM Orders  
WHERE CustomerID='ANTON' OR CustomerID='BOLID'  
GROUP BY CustomerID  
HAVING SUM(Freight) < 300;
```



# Subinterogări

---

- O subinterogare este o interogare încorporată într-o altă interogare
- Se poate folosi o subinterogare în clauza WHERE pentru a găsi înregistrările dintr-un tabel care se potrivesc cu înregistrările din alt tabel fără a folosi 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

---

- Compară o valoare scalară cu o mulțime de valori ce provin dintr-o singură coloană
- Sintaxa:

```
scalar_expression { = | < > | != | > | > = | ! > | < | < = | ! < }  
{ SOME | ANY } (subquery that has a result set of one column)
```

- Dorim să afișăm numele tuturor produselor pentru care se găsește **cel puțin o înregistrare** în tabelul *Distribuție* care îndeplinește condiția '*cantitate=23*':

```
SELECT nume FROM Produse  
WHERE cod_p = ANY(SELECT cod_p FROM Distribuție WHERE cantitate=23);  
SAU  
SELECT nume FROM Produse  
WHERE cod_p IN (SELECT cod_p FROM Distribuție WHERE cantitate=23);
```



# ALL

---

- Compară o valoare scalară cu o mulțime de valori ce provin dintr-o singură coloană
- Sintaxa:

scalar\_expression { = | <> | != | > | >= | !> | < | <= | !< }

ALL (subquery that returns a result set of one column)

- Dorim să afișăm toate produsele care au prețul **diferit de prețul minim al tuturor categoriilor de produse**:

```
SELECT nume, preț FROM Produse WHERE preț <> ALL  
(SELECT MIN(preț) FROM Produse GROUP BY cod_c);
```

SAU

```
SELECT nume, preț FROM Produse WHERE preț NOT IN  
(SELECT MIN(preț) FROM Produse GROUP BY cod_c);
```





# 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;
```