**DataBase** – colectie de date care este organizata intr-o maniera care faciliteaza accesarea, managementul si actualizarea datelor.

**PrimaryKey** – o coloana din tabel care idetifica obiectele ca unice.

Keya:

* Nu poate fi null
* E unica pentru orice obiect

Fiecare tabel are o singura keye primara, ce poate fi simpla sau compusa.

**SQL** nu este un limbaj de programare! Si nici un database language. El este un standart,un limbaj(nu de programare) pentru alte limbaje si alte limbaje il folosesc, dar il imbunatataesc sau mai adauga ceva pe langa el.

**Interogari**

**SHOW DATABASES –** arata toate bazele de date existente

**SHOW TABLES** – arata toate tabelele existente in baza de date selectate

**SHOW COLUMNS FROM TABEL** – arata informatii despre atributele din TABEL, si anume Field(nume), Type, NULL, Key(e cheie sau nu), Default(valoare default), Extra(info suplimentara)

**SELECT ATRIBUT1,ATRIBUT2,.., FROM TABEL** – arata coloanele ce apartin atributelor date din tabelul Tabel

[**\***] - se refera mereu la toate elementele, insemnand **all**

**SELECT *DISTINCT* ATRIBUT1,ATRIBUT2,.., FROM TABEL** – distinct se pune intre select si atribute si se asigura ca sa nu fie returnate de 2 ori valori ce se repeta.

**SELECT ATRIBUT FROM TABEL *LIMIT X*** – returneaza doar un nr de X elemente, si anume primele X. Limit e in mysql, in sql e TOP si se pune direct dupa select.

**SELECT ATRIBUT FROM TABEL** ***OFFSET X*** **LIMIT Y** – returneaza y elemente incepand cu al X element. Numararea se incepe de la 0 mereu! Deci offset 3 inseamna ca se incepe de la al 4, sau putem spune ca se ignora primele 3 s. Ordinea de amplasarae a lui offset si limit nu importa. IN SQL se pune OFFSET NR ROWS

In SQL server nu putem combina TOP cu OFFSET ROWS, si se pune ROWS FETCH NEXT X ROWs ONLY

SELECT \* FROM Laborator.dbo.pc\_uri ORDER BY Cod OFFSET 7 ROWS FETCH NEXT 3 ROWS ONLY;

**SELECT ATRIBUT FROM TABEL** **LIMIT Y, X –** e echivalent cu codul de sus. OFFSET poate fi ignorat dupa limit si pus doar , Nu putem inschimb ignora limit si pune offset.

**Uneori cand avem mai multe tabele, e mai comod sa folosim . pentru a arata din care tabel e atributul.**

**SELECT TABEL.ATRIBUT FROM TABEL**

**SELECT ATRIBUT,.., FROM TABLE *ORDER BY* ATRIBUT –** ordoneaza tabelul in ordine alfabetica/numerica in dependenta de atributul selectat. Daca doua elemente incep cu o litera identica, se compara dupa a 2 litera si tot asa. Nu putem sa ordonam fara a avea selectata coloana:

SELECT DISTINCT \* FROM Customers WHERE Age BETWEEN 20 AND 30 ORDER by age; **Nu il putem pune inainte la Where**

**SELECT \* FROM TABLE *ORDER BY* ATRIBUT1,ATRIBUT2 –** ordoneaza tabelul dupa 2 atribute. Intai se ordoneaza dupa primul atribut, si daca se gasesc valori identice, se aranjeaza deja dupa al 2.

**WHERE Conditie** – e folosit pentru a selecta doar acele date inregistrari care corespund unor anumite conditii.

**SELECT ATRIBUT FROM TABEL *WHERE CONDITIE*** –

De ex: SELECT \* FROM STUDENT WHERE ID < 10

**Functii**

* **Concat(Atribut1, ’,’, Atribut2...) –** combina atribute in unul si le separa prin ce punem intre ele. Numele noului atribut va fi identic cu cel al functiei.

SELECT CONCAT(first\_name, ' ', last\_name) FROM Customers;

* **AS –** ca sa nu fie numele noului atribut identic cu numele functiei

SELECT CONCAT(first\_name, ' ', last\_name) AS First\_Last FROM Customers;

* Putem efectua si operatii asupra atributelor selectate, de ex:

SELECT **Age+30** From Tabel;

Numele atributului va fi Age+30, si cu AS il modificam.

* **UPPER**(Atribut) – face in litere mari
* **LOWER**(Atribut) – face litere mici
* **SQRT**(Atribut) - radical
* **AVG(Atribut) –** returneaza o linie cu o coloana ce contine vlorile medii ale tuturor valorilor ale obiectului ale acestui atribut
* **POW(Atribut,putere)**
* **SUM(Atribut) –** returneaza suma atributului dat pentru toate obiectele, intr-o linie si coloana.
* **MIN(atribut) –** valoarea minima a tuturor obiectelor a atributului dat

**Reguli de sintaxa**

* SQL nu e sensibil la litere mari si nici la spatii. Putem folosi fie litere mari, fie mici la comenzi, e tot aceeasi. Nici spatiile sau randurile nu le vede.
* Dupa orice interogare se pune ;
* Pentru a spune programului ca un cuvant e un string, nu cuvant cheie, folosim apostrofe ' '

Ex: SELECT \* FROM STUDENTI WHERE Scoala='Alexandru cel Bun'

* Daca textul contine ', folosim '' pentru a arata ca ' face parte din string

Ex: ELECT \* FROM CITIES WHERE City\_name='L***’’***Empereur'

**Operatori**

* Operatori folositi la conditii(relationari):

= !=

>= <= < >

Nu exista **== caci = e identic si cu ==**

BETWEEN x **AND** y– intre un interval selectat ( x si y) , interval inchis.

* Operatori logici(returneaza true sau false ca in limbajele de programare)
* AND
* OR
* IN – returneaza true daca operandul se gaseste in lista. Folosim

IN (val1, val2 ...)

* NOT

**Like operator:**

Este folosit pentru a identifica valori in dependenta de anumite caractere pe care le contin. Se foloseste dupa WHERE

% - pot fi oricate caractere, si 0 macar

\_ - poate urma doar un caracter

[ABC] – oricare din acestea

De ex:

A%B - cuvant ce incepe cu A si poate avea dupa el oricate alte caractere si se termina cu B

A\_ - cuvantul incepe cu A si are dupa el inca un caracter obligatoriu.

[ABC]% - cuvantul poate incepe cu A sau B sau C si poate avea oricate caractere mai departe

SELECT \* FROM Tabel WHERE Name LIKE ’A%’ – toate numele ce incep cu A

**SubInterogari**

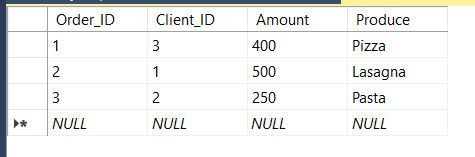
**Sunt interogari in interiorul altor interogari. Se iau mereu intre paranteze.**

SELECT Salary FROM TableWHERE **Salary>=(SELECT AVG(Salary) from table);**

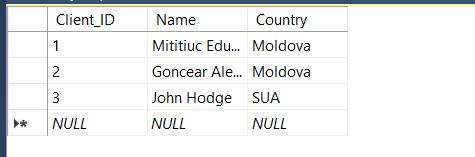
**Join, Tables Operators**

In SQL, tabelele pot fi combinate(join), fiind creat un nou tabel virtual cu toate datele din ambele tabele, doar ca combinate. De exemplu, fie ca avem 2 tabele:

Comenzi



Clienti



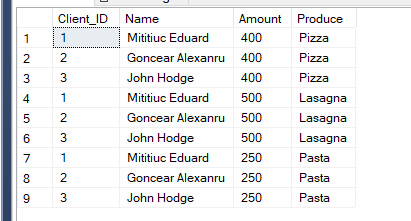
Am putea crea un tabel cu atributele:

Client\_ID, Name, Amount, Produce

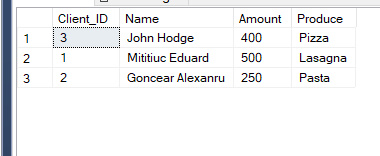
Asa facem un tabel cu anumite atribute din alte 2 tabele. E necesar doar sa selectam mai multe atribute:

**SELECT Clients.Client\_ID, Clients.Name,Orders.Amount,Orders.Produce FROM Clients,Orders ;**

Insa, fiecare obiect cu atributele alese din primul tabel se va combina cu fiecare obiect cu atributele alese din cel de al doilea tabel:



E de ajuns sa punem conditia ca fiecare obiect din tabelul 1 ce se va lua,ca virtual, sa aiba ID la client identic ca la obiectul din al 2 tabel,tot virtual, ca sa stergem asa combinatiile ce combina clienti diferiti.

:

SELECT Clients.Client\_ID, Clients.Name,Orders.Amount,Orders.Produce FROM Clients,Orders **WHERE Clients.Client\_ID=Orders.Client\_ID;**

* Unele tabele ar putea avea nume prea lungi, si nu e comod sa folosim mereu numele lor pentru accesa attribute. De aceea, putem sa le scurtam in interogare prin AS.

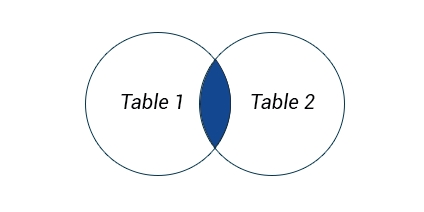
SELECT **Ord**.order\_id, **Ord**.item, **Ord**.amount, **ct**.customer\_id, **ct**.first\_name,**ct**.last\_name FROM **Customers AS ct**, **Orders as Ord** WHERE **ct**.customer\_id = **Ord**.customer\_id;

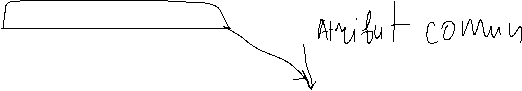
**Join**

Join ne ajuta sa unim mai multe attribute din diferite tabele in unul singur. Putem face acest lucru si fara Join, dar Join ne ofera mai multe poibilitati.

* **INNER JOIN** sau JOIN(care e INNER by default), uneste obiectele pe baza conditiei din ON, si le afiseaza doar daca conditia e true. Join, fara ON, combina fiecare obiect din primul tabel cu toate din al 2 tabel, dar ON va face ca o intersectie, adica se va asigura ca sa aiba un anumit atribut comun ce corespunde conditiei specificate, dar pune NULL la atributele ce ar fi trebuit sa vina din tabelul drept.

SELECT tabel1.A1,tabel2.A2,..., FROM tabel1 **INNER JOIN** tabel2 **ON** tabel1.A1 = tabel2.A;





* **LEFT JOIN –** combina fiecare obiect din Tabel1 cu fiecare obiect din Tabel2 pe baza conditie din ON, insa, daca un obiect din Tabel1 nu gaseste nici-un obiect din Tabel2 care sa corespunda conditiei, va returna acel obiect din Tabel2 cu Null la acel atribut ce nu corespunde conditiei. Fara ON face o simpla combinare. Se combina tabelul din stanga lui JOIN cu cel din dreapta.







* **RIGHT JOIN –** exact ca si LEFT JOIN, doar ca combina fiecare obiect din tabelul drept cu cele din tabelul stang, si daca nu gaseste nici-un obiect din tabelul stang ce ar corespunde conditie din ON, returneaza obiectul ales din dreapta, dar pune NULL la atributele ce ar fi trebuit sa vina din tabelul stang.



* **FULL JOIN –** afiseaza elementele care corespund conditie, si pe cele din primul tabel care nu au gasit o pereche potrivita si pe cele din al 2

Tabel, in mod separat.



**Union**

Este folosit pentru a uni mai multe randuri a 2 tabele diferite intr-un singur tabel. Insa, ambele tabele trebuie sa aiba acelasi numar de atribute, ele sa fie de acelasi tip si sa fie in aceeasi ordine. Union nu selecteaza randurile identice!

SELECT \* FROM TABEL1

UNION

SELECT \* FROM TABEL2

Daca de exemplu un tabel nu are un atribut din alt tabel, se poate pune NULL in loc de acea coloana.

SELECT A1, A2,A3 FROM TABEL1

UNION

SELECT A1, A2, NULL FROM TABEL2

**Union All**

Functioneaza exact ca Union, doar ca selecteaza toate elementele indiferent ca se dubleaza datele sau nu. E mai rapid.

**Alte operatii**

* **Minus** – se selecteaza elementele ce apar in tabel1 si nu sunt in tabel2

**SELECT \* FROM Tabel1**

**MINUS**

**SELECT \* FROM Tabel2**

* **Intersect**
* **Except – este opusul lui intersect, adica se returneaza elementele ce nu coincid.**
* **SEMI-JOIN – exact ca join, doar ca returneaza doar elementele dintr-un tabel.**

**Modificarea obiectelor din tabel**

* INSERT INTO tabel VALUES(val1,val2,..); - insereaza o valaore in tabel
* INSERT INTO tabel VALUES

(val1,va2...),

(val3,val4)

....

Insereaza mai multe valori

* ISERT INTO TABEL(atib2,atrib1,atrib3)

VALUES

(val2,val1,val3)

(val22,bal11,val33)

Adauga randuri, dar adaugam atributele in interogare in ordiena aparitiei lor la TABEL intre ()

Putem folosi si mai putine atribute, dar ele trebuie sa aiba valori default,care sa se seteze automat.

* UPDATE TABEL

SET Atrib1=val1, Atrib2=val2...

WHERE Conditie

Modifica valoriel lui Atrib1 si Atrib2 cu val1 si val2 unde se respecta WHERE

Fara where, se vor modifica aceste atribute la toate obiectele.

* DELETE FROM tabel WHERE Conditie – sterge liniile ce corespund conditie din where. Fara where se sterg toate liniile!

**Tipuri de date**

* Int
* Float(m,n) – m cifre in total(pana si dupa virgula) dintre care n dupa virgula
* Double(m,n)
* long
* Date – data doar
* Datetime – data si ora
* Time – ora
* Timestamp – calculeaza timpul de la 1 ianuarie 1970 timpul
* Char(n) – valori ce pot avea doar n caractere
* Varchar(n) -valori ce pot avea maxim n caractere
* **BLOB –** date binare, ca imagini, fisiere etc.
* **Text –** texte lungi

**Crearea Tabelelor**

* CREATE TABLE Nume(

Atr1 type(size),

Atr2 type(size),

Atr3 type(size)

)

**Constrangeri**

* **Primary Key**

CREATE TABLE Test(

ID long,

Nume varchar(20),

Age int,

**Primary key(ID)**

)

Sau

CREATE TABLE Test(

ID long **PRIMARY KEY**,

Nume varchar(20),

Age int,

)

* **NOT NULL**
* **UNIQUE** – face ca intr-o coloana fiecare obiect sa aiba valori unice.
* **DEFAULT valoare** - setam o valoare default daca nici una nu e furnizata
* **CHECK** (conditie)– verificam daca data e valida sau nu dupa o expresie

CREATE TABLE tabel(

Age int CHECK(Age>18)

)

* **AUTO\_INCREMENT** – face ca valaorea sa creasca automat cu o unitate. Cand inseram datele, trebuie sa specificam dupa numele la tabel ce atribute vom introduce, si sa il excludem pe cel care are aceasta proprietate.(MYSQL)

Pentru a crea un primary key(in sql) IN LOC DE auto\_increment, folosim IDENTITY(X,Y):

**X – valoarea de start**

**Y – valoarea cu care se va incrementa**

Cod int NOT NULL IDENTITY(1,1) PRIMARY KEY

**Modificarea propriu zisa a relatiei**

* ALTER TABLE Tabel ADD Atribut data\_type; - adauga un atribut nou
* ALTER TABLE Tabel DROP COLUMN Atribut; - sterge o coloana
* DROP TABLE Tabel – sterge un tabel
* ALTER TABLE Tabel RENAME Atribut1 TO Atribut2 – modifica numele unui atribut
* ALTER TABLE Tabel RENAME TO Tabel; - modifica numele unui tabel
* ALTER TABEL Tabel ADD [CONSTRAINT NumeConstraint] Constrangere
* ALTER TABLE tabel ALTER COLUMN Atribut1 TYPE\_DATA – modifica tipul de date al unui atribut. Putem sa mai adaugam, pe langa tipul de date, si SET DEFAULT
* ALTER TABLE tabel ALTER COLUMN coloana DROP DEFAULT – scoate constrangerea default. Putem sterge asa orice constrangere
* ALTER TABLE tabel ADD [CONSTRAINT nume\_constraint] PRIMARY KEY(Atribut); - seteaza o primary key cu constrangere
* ALTER TABLE tabel DROP CONSTRAINT nume\_contrangere - sterge o constrangere
* ALTER TABLE tabel ADD CONSTRAINT Nume Constrangere FOR Column
* ALTER TABLE tabel DROP PRIMARY KEY – sterge keia primara

**Vederi(Views)**

O vedere este un tabel virtual, care nu exista in memorie, ce care este bazat pe o multime rezultata dintr-o instructiune SQL. Pot contine date din diferite tabele.

Vederile permit:

* Sa structuram datele intr-un anumit mod
* Sa restrictionam datele in asa fel incat userul sa poata accesa sau modifica doar anumite din ele.
* Sa sumarizam datele din tabele si sa facem rapoarte pe baza lor.
* vedere se creaza asa:

**CREATE VIEW Nume AS**

**SELECT Atr1, Atr2,.., FROM Tabel;**

O vedere este o simpla interogare pe care o facem noi, dar care e salvata si poate fi utilizata. Nu e un tabel!

O putem folosi exact asa ca un tabel simplu:

SEELCT \* FROM View;

O vedere mereu va afisa datele actualizate, caci DBMS foloseste istructiunea vederii pentru a forma de fiecare data tabelul dorit.

* Putem actualiza o vedere:

CREATE OR REPLACE VIEW Nume AS....

* Putem sterge o vedere prin

DROP VIEW Vedere

* Putem modifica view si prin:

ALTER VIEW nume AS [noua seelctie];

**Declarare si lucrul cu variabile**

* Declararea lor:

**DECLARE @Nume\_Var Tip;**

* Setarea lor

**SET @Nume\_Var = Valoare;**

* Afisarea lor:

**SELECT @Nume\_var;**

* Variabilele pot stoca si datele pastrate de alte interogari:

**SET @Variabila = (SELECT AVG(pret) FROM calculatoare.dbo.pc\_uri );**

**IIF(if) statement**

SELECT IIF(conditie,val in caz de true,val in caz de false);

SELECT IIF(@Variabila>500,'YES','NO');

**Crearea Tabelelor**

* Pentru a defini un atribut ca foreign key, folosim dupa el REFERENCES Tabel(Atribut), adica el va fi asociat cu atributul Atribut din tabelul Tabel

...

Cod\_Proprietar int NOT NULL REFERENCES Proprietari(Cod\_Proprietare) ON DELETE CASCADE;

...

ON DELETE CASCADE – la stergerea unui element de tip Cod\_Proprietar din Tabelul Proprietari, va implica si stergerea obiectului Cod\_Proprietar in tabelul de baza.

* Primary Key si foreign Key pot fi definite si la final, si la declararea atributelor:

**CREATE TABLE Angajati(**

**ID long NOT NULL PRIMARY KEY,**

**Nume varchar(20),**

**Prenume varchar(20),**

**Adresa varchar(25),**

**ID\_Shop long REFERENCES Shops(ID\_Shop)**

**);**

Sau

**CREATE TABLE Angajati(**

**ID long NOT NULL,**

**Nume varchar(20),**

**Prenume varchar(20),**

**Adresa varchar(25),**

**ID\_Shop long,**

**PRIMARY KEY(ID),**

**FOREIGN KEY(ID\_Shop) REFERENCES Shops(ID\_Shop)**

**);**

* Putem pune la final si CHECK, pentru a introduce constrangeri, de ex:

CHECK(ID\_SHOP > 0)

**INTO**

Adauga toate coloanele dintr-un tabel in altul.

SELECT column1, column2, column3, ...  
INTO newtable   
FROM oldtable

**Truncate**

Sterge toate tuplurile dintr-un tabel

TRUNCATE TABLE Categories;

**Constraint**

E folosit pentru a oferi un nume unei constrangeri

Poate fi folosita si la declararea atributului, si dupa

**CREATE TABLE Angajati(**

**ID long CONSTRAINT nume1 NOT NULL,**

**Nume varchar(20),**

**Prenume varchar(20),**

**Adresa varchar(25),**

**ID\_Shop long,**

**CONSTRAINT nume2 PRIMARY KEY(ID),**

**CONSTRAINT nume3 FOREIGN KEY(ID\_Shop) REFERENCES Shops(ID\_Shop)**

**);**

E important sa le dam nume ca sa le putem sterge sau modifica mai usor.

**ON DELETE/ ON UPDATE**

Sunt utilizate ca atunci cand se face vreo modificare sau stergere din tabelul parinte, sa fie afectate in vre-un fel si datele din tabelul copil.

**CASCADE** – daca o valoare in tabelul parinte e stearsa/modificata, atunci ea e stearsa/modificata si in tabelul copil.

**SET NULL** – la stergerea/modificarea unei valori din tabelul parinte, in locul valorii date in tabelul copil se pune NULL

**SET DEFAULT** - – la stergerea/modificarea unei valori din tabelul parinte, in locul valorii date in tabelul copil se pune valoarea default

**NO ACTION** - daca o valoare in tabelul parinte e stearsa/modificata, atunci nu se intampla nimic in tabelul copil cu ea.

CREATE TABLE inventory

( inventory\_id INT PRIMARY KEY,

product\_id INT NOT NULL,

quantity INT,

min\_level INT,

max\_level INT,

CONSTRAINT fk\_inv\_product\_id

FOREIGN KEY (product\_id)

REFERENCES products (product\_id)

**ON DELETE CASCADE**

);

**Tipuri de indecsi**

**Unique** / nonunique – se asigura ca nici-un rand din tabel nu au valori identice ale cheii

**Clustered** / nonclustered – index care pastreaza fiecare rand in ordinea apritie lui in tabel, avand o referinta la el. E ca o map.

**Crearea indexului**

CREATE tip\_index INDEX nume ON Tabel(Atribut key) WHERE conditie,

WHERE e optional.

Indecsii pot fi modificati

ALTER INDEX nume ON Tabel DISABLE/REBUILD/ABORD/PAUSE

Indecsii pot fi stersi

DROP INDEX nume ON Tabel

**Stergerea atributului ce e foreign key altundeva**

Pentru a sterge un atribut dintr-un tabel, care aici e primaru key, dar in alt tabel e foreign key, trebuie sa adaugam CASCADE:

ALTER TABLE Atribut DROP PRIMARY KEY CASCADE;

**TOP nr WITH TIES**

SELECT TOP 4 WITH TIES

ProductName, Price

FROM Products

ORDER BY Price

Daca elementul al 4 mai are un element identic cu el, sau mai mule, ce urmeaza exact dupa el, le va afisa si pe ele.

**HAVING**

Exact ca WHERE, doar ca e folosit ca filtru petru GROUP BY

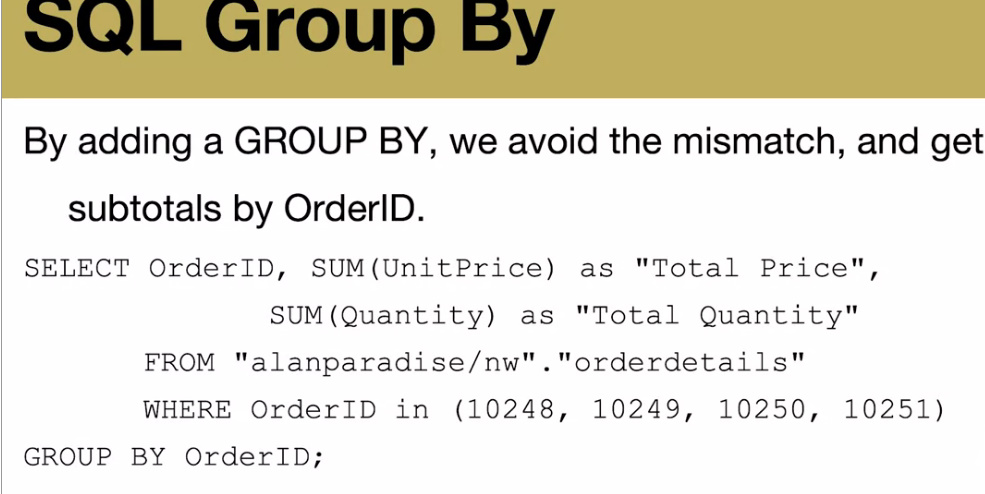
SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;

**GROUP BY**

Grupeaza coloanele dupa 1 sau mai multe atribute. Se returneaza un singur tuple pentru fiecare grup creat.

Daca se foloseste 2 atribute la GROUP BY, gen GROUP BY Country, City, atunci tuplurile se vor grupa dupa Tara si Oras, adica va aparea O tara si orasul ei o singura data. Tara poate deci de mai multe ori, caci are mai multe orase, dar fiecare oras doar odata.

* GROUP BY – el ii spune lui SQL care atribut anume e cel dupa care trebuie sa se faca gruparea. El se va asigura ca pentru fiecare valoare unica a unui atribut pus, sa se creeze o submultime



Deci, aici aratam ca se vor crea submultimi, pe baza la valorile lui OrderID, unice evident, si se va face suma pentru fiecare ID din OrderID care se gaseste prin tabel. **Putem avea oricate functii de grupare, important sa nu fie mai multe atribute de cat avem in GROUP BY**, caci daca punem mai multe atribute, dar la GROUP BY doar unul, nu e clar ce valoare sa se afiseze pentru celalalt, asa cum exista multe tupluri cu valori diferite.

* **Orice atribut din SELECT trebuie sa fie o group function sau un atribut pus in GROUP BY. ASTA E REGULA!!!**

**!!!!Atentie, cand definim o noua coloana, cu AS, unde folosim functii de group, nu vom folosi in Having numele acelei coloane, ci vom folosi functia:**

SELECT employeeid, count(orderid) AS "Orders"

          FROM "alanparadise/nw"."orders"

       GROUP BY employeeid

       HAVING count(orderid) > 100

       ORDER BY 2 desc;

SELECT employeeid, count(orderid) AS "Orders"

          FROM "alanparadise/nw"."orders"

       GROUP BY employeeid

       HAVING "Orders" > 100

       ORDER BY 2 desc;

**GROUP BY + ROLLUP/CUBE/SETS**

**ROLLUP –** genereaza subtotaluri, de ex:

GROUP BY ROLLUP(A1,A2,A3,A4), va forma tupluri pe baza la:

A1,A2,A3,A4

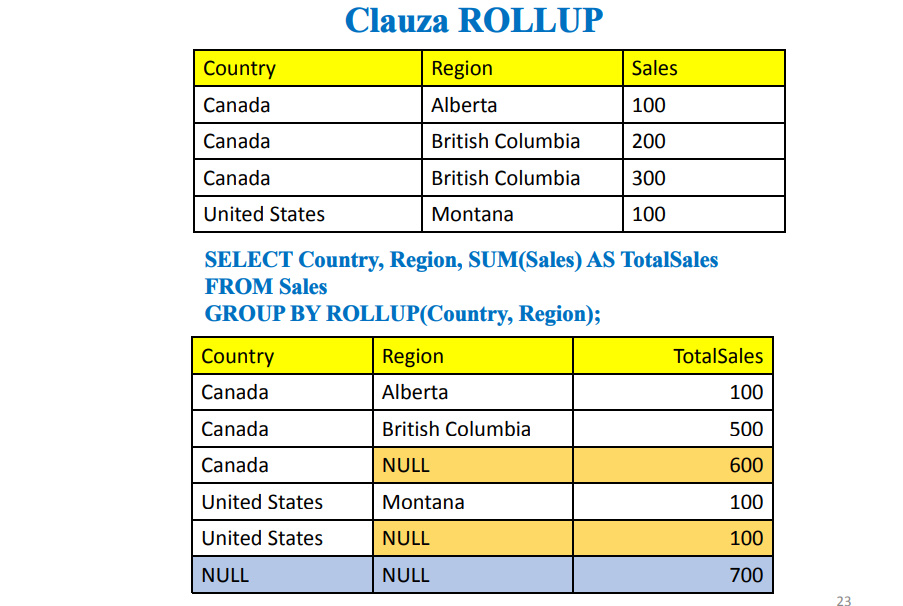
A1,A2,A3,NULL

A1,A2,NULL,NULL

A1,NULL,NULL,NULL,NULL

NULL,NULL,NULL,NULL

Deci se face total pentru toate 4, apoi pentru fiecare 3 si apoi fiecare 2 si fiecare 1.



**CUBE** – fiecare atribut va fi rand pe rand NULL

GROUP BY CUBE(A1,A2), vom avea:

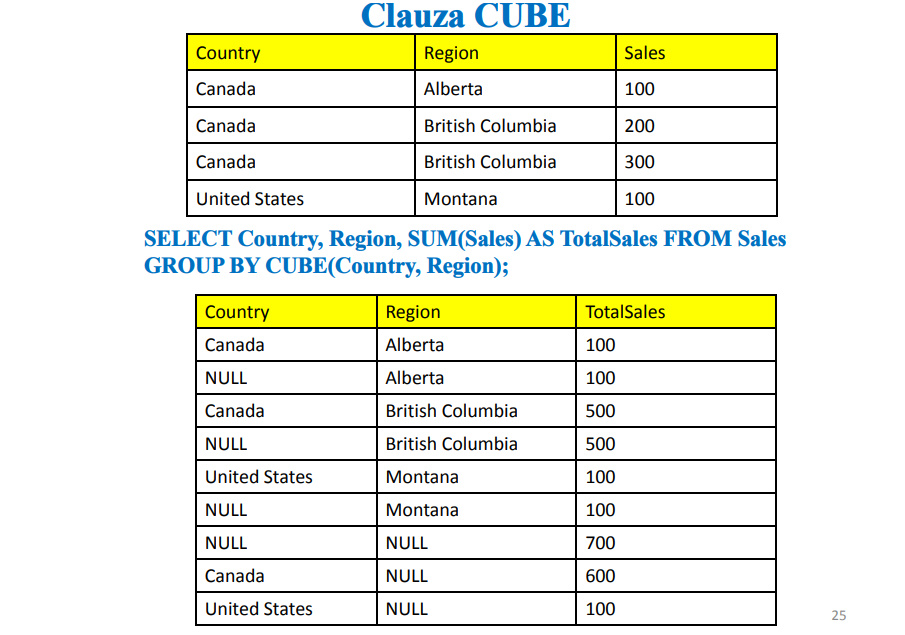
(A1,A2)

(NULL,A2)

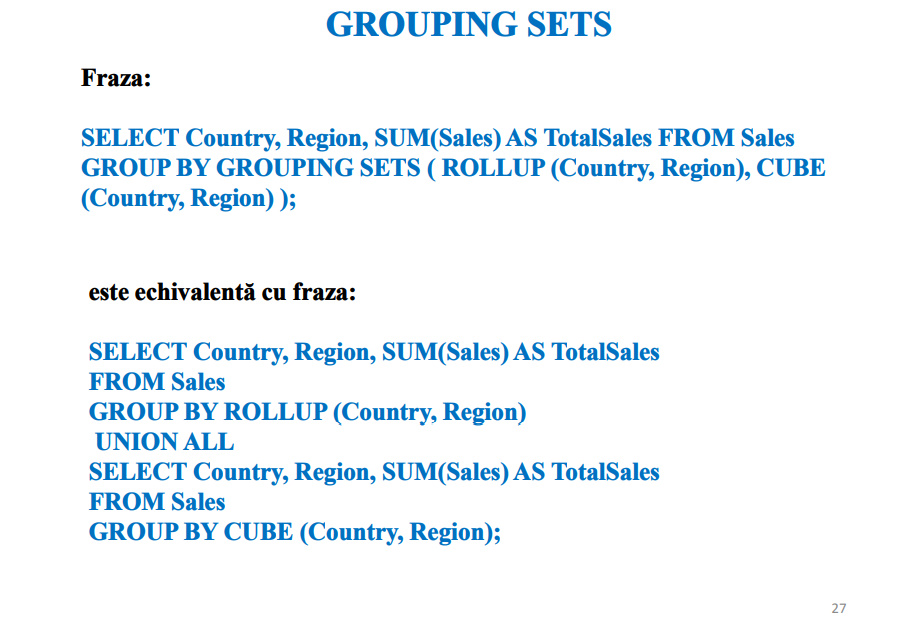
(A1,NULL)

(NULL,NULL)

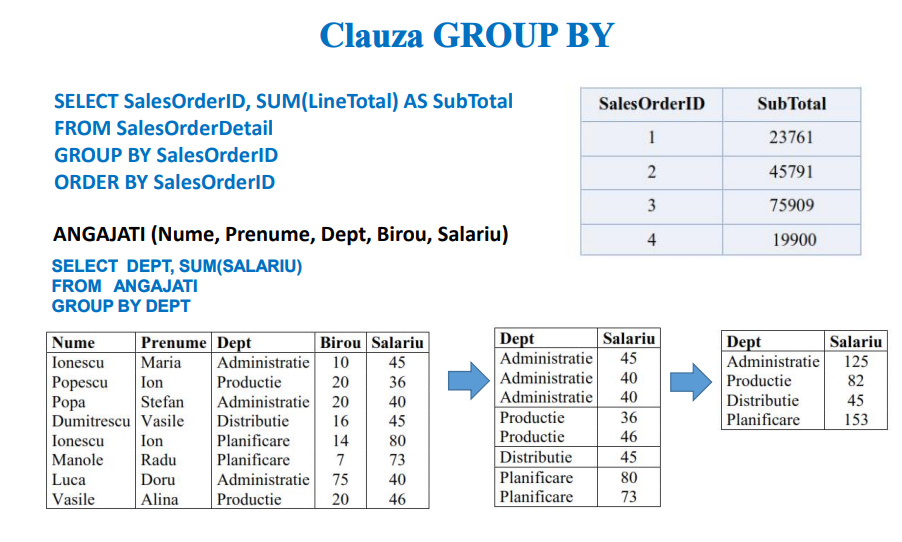
Asa vedem cate elemente de tip Atribut2 sunt si apoi cate de tip Atribut 1 sunt(total).



**SETS** – permite sa combinam mai multe GROUP BY in una







**WHERE**

**CONTAINS**

SELECT Model, Producător FROM calculatoare.dbo.produse WHERE CONTAINS(Model, 'te');

**ANY** – se arata tuplul daca se gaseste oricare care ar satisface conditia.

SELECT \*

FROM Teachers

WHERE age = ANY (

SELECT age

FROM Students

);

Deci, se va arata tuplul daca se gaseste orice student ce are o varsta egala cu a profesorului

**ALL**

SELECT \*

FROM Teachers

WHERE age > ALL (

SELECT age

FROM Students

);

Deci, se va arata tuplul daca age e mai mare ecat age la toti studentii

**Exists** – se arata tuplul daca exista macar un element gasit.

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price = 22);

**ORDER BY**

SELECT A1, SUM(A2) AS total FROM tabel ORDER BY A1 COLLATE collate\_name ASC|DESC OFFSET x ROWS FETCH NEXT y ROW ONLY

Daca se scriu mai multe coloane, rezultatul e sortat intai dupa prima, si daca se intalnesc valori identice la prima, atunci dupa a 2 si tot asa.

**NULL**

**NULLIF(expresie1, expresie2) –** returneaza NULL daca ambele expresii sunt egale

COALESCE(expresie1,...) – returneaza prima valoare a expresiei care nu e NULL

**COUNT**

COUNT(\*) – returneaza nr total de tupluri gasite

COUNT(ALL expression) – evalueaza expresia pentru fiecare tuplu din grup

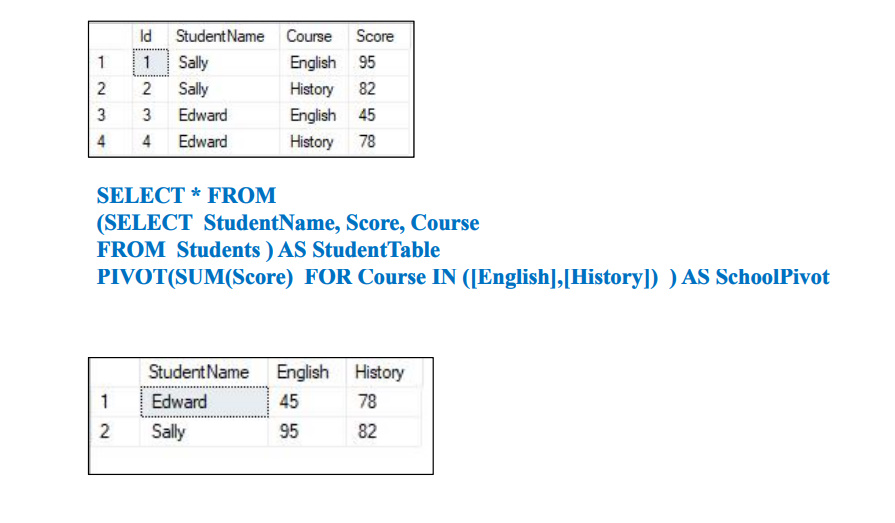
COUNT(DISTINCT expression) - calculeaza nr de valori unice

**PIVOT UNPIVOT**

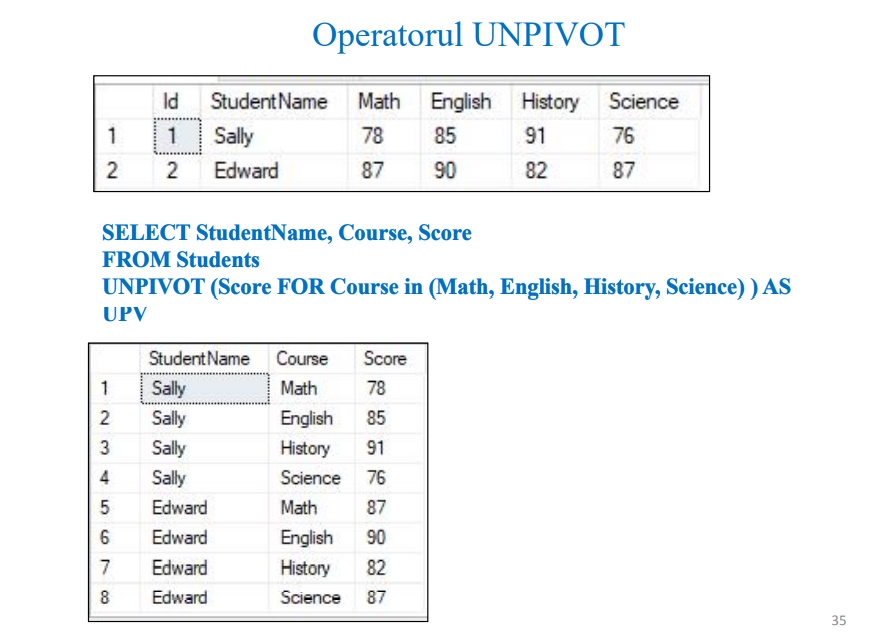
Ne permit sa cream un tabel incrucisat pe baza tuplurilor din alt tabel. Ele ne permi transformarea randurilor in coloane si invers.

Trebuie neaparat sa avem 2 selecturi, asa e structura obligatorie. adica SELECT ... FROM (SELECT ...)

**PIVOT** – transforma liniile in coloane. Adica transforma valorile unice ale unui atribut in coloane.

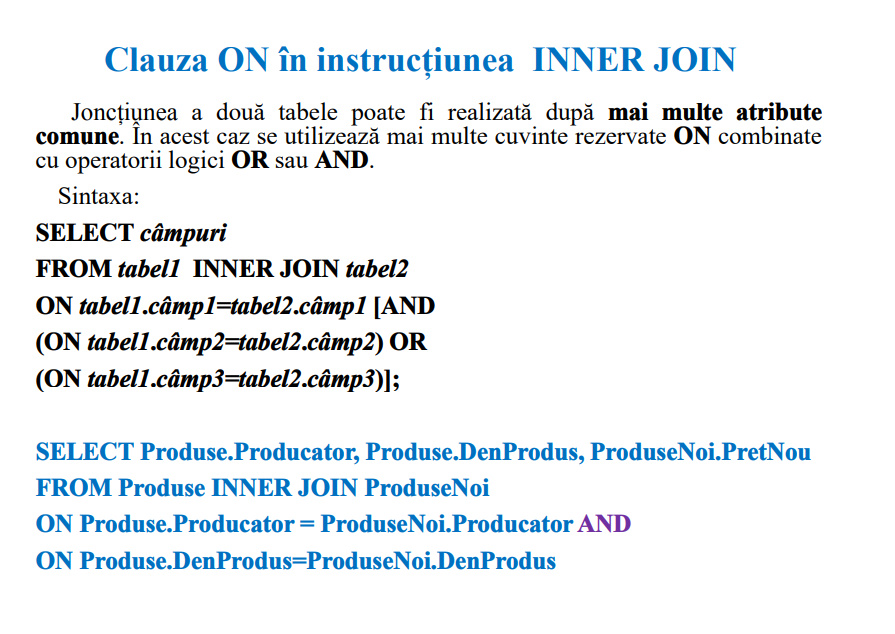


**UNPIVOT** – transforma coloanele in linii. Adica transforma valorile coloanelor intr- singura coloana.



**ON**

On se foloseste cu JOIN. WHERE se foloseste pentru a mai filtra datele obtinute dupa JOIN, insa ON pentru a impune conditia la combinare.



**Common Table Expression(CTE)**

* CTE – este o multime rezultat temporara cu nume ce poate fi utilizata intr-o instructiune SELECT, INSERT, UPDATE sau DELETE.
* Acesta se defineste prin adaugarea instructiunii WITH inainte de SELECT, INSERT ....
* WITH poate contine mai multe CTE separate prin ,
* WITH permite sa oferim unei subinterogari un nume.
* Aceasta poate fi folosita doar in interogarea existenta, nu si in altele!
* Sintaxa:

WITH temporaryTable (averageValue) as

(SELECT avg(Attr1)

FROM Table)

SELECT Attr1

FROM Table, temporaryTable

WHERE Table.Attr1 > temporaryTable.averageValue;

temporaryTable - Numele oferit subinterogarii(CTE table)

averageValue - Atributul prezent in acest CTE

* Putem folosi CTE cum dorim dupa ce l-am definit. El nu influenteaza in nici-un fel instructiunile de dupa el, si poaet fi folosit ca un tabel separat.
* Putem crea cate CTE vrem cu ,:

WITH totalSalary(Airline, total) as

(SELECT Airline, sum(Salary)

FROM Pilot

GROUP BY Airline),

airlineAverage(avgSalary) as

(SELECT avg(Salary)

FROM Pilot )

SELECT Airline

FROM totalSalary, airlineAverage

WHERE totalSalary.total > airlineAverage.avgSalary;

Dupa FROM scriem numele CTE, insa acest fapt nu va duce la crearea produsului cartezian. El e scris doar a specifica ca il vom folosi anume pe el mai departe.