Curs 4. ACCESS SQL

1.Expresii şi funcţii în interogări

Expresiile sunt instrumente folosite de programatori pentru a spori performanţele aplicaţiilor. Sunt formate dintr-o combinaţie de operatori, constante, variabile şi funcţii ce au ca rezultat o anumită valoare. Expresiile pot fi folosite în interogări, formulare, rapoarte, proprietăţi ale câmpurilor tabelelor sau ale controalelor din formulare şi rapoarte, în funcţii macro şi module. Ele pot fi folosite pentru a stabilii criterii, ca reguli de validare sau ca bază pentru coloanele calculate.
Exemplu: dacă într-o interogare introduceţi următoarea expresie în câmpul *Criteria* corespunzător unei coloane de tip Date/Time ea va returna numai înregistrările mai vechi de 3 zile: < Now ( ) + 3.

1.1.Părţile componente ale expresiilor

Expresiile pot fi private ca nişte propoziţii matematice, în componenţa cărora pot intra:

- **Operatori:** simboluri matematice;
- **Constante**: valori numerice sau şiruri de caractere ce nu îşi schimbă valoare;
- **Funcţii**: proceduri ce returnează o valoare;
- **Nume de câmpuri**: de obicei sunt introduse în paranteze drepte.

Operatori Joacă un rol important în cadrul expresiilor. Tipuri de operatori:

Operatori numerici
Sunt folosiţi pentru a efectua calcule matematice cu două sau mai multe valori numerice.

- ^ ridică un număr la o putere(de ex. $2^3=2^3$)
- * înmulţeşte două numere
- / împarte două numere şi returnează un număr real
- \ împarte două numere şi returnează un număr întreg
- **MOD** returnează restul împărţirii a două numere
- + adună două numere
- - scade două numere

Operatori de concatenare Sunt folosiţi pentru a lega două şiruri de caractere.

- **&** concatenează două şiruri de caractere
- + adună valorile a două câmpuri numerice. Poate fi folosit şi pentru a concatena şiruri de caractere

Operatori de comparaţie Sunt folosiţi pentru a compara valorile a două sau mai multe câmpuri şi/sau expresii.

- = verifică egalitatea a două valori
- <> verifică dacă două valori sunt diferite
- < verifică dacă o valoare este strict mai mică decât alta
- > verifică dacă o valoare este strict mai mare decât alta
- <= verifică dacă o valoare este mai mică sau egală cu alta
- >= verifică dacă o valoare este mai mare sau egală cu alta
- **Like** verifică dacă valoarea unui camp se potriveşte unui tipar(ex.: Like abc* se vor potrivi toate şirurile de caractere care încep cu "abc")
- **Between** verifică dacă valoarea unui câmp se află între alte două valori

Operatori logici
Efectuează operaţii logice.

- **Not** introduce o negaţie (de ex. Not T* introdus într-o interogare în câmpul *Criteria* al coloanei Nume va regăsi toţi studenţii a căror nume nu începe cu litera T)
- **And** efectuează conjuncţia a două valori

- **Or** efectuează disjuncţia a două valori
- **Xor** efectuează disjuncţia exclusivă a două valori
- **Eqv** verifică echivalenţa a două valori
- **Imp** operatorul "implică"

Precedenţa operatorilor

De multe ori, a calcula valoarea unei expresii implică efectuarea mai multor operaţii. Aceste operaţii se efectuează într-o anumită ordine predefinită, care este şi ordinea în care au fost prezentaţi operatorii de mai sus.
Dacă doriţi să schimbaţi această ordine (de exemplu să efectuaţi întâi o operaţie de adunare şi apoi o ridicare la putere), se folosesc parantezele pentru a preciza care operator va fi aplicat primul.

1.2.Constante

Nu îşi modifică valoarea pe parcursul rulării aplicaţiei. Se împart în trei grupe.

Constante predefinite

Sunt definite de programator, de obicei în module. Declaraţia unei constante predefinite începe cu cuvântul cheie CONST. După ce au fost declarate, ele pot fi folosite oriunde.
*Exemplu:*CONST Pi = 3.14

Constante intrinseci

Sunt constante furnizate de Visual Basic (cum ar fi *VbString* sau *VarType*) şi nu trebuie să fie declarate separat. Programatorul nu poate defini o constantă care să aibă acelaşi nume cu o constantă intrinsecă. Constantele intrinseci se pot folosi numai în module.

Constante sistem

Yes,No,On,Off şi *Nul* sunt cele cinci constante sistem. Ele pot fi folosite în orice obiect în baza de date, mai puţin în module.

1.3. Funcţii

Funcţiile pot intra în componenţa expresiilor. Ele acţionează ca şi operatorii dar nu sunt reprezentate prin simboluri. De exemplu, operatorul + şi funcţia Sum efectuează aceeaşi operaţie: cea de adunare. Diferenţa este aceea că funcţia *Sum* poate fi aplicată mai multor valori odată. Access vă pune la dispoziţie peste 160 de funcţii, dintre care câteva le puteţi vedea în câmpul *Totals* al grilei QBE a unei interogări de tip *Totals.*

1.4. Nume de câmpuri şi parametrii

Numele coloanelor din tabele sau cele ale parametrilor pot intra şi ele în componenţa unei expresii. De obicei, numele coloanelor trebuie incluse între paranteze drepte. Numele de parametrii sunt cele pe care le declaraţi în cutia de dialog Parameters a unei interogări. De asemenea, în expresii pot apărea şi aliasuri pentru numele coloanelor. Puteţi folosi numele unui camp într-o interogare numai dacă el se află în panoul ferestrei QBE sau dacă el este obţinut printr-o sub-interogare.

1.5.Localizarea expresiilor

In funcţie de locul în care introduceţi expresiile, acestea vor avea un anumit efect.
Dacă scrieţi o expresie în câmpul *Field* al grilei QBE a unei interogări, în modul *Dataseet View* va aparea o nouă coloană.
Dacă expresia este introdusă în câmpul *Criteria* sau *Total* al grilei QBE corespunzător unei coloane, asupra coloanei respective vor fi impuse limitări sau se vor efectua operaţii de agregare. După ce aţi introdus o expresie într-o celulă a grilei QBE aţi apăsat tasta Enter, Access va parcurge expresia pentru a o verifica din punct de vedere sintactic. Dacă va fi detectată vreo eroare "gramaticală", Access vă va indica porţiunea din expresie unde se află eroarea respectivă. Nu veţi putea rula interogarea până când problema nu va fi rezolvată.

1.6. Lucrul cu date şi ore

In Access, expresile pot conţine şi date sau ore. Acestea sunt stocate ca numere reale pe 64 de biţi. După ce o data/oră a fost introdusă într-o expresie Access parcurge expresia şi recunoaşte formatul respectiv, incuzând data(ora) între două caractere diez (#).
Access vă pune la dispoziţie aproximativ 22 de funcţii pentru lucrul cu date şi ore.

Exemplu: Interogarea Vechime foloseşte funcţiile *Year* şi *Now* pentru a calcula câţi ani au trecut de la angajarea unui profesor şi până în prezent. *Year()* returnează anul dintr-o dată completă, iar *Now()* returnează data/ora curentă.

Există câteva formate predefinite pentru afişarea datelor/orelor în formulare, rapoarte, în modul *Datasheet View* al unei tabele sau interogări etc. Tabelul următor prezintă o listă cu aceste formate, care se poate vizualiza şi în câmpul proprietăţii *Format* a unei coloane de tip date/time a unei tabele (în modul *Table Design*).

Format	Afişare (exemplu)
General	01/01/06 12:00:00 AM
Short Date	01/01/06
Medium Date	1-Jan-06
Long Date	Thursday, January 1,2006
Short Time	00:00:00
Medium Time	12:00 AM
Long Time	12:00:00 AM

Prin intermediul expresiilor se poate personaliza modul cum vor fi afişate datele/orele.

Format	Afişare
d	Ziua din lună, între 1 şi 31
dd	Ziua din lună, între 01 şi 31
ddd	Ziua săptămânii (primele trei litere din ziua săptămânii în limba engleză; ex: Mon pentru luni, Tue pentru marţi, etc.)
dddd	Ziua săptămânii în limba engleză (întreg cuvântul; ex: Monday pentru luni, Tuesday pentru marţi, etc.)
w	Ziua săptămânii, între 1 şi 7
ww	Săptămâna din an, între 1 şi 52
m	Luna anului, între 1 şi 12
mm	Luna anului, între 01 şi 12
mmm	Primele trei litere din luna anului în limba engleză (ex: Jan pentru ianuarie, Feb pentru februarie etc.)
mmmm	Luna anului în limba engleză (întreg cuvântul; ex: January pentru ianuarie, February pentru februarie etc.)
q	Trimestrul anului, între 1 şi 4
y	Ziua din an, între 1 şi 365
Yy	Anul, între 00 şi 99
Yyyy	Anul, între 0100 şi 9999
H	Ora, între 1 şi 23
Hh	Ora, între 01 şi 23
N	Minutul, între 1 şi 59
Nn	Minutul, între 01 şi 59
S	Secunda, între 1 şi 59
Ss	Secunda, între 01 şi 59
AM/PM	AM pentru ore între miezul nopţii şi miezul zilei; PM pentru ore între miezul zilei şi miezul nopţii
am/pm	am pentru ore între miezul nopţii şi miezul zilei; pm pentru ore între miezul zilei şi miezul nopţii
A/P	A pentru ore între miezul nopţii şi miezul zilei; P pentru ore între miezul zilei şi miezul nopţii

Semnul slash (/) este folosit, de obicei, ca separator pentru dată, iar două puncte (:), ca separator pentru oră.
Folosind aceste formate, se pot crea expresii care să returneze valori de genul: Luna angajarii este Feb.

*Exemplu:*Introduceţi câmpul Field al grilei QBE expresia:

```
Luna:"Luna angajarii este"&Format([DataAngajarii], "mmm")
```

unde:

- Luna: este numele coloanei care va conţine, în tabela de rezultate, valorile expresiei;

- "Luna angajarii este" este textul care va apărea înaintea lunii angajării;
- Format ([DataAngajarii], "mmm") converteşte valoarea stocată în câmpul DataAngajării la formatul mmm.

Notă: Dacă observaţi că informaţiile returnate sunt eronate, verificaţi data şi ora sistemului (în Control Panel, la secţiunea Date/Time), deoarece funcţiile Now() şi Date() se bazează pe acestea.

1.7. Funcţia IIF în expresii

```
IIF (<expresie>, valoare1,   valoare2)
```
unde:

- `<expresie>` este o expresie a cărei valoare de adevăr este evaluată pentru fiecare înregistrare în parte;
- `valoare1` este valoarea returnată dacă <expresie> este adevărată;
- `valoare2` este valoarea returnată dacă <expresie> este falsă.

Exemplu: Pentru fiecare student dorim să aflăm dacă a promovat sau nu toate cursurile opţionale la care s-a înscris (adică, dacă nota minimă obţinută este sau nu mai mare sau egală cu cinci). In tabela cu rezultatele interogării vom avea o coloană, numită Situaţia, care va avea valoarea "promovat" sau "nepromovat".

```
Situaţia: IIf (Min ([Nota])>=5, "Promovat", "Nepromovat")
```

1.8. Crearea expresiilor cu Expression Builder

Permite crearea de expresii complicate folosind doar mouse-ul. El poate fi lansat de oriunde.
Poate fi introdusă o expresie (câmpuri pentru specificarea regulilor de validare, a surselor de date, a criteriilor etc.), după caz, fie făcând clic pe butonul …ce apare în dreapta câmpului, fie făcând clic dreapta în celula respectivă, şi alegând din meniul context comanda *Build.*
Expression Builder se prezintă sub forma unei ferestre modale, împărţită în două. In partea superioară se află un câmp de editare în care va fi scrisă expresia, precum şi butoane pentru introducerea rapidă a operatorilor aritmetici, logici, de concatenare şi de comparaţie. In partea de jos există trei coloane. In cea din stânga veţi găsi toate obiectele care conţin elementele ce ar putea intra în componenţa unei expresii: tabele, interogări, formulare, rapoarte, funcţii, constante, operatori, expresii predefinite. Să presupunem că veţi dori să folosiţi funcţia Date () în cadrul expresiei. Pentru aceasta, faceţi dublu-clic pe folder-ul Functions şi în coloana din mijloc veţi vedea toate categoriile de funcţii predefinite pe care Access vi le pune la dispoziţie. Apoi, faceţi clic pe grupul care conţine funcţia anume *Date/Time*. In coloana din dreapta vor apărea toate funcţiile predefinite care se ocupă cu lucrul cu date şi ore. Aici, făcând dublu clic pe funcţia *Date,* ea va apărea în câmpul de editare în partea de sus a cutiei de dialog. Astfel, puteţi introduce în câmpul de editare toate elementele care compun expresia dorită. După ce aţi terminat, apăsaţi butonul OK şi expresia va fi introdusă în celula respectivă.

2. Access Jet Engine: procesarea interogărilor

Este important pentru un programator să înţeleagă modul în care motorul Jet Engine procesează interogările, deoarece astfel, aplicaţiile sale pot fi mult mai rapide în execuţie.
Motorul Access Jet Engine procesează interogările în trei etape: *compilarea, optimizarea* şi *execuţia.* Fiecare dintre aceste etape trebuie să fie efectuată cu succes, altfel procesorul este oprit şi apare un mesaj de eroare. Acest mesaj poate fi interceptat de către programator şi personalizat înainte de a fi afişat.

2.1. Definirea unei interogări

Prin definirea unei interogări se înţelege crearea sa cu ajutorul unuia dintre instrumentele furnizate de Access: fereastra QBE, limbajul Access SQL sau Data Access Objects (DAO). Access transpune apoi interogările în instrucţiuni SQL şi le transmite mai departe motorului Jet Query Engine, pentru a fi procesate.

2.2. Compilarea unei interogări

Prima operaţie pe care o efectuează motorul Jet Engine este verificarea sintaxei interogării SQL şi returnarea unui mesaj de eroare, dacă este cazul. Urmează parcurgerea interogării pentru a face legătura între numele invocate şi coloanele corespunzătoare din tabelele pe care se bazează interogarea. Pe urmă, interogarea este compilată într-un format intern, după care urmează faza de preoptimizare. Acum, interogarea este împărţită în elemente fundamentale (tabele de bază, coloane ce vor apărea în tabela de rezultate, restricţii, coloane de asociere (Join), coloane după care vor fi sortate rezultatele). Astfel, se reduce complexitatea interogării, crescând probabilitatea ca optimizatorul să îmbogăţească semnificativ structura şi performanţa ei.

2.3. Optimizarea unei interogări

Optimizatorul este una dintre cele mai complexe componente ale motorului Jet Query Engine, el fiind responsabil de alegerea unei strategii optime de execuţie a interogării. Alegerea se bazează pe crearea unei liste cu strategiile de execuţie posibile şi pe evaluarea timpilor de execuţie necesari fiecăreia. Optimizatorul va alege strategia pentru care timpul de execuţie este minim. Fiecare strategie constă într-o însuşire de operaţiuni ce trebuie efectuate şi care necesită, fiecare, un anumit timp de execuţie. Operaţiile care pot afecta în cea mai mare măsură timpul total de execuţie a interogării sunt parcurgerea tabelelor de bază şi efectuarea asocierilor. Pentru fiecare dintre acestea, optimizorul poate allege dintre mai multe strategii, pe cea mai performantă.

2.4. Execuţia unei interogări

După ce a fost elaborat planul optim, motorul motorului Jet Query Engine execută acest plan pas cu pas şi returnează rezultatele. Aveţi posibilitatea de a alege între două tipuri de tabele-rezultat: **dynaset** şi **snapshot**.
Un **dynaset** este o tabelă de rezultate care permite modificarea datelor. Când Jet Engine rulează o interogare ce va avea ca rezultat un dynaset, el creează în memorie un şir de valori-cheie unice care indică datele din tabele de bază. Valorile coloane ce compun un dynaset nu sunt cifre citite decât atunci când este necesar (de exemplu: când utilizatorul îl parcurge pentru a vedea o anumită linie), nefiind astfel nevoie de prea mult spaţiu în memorie.
La crearea unui **snapshot**, motorul Jet Engine încarcă în memorie toate rezultatele interogării. Astfel, dacă memoria nu este suficientă, datele vor fi scrise şi pe disc, fiind ştiut faptul că accesul la datele de pe disc este mult mai lent decât cel la datele din memorie.
In concluzie, tabelele dynaset sunt mult mai eficiente atunci când interogarea returnează multe date. Dacă interogarea returnează un număr redus de coloane şi linii, este mai eficient să folosiţi tabelele snapshot. Este important faptul că tabelele snapshot, nu permit modificarea datelor.

2.5. Strategii de parcurgere a tabelelor de bază

- *Parcurgere secvenţială*: Cea mai lentă metodă, sunt citite toate înregistrările din tabela de bază şi, pentru fiecare înregistrare sunt verificate restricţiile. Această metodă este aleasă pentru tabele mici şi neindexate.
- *Parcurgerea pe indecşi*: Este aleasă atunci când există restricţii asupra unei coloane indexate dintr-o tabelă de bază. După ce sunt selectate doar înregistrările pentru care sunt verificate aceste restricţii, aceste înregistrări vor fi supuse şi restului de restricţii (dacă există). Această metodă este eficientă atunci când tabelele de bază sunt mai mari, astfel încât deşi se vor citi aceleaşi înregistrări de mai multe ori, timpul total este mai mic decât cel necesar pentru a citi întreaga tabelă.
- Tehnica restricţionării *Rushmore*: Este aplicată atunci când există restricţii asupra mai multor coloane indexate. Prin folosirea mai multor indecşi, motorul Jet reduce considerabil numărul de înregistrări ce trebuie citite. In funcţie de tipul restricţiei, Jet Engine va efectua una dintre următoarele trei operaţii:
    - **Intersecţia indecşilor:** pentru restricţii de forma:
      `col1 = <exp> AND col2 = <exp>`
      unde `col1` şi `col2` sunt coloane indexate. Pentru fiecare restricţie este creată câte o mulţime de rezultate, care apoi sunt intersectate pentru a găsi înregistrările care verifică ambele restricţii.
    - **Reuniunea indecşilor:** pentru restricţii de forma:
      `col1 = <exp> OR col2 = <exp>`
      unde `col1` şi `col2` sunt coloane indexate. Pentru fiecare restricţie este creată câte o mulţime de rezultate, care apoi sunt reunite pentru a găsi înregistrările care verifică cel puţin una din cele

două restricţii.

- **Numărul indecşilor:** pentru interogări ce returnează numărul înregistrărilor ce verifică o anumită mulţime de restricţii:

```
SELECT Count (*) FROM Tabela
   WHERE col1 = <exp> AND col2 = <exp>
```

Pentru astfel de interogări, Jet Engine nu trebuie decât să citească paginile cu indecşi.

2.6. Strategii de efectuare a asocierilor

Pe baza unor statistici asupra tabelelor, motorul Jet Engine alege una dintre următoarele strategii de asociere, atunci când interogarea se bazează pe mai multe tabele:

- **Asociere prin iteraţii imbricate:** Folosită când nu există indecşi şi tabelele conţin puţine înregistrări.
- **Asociere prin index:** Parcurge înregistrările dintr-o tabelă şi caută înregistrările corespunzătoare în cea de-a doua tabelă, folosindu-se de un index. Este aleasă când cea de-a două tabelă este mică şi are coloana de asociere indexată sau când prima tabelă e mică şi puternic restricţionată.
- **Asociere de tip lookup:** Este similară prin index, cu diferenţa că înainte de efectuarea asocierii din tabela a doua sunt selectate şi sortate doar coloanele ce vor apărea în tabela de rezultate. Este folosită când cea de-a doua tabelă nu are coloana de asociere indexată.
- **Asociere de tip merge:** Sortează cele două tabele după coloanele de asociere şi apoi le combină parcurgându-le simultan. Este folosită când cele două tabele sunt la fel de mari şi rezultatele trebuie să fie în funcţie de coloane de asociere.
- **Asociere de tip index-merge:** Este similară unei asocieri de tip merge, cu diferenţa că, pentru a ordona cele două tabele, sunt folosiţi indecşii. Este aleasă atunci când ambele tabele au coloana de asociere indexată şi cel puţin una dintre cele două nu conţine valoarea NULL.

2.7. Statistici

Pentru a putea evalua diferitele strategii de parcurgere a tabelelor sau de asociere, motorul Jet Engine foloseşte următoarele statistici asupra fiecărei tabele pe care se bazează interogarea:

- Numărul înregistrărior din tabelă;
- Numărul paginilor ocupate de tabelă;
- Indecşi: dacă sunt unici; numărul paginilor cu indecşi; dacă există valoarea NULL în coloanele indexate.

2.8. Greşeli frecvente

- **Expresii în coloanele rezultat:** Motorul Jet Engine nu poate optimiza interogările care conţin expresii de tip IIF pentru coloanele rezultat. Folosiţi expresiile la nivelul controalelor din rapoartele sau formularele care vor folosi rezultatele interogării.
- **Prea multe coloane cu clauza GROUP BY:** Când creaţi o interogare de tip Totals, impuneţi clauza GROUP BY asupra cât mai puţine coloane. Astfel, timpul de execuţie va fi mai redus.
- **Indexarea coloanei de asociere pentru o singură tabelă:** Când folosiţi o asociere între două tabele, indexaţi câmpurile de asociere din ambele tabele.
- **Folosirea ineficientă a indecşilor:** Dacă nu se efectuează modificări frecvente asupra datelor, este bine să folosiţi indecşi pentru coloanele de asociere sau pentru coloanele asupra cărora se impun restricţii. Altfel, indecşii pot îngreuna sensibil operaţiile de actualizare a datelor.

2.9. Importanţa compactării

Compactarea unei baze de date este o operaţie asemănătoare defragmentării discurilor fizice. Pe măsură ce ştergeţi obiecte sau înregistrări din baza de date, vor rămâne spaţii nefolosite care măresc dimensiunile fişierului şi scad performanţele acţiunilor. Singurul mod în care pot fi eliminate aceste spatii este folosirea comenzii *Database Utilities* ▏*Compact Database.* Astfel, vor fi reânoite statisticile asupra tabelelor, vor fi reordonate înregistrările din tabele, iar spaţiul liber va putea fi folosit. In consecinţă, vor creşte şi performanţele interogărilor, despre care am mai spus că cel mai important instrument pus la dispoziţia programatorilor şi fără de care bazele de date nu şi-ar mai afla utilitatea.

3. Access SQL

Limbajul de interogare a bazelor de date SQL (*Structured Query Language*) a fost creat pe la începutul anilor ’70 în laboratoarele de cercetare ale firmei IBM pentru a implementa modelul relaţional al lui E.F.Codd. Access SQL conţine intrucţiuni legate de securitate şi acces concurent (cum ar fi COMMIT, GRANT sau LOCK) sau instrucţiuni DDL pentru definirea datelor şi, pe de altă parte, include instrucţiunea TRANSFORM şi declaraţia PARAMETERS.
Access vă dă posibilitatea de a folosi instrucţiuni SQL în multe situaţii. Astfel, sursa de date pentru formulare, rapoarte sau controale poate fi o instrucţiune SELECT în locul unei interogări. Astfel, baza de date va conţine mai puţine obiecte, dar aceasta în detrimentul vitezei cu care sunt regăsite datele.
De asemenea, în Access unele tipuri de interogări pot fi create numai cu ajutorul instrucţiunilorSQL: interogări de tip Union, de definire a datelor sau pentru comunicarea cu baze de date externe (ODBC).
Instrucţiunile SQL pot fi introduse în două moduri: în fereastra *SQL View* (pe care o deschideţi din modul *Design View* al unei interogări alegând comanda *View* | *SQL*) sau în cadrul codului aplicaţiilor.
Atunci când scrieţi instrucţiunea SQL este bine (dar nu obligatoriu) să respectaţi convenţia conform căreia cuvintele cheie sunt scrise cu litere mari, iar restul elementelor cu litere mici. Tot pentru claritate, este bine să plasaţi fiecare clauză a unei instrucţiuni pe câte o linie separată. Fiecare instrucţiune în Access se termină cu punct şi virgulă (;). Access va procesa totuşi instrucţiunea chiar şi în lipsa acestui terminator.

3.1. Instrucţiunea SELECT

Instrucţiunea SELECT este cea mai importantă şi mai folosită instrucţiune SQL. Cu ajutorul ei puteţi regăsi informaţia stocată în baza de date. Sintaxa instrucţiunii SELECT este următoarea:

```
SELECT lista_de coloane
FROM lista_de_tabele
[WHERE criterii]
[ORDER BY lista_coloane]
```

Orice instrucţiune SQL trebuie să includă clauzele SELECT şi FROM. Clauzele WHERE şi ORDER BY sunt opţionale.

3.1.1. Clauza SELECT

Se foloseşte pentru a preciza care coloane vor fi incluse în tabela de rezultate. (In fereastra QBE realizaţi acest lucru introducând coloanele în grilă şi validând caseta Show.) Ca şi în fereastra QBE, folosind asteriscul (*) veţi include în tabela de rezultate toate coloanele unei tabele. Sintaxa clauzei SELECT este:

```
SELECT   {* | expresial   [AS alias] [, expresia2 [AS alias2] [,…] ] }
```

Expresiile pot fi nume de coloane, coloane calculate sau funcţii agregat. Numele de coloane care conţin caractere nealfanumerice sau spaţii să fie incluse între paranteze drepte (a nu se face confuzie cu convenţia de notaţie pentru sintaxă). Puteţi specifica un nume alternativ pentru o coloană sau expresie (ca şi în grila QBE) adăugând “`AS alias`” după numele coloanei sau după expresie.

Exemplu: pentru a returna NrMatricol şi NumeStudent concatenate şi PrenumeStudent (cu un spaţiu între ele) cu titlul NumePrenume:

```
SELECT NrMatricol,   NumeSt  & PrenumeSt  As  [NumePrenume]
```

Dacă interogarea se va baza pe mai multe tabele şi rezultatele vor include o coloană care are acelaşi nume în cel puţin două dintre aceste tabele, va trebui să specificaţi tabela din care provine coloana astfel: Tabela.coloana

Exemplu: pentru a selecta coloana IdCurs din tabela Curs, scrieţi: `SELECT Curs.IdCurs`

3.1.2. Clauza FROM

In clauza FROM specificaţi tabelele şi/sau interogările din care doriţi să selectaţi datele. Dacă interogarea se bazează pe mai multe tabele, trebuie să specificaţi şi tipul de asociere. Sintaxa este următoarea:

```
FROM tabela_sau_interogare  [AS alias]
```

Ex. Interogarea care va selecta toate coloanele şi toate liniile din tabela Curs:

```
SELECT *
FROM Curs;
```

Ca şi în cazul coloanelor şi expresiilor, puteţi stabili aliasuri şi pentru ambele. (acest lucru e util mai ales când aveţi de-a face cu o asociere self-join).

Ex. Următoarea interogare va returna datele din coloanele Nume şi Catedra din tabela Profesor:

```
SELECT Nume, Catedra
FROM Profesor;
```

3.1.3. Clauza WHERE

Permite să impuneţi criterii pentru a filtra înregistrările returnate de o interogare. Ea corespunde liniilor Criteria şi Or din grila QBE. Coloanele la care faceţi referire în clauza WHERE nu trebuie neapărat să fie incluse şi în lista de coloane a clauzei SELECT. (In grila QBE devalidaţi caseta *Show* pentru a nu include în rezultatele interogării coloanelor asupra cărora impuneţi restricţii). Sintaxa clauzei WHERE este următoarea:

```
WHERE expresia1  [AND |OR  expresia2  [,…]]
```

Ex. Dacă vom dori să vedem profesorii şi catedrele lor pentru care IdTitlu=4, interogarea va fi următoarea:

```
SELECT Nume, Catedra
FROM Profesor
WHERE IdTitlu=4;
```

Expresiile din clauza WHERE au acelaşi format cu cele din grila QBE. *Exemple:*

```
WHERE Catedra = "Informatica"
WHERE Nota Between 5 And  8
WHERE [Data Angajarii] > DateAdd ("yyyy", -5, Date)
```

Ultima expresie verifică dacă data angajării este mai recentă de cinci ani.

Observaţii:

- Şirurile de caractere din clauza WHERE trebuie să fie incluse ori între ghilimele, ori între apostrofuri.
- Datele trebuie să fie incluse între caractere diez (#).
  Exemplu: `WHERE [Data Angajarii] > #10/23/1995#`
- Folosiţi întotdeauna cuvântul cheie LIKE atunci când specificaţi şablon pentru şiruri de caractere.
  Exemplu: `WHERE NumeSt  LIKE  "p*"`

3.1.4. Clauza ORDER BY

Se sortează înregistrările returnate de interogare în funcţie de una sau mai multe coloane. Prin cuvinte cheie ASC şi DESC specificaţi dacă sortarea va fi făcută în sens crescător sau descrescător. Cauza ORDER BY corespunde liniei Sort din grila QBE. Ca şi acolo, precedenţa, e de la stânga spre dreapta. Ca şi în cazul clauzei WHERE, coloanele după care se face sortarea nu trebuie neapărat să fie incluse în lista clauzei SELECT. Coloanele specificate în

clauza ORDER BY pot fi doar numerice, de tip text sau date/time. Sintaxa este următoarea:

```
ORDER BY coloana1  [{ASC|DESC}] [, coloana2 [{ASC|DESC}] [,…]]
```

Exemplu: Pt. lista studenţilor grupei 112, sortaţi alfabetic în funcţie de nume şi de prenume, folosiţi interogarea:

```
SELECT *
FROM Student
WHERE Grupa = 112
ORDER BY NumeSt, PrenumeSt;
```

3.1.5. Asocieri între tabele

In urma procesului de normalizare a tabelelor bazei de date, veţi dori, de cele mai multe ori să creaţi interogări care să regăsească datele din mai multe tabele. Astfel, pentru ca rezultatele să fie cele dorite, va trebui să creaţi asocieri după una sau mai multe coloane. Altfel, veţi obţine un produs cartezian, lucru nedorit de cele mai multe ori (de exemplu, dacă aveţi două tabele cu câte 25 de linii fiecare, produsul lor cartezian va fi o tabelă cu 625 de linii).
In Access SQL puteţi crea asocieri între tabele atât în clauza WHERE, cât şi în clauza FROM a unei interogări.
Sintaxa pentru folosirea asocierilor în clauza WHERE este următoarea:

```
SELECT lista_de_coloane
FROM Tabel1, tabel2
WHERE tabela1.coloana1=tabela2.coloana2;
```

Această sintaxă nu ne dă psibilitatea creării unei asocieri outer join între cele două tabele. Iată care este sintaxa în cazul folosirii asocierilor în clauza FROM:

```
SELECT lista_de_coloane
FROM tabela1 {INNER |LEFT [OUTER} |RIGHT [OUTHER]} JOIN tabela2
ON tabela1.coloana1=tabela2.coloana2;
```

Vă recomandăm să folosiţi această variantă, ea fiind mai performantă. In plus, când creaţi în grila QBE o interogare bazată pe o asociere între două tabele, Access va genera o instrucţiune SQL.

Exemplu: Pentru a regăsi datele din coloanele Nume, Catedra şi Titlu din tabelele Profesor şi Titlu pentru profesori angajaţi după data de 1 Ian 1990, puteţi folosi oricare dintre următoarele interogări:

```
SELECT, Nume, Titlu, Catedra
FROM Profesor, Titlu
WHERE Profesor.IdTitlu=Titlu.IdTitlu AND
[Data Angajarii]>#1/1/90#;
```

sau

```
SELECT Nume, Titlu, Catedra
FROM Profesor INNER JOIN Titlu
ON Profesor.IdTitlu = Titlu.IdTitlu
WHERE [DataAngajarii]>#1/1/90#;
```

Asocieri multiple

Ca şi atunci când folosiţi fereastra QBE, puteţi crea instrucţiuni SELECT bazate pe asocieri între mai multe tabele. Sintaxa simplificată a clauzei FROM, în acest caz, este:

```
FROM (tabela1 JOIN tabela2 ON conditional) JOIN tabela 3
ON conditia2) JOIN …)
```

Pentru simplificare, în sintaxa prezentată mai sus am omis să specificăm tipurile de asociere. Trebuie folosite unul dintre cuvintele cheie INNER, LEFT sau RIGHT.

Nu contează ordinea în care scrieţi asocierile în clauza FROM, deoarece motorul Jet Engine va alege ordinea optimă de efctuare a asocierilor. Nu acelaşi lucru se poate spune atunci când combinaţi asocieri cu unele outer. Pentru acest caz există reguli stricte pe care trebuie să le respectaţi:

1. Tabela din care nu se iau în considerare toate înregistrările în cazul unei asocieri outer nu poate participa şi la o asociere inner.
2. Tabela din care nu se iau în considerare toate înregistrările în cazul unei asocieri outer nu poate participa cu acelaşi statut la o altă asociere outer.

Dacă nu respectaţi aceste reguli, atunci când folosiţi asocieri multiple veţi primi un mesaj de eroare şi interogarea nu va putea rula.

Ex. Dorim să vedem numărul matricol al studenţilor, cursurile opţionale la care aceştia s-au înscris şi profesorii care predau cursurile respective, cu menţiunea că vrem ca tabela de rezultate să conţină toate cursurile, indiferent dacă la ele s-a înscris sau nu vreun student. Pentru aceasta, am fi tentaţi să scriem următoarea interogare:

```
SELECT NrMatricol, Denumire, Nume
FROM ((Curs_Student INNER JOIN Curs_Prof
ON Curs_Student.IdCurs = Curs_Prof.IdCurs)
INNER JOIN Profesor
ON Curs_Prof.IdProf = Profesor.IdProf)
RIGHT JOIN Curs ON Curs_Student.IdCurs = Curs.IdCurs;
```

Din păcate, interogarea de mai sus nu va putea rula, deoarece nu respectă regula numărul 1. Pentru a rezolva această problemă vom crea două interogări separate:

1. Prima, să o numim Intermediara, va face legătura între tabelele Curs_Student, Curs_Profesor şi Profesor, folosind două asocieri inner:

```
SELECT NrMatricol, Curs_Prof.IdCurs, Nume
FROM ((Curs_Student INNER JOIN Curs_Prof
ON Curs_Student.IdCurs = Curs_Prof.IdCurs)
INNER JOIN Profesor
ON Curs_Prof.IdProf = Profesor.IdProf);
```

2. Cea de-a doua va combina rezultatele interogării Intermediara cu tabela Curs, folosind o asociere de tip right outer:

```
SELECT NrMatricol, Denumire, Nume
FROM Intermediara RIGHT JOIN Curs
ON Intermediara.IdCurs = Curs.IdCurs;
```

Auto-asocieri

Auto-asocierile sunt utile atunci când avem de-a face cu o relaţie recursivă (între o tabelă şi ea însăşi).

Ca şi în fereastra QBE, pentru a crea o asociere între o tabelă şi ea însăşi trebuie să folosiţi un alias. Spre exemplu, în tabela Profesor există coloana IdCoordonator ce păstrează identificatorul coordonatorului fiecărui profesor (dacă acesta există).

Exemplu: am creat interogarea Asociat, ce regăsea numele fiecărui profesor şi al coordonatorului său, dacă acesta există. Iată care este instrucţiunea SQL corespunzătoare acestei interogări:

```
SELECT Profesor_1.Nume, Profesor.Nume
FROM Profesor AS Profesor_1 LEFT JOIN Profesor ON
Profesor_1.IdCoordonator = Profesor.IdProf;
```

3.1.6. Predicatele ALL, DISTINCTROW şi DISTINCT

Pentru a rezolva problemele legate de apariţia înregistrărilor duplicate, puteţi folosi în cadrul clauzei SELECT, înaintea listei de coloane, unul dintre predicatele: ALL, DISTINCTROW şi DISTINCT. Sintaxa este următoarea:

```
SELECT [{ALL|DISTINCTROW |}] lista_de_coloane
```

ALL este predicatul implicit şi este folosit pentru a returna toate înregistrările ce întrunesc criteriile specificate. In grila QBE, daţi în acest scop câmpurile Unique Values şi Unique Records din pagina de proprietăţi a interogării valoarea No.
Dacă apare DISTINCT, Access va elimina din tabela de rezultate înregistrările duplicate create pe baza listei de coloane din clauza SELECT. Dezavantajele folosirii predicatului DISTINCT sunt încetinirea execuţiei interogării şi faptul că datele din tabela de rezultate nu pot fi modificate. Echivalent, în grila QBE daţi proprietăţii Unique Values valoarea Yes.
Prin folosirea predicatului DISTINCTROW, Access va elimina din tabela de rezultate toate înregistrările duplicat pe baza tuturor coloanelor din tabele. In acest scop, în grila QBE daţi proprietăţii Unique Records valoarea Yes. Un lucru important este acela că datele din tabela de rezultate a unei interogări ce foloseşte predicatul DISTINCTROW vor putea fi actualizate.

3.1.7. Predicatul TOP

Predicatul TOP este folosit pentru a returna primele n sau primele n% înregistrări dintr-o tabelă de rezultate. In acest scop, folosiţi în grila QBE proprietatea *Top Values.*
Este logic şi recomandabil să folosiţi predicatul TOP numai împreună cu clauza ORDER BY. Astfel, veţi obţine primele înregistrări pe baza strategiei optime alese de motorul Jet Engine (deci, nici măcar aleator, în adevăratul sens al cuvântului).

Notă: Valorile Null sunt considerate de predicatul TOP a fi cele mai mici în ordinea alfabetică, cronologică sau numerică.

Predicatul TOP poate fi folosit singur sau împreună cu cuvântul cheie PERCENT. De asemenea, el poate fi combinat cu predicatele ALL, DISTINCT sau DISTINCTROW.
Sintaxa este următoarea:

```
SELECT [ {ALL |DISTINCTROW | DISTINCT}]  [TOP n [PERCENT] ]
Lista _de_coloane
```

Spre exemplu, pentru a vedea primii 15% studenţi, în ordinea notelor obţinute la cursul opţional “engleză”, vom folosi interogarea:

```
SELECT TOP 15 PERCENT Student.NrMatricol, NumeSt& " "&PrenumeSt AS Nume, Nota
From Student INNER  JOIN Curs_Stud
OL Student.NrMatricol = Curs_Stud.NrMatricol
ORDER BY Nota DESC;
```

Predicatul TOP este procesat după ce au fost aplicate criteriile, asocierile, sortările şi agregările.

Notă: Dacă, de exemplu, ultima notă cu care studenţii se înscriu în primii 15% ar fi 8, iar numărul studenţilor cu această notă sau cu o notă mai mare depăşeşte maximul impus de 15% ei tot vor fi impuşi în rezultatele interogării.

3.1.8. Declaraţia WITH OWNERACCESS OPTION

Prin această declaraţie, puteţi da utilizatorilor interogării create de dumneavoastră dreptul de a o rula, chiar dacă ei nu au drepturi asupra tabelelor pe care se bazează interogarea. Echivalent, în grila QBE, daţi proprietăţii Run Permissions valoarea Owner’s. Omiterea declaraţiei corespunde valorii User’s.

Sintaxa este următoarea:

```
SELECT lista_de_coloane
FROM Lista_de_tabele
[WHERE criterii ]
```

```
[ORDER BY lista_coloane]
[WITH OWNERACCESS OPTION]
```

Declaraţia WITH OWNERACCESS OPTION funcţionează numai pentru interogări salvate. Folosită într-o instrucţiune SQL introdusă direct în câmpul proprietăţii Record Source a unui formular, nu va avea nici un efect.

3.2. Agregarea datelor

Despre funcţiile agregat şi utilizarea lor am vorbit pe larg în capitolul III, în contextul creeării interogărilor cu ajutorul grilei QBE.
In Access SQL puteţi construi trei tipuri de interogări agregat:

- Interogări simple, bazate pe o instrucţiune SELECT fără clauza GROUP BY;
- Interogări simple, bazate pe o instrucţiune SELECT cu clauza GROUP BY;
- Interogări de tip Crosstab, create cu ajutorul instrucţiunii TRANSFORM.

3.2.1. Interogări agregat fără clauza GROUP BY

Pentru a crea statistici asupra tuturor înregistrărilor ce satisfac criteriile din clauza WHERE, introduceţi în clauza SELECT numai funcţia agregat. Să presupunem că dorim să aflăm numărul total de profesori din tablela Profesori, data angajării profesorului cu vechimea cea mai mare şi data angajării profesorului cu vechimea cea mai mică. Pentru aceasta vom folosi următoarea interogare agregat:

```
SELECT Count (*) AS TotalProf, Min ([DataAngajarii]) AS [Prima Angajare], Max
([DataAngajarii]) AS [Ultima Angajare]
FROM  Profesor;
```

3.2.2. Interogări agregat cu clauza GROUP BY

Clauza GROUP BY este folosită pentru a defini grupuri de înregistrări pentru care să se aplice o funcţie de agregare. Sintaxa unei instrucţiuni SELECT ce conţine clauza GROUP BY este următoarea:

```
SELECT lista_de_coloane
FROM lista_de_tabele
[WHERE criterii]
[GROUP BY grupuri]
[HAVING proprietati]
[ORDER BY lista_coloane]
```

Sintaxa clauzei GROUP BY este:

```
GROUP BY expresia1 [, expresia2 [,…]]
```

Expresiile ce apar în clauza GROUP BY pot conţine nume de coloane, coloane calculate sau constante. Coloanele din clauza SELECT a unei interogări ce conţine clauza GROUP BY trebuie sa fie ori argumentele unei funcţii agregat, ori trebuie să figureze în clauza GROUP BY.

Exemplu: următoarea interogare va calcula pentru fiecare student media obţinută la cursurile opţionale:

```
SELECT NrMatricol, Avg(Nota) AS Media
FROM Curs_Stud
GROUP BY NrMatricol;
```

Dacă în clauza GROUP BY sunt specificate mai multe coloane, grupuri vor fi definite de la stânga spre dreapta, ca în cazul clauzei ORDER BY. Clauza GROUP BY ordonează automat valorile în sens crescător. Dacă doriţi ca grupurile să fie sortate descrescător, folosiţi clauza ORDER BY cu cuvântul cheie DESC.

Exemplu: Dacă dorim să vedem pentru fiecare catedră numărul de profesori din fiecare categorie salarială, astfel încât rezultatele să fie sortate crescător după catedre şi descrescător după salarii, folosim:

```
SELECT Catedra, Salariu, Count(*) AS TotProf
FROM Profesor INNER JOIN Titlu
ON Profesor.IdTitlu=Titlu.IdTitlu
GROUP BY Catedra, Salariu
ORDER BY catedra, Salariu DESC;
```

Sortarea se poate face şi după coloanele agregat. Clauza GROUP BY poate conţine până la 10 câmpuri. Nu adăugaţi câmpuri inutile în clauza GROUP BY, deoarece aceasta va duce la încetinirea execuţiei interogării.

Clauza HAVING

O interogare agregat poate conţine atât o clauză WHERE, cât şi o clauză HAVING, amândouă fiind folosite pentru a stabili criterii pentru filtrarea datelor. Diferenţa este aceea că orice criteriu din clauza WHERE este aplicat datelor înaintea grupării (agregării) lor, în timp ce criteriile din clauza HAVING sunt aplicate după operaţia de agregare (deci grupurilor). Astfel, filtrarea se bazează pe informaţiile statistice calculate pentru fiecare grup.
Sintaxa clauzei HAVING este asemănătoare cu cea a clauzei WHERE.

```
HAVING expresia1 [{AND|OR} expresia2 [,…]]
```

Exemplu. Dacă dorim să calculăm, pentru fiecare student, media notelor mai mari decât 4 şi să returnăm numai acele înregistrări pentru care această medie este 7, vom scrie:

```
SELECT NrMatricol, Avg(Nota) AS Media
FROM Curs_Stud
WHERE Nota > 4
GROUP BY NrMatricol
HAVING Avg(Nota) > 7;
```

3.2.3.Interogări de tip Crosstab

Pentru a crea totalurile într-un format tabelar, Access SQL vă pune la dispoziţie instrucţiunea TRANSFORM, a cărei sintaxă este următoarea:

```
TRANSFORM funcţie_agregat
Instructuine_select
PIVOT camp_pivot [IN (valoare1 [, valoare2 [,…]])];
```

Funcţia agregat va da valorile fiecărei cellule a tabelului. Câmpul pivot este cel care va da numele coloanelor tabelului. Valorile clauzei opţionale IN pot fi folosite pentru specificarea (fixarea) numelor de coloane ce vor apărea în tabel.

Transformarea unei interogări de tip Group By într-o interogare de tip Crosstab

Nefiind o instrucţiune SQL standard, instrucţiunea TRANSFORM poate părea greu de construit. De aceea, vom porni de la o instrucţiune SELECT ce conţine clauza GROUP BY şi o vom transforma într-o instrucţiune TRANSFORM. Pentru a putea face asta, instrucţiunea SELECT nu trebuie să aibă o clauză HAVING, iar în clauza GROUP BY trebuie să existe cel puţin două câmpuri. In plus, câmpul folosit pentru a da numele coloanelor nu trebuie să aibă mai mult de 254 de valori distincte.

Exemplu: Vom porni acum de la următoarea interogare agregat simplă, pe care o vom transforma în interogare de tip crosstab:

```
SELECT Catedra, Titlu, Count(*)
FROM Profesor INNER JOIN Titlu
ON Profesor.IdTitlu=Titlu.IdTitlu
GROUP BY Catedra, Titlu;
```

Paşii necesari transformării:

1. Incadraţi precedenta instrucţiune SELECT între cuvintele cheie TRANSFORM şi PIVOT, ca mai jos:

```
TRANSFORM
SELECT Catedra, Titlu, Count (*)
FROM Profesor INNER JOIN Titlu
ON Profesor.IdTitlu=Titlu.IdTitlu
GROUP BY Catedra, Titlu
PIVOT;
```

2. Treceţi funcţia agregat (care va da valorile celulelor tabelului) în clauza TRANSFORM:

```
TRANSFORM Count(*)
SELECT Catedra, Titlu
FROM Profesor INNER JOIN Titlu
ON Profesor.IdTitlu=Titlu.IdTitlu
GROUP BY Catedra, Titlu
PIVOT;
```

3. Mutaţi câmpul ce va da numele coloanelor tabelului din clauza GROUP BY în clauza PIVOT. Stergeţi acest camp şi din clauza SELECT:

```
TRANSFORM Count(*)
SELECT Catedra
FROM Profesor INNER JOIN Titlu
ON Profesor.IdTitlu=Titlu.IdTitlu
GROUP BY Catedra
PIVOT  Titlu;
```

Notă: Pentru a formata datele produse de o astfel de interogare trebuie să folosiţi grila QBE.

Calcularea totalurilor pe linii

Prin adăugarea unei funcţii agregat în clauza SELECT puteţi introduce în tabelul cu rezultatele unei interogări crosstab o coloană care să păstreze totalurile de linii.

Exemplu: Se va returna şi o coloană numită TotalProf, cu număr total de profesori ai fiecărei catedre:

```
TRANSFORM Count(*)
SELECT Catedra, Titlu, Count (*) AS TotalProf
FROM Profesor INNER JOIN Titlu
ON Profesor.IdTitlu=Titlu.IdTitlu
GROUP BY Catedra
       PIVOT Titlu;
```

Puteţi adăuga mai multe astfel de coloane pentru a calcula tot felul de totaluri, incluzând în clauza SELECT funcţiile agregat necesare.

Clauza IN

Clauza IN este utilă când doriţi să specificaţi ordinea coloanelor în tabelul cu rezultatele interogării, să includeţi sau să excludeţi coloanele din acest tabel. Sintaxa este următoarea:

```
PIVOT coloana_pivot [IN (valoarea1 [, valoarea2 [,…]])]
```

Exemplu: Dacă în tabelul cu rezultate dorim să sortăm coloanele în ordinea importanţei titlurilor şi totodată, să excludem coloana corespunzătoare preparatorilor, clauzele PIVOT şi IN vor fi:

```
PIVOT Titlu IN ("asistent", "lector", "lector dr", "conferenţiar dr", "profesor dr")
```