

INTEROGAREA BAZELOR DE DATE. LIMBAJUL SQL

Sumar

- Prezentarea generală a limbajului SQL
- Crearea tabelor și declararea restricțiilor
- Comenzi de actualizare a tabelor
- Structura de bază a frazei select
- Joncțiuni
- Funcții-agregat: COUNT, SUM, AVG, MAX, MIN
- Sub-consultări. operatorul IN
- Reuniune, intersecție, diferență
- Gruparea tuplurilor. Clauzele GROUP BY și HAVING

6.1 PREZENTAREA GENERALĂ A LIMBAJULUI SQL

După două capitole de luptă surdă cu bazele de date, a devenit aproape evident că folosim bazele de date pentru că avem memoria prea scurtă. Deoarece suntem o lume așezată pe un morman de hârtii & hârțoage, ne este cu neputință să reconstituim ceea ce am făcut adineori, dară-mi-te ieri, săptămâna trecută sau acum un an sau cinci. Necazul e că, de cele mai multe ori, trebuie să știm nu numai ce-am făcut noi, dar ce-au făcut și colegii și partenerii de afaceri. Dacă la oameni mai putem trece cu vederea, în cazul bazelor de date încrederea este elementul cheie (primară, străină...). Așa încât și în capitolul 4 și în capitolul 5 am fost foarte interesați să aflăm cum definim cât mai multe restricții, astfel să diminuăm riscul preluării în baza de date a unor informații eronate. Apoi am văzut cum inserăm, modificăm și ștergem date într-o bază. Tangențial am abordat și câteva modalități de a obține informații de o manieră asistată, grafică.

Chiar dacă am încercat să păstrăm discuția la un nivel general, exemplificările au fost făcute pe calapodul ACCESS-ului. Cei care au experiență în ACCESS și vor să treacă, forțați de împrejurări, pe un SGBD mai performant, gen PostgreSQL, Oracle etc. (trecere care echivalează cu trecerea de la amatori la semi sau chiar profesioniști de-a binelea) vor fi bruscați de diferențele de interfață, opțiuni etc.

Pentru atenuarea șocurilor trecerilor de la un SGBD la altul, specialiștii bazelor de date s-au gândit la un limbaj universal dedicat creării tabelor, definirii restricțiilor, creării utilizatorilor și grupurilor de utilizatori, definirii drepturilor fiecărui utilizator/grup la obiecte din bază, actualizării înregistrărilor din tabele (inserare, modificare, ștergere) și, mai ales, extragerii și prelucrării datelor din bază. Acest limbaj a fost standardizat încă din 1986 (în SUA) și 1989 (la nivel mondial – ISO) și se numește SQL.

Haideți să revenim la interogarea FACTURI_DUPA_20IUNIE2005 din figura 5.40 (a cărei execuție produce rezultatul din figura 5.39). Dacă din modul proiectare (**Design**) alegem opțiunea SQL View, o să ne trezim cu o fereastră în care apare din senin o comandă cel puțin ciudată:

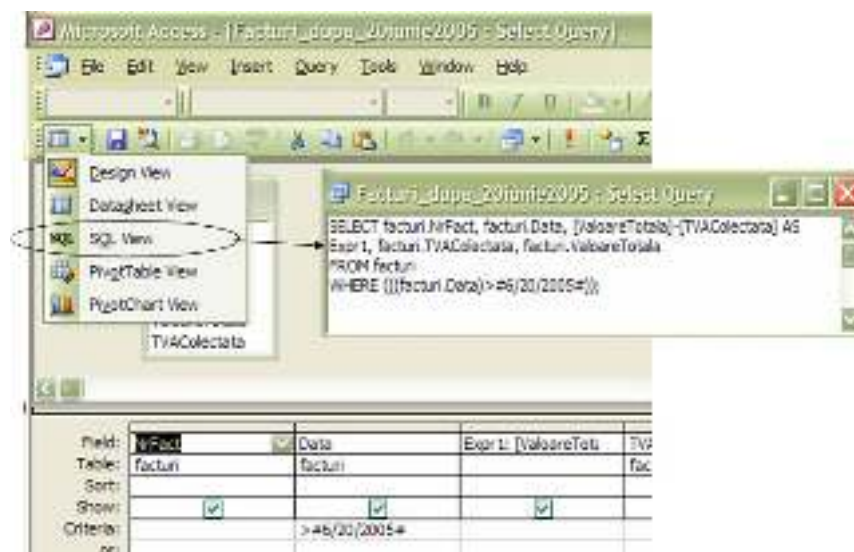


Figura 6. 1. O interogare SQL de-a gata

```
SELECT facturi.NrFact, facturi.Data, [ValoareTotala]-[TVAColectata] AS Expr1,  
       facturi.TVAColectata, facturi.ValoareTotala  
FROM facturi  
WHERE (((facturi.Data)>#6/20/2005#));
```

Ei bine, acest SELECT este chiar o comandă SQL. Dar s-o luăm la pas temeinic. Un mobil determinant al succesului bazelor de date relaționale l-a constituit, fără nici o îndoială, SQL. De la începuturile bazelor de date, s-a pus problema elaborării unui limbaj universal special dedicat bazelor de date, limbaj care să permită, în egală măsură, definirea relațiilor (tabelelor), declararea restricțiilor, modificarea datelor din tabele, precum și extragerea informațiilor din cele mai diverse din datele existente în bază. În 1970, E.F.Codd sugera "adoptarea unui model relațional pentru organizarea datelor [...] care să permită punerea la punct a unui sub-limbaj universal pentru gestiunea acestora, sub-limbaj care să fie, în fapt, o formă aplicată de calcul asupra predicatelor".

De atunci până în prezent au fost propuse numeroase limbaje pentru lucrul cu bazele de date, denumite, în general, *limbaje de interogare*. Dintre cele teoretice, cele mai cunoscute sunt algebra relațională și calculul relațional, iar dintre cele practice (comerciale) Quel, QBE și SQL. De departe, cel mai important este SQL, fundamentat de algebra relațională.

După mulți autori, momentul decisiv în nașterea SQL ca limbaj îl constituie lansarea proiectului System /R de către firma IBM, eveniment ce a avut loc în 1974. Tot în 1974 Chamberlin și Boyce au publicat o lucrare în care este prezentată forma unui limbaj structurat de interogare, "botezat" SEQUEL (Structured English as QUery Language). În 1975 Chamberlin, Boyce, King și Hammer publică un articol în care prezintă sub-limbajul SQUARE, asemănător SEQUEL-ului, dar care utiliza expresii matematice și nu cuvinte din limba engleză. Autorii celor două studii au demonstrat că limbajele SEQUEL și SQUARE sunt complete din punct de vedere relațional.

Un colectiv de autori, condus de Chamberlin, elaborează în 1976 o nouă lucrare în care se face referire la SEQUEL 2, acesta fiind "preluat" ca limbaj de interogare al SGBD-ului System /R al firmei IBM. În 1980 Chamberlin schimbă denumirea SEQUEL în SQL - Structured Query Language (Limbaj Structurat de Interogare), dar și astăzi mulți specialiști pronunță SQL ca pe predecesorul său. Anii următori au înregistrat apariția a o serie întreagă de lucrări care l-au perfecționat, ultimul deceniu consacrându-l ca pe cel mai răspândit limbaj de interogare a BDR, fiind prezent în numeroase "dialecte" specifice tuturor SGBDR-urilor actuale, de la DB2 la Microsoft SQL Server, de la Oracle la FoxPro și ACCESS.

American National Standard Institute publică în 1986 standardul SQL ANSI X3.136-1986. Este un standard care se bazează, într-o mare măsură, pe "dialectul" SQL al produsului DB2 de la IBM. În 1989 are loc revizuirea extinderea acestui standard, "născându-se" SQL-89, care mai este denumit și SQL1. Deși recunoscut ca bază a multor SGBDR-uri comerciale, SQL1 și-a atras numeroase critici. În plus, variantele comercializate de diferiți producători, deși asemănătoare în esență, erau (și sunt) incompatibile la nivel de detaliu. Pentru a umple golurile SQL1, ANSI a elaborat în 1992 "versiunea" SQL2, specificațiile fiind prezentate la un nivel mult mai detaliat (dacă SQL1 se întindea pe numai 100 de pagini, SQL2 a fost publicat în aproape 600). IBM a avut un aport incontestabil la apariția și maturizarea SQL, fiind un producător cu mare influență în "lumea" SGBD-urilor, iar produsul său, DB2, este unul din standardele de facto ale SQL.

Standardul SQL:1999 a fost amânat de câteva ori până la publicarea sa, iar cea mai recentă versiune este SQL:2003. Actualmente, principalele orientări ale SQL vizează transformarea acestuia

într-un limbaj complet, în vederea definirii și gestionării obiectelor complexe și persistente. Aceasta include: generalizare și specializare, moșteniri multiple, polimorfism, încapsulare, tipuri de date definite de utilizator, trigger (declanșatoare) și proceduri stocate, suport pentru sisteme bazate pe gestiunea cunoștințelor, expresii privind interogări recursive și instrumente adecvate de administrare a datelor.

La momentul actual, SQL reprezintă cel mai important limbaj în domeniul bazelor de date, atât prin gama comenzilor și opțiunilor de care dispune, dar mai ales datorită faptului că s-a reușit standardizarea sa și portarea pe toate Sistemele de Gestiune a Bazelor de date semnificative. Cu atât mai mult, cu cât, spre deosebire de majoritatea covârșitoare a altor limbaje, poate fi deprins relativ ușor de neinformaticieni și utilizat pentru chestiuni de mare finețe de către profesioniști. Acest capitol se dorește a fi o prezentare a elementelor esențiale prin care, dată fiind structura unei baze de date relaționale, pot fi formulate interogări (fraze SELECT) prin care se obțin răspunsuri la gamă eterogenă de întrebări. În plus, sunt evocate pe scurt comenzile pentru actualizarea unei tabel (INSERT, UPDATE, DELETE), precum și cele de declarare a structurii bazei de date (CREATE TABLE).

Din punctul de vedere al utilizatorului final, obiectivul principal al SQL constă în a oferi utilizatorului mijloacele necesare formulării unei consultări numai prin descrierea rezultatului dorit, cu ajutorul unei aserțiuni (expresie logică), fără a fi necesară și explicitarea modului efectiv în care se face căutarea în BD. Altfel spus, utilizatorul califică (specifică) rezultatul iar sistemul se ocupă de procedura de căutare.

Deși este referit, în primul rând, ca un limbaj de interogare, SQL este mult mai mult decât un instrument de consultare a bazelor de date, deoarece permite, în egală măsură:

- Definirea datelor
- Consultarea BD
- Manipularea datelor din bază
- Controlul accesului
- Partajarea bazei între mai mulți utilizatori ai acesteia
- Menținerea integrității BD.

Deși toate clasificările îl încadrează la limbaje de generația a IV-a, SQL nu este, totuși, un limbaj de programare propriu-zis, prin comparație cu Basic, Pascal, C, COBOL etc. SQL nu conține (până la SQL3) instrucțiuni/comenzi pentru codificarea structurilor alternative și repetitive, cu atât mai puțin facilitați de lucru cu obiecte vizuale, specifice formularelor de preluare a datelor (căsuțe-text, butoane radio, liste, butoane de comandă etc.). Din acest punct de vedere, poate fi referit ca sub-limbaj orientat pe lucrul cu bazele de date. Comenzile sale pot fi, însă, inserate în programe redactate în limbaje de programare "clasice".

Principalele atuuri ale SQL sunt:

1. Independența de producător, nefiind o tehnologie "proprietary".
2. Portabilitate între diferite sisteme de operare.
3. Standardizare.
4. "Filosofia" sa se bazează pe modelul relațional de organizare a datelor.
5. Este un limbaj de nivel înalt, cu structură ce apropie de limba engleză.
6. Furnizează răspunsuri la numeroase interogări simple, ad-hoc, neprevăzute inițial.
7. Constituie suportul programatic pentru accesul la BD.
8. Permite multiple imagini asupra datelor bazei.
9. Este un limbaj relațional complet.

10. Permite definirea dinamică a datelor, în sensul modificării structurii bazei chiar în timp ce o parte din utilizatori sunt conectați la BD.
11. Constituie un excelent suport pentru implementarea arhitecturilor client-server.

Principalele comenzi ale SQL, care se regăsesc, într-o formă sau alta, în multe dintre SGBDR-urile actuale sunt prezentate în tabelul 6.1.

Tabel 6.1. Clase de comenzi SQL

Comandă	Scop
Pentru manipularea datelor	
SELECT	Extragerea datelor din BD
INSERT	Adăugarea de noi linii într-o tabelă
DELETE	Ștergerea de linii dintr-o tabelă
UPDATE	Modificarea valorilor unor atribute
Pentru definirea bazei de date	
CREATE TABLE	Adăugarea unei noi tabele în BD
DROP TABLE	Ștergerea unei tabele din bază
ALTER TABLE	Modificarea structurii unei tabele
CREATE VIEW	Crearea unei tabele virtuale
DROP VIEW	Ștergerea unei tabele virtuale
Pentru controlul accesului la BD	
GRANT	Acordarea unor drepturi pentru utilizatori
REVOKE	Revocarea unor drepturi pentru utilizatori
Pentru controlul tranzacțiilor	
COMMIT	Marchează sfârșitul unei tranzacții
ROLLBACK	Abandonează tranzacția în curs

6.2 CREAREA TABELELOR ȘI DECLARAREA RESTRICȚIILOR

Limbajul SQL prezintă o serie întreagă de opțiuni care permit crearea tabelelor și modificarea valorilor unor atribute pentru tuplurile unei relații. Mai mult, standardul SQL dispune de opțiuni clare privind specificarea unor restricții legate de cheile primare, cheile străine etc. Baza de date pe care o vom lua în discuție pe parcursul acestui capitol este cea prezentată în capitolul 4. Pentru crearea unei tabele comanda SQL este, natural, **CREATE TABLE** prin care se declară numele tabelei, numele, tipul și lungimea fiecărui atribut, precum și restricțiile. Astfel, primele două tabele din baza de date, **CODPOST_LOC** și **CLIEȚI** pot fi create astfel:

```
CREATE TABLE codPost_loc (
    CodPostal CHAR(6) CONSTRAINT pk_cp PRIMARY KEY,
    Localitate CHAR (35) NOT NULL,
    Judet CHAR(25) NOT NULL );

CREATE TABLE clienti (
    CodClient INTEGER CONSTRAINT pk_cp PRIMARY KEY,
    NumeClient CHAR (30) NOT NULL CONSTRAINT un_numeclient UNIQUE,
    Adresa CHAR(60),
    CodPostal CHAR(6) NOT NULL CONSTRAINT ref_cl_codpost
    REFERENCES codpost_loc (CodPostal) );
```

Se pot observa cu ușurință clauzele **PRIMARY KEY** folosite pentru declararea cheilor primare, **UNIQUE** pentru cheile alternative, **NOT NULL** pentru interzicerea valorilor nule, precum și

REFERENCES pentru declararea cheii străine. În plus, opțiunea CONSTRAINT ne ajută să “botezăm” fiecare restricție după cum dorim.

Spre deosebire de alte SGBD-uri, în ACCESS comenzile nu pot fi introduse direct, ci incluse în module de program. Iată modulul CREARETABELE():

```
Sub CreareTabele()
    Dim dbs As Database
    'Set dbs = OpenDatabase("Z:\ISA_2006_Z\BD\vinzari.mdb")
    Set dbs = CurrentDb
    dbs.Execute "CREATE TABLE codPost_loc " _
        & "(CodPostal CHAR(6) CONSTRAINT pk_cp PRIMARY KEY, " _
        & " Localitate CHAR (35) NOT NULL, " _
        & " Judet CHAR(25) NOT NULL) ;"
    dbs.Execute "CREATE TABLE clienti " _
        & "(CodClient INTEGER CONSTRAINT pk_cp PRIMARY KEY, " _
        & " NumeClient CHAR (30) NOT NULL CONSTRAINT un_numecient UNIQUE, " _
        & " Adresa CHAR(60), " _
        & " CodPostal CHAR(6) NOT NULL CONSTRAINT ref_cl_codpost REFERENCES codpost_loc (CodPostal) ) ;"
    dbs.Execute "CREATE TABLE facturi " _
        & "(NrFact INTEGER CONSTRAINT pk_facturi PRIMARY KEY, " _
        & " CodClient INTEGER NOT NULL CONSTRAINT ref_fact_cl REFERENCES clienti (CodClient), " _
        & " Data DATETIME NOT NULL, " _
        & " ValoareTotala NUMERIC NOT NULL, " _
        & " TVAColectata NUMERIC NOT NULL) ;"
End Sub
```

Deși în capitolul 4 spuneam că structura unei baze de date este constantă, există situații în care trebuie să:

- adăugăm un atribut;
- eliminăm un atribut;
- schimbăm tipul unui atribut;
- modificăm lungimea unui atribut;
- să declarăm o nouă restricție
- să anulăm o restricție în vigoare.

Deoarece baza de date este în uz, nu ne putem permite să ștergem și apoi să recreăm tabelele, pentru că aceasta echivalează cu pierderea iremediabilă a înregistrărilor existente. Așa că este necesară folosirea comenzii ALTER TABLE. Dacă în tabela CLIENȚI se dorește păstrarea și a codului fiscal al fiecărui furnizor, este necesară adăugarea atributului **CodFiscal**, care este un șir de caractere (un număr precedat de litera R, dacă clientul respectiv este plătitor de TVA) de lungime 10 caractere. Comanda utilizată este:

ALTER TABLE CLIENȚI ADD CodFiscal CHAR(10)

În SQL ștergerea unei tabele din baza de date este realizabilă cu ajutorul comenzii DROP TABLE. Iată cum ar putea fi lansate comenzile de ștergere ale celor trei tabele pe care abia le-am creat acum jumătate de pagină:

```
Sub StergeTabele()
    Dim dbs As Database
    Set dbs = OpenDatabase("Z:\ISA_2006_Z\BD\vinzari.mdb")
    dbs.Execute "DROP TABLE facturi ;"
    dbs.Execute "DROP TABLE clienti ;"
    dbs.Execute "DROP TABLE codpost_loc ;"
End Sub
```

Am trecut în revistă până în acest moment principalele clauze ale comenzii CREATE TABLE (și ALTER TABLE) pentru declararea cheilor primare, alternative și străine, și valorilor nenule.

Nemeritat, a fost omisă clauza CHECK prin care putem defini restricții utilizator sub forma regulilor de validare la nivel de atribut sau înregistrare. Astfel, în tabela CLIEŢI valorile atributului **CodClient** trebuie să înceapă de la 1001, iar numele clientului se doreşte a fi scris cu majuscule întotdeauna. Aceste două reguli de validare pot fi definite în SQL atât în momentul creării:

```
CREATE TABLE clienti (
    CodClient INTEGER CONSTRAINT pk_cp PRIMARY KEY
    CONSTRAINT ck_codclient CHECK (CocClient > 1000),
    NumeClient CHAR (30) NOT NULL CONSTRAINT un_numeclient UNIQUE
    CONSTRAINT ck_numeclient CHECK (NumeClient = UPPER(NumeClient)),
    Adresa CHAR(60),
    CodPostal CHAR(6) NOT NULL CONSTRAINT ref_cl_codpost
    REFERENCES codpost_loc (CodPostal) );
```

cât şi ulterior prin ALTER TABLE. Din păcate, ACCESSul nu este prea îngăduitor în această privinţă, clauza CHECK fiind interzisă. Singura modalitate de declarare a regulilor este cea procedurală:

```
Sub Reguli_Attribute()
    Dim dbs As Database, tdf As TableDef, fld As Field
    Set dbs = CurrentDb

    ' CLIENTI.CodClient > 1000
    Set tdf = dbs.TableDefs("clienti")
    Set fld = tdf.Fields("CodClient")
    fld.ValidationRule = "[CodClient] > 1000"
    fld.ValidationText = "Cel mai mic cod de client acceptat este 1001 !"

    ' CLIENTI.NumeClient se scrie numai cu majuscule
    Set tdf = dbs.TableDefs("clienti")
    Set fld = tdf.Fields("NumeClient")
    fld.ValidationRule = "StrComp(UCase([NumeClient]), [NumeClient], 0) = 0"
    fld.ValidationText = "Literale din numele clientului sunt obligatoriu majuscule !"

    ' Prima litera din CLIENTI.Adresa este majuscula. Restul, la alegere !
    Set tdf = dbs.TableDefs("clienti")
    Set fld = tdf.Fields("Adresa")
    fld.ValidationRule = "StrComp(LEFT(UCase([Adresa]),1), LEFT([Adresa],1), 0) = 0"
    fld.ValidationText = "Prima litera din adresa clientului trebuie sa fie majuscula !"

    'Data facturii
    Set tdf = dbs.TableDefs("facturi")
    Set fld = tdf.Fields("Data")
    fld.ValidationRule = "[Data] BETWEEN #1-1-2005# AND #12-31-2010#"
    fld.ValidationText = "Data facturii trebuie sa se incadreze in intervalul 1 ian.2005 - 31 dec.2010 !"

End Sub
```

Despre reguli la nivel de înregistrare, ce să mai vorbim... Astfel, dacă în tabela FACTURI s-ar dori instituirea regulii după care TVA-ul poate fi cel mult egal cu 0,19/1,19 din valoarea totală a fiecărei facturi (prietenii ştiu de ce !), restricția ar putea fi definită la creare astfel:

```
CREATE TABLE facturi (
    NrFact INTEGER CONSTRAINT pk_facturi PRIMARY KEY,
    CodClient INTEGER NOT NULL CONSTRAINT ref_fact_cl REFERENCES clienti (CodClient),
    Data DATE NOT NULL,
    ValoareTotala NUMERIC NOT NULL,
    TVAColectata NUMERIC NOT NULL),
    CONSTRAINT ck_tva_valtot CHECK (TVAColectata <= ValoareTotala * 0.19 / 1.19) );
```

ACCESS-ul este, după cum vă imaginați, impasibil la grațiile (de tip CHECK) ale SQL-ului, așa că singura soluție e tot cea procedurală:

```
Sub Regula_inregistrare()
```

```

Dim dbs As Database, tdf As TableDef
Set dbs = CurrentDb
Set tdf = dbs.TableDefs("facturi")
tdf.ValidationRule = "[TVAColectata] <= [ValoareTotala] * 0.19 / 1.19"
tdf.ValidationText = "TVA poate fi maxim 0,19/1,19 din valoarea totala a facturii !"
End Sub

```

6.3 COMENZI DE ACTUALIZARE A TABELELOR

SQL prezintă comenzi specifice pentru modificarea conținutului unei tabele, înțelegând prin aceasta trei acțiuni prin care se actualizează baza: a) adăugarea de noi linii la cele existente într-o tabelă; b) ștergerea unor linii, c) modificarea valorii unui atribut.

6.3.1 Adăugare

Să presupunem că, la un moment dat, întreprinderea vinde produse și firmei RODEX SRL care are sediul pe strada Sapienței, nr.44 bis, în localitatea Iași. Acest nou client trebuie "introdus" în baza de date, operațiune care în SQL, se realizează prin comanda:

INSERT INTO clienti VALUES (1009, 'RODEX SRL', 'Sapienței, 44 bis', '706600')

Fraza **INSERT** de mai sus poate fi scrisă și sub forma:

INSERT INTO clienti (CodClient, NumeClient, Adresa, CodPostal)
VALUES (5009, 'RODEX SRL', 'Sapienței 44 bis', 706600)

După cum se observă, după numele tablei (CLIENTI) au fost enumerate toate atributele pentru care se introduc valori prin clauza **VALUES**. Dacă nu s-ar fi cunoscut adresa clientului RODEX, atunci fraza **INSERT** ar fi avut una din formele:

INSERT INTO clienti (CodClient, NumeClient, Adresa, CodPostal)
VALUES (5009, "RODEX SRL", NULL, '6600') sau

INSERT INTO clienti (CodClient, NumeClient, CodPostal) VALUES (5009, 'RODEX SRL', '6600')

În noua linie a tablei CLIENTI valoarea atributului **Adresa** va fi NULL.

Conținutul celor trei tabele din finalul capitolului 4 a fost realizat în ACCESS prin modulul următor în care comanda INSERT este argumentul unei comenzi *DoCmd.RunSQL*:

Sub Inserturi()

```

' CODPOST_LOC
DoCmd.RunSQL ("INSERT INTO codPost_loc VALUES ('706600', 'Iasi', 'Iasi') ;")
DoCmd.RunSQL ("INSERT INTO codPost_loc VALUES ('706610', 'Iasi', 'Iasi') ;")
DoCmd.RunSQL ("INSERT INTO codPost_loc VALUES ('705300', 'Focsani', 'Vrancea') ;")
DoCmd.RunSQL ("INSERT INTO codPost_loc VALUES ('705725', 'Pascani', 'Iasi') ;")
DoCmd.RunSQL ("INSERT INTO codPost_loc VALUES ('706750', 'Tg.Frumos', 'Iasi') ;")

' CLIENTI
DoCmd.RunSQL ("INSERT INTO clienti VALUES (1001, 'TEXTILA SA', 'Bld. Copou, 87', '706600') ;")
DoCmd.RunSQL ("INSERT INTO clienti VALUES (1002, 'MODERN SRL', 'Bld. Gării, 22', '705300') ;")
DoCmd.RunSQL ("INSERT INTO clienti VALUES (1003, 'OCCO SRL', NULL, '706610') ;")
DoCmd.RunSQL ("INSERT INTO clienti VALUES (1004, 'FILATURA SA', 'Bld. Unirii, 145', '705300') ;")
DoCmd.RunSQL ("INSERT INTO clienti VALUES (1005, 'INTEGRATA SA', 'I.V.Viteazu, 115', '705725') ;")
DoCmd.RunSQL ("INSERT INTO clienti VALUES (1006, 'AMI SRL', 'Galapiului, 72', '706750') ;")
DoCmd.RunSQL ("INSERT INTO clienti VALUES (1007, 'AXON SRL', 'Silvestru, 2', '706610') ;")
DoCmd.RunSQL ("INSERT INTO clienti VALUES (1008, 'ALFA SRL', 'Prosperității, 15', '705725') ;")

' FACTURI
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111111, 1003, #6-17-2005#, 17000, 0) ;")

```



```

DoCmd.RunSQL ("INSERT INTO facturi VALUES (111112, 1001, #6-17-2005#, 2850, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111113, 1004, #6-18-2005#, 5850, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111114, 1003, #6-18-2005#, 2850, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111115, 1008, #6-18-2005#, 35700, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111116, 1008, #6-19-2005#, 8700, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111117, 1006, #6-20-2005#, 1100, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111118, 1007, #6-23-2005#, 15000, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111119, 1005, #6-24-2005#, 4720, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111120, 1003, #6-24-2005#, 3000, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111121, 1001, #6-24-2005#, 4250, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111122, 1007, #6-24-2005#, 8750, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111123, 1006, #6-25-2005#, 66000, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111124, 1004, #6-25-2005#, 38600, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111125, 1003, #6-30-2005#, 1280, 0);")
DoCmd.RunSQL ("INSERT INTO facturi VALUES (111126, 1002, #6-01-2005#, 54250, 0);")

```

End Sub

După cum se observă, deliberat TVA colectată este declarată zero la toate liniile inserate în FACTURI. Aceasta pentru a avea motiv de modificare (UPDATE) – vezi paragraful pe peste linia curentă.

6.3.2 Modificarea valorilor unor atribute

Pentru modificarea valorilor unuia sau multor atribute dintr-o tabelă, comanda utilizată este **UPDATE** care are formatul general: **UPDATE tabelă SET atribut = expresie WHERE predicat**

Ca rezultat, vor fi modificate valorile atributului specificat, noile valori ale acestuia fiind cele care rezultă în urma evaluării expresiei; modificarea se va produce pe toate liniile tabelii care îndeplinesc condiția specificată în predicat. Astfel comanda:

UPDATE facturi SET TVACollectata = INT(ValoareTotala * 19 / 1.19) / 100

va stabili valoarea TVA colectată pentru toate facturile (lipsește clauza WHERE, deci vor fi afectate toate liniile din tabela FACTURI) pe 0.19/1.19 din valoarea totală a fiecărei facturi. Funcția INT extrage doar partea întreagă dintr-un număr real (se elimină, deci, partea fracționară), iar prin împărțirea rezultatului la 100 ne asigurăm că TVA va avea două poziții la partea fracționară.

În ACCESS această actualizare se poate face în două moduri. Grafic, putem crea o interogare de tip pe calapodul clasic al construirii machetelor (prezentat în paragraful 5.4), după “aducerea” tabelii FACTURI în machetă, selectând din meniul **Query Type** a opțiunii *Update Query* – vezi partea stângă a figurii 6.2. Partea dreaptă a figurii prezintă noua formă a machetei, în care apare linia **Update To** și dispar **Sort** și **Show**. La rubrica **Update To** se introduce expresia de calcul a valorilor atributului **TVACollectată**. Cea de-a doua modalitate este cea procedurală, comanda prin care se lansează UPDATE-rile fiind, ca și în cazul inserării, *DoCmd.RunSQL*.

Sub Updateuri()

```

DoCmd.RunSQL ("UPDATE facturi SET TVACollectata = INT(ValoareTotala * 19 / 1.19) / 100 ;")
DoCmd.RunSQL ("UPDATE facturi SET TVACollectata = 0 WHERE NrFact = 111117 ;")
DoCmd.RunSQL ("UPDATE facturi SET TVACollectata = INT(ValoareTotala * 9 / 1.09) / 100 " & _
" WHERE NrFact IN (111118, 111122);")

```

End Sub

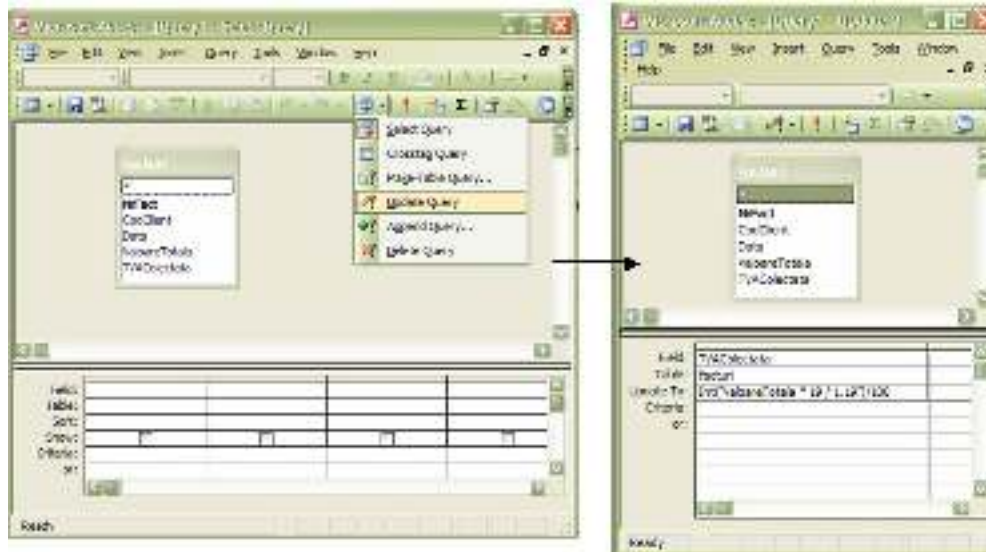


Figura 6.2. Interogare de modificare

Modulul de mai sus, specific ACCESSului conține, pe lângă comanda UPDATE explicată, alte două, prin care facturi 111117 îi este declarat procentul zero de TVA, iar a doua indică faptul că toate produsele/serviciile din facturile 111118 și 111122 au procentul de TVA de 9%. Deși cei mai mulți utilizatori ar opta pentru varianta grafică, va asigurăm că varianta procedurală este mai productivă atunci când sunt necesare actualizări repetate, așa cum este cazul de mai sus.

6.3.3 Ștergeri

Operațiunea de eliminarea a una sau mai multe linii dintr-o tabelă, pe baza unui predicat, se realizează în SQL prin comanda **DELETE** care are sintaxa: **DELETE FROM nume-tabelă WHERE predicat**

Dacă am dori să eliminăm din tabela CLIENȚI linia aferentă clientului MODERN SRL (cod 1002), comanda ar fi:

DELETE FROM clienti WHERE CodClient = 1002

În ACCESS, similar interogărilor pentru modificare pot fi create și interogări pentru ștergere, în care se specifică criteriul pe care trebuie să-l satisfacă liniile pentru a fi șterse din tabela indicată (fiind în criză de spațiu, nu vom mai prezenta nici o figură în acest scop). Prin program, dacă am dori eliminarea tuturor înregistrărilor din tabelele bazei, apoi re-adăugarea și re-modificarea lor, ne-am putea servi de blocul următor:

```
Sub Stergeri()
    DoCmd.RunSQL ("DELETE FROM facturi ;")
    DoCmd.RunSQL ("DELETE FROM clienti ;")
    DoCmd.RunSQL ("DELETE FROM codPost_loc ;")
    Call Inserturi
    Call Updateuri
End Sub
```

6.4 STRUCTURA DE BAZĂ A FRAZEI SELECT

Fără îndoială, cea mai gustată parte din SQL este cea legată de interogarea bazei, adică de obținerea de informații din cele mai diverse, prin prelucrări, grupări etc. Ceea ce a fost prezentat în

paragraful 5.4 este doar o părticică din ceea ce poate fi “stors” dintr-o bază de date. Așadar, în SQL o interogare se formulează printr-o frază **SELECT**. Aceasta prezintă trei clauze principale: **SELECT**, **FROM** și **WHERE**.

- **SELECT** este utilizată pentru desemnarea listei de atribute (coloanele) din rezultat;
- **FROM** este cea care permite enumerarea relațiilor din care vor fi extrase informațiile aferente consultării;
- prin **WHERE** se desemnează condiția (predicatul), simplă sau complexă, pe care trebuie să le îndeplinească liniile tabelor (enumerate în clauza **FROM**) pentru a fi extrase în rezultat.

La modul general (și simplist) o consultare simplă în SQL poate fi prezentată astfel: **SELECT C1, C2, ..., Cn FROM R1, R2, ..., Rm WHERE P**.

Execuția unei fraze **SELECT** se concretizează în obținerea unui rezultat de forma unei tabele (relații). Când clauza **WHERE** este omisă, se consideră implicit că predicatul **P** are valoarea logică “adevărat”. Dacă în locul coloanelor **C1, C2, ... Cn** apare simbolul “*”, în tabela-rezultat vor fi incluse toate coloanele (atributele) din toate relațiile specificate în clauza **FROM**. De asemenea, în tabela-rezultat, nu este obligatoriu ca atributele să prezinte nume identic cu cel din tabela enumerată în clauza **FROM**. Schimbarea numelui se realizează prin opțiunea **AS**.

Uneori, rezultatul interogării “încalcă” poruncile modelului relațional. Conform restricției de entitate, într-o relație nu pot exista două linii identice. Or, în SQL, rezultatul unei consultări poate conține două sau mai multe tupluri identice. Pentru eliminarea liniilor identice este necesară utilizarea opțiunii **DISTINCT**: **SELECT DISTINCT C1, C2, ..., Cn FROM R1, R2, ..., Rm WHERE P**

Am început acest capitol prin a arăta fraza **SELECT** care se ascunde în spatele unei consultări “grafice” **ACCESS**. Fraza:

```
SELECT facturi.NrFact, facturi.Data, [ValoareTotala]-[TVACollectata] AS ValFaraTVA,  
facturi.TVACollectata, facturi.ValoareTotala  
FROM facturi WHERE (((facturi.Data)>#6/20/2005#));
```

conține și un câmp calculat – **ValFărăTVA**, al cărui nume se specifică prin clauza **AS**, și ale cărui valori se determină prin relația **ValoareTotală – TVACollectată**. Predicatul de selecție (clauza **WHERE**) asigură extragerea în rezultat doar a facturilor emise după 20 iunie 2005.

Cum se poate introduce o interogare în **ACCESS** fără a o “desena” cu proiectantul de machete pentru interogări prezentat în paragraful 5.4 ? Mai întâi creăm o interogare foarte simplă – să-i zicem – **INTEROG**. După modelul indicat în stânga figurii 6.3 îi vizualizăm definiția folosind opțiunea **SQL View**, iar apoi în fereastra care apare înlocuim fraza **SELECT** cu cea care ne interesează (frază care nu are, probabil, nici o legătură cu definiția actuală a interogării), după care se apasă butonul **Run** (semnul mirării).

Iar dacă vrem să facem același lucru prin program, folosim modulul următor:

```
Sub interogareBETWEEN()  
Dim consultare As QueryDef  
Set consultare = CurrentDb.QueryDefs("interog")  
consultare.SQL = "SELECT * FROM facturi WHERE nrfact BETWEEN 111120 AND 111124 ;"  
DoCmd.OpenQuery ("interog")  
End Sub
```

Modulul *interogareBETWEEN()* declară (prin **Dim**) obiectul *consultare* ca fiind definiția unei interogări (**QueryDef**), preia (prin comanda **Set**) în *consultare* fraza **SELECT** care constituie definiția

interogării create anterior – *interog* –, modifică definiția acesteia prin linia *consultare.SQL* = “*SELECT ...*” și, în final, execută noua variantă a interogării, rezultatul fiind similar variantei din figura 6.3.

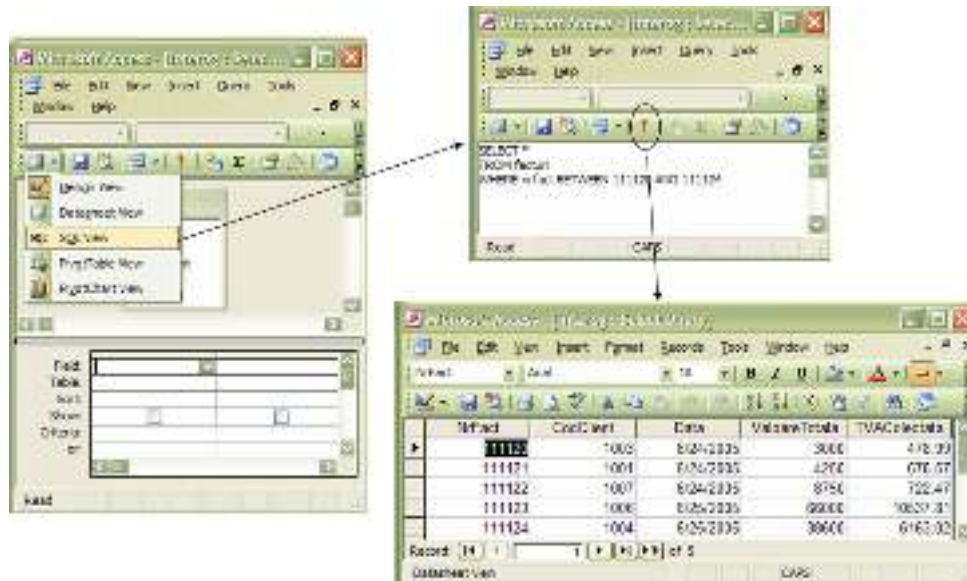


Figura 6.3. Schimbarea definiției unei interogări și re-execuția sa

Firește, noile definiții pot fi salvate sau se poate renunța la ele, mai ales atunci lucrăm cu module ce pot apelate ori de câte ori este nevoie.

Operatorul BETWEEN

Fraza SELECT de mai sus conține operatorul BETWEEN care va extrage din tabelă numai facturile cu numerele cuprinse între 111120 și 111124. Fără BETWEEN predicatul ar trebui scris WHERE nrfact >= 111120 AND nrfact <= 111124.

Operatorul LIKE

Operatorul LIKE se folosește pentru a compara un atribut de tip șir de caractere (ex. **NumeClient**, **Adresa**, **Localitate**) cu un literal (constantă de tip șir de caractere). Astfel, dacă se dorește obținerea unei tabele-rezultat care să conțină numai clienții ai căror nume începe cu litera M, putem scrie predicatul din clauza **WHERE** sub forma: **NumeClient LIKE "M%"**. Deoarece după litera M apare semnul "%" (sau "*"), se vor extrage din tabela **CLIEŢI** toate tuplurile pentru care valoarea atributului **NumeClient** începe cu litera M, indiferent de lungimea acestuia (ex. MELCRET, MIGAS, MITA, MATSUSHITA etc.). Despre semnul "%" (sau "*") se spune că este un specificator multiplu, joker sau mască. Un alt specificator multiplu utilizat în multe versiuni SQL este liniuța-de-subliniere ("_") sau semnul de întrebare ("?"). Spre deosebire de "%", "_" substituie un singur caracter. Diferența dintre cei doi specificatori multipli este pusă în evidență în continuare. Astfel, dacă interesează care sunt clienții ai căror nume începe cu litera A și sunt societăți cu răspundere limitată (SRL-uri), fraza SELECT care furnizează răspunsul este:

SELECT * FROM CLIEŢI WHERE NumeClient LIKE "A_ SRL%"

(vezi partea stângă a figurii 6.4). Dacă s-ar fi utilizat simbolul "%" de maniera următoare:

SELECT * FROM CLIEŢI WHERE NumeClient LIKE "A%SRL%"

rezultatul ar fi fost cel din partea dreaptă a figurii.

CodClient	NumeClient	Adresa	CodPostal
1006	AMI SRL	Galatului, 72	706750

CodClient	NumeClient	Adresa	CodPostal
1008	ALFA SRL	Prospetului, 15	705725
1006	AMI SRL	Galatului, 72	706750
1007	AXON SRL	Silvestru, 2	706610

Figura 6.4. Folosirea specificatorilor multipli

În concluzie, "_" sau "?" înlocuiesc (substituie) un singur caracter, în timp ce "%" sau "*" înlocuiesc un șir de caractere de lungime variabilă (între 0 și n caractere). Cei doi specificatori multipli pot fi utilizați împreună.

Operatorul IN

Un alt operator util este IN cu formatul general: **expresie1 IN (expresie2, expresie3, ...)**. Rezultatul evaluării unui predicat ce conține acest operator va fi "adevărat" dacă valoarea **expresiei1** este egală cu (cel puțin) una din valorile: **expresie2, expresie3, ...**

Spre exemplu, pentru a afla care sunt clienții din localitățile din județele Iași și Vaslui, fără utilizarea operatorului IN se scrie:

```
SELECT * FROM codpost_loc WHERE Judet = 'Iasi' OR Judet = 'Vaslui'
```

Iar utilizând IN:

```
SELECT * FROM codpost_loc WHERE Judet IN ("Iasi", "Vaslui")
```

Operatorul IS NULL

O valoare nulă este o valoare nedefinită. Este posibil ca la adăugarea unei linii într-o tabelă, valorile unor attribute să fie necunoscute. În aceste cazuri valoarea atributului pentru tuplul respectiv este nulă. Reamintim că, prin definiție, nu se admit valori nule pentru grupul atributelor care constituie cheia primară a relației. Pentru aflarea clienților pentru care nu s-a introdus adresa, se poate scrie:

```
SELECT * FROM clienti WHERE Adresa IS NULL
```

Cum în baza noastră de date, numai clientului OCCO SRL nu-i cunoaștem adresa, rezultatul interogării este cel din figura 6.5 (în ACCESS valorile NULL se afișează ca și cum ar conține spații).

CodClient	NumeClient	Adresa	CodPostal
1003	OCCO SRL		706610

Figura 6.5. Extragerea valorilor NULL

Observații

- Valoarea NULL nu se confundă cu valoarea zero (pentru attributele numerice) sau cu valoarea "spațiu" (pentru attributele de tip șir de caractere)
- Operatorul NULL se utilizează cu IS și nu cu semnul "=". Dacă s-ar utiliza expresia = NULL și nu expresia IS NULL, rezultatul evaluării va fi întotdeauna fals, chiar dacă expresia nu este nulă !

Opțiunile DISTINCT și ORDER BY

Dorim să aflăm județele în care firma are clienți. Este necesară parcurgerea relației CODPOST_LOC, singurul atribut care interesează fiind Judet:

```
SELECT DISTINCT Judet FROM codpost_loc
```

După cum se observă în partea stângă a figurii 6.6, SQL nu elimină dublurile automat, iar dacă se dorește ca în tabela-rezultat o localitate să figureze o singură dată, se utilizează opțiunea DISTINCT (rezultatul în dreapta figurii 6.6):

SELECT DISTINCT Judet FROM codpost_loc



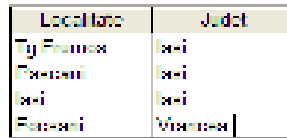
Judet
Iasi
Iasi
Iasi
Iasi
Iasi

Judet
Iasi
Vrancea

Figura 6.6. Fără și cu **DISTINCT**

În continuare vrem să obținem denumirea fiecărei localități și județul în care se află, dar liniile rezultatului trebuie ordonate în funcție de județ și, în cadrul aceluiași județ, în ordinea inversă a localității (de la Z la A), fraza **SELECT** se formulează după cum urmează, rezultatul fiind prezentat în figura 6.7.

**SELECT DISTINCT Localitate, Judet FROM codpost_loc
ORDER BY Judet ASC, Localitate DESC**



Localitate	Judet
Ty-Frumos	Iasi
Fetea	Iasi
Iasi	Iasi
Fetea	Vrancea

Figura 6.7. Clauza **ORDER BY**

Opțiunile **ASCENDING** (crescător) și **DESCENDING** (descrescător) indică deci modul în care se face ordonarea tuplurilor tabelului-rezultat al interogării. Prioritatea de ordonare este stabilită prin ordinea atributelor specificate în **ORDER BY**: ordonarea "principală" se face în funcție de valorile primului atribut specificat; în cadrul grupelor de tupluri pentru care valoarea primului atribut este identică, ordinea se stabilește după valoarea celui de-al doilea atribut specificat s.a.m.d. Dacă în **ORDER BY** lipsesc opțiunile **ASCENDING/DESCENDING**, ordonarea se face crescător.

6.5 JONCTIUNI

După cum afirmam și în paragraful 5.4 majoritatea informațiilor obținute dintr-o bază de date necesită "răsfoirea" simultană a două sau mai multe tabele. Interogarea *Fac_dupa20iunie2005v2* din figura 5.41 folosește trei tabele. Folosind opțiunea **SQLView** obținem o frază **SELECT** cu totul remarcabilă:

```
SELECT facturi.NrFact, facturi.Data, clienti.NumeClient, codPost_loc.Localitate,  
      [ValoareTotala]-[TVAColectata] AS Expr1, facturi.TVAColectata, facturi.ValoareTotala, *  
FROM (codPost_loc INNER JOIN clienti ON codPost_loc.CodPostal = clienti.CodPostal)  
      INNER JOIN facturi ON clienti.CodClient = facturi.CodClient  
WHERE (((facturi.NrFact)>#6/20/2005#) AND ((codPost_loc.Localitate)="Iasi"));
```

Clauza **FROM** vine acum în centrul atenției prin apariția clauzei **INNER JOIN**. Iată cum stau lucrurile: deoarece în lista pe care vrem să obținem se găsesc atribute plasate în cele trei tabele, în clauza **FROM** trebuie enumerate cele trei **tabele**; în fapt, după cum am văzut în capitolul 4, cele trei **tabele sunt legate prin restricții referențiale**, atributele de legătura fiind cheile străine – cheile primare. Astfel, legătura dintre **FACTURI** și **CLIENTI** se poate realiza prin intermediul atributului **CodClient** care este cheie primară în **CLIENTI** (tabela părinte) și cheie străină în **FACTURI** (tabela copil).

Legătura dintre aceste două tabele care prezintă un câmp comun se numește joncțiune și se simbolizează în SQL prin INNER JOIN:

```
SELECT *
FROM facturi INNER JOIN clienti ON facturi.CodClient=clienti.CodClient
```

Fără a intra în prea multe detalii teoretice, reținem că, ori de câte ori informațiile necesare și condițiile pe care trebuie să le îndeplinească acele informații privesc atribute aflate în tabele diferite, trebuie făcută joncțiunea acestor tabele. Când tabele nu pot fi joncționate direct, trebuie aduse “cu forța” în clauza FROM și tabelele care să completeze “lanțul”.

Ne interesează, spre exemplu, numărul și data facturilor emise clienților din județul Iași. Numărul și data facturilor se găsesc în tabela FACTURI (atributele **NrFact** și **Data**), însă pentru denumirea județului există un atribut (**Judet**) în tabela CODPOST_LOC. Cele două tabele nu pot fi joncționate direct, așa încât atragem în interogare și cea de-a treia tabelă – CLIEŢI:

```
SELECT NrFact, Data
FROM (facturi INNER JOIN clienti ON facturi.codclient=clienti.codclient)
     INNER JOIN codpost_loc ON codpost_loc.codpostal=clienti.codpostal
WHERE judet='Iasi'
ORDER BY NrFact
```

Scrisă sub formă de modul ACCESS – *InterogareJONCTIUNE1()* – și lansată prin “apăsarea butonului **Run**, fraza SELECT obține rezultatul din figura 6.8.

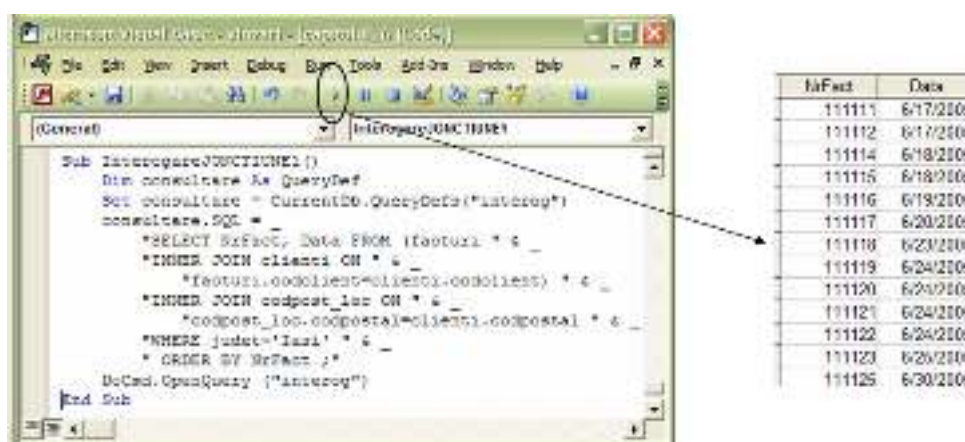


Figura 6.8. Un modul ACCESS cu frază SELECT ce joncționează cele trei tabele (plus rezultatul)

Lăsam să se înțeleagă, la un moment dat, că în SQL pot fi formulate interogări mult mai complexe decât se poate realiza cu ajutorul machetei din paragraful 5.4. Haideți să luăm o asemenea problemă, ce-i drept nu atât de complicată precum amenințăm: *Care sunt facturile emise în aceeași zi ca și factura 111113 ?* Dificultatea ține de faptul că cerința este formulată indirect, adică vrem să aflăm facturile emise într-o zi (**Data**), dar noi nu știm data etalon, ci factura-etalon.

Problema propusă poate fi rezolvată relativ ușor folosind o subconsultare, după cum va fi prezentat într-un paragraf viitor. Până una-alta, soluția pe care o avem în acest moment la îndemână se bazează pe *autojoncțiune*. Autojoncțiunea înseamnă joncțiunea unei tabele (FACTURI) cu ea-însăși, practic, joncțiunea a două instanțe ale unei aceleiași tabele. Pentru a joncțiune cele două instanțe trebuie să aibă pseudonime (aliasuri) diferite, în cazul nostru F1 și F2. Întrucât ne interesează facturi emise în *aceeași zi* cu 111113, autojoncțiunea se face după atributul **Data**:

```
SELECT *
```


**FROM facturi F1 INNER JOIN facturi F2 ON F1.data=F2.data
WHERE F2.NrFact = 111113**

Îată rezultatul – vezi figura 6.9. Rezultatul conține 10 coloane, cinci din prima instanță a tabelului FACTURI (F1) și cinci din a doua instanță (F2). ACCESSul e destul de inspirat să scrie înaintea fiecărui atribut din ce instanță provine.

F1.NrFact	F1.CodClient	F1.Data	F1.ValoareTotala	F1 TVAConsumibil	F2.NrFact	F2.CodClient	F2.Data	F2.ValoareTotala	F2 TVAConsumibil
111113	1004	6/10/2005	9990	994.02	111113	1004	6/10/2005	1000	994.02
111114	1003	6/10/2005	2890	465.04	111113	1004	6/10/2005	1000	994.02
111115	1008	6/10/2005	35700	5700	111113	1004	6/10/2005	1000	994.02

Figura 6.9. Facturile emise în aceeași zi ca și 111113

Toate liniile rezultatului respectă condiția de joncțiune - F1.data=F2.data. Jumătatea din dreapta a coloanelor se referă strict la factura etalon – 111113, iar cea din stânga la facturile din aceeași zi cu 111113, inclusiv factura-etalon. Pentru a răspunde punctual la problemă, precizăm attributele (coloanele) care ne interesează și eliminăm din rezultat factura 111113:

**SELECT F1.NrFact, F1.Data
FROM facturi F1 INNER JOIN facturi F2 ON F1.data=F2.data
WHERE F2.NrFact = 111113 AND F1.NrFact <> 111113**

6.6 FUNCȚII-AGREGAT: COUNT, SUM, AVG, MAX, MIN

Cu funcțiile agregat facem un prim pas spre analiza datelor din bază. Li se spune funcții agregat deoarece, în absența grupării (clauza GROUP BY – vezi ultimul paragraf din acest capitol) rezultatul unei asemenea funcții are forma unei tabeli cu o singură linie.

Funcția COUNT

Contorizează valorile unei coloane, altfel spus, numără, într-o relație, câte valori diferite de NULL are coloana specificată. Dacă în locul unui atribut apare semnul asterisc (*) se numără liniile rezultatului. Astfel, dacă vrem să aflăm câți clienți are firma vom folosi interogarea (rezultatul său se află în stânga figurii 6.10).

SELECT COUNT (CodClient) AS Nr_Clienti1, COUNT (*) AS Nr_Clienti2 FROM clienti

Nr_Clienti1	Nr_Clienti2
8	8

NrClienti1	NrClienti2
16	16

Figura 6.10. Niște COUNT-uri

Teoretic, în tabela CLIENȚI pot apărea și clienți cărora încă nu li s-a trimis încă nici o factură. Dacă vrem să aflăm răspunsul la întrebarea: *La câți clienți s-au trimis facturi ?*, am fi tentați să folosim interogarea:

**SELECT COUNT (*) AS NrClienti1, COUNT(CodClient) AS NrClienti2
FROM clienti INNER JOIN facturi ON clienti.CodClient=facturi.CodClient**

care, însă, ne va furniza un răspuns eronat (vezi partea dreaptă a figurii 6.10). În produsele care respectă recomandările standardelor SQL, rezultatul corect poate fi însă obținut prin utilizarea clauzei **DISTINCT** astfel:

SELECT COUNT (DISTINCT CodClient) FROM facturi

În ACCESS această opțiune nu este acceptată, așa încât mai așteptăm până la paragraful dedicat subconsultărilor.

Funcția SUM

Funcția **SUM** calculează suma valorilor unei coloane. Pentru a afla suma valorilor totale ale facturilor, soluția este cât se poate de simplă:

```
SELECT SUM (ValoareTotala) AS Total_ValoriFacturi FROM facturi
```

iar totalul valorilor pentru facturile trimise clientului AXON SRL este obținut astfel:

```
SELECT SUM (ValoareTotala) AS Total_Fact_AXON  
FROM facturi INNER JOIN clienti ON facturi.CodClient = clienti.CodClient  
WHERE NumeClient = 'AXON SRL'
```

Funcțiile MAX și MIN

Determină valorile maxime, respectiv minime ale unei coloane în cadrul unei tabele. Valorile cea mai mică și cea mai mare ale unei facturi se află astfel:

```
SELECT MIN(ValoareTotala), MAX(ValoareTotala) FROM facturi
```

Atenție ! La întrebarea *Care este factura emisă cu cea mai mare valoare ?* nu putem răspunde deocamdată. Varianta următoare nu este corectă: **SELECT NrFactura, MAX(ValoareTotala) FROM facturi**

La execuția acestei interogări se afișează un mesaj de eroare, soluția problemei fiind posibilă ceva mai pe finalul capitoului.

6.7 SUB-CONSULTĂRI. OPERATORUL IN

O altă facilită deosebit de importantă a limbajului SQL o constituie posibilitatea includerii (îmbricării) a două sau mai multe fraze **SELECT**, astfel încât pot fi formulate interogări cu mare grad de complexitate. Operatorul cel mai des întâlnit este **IN**. Astfel, revenind la o problemă anterioară - *Care sunt facturile emise în aceeași zi în care a fost întocmită factura 111113 ?* – în locul epuizantei auto-joncțiuni putem recurge la subconsultare:

```
SELECT * FROM facturi WHERE NrFact <> 111113 AND Data IN  
(SELECT Data FROM facturi WHERE NrFact=111113)
```

Sub-consultarea **SELECT Data FROM facturi WHERE NrFact = 111113** are ca rezultat o tabelă alcătuită dintr-o singură coloană (**Data**) și o singură linie ce conține valoarea atributului **Data** pentru factura 111113, ca în stânga figurii 6.11. Clauza **WHERE Data IN** determină căutarea în tabela **FACTURI** a tuturor tuplurilor (liniilor) care au valoarea atributului **Data** egală cu una din valorile tuplurilor (în cazul nostru, egală cu valoarea tuplului) din tabela obținută prin "sub-consultare" (în cazul nostru, tabela din stânga figurii). Cu alte cuvinte, în acest caz **WHERE Data IN** va selecta toate facturile pentru care data emiterii este 18/06/2005 – partea dreaptă a figurii 6.11.

Data	NrFact	CodClient	Data	ValoareTotala	TVAColectata
6/18/2005	111114	1003	6/18/2005	2850	455.04
	111115	1008	6/18/2005	35700	5700

Figura 6.11 Rezultatul sub-consultării (stânga) și al interogării

Dacă s-ar schimba condiția de selecție, în sensul că ne-ar interesa facturile emise în alte zile decât cea în care a fost întocmită factura 111113 operatorul de sub-consultare va fi **NOT IN**:

```
SELECT * FROM facturi WHERE Data NOT IN
      (SELECT Data FROM facturi WHERE NrFact = 111113)
```

Ca să încheiem paragraful cu o interogare mai prezentabilă, ne interesează clienții cărora li s-au trimis facturi întocmite în aceeași zi cu factura 111113:

```
SELECT DISTINCT NumeClient FROM clienti WHERE CodClient IN
      (SELECT CodClient FROM facturi WHERE Data IN
        (SELECT Data FROM facturi WHERE NrFact = 111113))
```

Am ilustrat modul în care pot fi imbricate (înlănțuite, incluse) trei fraze SELECT.

6.8 REUNIUNE, INTERSECȚIE, DIFERENȚĂ

Operatorul pentru *reuniune* este deci **UNION**. De remarcat că, la reuniune, SQL elimină automat dublurile, deci nu este necesară utilizarea clauzei **DISTINCT**. Operatorul **UNION** este prezent în toate SGBD-urile importante. Dacă două relații, R1 și R2 sunt *uni-compatibile*, adică au același număr de attribute care corespund sintactic (adică primul atribut din R1 este de același tip cu primul atribut din R2), se poate scrie **SELECT * FROM R1 UNION SELECT * FROM R2**.

În ceea ce ne privește, vrem să aflăm cum se numesc clienții cărora le-am emis facturi pe 23 sau pe 24 iunie 2005, avem la dispoziție două variante, una bazată pe operatorul logic OR:

```
SELECT DISTINCT NumeClient FROM clienti WHERE CodClient IN
      (SELECT DISTINCT CodClient FROM facturi
        WHERE Data = #6/23/2005# OR Data = #6/24/2005#)
```

și o altă bazată pe reuniune:

```
SELECT DISTINCT NumeClient FROM clienti WHERE CodClient IN
      (SELECT DISTINCT CodClient FROM facturi WHERE Data = #6/23/2005# )
UNION
SELECT DISTINCT NumeClient FROM clienti WHERE CodClient IN
      (SELECT DISTINCT CodClient FROM facturi WHERE Data = #6/24/2005#)
```

Pentru realizarea *intersecției* a două tabele unicompatibile, R1 și R2, în standardele SQL a fost introdus operatorul **INTERSECT**: **SELECT * FROM R1 INTERSECT SELECT * FROM R2**. Dacă în produsele profesionale, precum DB2 (IBM) sau Oracle operatorul este prezent, în schimb multe din cele din categoria “ușoară”, precum Visual Fox Pro și ACCESS **INTERSECT** rămâne un deziderat, funcționalitatea sa realizându-se prin subconsultări (operatorul **IN**) sau, uneori, prin joncțiune. Astfel, dacă dorim să aflăm cum se numesc clienții cărora le-am emis facturi și pe 23 și pe 24 iunie 2005, soluția cea mai la îndemână în ACCESS este:

```
SELECT DISTINCT NumeClient FROM clienti WHERE CodClient IN
      (SELECT DISTINCT CodClient FROM facturi WHERE Data = #6/23/2005# )
      AND CodClient IN
      (SELECT DISTINCT CodClient FROM facturi WHERE Data = #6/24/2005#)
```

Diferența dintre tabelele R1 și R2 (unicompatibile) se realizează utilizând operatorul **MINUS** sau **EXCEPT**, însă implementările sunt similare operatorului **INTERSECT**. Astfel, pentru a obține clienții cărora le-am emis facturi și pe 24, dar nu și pe 24 iunie 2005, soluția ACCESS este:

```
SELECT DISTINCT NumeClient FROM clienti WHERE CodClient IN
      (SELECT DISTINCT CodClient FROM facturi WHERE Data = #6/24/2005# )
      AND CodClient NOT IN
      (SELECT DISTINCT CodClient FROM facturi WHERE Data = #6/23/2005#)
```

6.9 GRUPAREA TUPLURILOR. CLAUZELE GROUP BY ȘI HAVING

În paragraful 5.4 făceam cunoștință cu o primă interogare în care era necesară gruparea liniilor. Ne interesa valoarea zilnică a vânzărilor într-o anumită perioadă, iar cadrul construirii interogării (*VinzariPeZile_interval_la_alegere*) lua forma din figura 5.42. Folosind din nou opțiunea **SQL View** să vedem fraza **SELECT** ce se ascunde în spatele machetei:

```
SELECT facturi.Data, Sum(facturi.TVAColectata) AS SumOfTVAColectata,
       Sum(facturi.ValoareTotala) AS SumOfValoareTotala,
       Sum([ValoareTotala]-[TVAColectata]) AS ValFaraTVA
FROM facturi
GROUP BY facturi.Data
HAVING (((facturi.Data) Between [Data initiala:] And [Data finala:]))
ORDER BY facturi.Data;
```

SQL permite utilizarea clauzei **GROUP BY** pentru a forma grupe (grupuri) de tupluri ale unei relații, pe baza valorilor comune ale unei coloane. În frazele **SELECT** formulate până în acest paragraf, prin intermediul clauzei **WHERE** au fost selectate tupluri din diferite tabele. Prin asocierea unei clauze **HAVING** la o clauză **GROUP BY** este posibilă selectarea anumitor grupe de tupluri ce îndeplinesc un criteriu.

Clauza GROUP BY

Rezultatul unei fraze **SELECT** ce conține această clauză este o tabelă care va fi obținută prin regruparea tuturor liniilor din tabelele enumerate în **FROM**, care prezintă o aceeași valoare pentru o coloană sau un grup de coloane. Formatul general este:

SELECT coloană 1, coloană 2, ..., coloană m FROM tabelă GROUP BY coloană-de-regrupare

Simplificăm problema, dorind o listă cu totalul zilnic al valorii facturilor emise. Fraza este ceva mai simplă:

SELECT Data, SUM (ValoareTotala) AS Total_Zilnic FROM facturi GROUP BY Data

Tabela-rezultat va avea un număr de linii egal cu numărul de date calendaristice distincte din tabela **FACTURI**. Pentru toate facturile aferente unei zile se va calcula suma valorilor, datorită utilizării funcției **SUM(ValoareTotala)**. Succesiunea pașilor este următoarea:

1. Se ordonează liniile tabelii **FACTURI** în funcție de valoarea atributului **Data** - figura 6.12.

NrFact	CndClient	Data	ValoareTotala	TVAColectata
111125	1032	6/1/2005	6420.00	8691.76
111112	1031	6/1/2005	2000.00	459.04
111111	1029	6/1/2005	17000.00	2714.20
111113	1034	6/10/2005	1000.00	934.00
111114	1033	6/10/2005	2000.00	459.04
111115	1030	6/10/2005	35700.00	5700.00
111116	1030	6/19/2005	8700.00	1389.07
111117	1036	6/20/2005	1100.00	0.00
111118	1037	6/23/2005	11000.00	1230.00
111123	1033	6/24/2005	3000.00	470.89
111121	1031	6/24/2005	4200.00	670.57
111122	1037	6/24/2005	8700.00	722.47
111119	1035	6/24/2005	4700.00	709.61
111123	1036	6/25/2005	6000.00	10637.81
111124	1034	6/25/2005	38600.00	6160.02
111122	1033	6/30/2005	1200.00	204.06

Figura 6.12. Pasul 1 al grupării

2. Se formează câte un grup pentru fiecare valoare distinctă a atributului **Data** - vezi figura 6.13.

	NrFact	CodClient	Data	ValoareTotala	TVAcolectata
grup 1	111126	1002	6/1/2005	54250.00	8661.76
grup 2	111112	1001	6/17/2005	2850.00	455.04
	111111	1003	6/17/2005	17000.00	2714.28
	111113	1004	6/18/2005	5850.00	934.03
grup 3	111114	1003	6/18/2005	2850.00	455.04
	111115	1008	6/18/2005	35700.00	5700.00
grup 4	111116	1008	6/19/2005	8700.00	1389.07
grup 5	111117	1006	6/20/2005	1100.00	0.00
grup 6	111118	1007	6/23/2005	15000.00	1238.53
	111120	1003	6/24/2005	3000.00	478.99
grup 7	111121	1001	6/24/2005	4250.00	678.57
	111122	1007	6/24/2005	8750.00	722.47
	111119	1005	6/24/2005	4720.00	753.61
grup 8	111123	1006	6/25/2005	66000.00	10537.81
	111124	1004	6/25/2005	38600.00	6163.02
grup 9	111125	1003	6/30/2005	1280.00	204.36

Figura 6.13. Al doilea pas al grupării

3. Pentru fiecare din cele nouă grupuri se calculează suma valorilor atributului **ValoareTotala**. Tabela rezultat va avea nouă linii, ca în figura 6.14.

Data	VinzariTotala
6/1/2005	54250
6/17/2005	19850
6/18/2005	44350
6/19/2005	8700
6/20/2005	1100
6/23/2005	16100
6/24/2005	20720
6/25/2005	104600
6/30/2005	1280

Figura 6.14. Rezultatul final al grupării

Dacă interesează numărul facturilor emise pentru fiecare client, răspunsul poate fi obținut prin interogarea:

```
SELECT NumeClient, COUNT(NrFact) AS NrFacturi_pe_Client
FROM facturi INNER JOIN clienti ON facturi.CodClient = clienti.CodClient
GROUP BY NumeClient
```

Până la standardul SQL:1999 și publicarea *Amendamentului OLAP* la acest standard, în SQL nu puteau fi calculate, prin **GROUP BY**, subtotaluri pe mai multe niveluri. Pentru aceasta este necesară scrierea de programe în SGBD-ul respectiv.

Clauza HAVING

Clauza **HAVING** permite introducerea unor restricții care sunt aplicate grupurilor de tupluri, deci nu tuplurilor "individuale", așa cum "face" clauza **WHERE**. Din tabela rezultat sunt eliminate toate grupurile care nu satisfac condiția specificată. Clauza **HAVING** "lucrează" împreună cu o clauză **GROUP BY**, fiind practic o clauză **WHERE** aplicată acesteia. Formatul general este:

```
SELECT coloană 1, coloană 2, .... , coloană m FROM tabelă
GROUP BY coloană-de-regrupare HAVING caracteristică-de-grup
```

Pentru facturile emise interesează valoarea zilnică a acestora (în funcție de data la care au fost întocmite), dar numai dacă aceasta (valoarea zilnică) este de mai mare de 40000 lei noi (RON).

```
SELECT Data, SUM(ValoareTotala) AS Vinzari_Zilnice
FROM facturi GROUP BY Data HAVING SUM(ValoareTotala) > 40000
```

La execuția acestei fraze, se parcurg cei trei pași descriși la începutul acestui paragraf, apoi, dintre cele nouă tupluri obținute prin grupare, sunt extrase numai cele care îndeplinesc condiția **SUM(ValoareTotala) > 40000**. Rezultatul final este cel din figura 6.15.

Data	Vinzari_Zilnice
6/1/2005	54250
6/18/2005	44400
6/25/2005	104600

Figura 6.15. Zilele cu vânzări mai mari de 40000 RON

Și acum, o interogare cu adevărat interesantă: *Să se afișeze ziua în care s-au întocmit cele mai multe facturi !* Iată soluția:

```
SELECT Data, COUNT(*) AS nr_facturilor FROM facturi
GROUP BY Data HAVING COUNT(*) >= ALL
(SELECT COUNT(*) FROM facturi GROUP BY Data)
```

Avem de-a face cu o subconsultare al cărei rezultat (stânga figurii 6.16) servește drept termen de comparație al predicatului (condiției) formulat în clauza HAVING.

rezultatul subconsultării	rezultatul consultării dacă nu s-ar fi folosit clauza HAVING	rezultatul consultării cu aplicarea clauzei HAVING
Expr1000	Data nr_facturilor	Data nr_facturilor
1	6/1/2005 1	6/24/2005 4
2	6/17/2005 2	
3	6/18/2005 3	
1	6/19/2005 1	
1	6/20/2005 1	
1	6/23/2005 1	
4	6/24/2005 4	
2	6/25/2005 2	
1	6/30/2005 1	

Figura 6.16. Clauza HAVING cu folosirea unei subconsultări

Cam atât pentru acest capitol, disciplină, semestru (“informatic”) și an (tot “informatic”) ! Vă mulțumim pentru răbdare, și vă asigurăm că cine a ajuns cu cititul (și înțelesul) până în acest punct, are toate șansele unui examen încununat de succes.