LIMBAJUL SQL

Instrucțiunile pentru definirea datelor

În limbajul de interogare SQL, instrucțiunile de definire a datelor sunt următoarele:

- **CREATE DATABASE nume_bază_de_date;**

Prima etapă în administrarea datelor o reprezintă crearea bazei de date. Majoritatea S.G.B.D.-urilor relaționale permit construirea bazei de date printr-o simplă apăsare a butonului mouse-ului. Există însă şi posibilitatea de definire a unei baze de date, folosind această instrucțiune SQL, dar lucrurile devin mult mai greoaie. Sintaxa comenzii nu este standardizată, putând varia în funcție de necesitățile utilizatorului şi de S.G.B.D.-ul folosit. Anumite sisteme, cum este şi ACCESS SQL, nici nu acceptă o astfel de instrucțiune.

La crearea unei baze de date trebuie luate în considerare anumite restricții stabilite de administratorul de sistem şi anume, cele referitoare la nivelul drepturilor de utilizare a instrucțiunilor în sistem şi cele care privesc stabilirea dimensiunilor predefinite pentru baza de date.

- **CREATE TABLE nume_tabelă**
  (câmp1 tip_dată [NOT NULL],
  câmp2 tip_dată [NOT NULL],
  câmp3 tip_dată [NOT NULL]...);

Plecând de la structura unei înregistrări şi de la tipurile de date asociate câmpurilor (definite în cadrul acesteia) utilizatorul poate să creeze o tabelă. Printre cele mai importante tipuri de date folosite amintim: Character, Memo, Number, Integer, Decimal, Logical, Date, OLE Object etc.

Numele tabelei trebuie să fie unic în cadrul bazei de date, neputând fi unul din cuvintele rezervate. Totodată acesta poate avea şi anumite restricţii privind: numărul de caractere din care este format, utilizarea anumitor simboluri, folosirea literelor mari sau mici, natura caracterului de început etc. Aceleaşi cerinţe apar şi pentru numele câmpurilor; în plus există posibilitatea duplicării lor în cadrul bazei de date, dar se păstrează unicitatea în tabelă. Clauza NOT NULL arată că în câmpul respectiv nu se memorează valori de tip NULL.

Exemplu:

Se creează tabela *Vânzări* cu următoarea structură a înregistrării: *număr* (tip numeric), *cod marfă* (tip numeric), *data vânzării* (tip dată calendaristică), *localitatea* (tip caracter). În câmpul data vânzării nu se memorează valori de tip NULL.

```
CREATE TABLE VANZARI
(Nr Number, Cod_m Number, Data_v Date NOT NULL, Localit Char);
```

- **ALTER TABLE nume_tabelă0**
  ADD nume_câmp tip_dată;

Această instrucţiune permite adăugarea unui câmp la o tabelă existentă. Nu este însă posibilă ştergerea unui câmp şi adăugarea (respectiv ştergerea) de câmpuri la nivelul bazei de date în ansamblu.

Exemplu:

Se adăugă tabelei *Personal_Vânzare* un nou câmp numit *Telefon*:

```
ALTER TABLE PERSONAL_VANZARE
ADD Telefon Integer;
```

- **DROP TABLE nume_tabelă;**

Este folosită pentru a şterge complet o tabelă dintr-o bază de date (inclusiv indecşii şi valorile asociate).

Exemplu:

Se şterge tabela Vânzări din baza de date *Aplicaţie*:

```
DROP TABLE VANZARI;
```

- **DROP DATABASE nume_bază_de_date;**

Este utilizată pentru a şterge o bază de date; există însă o multitudine de restricţii stabilite de administratorul sistemului privind această operaţie, mai ales în condiţiile existenţei unei reţele de calculatoare. De remarcat faptul că numeroase versiuni SQL nu includ această instrucţiune, ştergerea făcându-se mai uşor printr-o simplă apăsare pe butonul mouse-lui.

Instrucţiunile de selecţie a datelor

Cereri de interogare simple

Instrucţiunile de selecţie reprezintă una din categoriile cele mai importante ale limbajului de interogare SQL ACCESS. Indiferent dacă sunt simple sau complexe, punctul de plecare îl constituie fraza SELECT, prin care se regăsesc şi se afişează informaţiile dorite de utilizator.

Pentru definirea interogărilor de selecţie simple se utilizează următoarea sintaxă a instrucţiunii SELECT:

- **SELECT [domeniu] listă_selecţie**
  FROM nume_tabelă1, nume_tabelă2,...
  [WHERE criteriul_de_selecţie]
  [ORDER BY câmpuri_criteriu [ASC|DESC]];

- ***Domeniu***

Determină stabilirea modalităţii de manipulare a înregistrărilor din baza de date asupra căreia se efectuează selecţia şi poate fi:

ALL - permite includerea tuturor înregistrărilor ce îndeplinesc condiţiile impuse. Cum frazele SELECT tabelă şi SELECT ALL tabelă au practic acelaşi rezultat, calificativul ALL este destul de rar utilizat.

DISTINCT - are ca efect eliminarea înregistrărilor care conţin duplicate în câmpurile selectate; astfel se va afişa doar o apariţie a datei multiple.

DISTINCTROW - are în vedere înregistrările duplicate în ansamblul lor, nu numai pe cele care au câmpuri duplicate.

- ***Listă_selecţie***

Cuprinde toate câmpurile care vor apărea în tabela cu rezultatele interogării. Câmpurile adăugate în rândul Field din grila Query a machetei grafice QBE, care au marcată caseta de validare Show, sunt aceleaşi cu cele menţionate în lista de selecţie.

- ***Clauza FROM***

Specifică numele tabelei sau tabelelor care vor forma suportul interogării. Dacă în lista_selecţie se includ câmpuri din mai multe tabele, în faţa numelui acestora trebuie precizată tabela din care fac parte. Aşa cum arătam la regulile de sintaxă, pentru separarea numelor de tabele, se utilizează semnul "," (virgulă). Trebuie să precizăm faptul că în cadrul acestei clauze se pot menţiona pe lângă tabele, ca surse de informaţii pentru interogările SQL, şi interogări care au fost deja create.

- ***Clauza WHERE***

Face interogările mai selective, specificând faptul că vor fi afişate numai înregistrările care îndeplinesc criteriul descris. Parametrul criteriul_de_selecţie este o expresie care conţine un operator de tip text (şir) sau numeric, în funcţie de tipul câmpului. Clauza WHERE este opţională şi nu operează cu funcţii totalizatoare.

- ***Clauza ORDER BY***

Utilizată atunci când se doreşte ca rezultatele interogării să fie ordonate în mod crescător (ASC) sau descrescător (DESC). Sortarea este opţională şi se poate realiza după unul sau mai multe câmpuri_criteriu (definite drept chei de sortare). Componenta BY a clauzei nu poate să lipsească atunci când se doreşte sortarea rezultatelor interogării SQL ACCESS !

Exemplu:

Să se afişeze cifra de afaceri şi numărul mediu de salariaţi pentru firmele din Braşov, cu denumirile sortate invers alfabetic. Care va fi cifra de afaceri prognozată pentru fiecare societate, ştiind că procentul de creştere pentru anul în curs este de 10%?.

```
SELECT denumire, CA, nr_mediu_s, [CA]*1.1 AS CA_prognoz
FROM Indicatori
WHERE localit ="Brasov"
ORDER BY denumire DESC;
```

În cadrul frazelor SELECT apar frecvent operatorii (AND, OR, NOT, IN, BETWEEN, LIKE) care vor fi descrişi detaliat într-un capitol viitor. Apelând la aceştia este posibilă o construcţie mult mai amănunţită a interogărilor SQL şi o creştere a complexităţii lor.

Exemple:

a) Să se obţină o listă cu mărfurile în stoc din depozitele 2 şi 3, pentru care s-au stabilit preţurile de desfacere.

```
SELECT denumire, um, pret_d
FROM Stocuri
WHERE nr_d IN (2,3) AND stoc<>0 AND pret_d IS NOT NULL;
```

b) Să se afişeze lista cu studenţii din anul doi, trei şi patru, al căror nume începe cu litera A şi au obţinut note între 8 şi 10 la examenul de informatică. Va exista o sortare alfabetică după valorile câmpului nume, primul în structura tabelei.

```
SELECT nume, nota
FROM Situatie
WHERE an IN (2,3,4) AND nume LIKE "A*" AND nota BETWEEN 8 AND 10
ORDER BY 1;
```

În scrierea interogărilor de selecţie simple SQL ACCESS, este posibilă şi folosirea funcţiilor totalizatoare ce vor fi, de asemenea, explicate mai în detaliu în capitolele următoare. Cele mai importante funcţii din această categorie sunt:

- ***COUNT*** returnează numărul de înregistrări care respectă condiţiile stabilite prin clauza WHERE.
- ***SUM*** redă suma tuturor valorilor dintr-un câmp; operează numai cu valori numerice.
- ***AVG*** calculează valoarea medie a unui câmp numeric.
- ***MAX*** permite determinarea celei mai mari valori dintr-un câmp; nu operează în cadrul clauzei WHERE.
- ***MIN*** duce la obţinerea celei mai mici valori a unui câmp; rămâne valabilă şi aici restricţia privind clauza WHERE.

Exemple:

a) Să se afişeze câţi clienţi au încheiat contracte de leasing cu firma "X" între 1 ianuarie şi 30 iunie 1999.

```
SELECT COUNT(*) AS nr_contracte
FROM Contr
WHERE data_c BETWEEN #01/01/99# AND #06/30/99#;
```

b) Care este totalul cheltuielilor comune de secţie în luna august? Dar media lor?

```
SELECT SUM(ch_s) AS T_ch_s, AVG(ch_s) AS Medie_ch_s
FROM Calculatie
WHERE luna="august";
```

c) Care este cel mai mare, respectiv cel mai mic număr de absenţe înregistrate la firma "Z" de la începutul anului până la 31 martie 1999 de salariaţii care au profesia "economist".

```
SELECT MAX([nr_abs]) AS Max_abs, MIN([nr_abs]) AS Min_abs
FROM Personal
WHERE per IN ("ian","febr","mart") AND profesie="economist";
```

Cereri de interogare complexe

Limbajul de interogare SQL ACCESS permite, pe lângă definirea de interogări de selecţie simple, crearea unor interogări cu o structură complexă, cum ar fi cele în care regăsim funcţiile agregate, asocierile (JOIN) sau combinările (UNION). La acestea se adaugă folosirea instrucţiunilor în cadrul formularelor, rapoartelor sau a macro-urilor, precum şi pentru stabilirea parametrilor de interogare.

Funcţiile de grup (agregat)

Funcţiile de grup (agregat) permit construirea unor interogări SQL ACCESS complexe, prin care utilizatorul poate să efectueze diverse calcule pentru grupuri de înregistrări care au câmpuri cu aceeaşi valoare. În cazul utilizării lor se foloseşte următoarea formă a frazei SELECT:

- **SELECT [domeniu] [funcţie_agregată (nume_câmp) AS alias [,listă_ selecţie]**
  FROM nume_tabelă1, nume_tabelă2 ...
  GROUP BY câmp_de_grupare
  [HAVING criteriul_de_ grupare]
  [ORDER BY câmpuri_criteriu [ASC|DESC]];

Din structura instrucţiunii de mai sus putem observa că anumite clauze care au fost întâlnite la definirea interogărilor simple se regăsesc şi aici; apar însă şi elemente noi de sintaxă:

- *Listă selecţie*

Se va referi la una sau mai multe funcţii agregate care au ca argumente nume de câmpuri ale bazei de date. Există restricţia ca aceste câmpuri să fie întotdeauna de tip numeric.

- *AS alias*

Asociază un pseudonim (nume) rezultatului utilizării funcţiei agregat.

- *Clauza GROUP BY*

Precizează câmpul sau câmpurile pe baza cărora se va efectua gruparea înregistrărilor. În acelaşi timp, prin intermediul acestei clauze, se pot executa funcţiile agregate descrise în lista de selecţie pentru fiecare dintre grupări (constituite pe baza câmpurilor de grupare). Echivalentul acestei clauze în macheta grafică QBE de construcţie a interogării îl reprezintă rândul Total.

- *Clauza HAVING*

Se referă la criteriul care va fi aplicat câmpului-definit ca argument al funcţiei agregat. Altfel spus, când se foloseşte clauza GROUP BY şi este necesară şi o condiţie, se va utiliza clauza HAVING. Spre deosebire de WHERE, care acţionează înainte de a se efectua gruparea înregistrărilor, HAVING va opera după definirea acesteia. De remarcat

faptul că se admite utilizarea unei funcţii agregat care nu apare în lista de selecţie, precum şi apelarea la mai multe criterii de grupare.

Exemple:

a) Să se stabilească numărul mediu, respectiv totalul acţiunilor emise, în localităţile Bucureşti şi Ploieşti, pe categorii de valori nominale.

```
SELECT localitate, valoare_nominala, AVG([Nr_act]) AS Medie,
SUM([Nr_act]) AS Total
FROM Capitaluri
GROUP BY localitate, valoare_nominala
HAVING localitate IN ("Bucuresti", "Ploiesti");
```

b) Să se obţină lista clienţilor rău platnici care au acumulat o valoare a facturilor neachitate mai mare de 20 milioane lei.

```
SELECT denumire_client, SUM([Valoare_neachitata]) AS Total
FROM Creante
GROUP BY denumire_client
HAVING SUM (Valoare_neachitata) > 20000000;
```

c) Să se afişeze tipurile de imobile care au în medie valori de asigurare mai mari de 30000000 lei şi pentru care există mai mult de 50 de persoane care au dorit să-şi asigure un astfel de imobil.

```
SELECT tip_imobil, AVG(val_asigurata) AS Medie,
Count(*) AS Nr_total_de_asigurati
FROM Asigurari
GROUP BY tip_imobil
HAVING AVG(val_asigurata) > 30000000 AND COUNT(cod_asigurat) > 50;
```

Asocierile (interogările JOIN)

O facilitate deosebit de importantă a limbajului SQL o reprezintă posibilitatea de a grupa şi folosi date din tabele diferite. Operaţiile de asociere induse de clauza JOIN au ca rezultat producerea tuturor combinaţiilor posibile, pentru conţinutul informaţional al fiecărei tabele. Noile înregistrări care rezultă în urma joncţiunii vor deveni disponibile pentru selecţiile ulterioare. La o asociere pot participa mai mult de două tabele.

Putem distinge mai multe categorii de joncţiuni: CROSS (încrucişată)-mai puţin utilizată, cu rol în ilustrarea elementelor specifice proprietăţilor combinatorii ale tuturor asocierilor; ECHIVALENTĂ (echijoncţiune)-cea mai folosită, presupune folosirea clauzei WHERE (pentru selecţia înregistrărilor) asociată cu o egalitate dorită; NEECHIVALENTĂ (non echijoncţiune)-care, spre deosebire de precedenta, face apel în

clauza WHERE la oricare alt operator de comparare în afară de semnul egal ("="). Acest din urmă tip de joncțiune este în general foarte rar utilizat.

Sintaxa generală pentru joncțiunile echivalente şi neechivalente este următoarea:

```
SELECT [ domeniu] listă_selecție
    FROM nume_tabelă1 , nume_tabelă2...
        [WHERE criteriul_de_asociere]
            [ORDER BY câmpuri_criteriu [ASC|DESC]];
```

Deoarece în instrucțiunile SQL care descriu joncțiuni se utilizează câmpuri ce fac parte din tabele diferite, este necesară întotdeauna specificarea tabelei de care aparțin. Forma generală de descriere a unui astfel de câmp va fi următoarea: nume_tabelă.nume_câmp. Nu se lasă spații înainte sau după punct!

Exemple:

a) Să se calculeze şi să se afişeze dobânzile lunare acordate clienților în funcție de sumele depuse, pentru o rată anuală a dobânzii de 51%.

```
SELECT Cl.nume, Cont.suma_existenta*(0.51/12) AS dobanda
FROM Clienti AS Cl, Cont_depunere AS Cont
WHERE Cl.cod_client = Cont.cod_client
ORDER BY Cl.cod_client;
```

În exemplul de mai sus este prezentată o joncțiune de tip echivalent, între două tabele *Clienți* şi *Cont_depunere*, câmpul de asociere fiind *cod_ client.*

b) Să se afişeze suma facturată şi respectiv încasată pentru fiecare factură şi client.

```
SELECT Factura.nr_factura, Client.cod_client, Factură.suma_facturata,
Incasări.suma_incasata
FROM Factura, Client, Incasari
WHERE Factura.cod_client = Client.cod_client AND
Client.cod_client = Incasari.cod_client
ORDER BY Client.Cod_client;
```

În acest caz este vorba tot de o joncțiune echivalentă, dar construită între trei tabele *Încasări, Client* şi *Factură*, utilizându-se câmpul de asociere cod_client.

O altă abordare priveşte joncțiunile ca fiind: interne (INNER JOIN) şi externe (OUTER JOIN). Primele determină o asociere a înregistrărilor din tabele, astfel încât să rezulte un număr total de înregistrări egal cu produsul numărului de înregistrări din fiecare tabelă. Joncțiunile externe, la rândul lor, sunt de două tipuri: de stânga (LEFT OUTER JOIN) şi de dreapta (RIGHT OUTER JOIN) fiind destul de puțin utilizate. Echivalentul QBE al acestor categorii de joncțiuni este alegerea opțiunilor 1, 2 sau 3, din caseta Join Properties, care au fost explicate într-un capitol precedent.

În acest mod de abordare al joncţiunilor sintaxa va avea forma:

- **SELECT [domeniu] listă_selecţie**
  FROM nume_tabelă1
  {INNER|LEFT OUTER|RIGHT OUTER} JOIN nume_tabelă2
  ON criteriul_de_asociere
  [{INNER|LEFT OUTER|RIGHT OUTER} JOIN nume_tabelă3
  ON criteriul_de_asociere]...
  [WHERE criteriul_de_selecţie]
  [ORDER BY câmpuri_criteriu [ASC|DESC]];

Semnificaţia elementelor de sintaxă descrise mai sus este următoarea:

INNER, LEFT OUTER, RIGHT OUTER- se referă la tipurile de joncţiuni (INNER JOIN-internă de tip echivalent, LEFT OUTER JOIN-externă de stânga, RIGHT OUTER JOIN- externă de dreapta). De remarcat faptul că SQL ACCESS acceptă scrierea interogărilor externe fără specificarea explicită a lui OUTER.

JOIN - specifică tabela care va fi asociată (nume_tabelă2, nume_tabelă3...) tabelei precizată în clauza FROM.

ON criteriul de asociere - arată relaţia dintre câmpurile pe care se bazează joncţiunea. Unul se află în tabela asociată, iar celălalt există într-o altă tabelă din lista cu numele tabelelor. Expresia criteriul_de_asociere conţine un operator de comparaţie de tip egalitate (=) şi va returna valorile logice TRUE sau FALSE.

Clauzele **FROM, WHERE** şi **ORDER BY** care apar şi aici, au fost deja explicate pe larg atunci când s-a tratat problema interogărilor simple.

Exemplu:

Plecând de la aceeaşi cerinţă şi anume afişarea dobânzii lunare pentru sumele depuse în cont de clienţi s-au scris mai jos instrucţiunile SQL ACCESS pentru toate cele trei tipuri de asocieri : internă, externă de stânga, externă de dreapta.

a) asociere internă (INNER JOIN)

```
SELECT Cl1.nume_client, Cont1.perioada,
(Cont1.suma_existenta*Cont1.proc/12)/100 AS dobanda
FROM Clienti1 AS Cl1
INNER JOIN Cont_depunere1 AS Cont1 ON Cl1.cod_client=Cont1.cod_client
ORDER BY Cl1.cod_client;
```

În rezultatul joncţiunii se vor regăsi înregistrările pentru care câmpul de asociere cod_client va avea aceleaşi valori în tabela *Clienţi1* şi *Cont_depunere*.

b) asociere externă de stânga (LEFT OUTER JOIN)

```
SELECT Cl1.nume_client, Cont1.perioada
(Cont1.suma_existenta*Cont1.proc/12)/100 AS dobanda
FROM Clienti1 AS Cl1
LEFT OUTER JOIN Cont_depunere1 AS Cont1
ON Cl1.cod_client = Cont1.cod_client
ORDER BY Cl1.cod_client;
```

În acest caz, în urma interogării, vom obține toate înregistrările din tabela *Clienți1* și doar acele înregistrări din tabela *Cont_depunere* pentru care câmpul de asociere *cod_client* are aceleași valori cu cele din prima tabelă; în rest se afișează spații.

c) asociere externă de dreapta (RIGHT OUTER JOIN)

```
SELECT Cl1.nume_client, Cont1.perioada
(Cont1.suma_existenta*Cont1.proc/12)/100 AS dobanda
FROM Clienti1 AS Cl1
RIGHT OUTER JOIN Cont_depunere1 AS Cont1
ON Cl1.cod_client = Cont1.cod_client
ORDER BY Cl1.cod_client;
```

De data aceasta, ca rezultat al interogării, vom obține toate înregistrările din tabela *Cont_depunere* și doar acele înregistrări din tabela *Clienți1* pentru care câmpul de asociere *cod_client* are aceleași valori cu cele din prima tabelă; în rest se vor fi afișate spații.

Combinările (interogările UNION)

Când utilizatorul dorește să vadă rezultatele a mai multor interogări SELECT în același timp, prin combinarea ieșirilor lor, poate utiliza facilitatea UNION a limbajului de interogare SQL ACCESS. De remarcat faptul că nu există echivalent QBE pentru această instrucțiune.

Sintaxa generală pentru interogările UNION este:

- **SELECT lista_campuri FROM tabela1**
  UNION SELECT listă_campuri FROM tabela2
  [GROUP BY camp_de_grupare]
  [HAVING criteriul_de_agregare]
  [UNION SELECT listă_campuri FROM tabela3
  [GROUP BY camp_de_grupare]
  [HAVING criteriul_de_grupare]]
  [UNION...]
  [ORDER BY camp_criteriu_de_sortare];

Există mai multe restricții pentru instrucțiunile care generează interogări UNION şi anume: numărul de câmpuri din lista de câmpuri asociată fiecărei instrucțiuni SELECT şi UNION SELECT trebuie să fie acelaşi; este permisă doar o dată utilizarea clauzei ORDER BY, după ultima instrucțiune UNION SELECT; secvența de nume din fiecare listă de câmpuri trebuie să corespundă unor intrări identice. Când se foloseşte UNION, automat se vor elimina duplicatele ce apar în urma combinării. In cazul folosirii domeniului ALL se vor lua în considerare şi valorile duplicate.

Exemple:

a) Să se afişeze numele şi prenumele colaboratorilor firmei "ABC" din anii 1997 şi 1998 care au avut sub 30 de ani; ordonarea în listă se face după nume.

```
SELECT nume, prenume, varsta FROM Sit97
UNION SELECT nume, prenume, varsta FROM Sit98
ORDER BY nume;
```

b) Care a fost media de vârstă a colaboratorilor studenți în anii 1997 şi 1998?

```
SELECT AVG(varsta) AS Medie FROM Sit97
GROUP BY categorie
HAVING categorie = "student"
UNION SELECT AVG(varsta) AS Medie FROM Sit98
GROUP BY categorie
HAVING categorie = "student";
```

Alte categorii de interogări complexe

Interogările Parametru reprezintă un alt tip de interogare complexă pentru care există instrucțiuni SQL asociate. Cu toate acestea utilizatorii preferă varianta de folosire a facilităților machetei grafice QBE şi a meniurilor asociate. În vederea respectării standardului SQL, tipurile de date pentru aceşti parametrii nu sunt aceleaşi cu cele folosite în ACCESS.

Sintaxa simplificată:

- **PARAMETERS listă_campuri_parametru;**
  SELECT DISTINCTROW...;

Exemplu:

```
PARAMETERS nume Text;
SELECT Clienti.cod_client, Clienti.nume
FROM Clienti;
```

Instrucțiunile SQL ACCESS mai pot fi utilizate pentru: proprietatea RECORD SOURCE a formularelor şi rapoartelor. înlocuindu-se numele interogării în caseta Record Source cu instrucțiunea SQL dorită; proprietatea ROW SOURCE a formularelor, în listele obişnuite

şi în cele derulante, obţinându-se un control asupra ordinii câmpurilor; argumentul acţiunii macro RunSQL(), existând însă restricţia de a utiliza doar instrucţiuni SQL care creează interogări acţiune.

Instrucţiunile pentru manipularea datelor

Foarte utile în exploatarea unei baze de date, aceste instrucţiuni se implementează prin interogările acţiune. Este însă necesară o mare atenţie în utilizarea lor deoarece, efectele acţiunii lor sunt permanente (ireversibile), influenţând inclusiv integritatea referenţială a bazei de date.

Cele mai importante instrucţiuni sunt: CREATE, INSERT, UPDATE şi DELETE.

- **SELECT [domeniu] (câmp1,câmp2...)**
  INTO tabela_nouă
  FROM tabela_sursă
  [WHERE criteriul_de_adăugare];

Se materializează într-o interogare acţiune de creare a unei tabele noi, plecând de la structura şi conţinutul uneia deja existente. Utilizatorul poate stabili anumite criterii asupra înregistrărilor ce vor fi adăugate în noua tabelă.

Exemplu:

Să se creeze o tabelă cu numele *Zona_Vânzări*, plecând de la tabela deja existentă *Vânzări*, în care să regăsim doar mărfurile care au fost desfăcute în localităţile Bucureşti şi Cluj:

```
SELECT DISTINCTROW Cod_m, Localit
INTO Zona_Vanzari
FROM Vanzari
WHERE Localit = "Bucuresti" OR Localit="Cluj";
```

- **INSERT**

Se foloseşte pentru adăugarea de înregistrări dintr-o tabelă în alta. Prin această interogare de adăugare nu se pot insera date dintr-o tabelă în ea însăşi; operaţia ar fi totuşi posibilă printr-o selectare prealabilă a datelor iniţiale într-un tabel temporar, urmată de modificarea şi readucerea lor în tabelul de la care s-a plecat.

Există două forme ale instrucţiunii şi anume: INSERT...VALUES şi INSERT...SELECT.

a) INSERT INTO nume_tabelă (câmp1, câmp2...)
VALUES (valoare1,valoare2...);

În acest caz se adaugă o înregistrare într-o tabelă, menţionându-se câmpurile şi valorile asociate acestora. Ca particularitate se remarcă inserarea unei singure înregistrări la un

moment dat. Prima formă a lui INSERT se utilizează pentru operații simple care presupun lucrul cu un număr redus de înregistrări. După lansarea în execuție a interogării apare un mesaj de avertizare privind adăugarea noii înregistrări în baza de date și caracterul ireversibil al acestei operații.

În cadrul acestui tip de inserare a datelor trebuie să se respecte următoarele reguli:

- valorile menționate în clauza VALUES vor avea aceeași natură cu câmpurile specificate în clauza INTO;
- mărimea valorii corespunzătoare fiecărui câmp va fi mai mică decât dimensiunea câmpului;
- nu va fi obligatorie specificarea denumirii câmpurilor, deoarece SQL ACCESS va asocia listei de valori câmpurile în ordinea din structura înregistrării (prima valoare se va introduce în primul câmp, a doua valoare în al doilea câmp ș.a.m.d.);
- dacă un câmp are definiția NOT NULL va fi obligatorie introducerea unei valori pentru acesta.

Exemplu:

Se adaugă în tabela *Mărfuri* o înregistrare care respectă structura: *cod marfă, denumire, categorie, unitate de măsură, cantitate vândută, preț vânzare.*

```
INSERT INTO Marfuri
(Cod_m, Den, Categ, UM, Cant_v, Pret_v)
VALUES (1000, "Sapun", "Cosmetice", "Buc", 50, 7000);
```

b) INSERT INTO tabelă_destinație (câmp1,câmp2...)
SELECT [domeniu] câmp1,câmp2...
FROM tabelă_sursă
WHERE criteriul_de_adăugare;

În acest caz este posibil să se copieze selectiv înregistrări dintr-o tabelă într-una sau în mai multe tabele.

Regulile menționate la instrucțiunea INSERT...VALUES rămân valabile și aici. În plus se mai adaugă următoarele:

- fraza SELECT nu poate extrage înregistrări din tabela destinație;
- numărul și natura câmpurilor menționate în clauza INTO trebuie să fie aceleași cu numărul și natura câmpurilor returnate de instrucțiunea SELECT;
- dacă nu se introduce clauza WHERE, toate înregistrările din tabela sursă vor fi adăugate în tabela destinație.

Exemplu:

Se inserează valorile câmpurilor: *număr, nume, prenume* şi *studii* doar pentru agenţii de vânzare care sunt studenţi. Selecţia se face din tabela sursă *Agent_Vânzare*, iar destinaţia o constituie tabela *Studii* (care trebuie creată în prealabil):

```
CREATE TABLE Studii
(nr Number, nume Text, pren Text, std Text);
```

Apoi instrucţiunea propriu-zisă de inserare:

```
INSERT INTO STUDII ( nr, nume, pren, std )
SELECT nr, nume, pren, std
FROM Agent_vanzare
WHERE STD = "student";
```

- **DELETE FROM nume_tabela**
  [WHERE criteriul_de_stergere];

Se materializează în interogarea acţiune de ştergere parţială sau totală a înregistrărilor din tabele. Nu este utilizată pentru ştergerea de valori din câmpuri individuale, ci va acţiona doar asupra înregistrărilor în totalitatea lor. În acelaşi timp se va şterge doar conţinutul tabelei nu şi aceasta (pentru eliminarea tabelei se va apela la instrucţiunea DROP TABLE).

Ca şi instrucţiunea INSERT, operaţia de ştergere a înregistrărilor dintr-o tabelă poate duce la apariţia unor probleme de integritate referenţială în alte tabele. Clauza WHERE restricţionează domeniul de ştergere în funcţie de cerinţele utilizatorului.

Exemple:

Să se şteargă integral conţinutul tabelei *Mărfuri*:

```
DELETE *
FROM Marfuri;
```

Să se şteargă din tabela *Agent_Vânzare* înregistrările care privesc persoanele angajate înainte de 1 ianuarie 1990:

```
DELETE *
FROM Agent_Vanzare
WHERE Data_a < #01/01/90#;
```

- unde *Data_a* reprezintă câmpul data angajării.

- **UPDATE nume_tabelă**
  SET nume_câmp1 = valoare1
  [,nume_câmp2 = valoare2]...
  [WHERE criteriul_de_actualizare];

Are atât scopul de a insera noi înregistrări, cât şi de a modifica valorile câmpurilor din înregistrările existente. Ca şi în cazul instrucţiunii INSERT, se va urmări dacă în câmpul cu valori de actualizat sunt permise numai valori unice.

Atunci când se doreşte actualizarea datelor din mai multe câmpuri se foloseşte virgula ca separator între câmpuri şi valorile acestora. Se pot utiliza mai multe condiţii WHERE apelând la operatorul logic AND pentru a limita actualizarea la înregistrări mai bine specificate.

Exemple:

a) Să se modifice în tabela *Mărfuri* categoria de marfă din "detergent" în "detergenţi" :

```
UPDATE Marfuri
SET Categ = "detergenti"
WHERE Categ = "detergent";
```

b) Să se actualizeze valorile câmpurilor categorie şi unitate de măsură, astfel încât categoria de marfă "detergent" să devină "detergenţi", iar unitatea de măsură din "cutii" în "cutie":

```
UPDATE Marfuri
SET Categ = "detergenti", UM = "cutie"
WHERE Categ = "detergent" AND UM = "cutii";
```

Cereri de interogare imbricate

Scrierea unei interogări în cadrul alteia duce la apariţia unei subinterogări; setul de rezultate obţinut de la o interogare va constitui argument pentru o alta. Utilizatorul poate astfel să creeze legături între mai multe interogări SQL ACCESS, pe baza unor câmpuri unice, cu rol de căutare în structura tabelelor. Subinterogările înlocuiesc interogările imbricate din versiunile precedente, cu performanţe mult îmbunătăţite. Pot fi construite şi prin varianta de lucru a machetei grafice QBE ACCESS.

Cea mai simplă subinterogare are sintaxa următoare:

-
```
SELECT *
    FROM Tabelă1
        WHERE Tabelă1.nume_ câmp =
                (SELECT nume_camp
                    FROM Tabelă2
                        WHERE criteriul_de_selecţie);
```

Tabela1 şi *Tabela2* vor avea un câmp comun (nume_câmp) care va reprezenta de fapt câmpul de legătură ce stă la baza construirii subinterogării. Clauza SELECT din subinterogare va avea acelaşi număr de câmpuri şi de natură similară cu cele din clauza WHERE a interogării externe.

Exemplu:

Să se afişeze toate informaţiile despre furnizorul care a livrat firmei materiale în valoare de 25 milioane lei.

```
SELECT cod_fz, denumire, adresa
FROM Furnizor
WHERE [Furnizor].[cod_fz] = (SELECT cod_fz
FROM Materiale
WHERE valoare = 25000000);
```

Când utilizatorul creează legături de subinterogare între două tabele, pentru care s-a construit şi o joncţiune, atunci sintaxa se modifică astfel:

- **SELECT listă_câmpuri**
  FROM tabelă1, tabelă2
  WHERE tabelă1.nume_ camp = tabelă2. nume_camp
  AND tabelă1. nume_camp = (SELECT nume_camp
  FROM tabelă2
  WHERE criteriul_de_selecţie);

În prima clauză WHERE s-a descris condiţia de asociere pe câmpul nume câmp care, mai departe după operatorul logic AND, a fost utilizat şi drept câmp de efectuare a subinterogării.

Exemplu:

Care este clientul care a solicitat o comandă de biscuiţi în valoare de 2 milioane lei?

```
SELECT Client.cod_client, Client.denumire, Client.adresa
FROM Client, Comenzi
WHERE [Client].[cod_client] = [Comenzi].[cod_client] AND
[Client].cod_client = (SELECT cod_client
FROM Comenzi
WHERE valoare = 2000000);
```

În anumite situaţii, subinterogările conţin în structura lor şi funcţii totalizatoare (SUM, COUNT, MIN, MAX, AVG), ceea ce determină o creştere a complexităţii în utilizare. În acelaşi timp este de remarcat faptul că subinterogările se pot imbrica, ceea ce însemnă a se "lega" mai mult de două interogări. Pe lângă clauza WHERE amintită anterior se folosesc şi clauzele GROUP BY şi HAVING cu rolul deja descris în subcapitolele precedente.

Exemple:

a) Ce mărci de maşină au valoarea vânzărilor peste medie?

```
SELECT M.nr_masina, M.marca, M.pret*C.cantitate AS Valoare
FROM Masini AS M, Cumparare AS C
WHERE M.nr_masina = C.nr_masina
AND M.pret*C.cantitate > (SELECT AVG(M.pret*C.cantitate)
FROM Masini AS M, Cumparare AS C
WHERE M.nr_masina = C.nr_masina);
```

b) Să se afişeze mărcile de maşini care au preţuri de vânzare mai mici decât media

```
SELECT marca, AVG(pret) AS Medie
FROM Masini
GROUP BY marca
HAVING AVG(pret) < (SELECT AVG(pret)
FROM Masini);
```

Un caz special în construirea unei subinterogări îl reprezintă utilizarea cuvântului cheie IN prin sintaxa:

- **SELECT listă_câmpuri**
  FROM listă_tabele
  WHERE [nume_tabelă.] nume_câmp IN
  (SELECT nume_camp
  [GROUP BY câmp_de_grupare]
  [HAVING criteriul_de_grupare]
  [ORDER BY câmpuri_criteriu [ASC|DESC]];

Clauza IN, adeseori folosită, caută potriviri în setul de informaţii care îi urmează, ducând în final la obţinerea unei ieşiri pe mai multe linii ale subinterogării.

Exemplu:

Care din salariaţii firmei cu reţineri pe statul de plată au un câştig brut peste medie?

```
SELECT R.cod_salariat, R.fel_retinere, R.suma_datorata
FROM Retineri AS R
WHERE R.cod_salariat IN
(SELECT D.cod_salariat
FROM Descriere_angajati AS D, Pontaj AS P
WHERE D.cod_salariat = P.cod_salariat AND P.ore_lucrate*D.tarif_ora >=
(SELECT AVG(P.ore_lucrate*D.tarif_ora) AS Medie_castig
FROM Descriere_angajati AS D, Pontaj AS P
WHERE D.cod_salariat = P.cod_salariat));
```

Restricționarea subinterogărilor

Domeniul de obținere a rezultatelor unei subinterogări poate fi influențat prin precizarea unuia din cuvintele cheie: ALL, ANY şi respectiv EXIST. În general pentru utilizatorul de SQL este destul de dificil să aprecieze corect diferențele dintre aceste clauze.

ALL

Se preiau rezultatele subinterogării şi, dacă acestea îndeplinesc condiția cerută, se returnează valoarea logică True; este folosit de obicei ca o dublă negație.

Exemplu:

Să se afişeze lista cărților distribuite de editura "XZ" în provincie.

```
SELECT titlul_cartii, anul_aparitiei, pret
FROM Carti
WHERE Oras_distributie <> ALL
(SELECT Oras_distributie
FROM Carti
WHERE Oras_distributie="Bucuresti");
```

ANY

Are în vedere compararea valorii de ieşire a subinterogării cu fiecare înregistrare din interogarea externă. Dacă pentru fiecare înregistrare din interogare există un rezultat al subinterogării se va returna valoarea logică True. Spre deosebire de clauza IN poate fi folosit cu diverşi operatori relaționali. Cuvântul cheie SOME are acelaşi rol şi caracteristici ca ANY.

Exemplu:

Să se stabilească dacă există în bibliotecă măcar o carte din domeniul informaticii:

```
SELECT titlul_cartii, domeniul, anul_aparitiei, pret
FROM Carti
WHERE domeniul= ANY
(SELECT domeniul
FROM Carti
WHERE domeniul ="informatica");
```

Înlocuirea lui ANY cu SOME produce un rezultat identic:

```
SELECT titlul_cartii, domeniul, anul_aparitiei, pret
FROM Carti
WHERE domeniul= SOME
(SELECT domeniul
FROM Carti
WHERE domeniul ="informatica");
```

EXISTS

Foloseşte subinterogarea ca pe o condiţie, analizând setul de rezultate al acesteia şi returnând valoarea False dacă nu există nici o ieşire. Se poate astfel verifica existenţa anumitor înregistrări şi controla ansamblul răspunsurilor date de interogare.

Exemplu:

Să se verifice dacă în cadrul bibliotecii există şi cărţi apărute înainte de 1 ianuarie 1990:

```
SELECT titlul_cartii, anul_aparitiei
FROM Carti
WHERE EXISTS
(SELECT *
FROM Carti
WHERE anul_aparitiei < 1990);
```

Fiecare cuvintele cheie ANY, ALL şi EXISTS poate să fie utilizat pentru restricţionarea interogărilor şi în forma cu negaţie – adică prin folosirea operatorului logic NOT.