Curs 5. ACCESS SQL (2)

1.Interogări de tip Union

Interogările de tip *Union* (reuniune) pot fi create în Access numai folosind SQL. Union nu este o instrucţiune sau o clauză SQL, ci un operator folosit pentru a "alătura" rezultatele a două sau mai multe interogări "compatibile".
Două interogări sunt compatibile dacă returnează acelaşi număr de coloane. De regulă, coloanele returnate de interogările ce vor fi reunite folosind operatorul UNION trebuie să aibă aceeaşi denumire şi acelaşi tip de date. Astfel, Access va folosi următoarele reguli pentru a le reuni:

- Numele coloanelor returnate sunt diferite→în tabela de rezultate vor apare numele coloanelor returnate de prima interogare;
- Dacă tipurile de date sunt diferite, Access le va converti la un tip de date compatibil cu amândouă. Astfel, pentru o coloană de tip integer şi una de tip long integer, tipul folosit de Access va fi long integer; pentru o coloană de tip text şi una numerică, Access va alege tipul text, iar pentru o coloană de tip *date/time* şi una de tip *yes/no*, Access va alege tot tipul text.

Notă: Nu putem folosi coloane de tipul Memo sau OLE Object într-o interogare Union.

Sintaxa:

```
Instructiune_select1
UNION [ALL]
Instrucţiune_select2
[UNION [ALL]
Instrucţiune_select3]
[… .]
```

Datele din tabela de rezultate a unei astfel de interogări nu pot fi modificate. Trebuie să avem grijă la ordinea în care specificăm coloanele în clauza SELECT a fiecărei interogări.

Exemplu: Presupunem că în baza de date există o tabelă cu profesorii pensionaţi (numită ProfPensionari), ce conţine aceleaşi coloane ca şi tabela Profesor. Creăm o interogare care să returneze datele din tabelele Profesor şi ProfPensionari:

```
SELECT Nume, Catedra, IdTitlu
FROM Profesor
UNION
SELECT Nume, Catedra, IdTitlu
FROM ProfPensionari;
```

1.1.Opţiunea TABLE

Atunci când dorim să includem în tabela de rezultate toate coloanele tabelelor pe care le reunim, folosim o sintaxă prescurtată, înlocuind instrucţiunea:

```
SELECT  *  FROM tabela
```

cu

```
TABLE tabela
```

Exemplu: Următoarele două interogări sunt echivalente:

```
SELECT  *  FROM Profesor
UNION
SELECT  *  FROM ProfPensionari;
```

şi

```
TABLE Profesor
UNION
TABLE ProfPensionari;
```

1.2. Opţiunea ALL

In mod implicit, Access va elimina înregistrările duplicate din tabela de rezultate a unei interogări Union. Dacă nu dorim ca acest lucru să se întâmple, folosim opţiunea ALL după operatorul UNION.

1.3. Sortarea rezultatelor

Putem folosi o clauză ORDER BY în cadrul ultimei instrucţiuni SELECT a unei interogări Union pentru a ordona tabela de rezultate. Dacă denumirile coloanelor diferă între tabelele ce sunt reunite, coloanele din clauza ORDER BY trebuie să se regăsească în clauza SELECT a primei interogări.
Dacă includem clauza ORDER BY în mai multe dintre instrucţiunile SELECT ce compun interogarea de tip Union, toate aceste clauze vor fi ignorate, cu excepţia celei din ultima instrucţiune SELECT.

2. Subinterogări

Prin folosirea unei subinterogări înţelegem includerea unei instrucţiuni SELECT în cadrul altei instrucţiuni SQL. De regulă, subinterogările sunt incluse în clauza WHERE a unei instrucţiuni SQL, dar ele pot apărea şi în clauza SELECT.

Notă: De multe ori, putem ajunge la acelaşi rezultat folosind fie o subinterogare, fie o asociere.
În grila QBE putem introduce o subinterogare fie în câmpul *Criteria*, fie în câmpul *Field*.

Ce se poate realiza cu ajutorul subinterogărilor:

1. Să verificăm dacă anumite valori se află printre datele conţinute într-o tabelă sau returnate de o interogare. Sintaxa:

```
Expresie [NOT] IN (subinterogare)
```

Exemplu: Dorim să vedem profesorii (împreună cu titlurile şi catedrele lor) care au salariul mai mare de 1.000.000 de lei. Putem folosi prima dintre interogările de mai jos (care conţine o subinterogare), ori cea de-a doua (bazată pe un join).

```
SELECT Nume, IdTitlu, Catedra
FROM Profesor
WHERE IdTitlu IN (SELECT IdTitlu FROM Titlu
WHERE Salariu  >=  1000000)
```
sau
```
SELECT Nume, IdTitlu, Catedra
FROM Profesor INNER JOIN Titlu
ON Profesor.IdTitlu = Titlu.IdTitlu
WHERE Salariu >= 1000000;
```

Acest tip de subinterogare nu poate returna decât o singură coloană; altfel, vom obţine o eroare.
Pentru a verifica dacă anumite valori nu se află într-o listă, folosim operatorul NOT.

2. Să comparăm o valoare cu datele returnate de o interogare. Sintaxa este:

```
Comparaţie [{ANY|SOME|ALL}] (subinterogare)
```

Şi în acest caz, subinterogarea nu trebuie să returneze decât o singură coloană.

Exemplu: Dorim să aflăm numele, titlul şi catedra profesorilor care au salariul mai mare decât cel al profesorului cu numele “Cristea George”.

```
SELECT Nume, Titlu, Catedra
FROM Profesor INNER JOIN Titlu
ON Profesor.IdTitlu =  Titlu.IdTitlu
WHERE Salariu > (SELECT Salariu
FROM Profesor INNER JOIN Titlu
ON Profesor.IdTitlu = Titlu.IdTitlu
WHERE Nume  = “Cristea George”);
```

In cazul în care am dori să facem comparaţii cu mai multe valori returnate de o subinterogare, putem utiliza următoarele predicate:

- ANY sau SOME: comparaţia e adevărată dacă există cel puţin o valoare returnată de subinterogare pentru care ea să fie adevărată;

- ALL: comparaţia e adevărată dacă ea este adevărată pentru orice valoare returnată de subinterogare.

Notă: Dacă nu folosim nici unul dintre predicatele ANY, SOME sau ALL, trebuie să ne asigurăm de faptul că subinterogarea returnează o singură valoare.

Exemplu: Dacă am fi dorit să aflăm numele, titlul sau catedra profesorilor care au salariul mai mare decât cel puţin un alt profesor (deci care nu au salariul cel mai mic), am fi putut scrie interogarea:

```
SELECT Nume, Titlu, Catedra
FROM Profesor INNER JOIN Titlu
ON Profesor.IdTitlu = Titlu.IdTitlu
WHERE Salariu > ANY (SELECT Salariu FROM Titlu);
```

Dacă dorim numele,titlul şi catedra profesorilor care au salariul mai mare decât orice alt profesor, folosim interogarea:

```
SELECT Nume, Titlu, Catedra
FROM  Profesor INNER JOIN Titlu
ON Profesor.IdTitlu = Titlu.IdTitlu
WHERE Salariu >= ALL (SELECT Salariu FROM Titlu);
```

Notă: Predicatele ANY,MORE şi All iau în considerare şi valorile null (spre deosebire de funcţiile agregat Min() şi Max()).

3. Să stabilim criterii în funcţie de existenţa înregistrărilor returnate de o subinterogare. Pentru aceasta, vom folosi predicatul EXIST în cadrul sintaxei:

```
[NOT] EXIST (subinterogare)
```

Subinterogările prezentate în această secţiune au fost independente de interogarea "exterioară" (cea care le folosea). Putem crea însă şi subinterogări "corelate" cu interogarea exterioară prin intermediul unei coloane de legătură.
Fiecare dintre cele trei tipuri de subinterogări poate fi corelată. Totuşi, de cele mai multe ori, subinterogările care folosesc predicatul EXIST sunt corelate.

Exemplu: Dorim să aflăm care sunt cursurile la care nu s-a înscris nici un student. Pentru aceasta, vom folosi o subinterogare ce conţine predicatul NOT EXISTS:

```
SELECT IdCurs, Denumire
FROM Curs
WHERE NOT EXISTS  (SELECT  *  FROM  Curs_Stud
        WHERE IdCurs=Curs.IdCurs);
```

Subinterogarea de mai sus este corelată deoarece ea utilizează în clauza WHERE date din interogarea exterioară.

Notă: Nu există restricţii asupra numărului de coloane returnate de subinterogările de acest tip, deoarece nu interesează numai existenţa înregistrărilor.

2.1. Subinterogările în clauza SELECT

După cum am arătat cel mai adesea vom introduce subinterogări în clauza WHERE a unei interogări. Putem folosi însă şi în clauza SELECT o subinterogre care să returneze o singură valoare.

Exemplu: Scriem o interogare care ne spune, pentru fiecare curs opţional, dacă există sau nu studenţi înscrişi la el.

```
SELECT IdCurs, Denumire,
IIf(EXISTS  (SELECT  *  FROM  Curs_Stud
WHERE IdCurs=Curs.IdCurs),  "Da", "Nu") AS [Studenţi înscrisi?]
FROM Curs;
```

2.2. Subinterogări pentru găsirea duplicatelor dintr-o tabelă

Să presupunem că avem o tabelă, numită *DuplCurs*, ce conţine aceleaşi coloane ca şi tabela *Curs*, dar şi înregistrări duplicate, astfel încât nu putem alege coloana *IdCurs* drept cheia primară a tabelei. Putem identifica înregistrările duplicate cu ajutorul următoarei subinterogări corelate:

```
SELECT  *
FROM DuplCurs
WHERE IdCurs IN (SELECT IdCurs FROM DuplCurs
GROUP BY IdCurs
HAVING Count(*)>1)
ORDER BY IdCurs;
```

Tabela cu rezultatele acestei interogări va conţine toate coloanele din tabela *Curs*, dar numai acele înregistrări ce apar de mai multe ori.

3.Interogări cu parametri

Ca şi fereastra QBE, Access SQL ne permite să scriem interogări conţinând parametri ale căror valori urmează a fi specificate în timpul rulării. Pentru aceasta, vom folosi declaraţia PARAMETERS, a cărei sintaxă este:

```
PARAMETERS parametrul1 tip_date1 [, parametrul2 tip_date2 [,…]];
```

Exemplu: Creăm interogarea *MediaParam* care regăsea mediile studenţilor al căror nume de familie începe cu o literă specificată de utilizator.

```
PARAMETERS  [Introduceţi prima literă a numelui] Text;
SELECT Student.NumeStudent, Student.PrenumeStudent, Avg(Curs_Stud.Nota)
As Media
FROM Student INNER JOIN Curs_Stud
ON Student.NrMatricol = Curs_Stud.NrMatricol
GROUP BY Student.NumeStudent, Student.PrenumeStudent
HAVING (((Student.NumeStudent) Like [Introduceţi prima literă a numelui ] & "*"));
```

4. Folosirea surselor de date externe

Există trei moduri în care puteţi face referire la surse de date aflate fizic înafara unei baze de date Access, prin intermediul unei instrucţiuni SQL:

- folosind tabele legate;
- folosind clauza IN;
- prin referiri directe la tabelele externe.

4.1. Tabele legate

Aceasta este cea mai simplă şi eficientă metodă de a folosi tabele externe drept sursă de date. O dată ce o tabelă a fost legată la o bază de date Access, o puteţi folosi în instrucţiuni SQL ca pe orice altă tabelă din baza de date.

4.2. Clauza IN

Folosirea clauzei IN este un mod simplu de a face referire la una sau mai multe tabele care nu sunt legate la baza de date Access curentă, dar se află într-o aceeaşi bază de date Access, ODBC sau în acelaşi subdirector pentru un anumit tip de bază de date.

Baze de date Access

In cazul tabelelor nelegate ce se află într-o altă bază de date Access decât cea curentă, vom folosi sintaxa:

```
FROM lista_tabele IN "calea_si_baza_de_date"
```

Exemplu: Următoarea instrucţiune SQL selectează datele din tabela *Profesor* aflată în baza de date *Backup.mdb* care se găseşte în directorul c:\Bazededate:

```
SELECT * FROM Profesor
IN "c:\Baze_de_date\ Backup.mdb";
```

Baze de date externe

In cazul în care dorim să accesăm informaţia dintr-o bază de date externă, folosim una dintre sintaxele:

```
FROM lista_tabele IN "calea" "produs;"
```

sau

```
FROM lista_tabele IN " " [produs; DATABASE=calea;]
```

produs poate fi: dBaseIII, dBaseIV, dBase 5, Paradox 3.x, Paradox 4.x, Paradox 5.x, FoxPro 2.0, FoxPro 2.5, FoxPro 2.6, FoxPro ..., Btrieve, Excel ..., iar calea conţine:

- pentru fişiere Excel, calea completă, numele şi extensia tabelei;
- pentru fişiere Btrieve, calea completă, numele fişierului DDF şi extensia;
- pentru restul produselor, doar calea până la subdirectorul ce conţine tabelele.

Notă: In cazul celei de-a doua sintaxe de mai sus, parantezele drepte sunt obligatorii

Exemplu: Următoarele instrucţiuni SQL regăsesc înregistrările din tabela *Profesor* a unei baze de date FoxPro 3.0:

```
SELECT * FROM Profesor
IN "c: \Baze_de_date" "FoxPro 3.0;";
SELECT * FROM Profesor
IN " " [FoxPro 3.0; DATABASE=c: \Baze_de_date;];
```

Baze de date ODBC

```
FROM lista_tabele IN " " [ODBC; sir_de_conectare;]
```

Notă: Parantezele drepte sunt obligatorii.

Şirul de conectare depinde de driverul ODBC folosit. Pentru Microsoft SQL Server el va fi de forma:
DSN=sursa_de_date;UID=in_utilizator;PWD=parola; DATABASE=baza_de_date

unde sursa_de_date(DSN) este numele sursei de date pe care aţi specificat-o folosind programul pentru administrarea drivelor ODBC şi care poate face referire la una sau mai multe baze de date (Dacă face referire la o singură bază de date, nu mai trebui să folosiţi parametrul DATABASE). Parametrii UID şi PWD sunt opţionali (din motive de securitate, este bine să nu includeţi parola în şirul de conectare). Dacă omiteţi unii dintre aceşti parametri, ei vă vor fi ceruţi la rularea interogării.

Exemplu: Următoarea instrucţiune SQL regăseşte datele din tabela Profesor ce face parte dintr-o sursă de date SQL Server numită SQLBackup:

```
SELECT * FROM Profesor
IN " " [ODBC;DSN=SQLBackup;UID=DanB;];
```

La executarea interogării vă va fi cerută parola utilizatorului DanB.

Notă: Dacă lucraţi cu baze de date QDBC, este preferabil să folosiţi tabele legate, deoarece astfel interogările vor fi mai rapide în execuţie.

4.3. Referiri directe la tabele externe

Pentru cazul în care dorim să lucrăm cu surse de date care se află în subdirectoare sau baze de date diferite (sau de tipuri diferite), Access ne pune la dispoziţie metoda referirii directe.
Tabelul următor prezintă sintaxa pentru diferitele tipuri de surse de date:

Sursa de date	Sintaxa
Baze de date Access	[cale_si_baza_de_date].tabela
Baze de date externe	[produs;DATABASE=calea;].tabela
Baze de date ODBC	[ODBC, şir_de_conectare;].tabela

Notă: Parantezele drepte sunt obligatorii

Exemplu: folosim două asocieri între trei tabele: tabela Access externă Curs, tabela SQL Server Curs_Prof şi tabela FoxPro 3.0 Profesor:

```
SELECT Denumire, Nume, Catedra
FROM (c: \Baze_de_date \Backup.mdb].Curs INNER JOIN
([ODBC; DSN=SQLBackup;UID=DanB;].Curs_Prof INNER JOIN
[FoxPro 3.0; DATABASE=c: \Baze_de_date1;].Profesor
ON Curs_Prof.IdProf=Profesor.IdProf)
ON Curs.IdCurs=Curs_Prof.IdCurs;
```

Notă: Deşi corectă, interogarea de mai sus nu este foarte eficientă. In general, este bine să evitaţi să folosiţi prea multe asocieri între tabele ce provin din surse de date diferite. Vă repetăm că cea mai performantă metodă este folosirea tabelelor legate.

5. Instrucţiuni pentru actualizarea datelor

Access SQL pune la dispoziţie patru comenzi pentru actualizarea datelor: UPDATE, DELETE, INSERT INTO şi SELECT INTO. Acestea corespund alegerii, pentru o interogare creată în fereastra QBE, tipului Update, Delete, Append şi, respectiv, Make Table.

5.1. Instrucţiunea UPDATE

Este folosită pentru a modifica valoarea uneia sau mai multor coloane dintr-o tabelă. Sintaxa este:

```
UPDATE tabela_sau_interogare
SET coloana1=expresia1[, coloana2=expresia2[,…]]
[WHERE criterii];
```

Expresiile din clauza SET pot fi constante sau obişnuite pe baza unor calcule Access. SQL vă permite folosirea asocierilor în clauza UPDATE, dar nu şi a subinterogărilor.

Exemplu: Următoarea interogare va mări cu 5% salariile profesorilor catedrei “Matematica”:

```
UPDATE Profesor INNER JOIN Titlu
ON Profesor.IdTitlu=Titlu.IdTitlu
SET Salariu=[Salariu]+[Salariu]*0.05
WHERE Catedra=”Matematica”;
```

5.2. Instrucţiunea DELETE

Este folosită pentru a şterge înregistrări dintr-o tabelă şi are următoarea sintaxă:

```
DELETE [tabela.*]
FROM clauza
[WHERE criterii];
```

Notă: Dacă interogarea se bazează pe o singură tabelă, nu e necesar să specificaţi tabela.* în clauza DELETE, deoarece oricum vor fi şterse complet înregistrările ce întrunesc criteriile clauzei WHERE (şi nu doar valorile unor coloane pentru aceste înregistrări).

Exemplu: Următoarea interogare va şterge din tabela Profesor toţi profesorii catedrei "Matematica":

```
DELETE
FROM Profesor
WHERE Catedra="Matematica"
```

Dacă în clauza FROM specificaţi mai multe tabele, trebuie să respectaţi următoarele reguli:

- pentru a şterge înregistrări dintr-o tabelă pe baza datelor din alte tabele, puteţi folosi asocieri în clauza FROM sau subinterogări în clauza Where;
- puteţi şterge înregistrări din mai multe tabele cu o singură instrucţiune DELETE, dacă între acestea există o relaţie de tip 1:1;
- puteţi şterge înregistrări din mai multe tabele legate prin relaţii de tip 1:m, prin mai multe instrucţiuni DELETE sau printr-o singură instrucţiune DELETE, dacă atunci când aţi definit relaţiile aţi validat opţiunea "Cascade Deletes".

Exemplu: Ştergem din tabela Profesor profesorii care au salariul mai mic de 1000000 lei:

```
DELETE Titlu.*
FROM Titlu INNER JOIN Profesor ON Titlu.IdTitlu =
Profesor.IdTitlu
WHERE (((Titlu.Salariu) < 1000000));
```

Notă: Dacă doriţi să ştergeţi valoarea uneia sau mai multor coloane, dar nu o întreagă înregistrare, folosiţi o instrucţiune UPDATE în care să daţi câmpurilor respective valoarea null.

5.3. Instrucţiunea SELECT INTO

Creează o nouă tabelă folosind înregistrări din altă tabelă sau interogare. Sintaxa:

```
SELECT coloana1 [, coloana2[,…]] INTO tabela_noua
FROM lista_de_tabele
[WHERE criterii]
[ORDER BY coloane];
```

Exemplu: Interogarea MakeProfBack ce crează tabela ProfMate, cu profesorii catedrei "Matematici", conţinând câmpurile IdProf, Nume şi Titlu:

```
SELECT Profesor.IdProf, Profesor.Nume, Titlu.Titlu,
Profesor. Catedra INTO ProfBack
FROM Titlu INNER JOIN Profesor
ON Titlu.IfTitlu = Profesor.IdTitlu
Where (((Profesor.Catedra)="Matematica"));
```

Notă: Tabelele create cu ajutorul instrucţiunii SELECT INTO nu vor avea chei primare, indecşi sau orice alte proprietăţi speciale, altele decât cele implicite folosite la crearea unei tabele noi.

5.4. Instrucţiunea INSERT INTO

Putem copia liniile dintr-o tabelă (interogare) în altă tabelă. De asemenea, putem adăuga o înregistrare la o tabelă specificând lista valorilor pe care aceasta le va conţine. Sintaxele pentru pentru cele două tipuri de instrucţiuni:

```
INSERT INTO tabela
instructiune_select;

INSERT INTO tabela [(coloana1 [, coloana2 [,…]])]
VALUES (valoarea1 [, valoarea 2 [, …]]);
```

Instructiune_select poate fi orice instrucţiune SELECT ce poate conţine clauze GROUP BY, asocieri, operatori UNION şi chiar subinterogări.

Exemplu: am creat interogarea AppINSERT INTO ProfBack ce adaugă profesorii catedrei de Informatică la tabela ProfMate pe care am creat-o rulând interogarea MakeProfBack. Iată care este instrucţiunea SQL corespunzătoare.:

```
SELECT Profesor.IdProf, professor.Nume, Titlu.Titlu,
Professor.Catedra
FROM Titlu RIGHT JOIN Profesor
ON Titlu.IdTitlu = Profesor.IdTitlu
WHERE Profesor.Catedra=”Informatica”;
```

Dacă în cea de-a doua sintaxă omiteţi să specificaţi coloanele, valorile din clauza VALUES trebuie să fie date exact în ordinea în care apar coloanele în definiţia tabelei.

6. Instrucţiuni pentru definirea datelor

Interogările DDL (*Data Definition Language*) pentru definirea datelor reprezintă unul dintre modurile în care puteţi defini structura tabelelor în Access. Celelalte două moduri sunt: folosirea ferestrelor Table Design şi Table Datasheet şi prin intermediul obiectelor DAO (Data Access Objects).
Deşi interogările DDL nu vă oferă prea multe facilităţi, cum ar fi definirea regulilor de validare, a valorilor implicite etc., ele au avantajul de a avea la bază un limbaj standard foarte larg folosit –SQL.
Nu puteţi crea interogări DDL cu ajutorul ferestrei QBE, ci doar în fereastra SQL View.
Cele patru instrucţiuni DDL oferite de Access SQL sunt: CREATE TABLE, ALTER TABLE, CREATE INDEX şi DROP

6.1. Instrucţiunea CREATE TABLE

Creează scheletul unei noi tabele şi are următoarea sintaxă:

```
CREATE TABLE tabela (coloana1 tip1 [(dimensiunea1)]
[CONSTRAINT restrictie_coloana1] [,coloana2 tip2
[(dimensiunea2)] [CONSTRAINT restrictie_coloana2]] [,…]
[CONSTRAINT restrictie1_tabela [, restrictie2_tabela [,…]]]);
```

Tipul de date corespunzător unei coloane:

Tip de date Jet Engine SQL	Corespondentul în modul Table Design
BIT, BOOLEAN, LOGICAL, LOGICAL1, YESNO, BYTE, INTEGER1	Yes/No sau Number cu FieldSize=Byte
COUNTER, AUTOINCREMENT	Autonumber cu FieldSize=Long Integer
CURRENCY, MONEY	Currency
DATATIME, DATE, TIME	Date/Time
SHORT, INTEGER2, SMALLINT	Number, cu FieldSize=Integer
LONG, INT, INTEGER, INTEGER4	Number, cu FieldSize=Long
SINGLE, FLOAT4, IEEESINGLE, REAL	Number, cu FieldSize=Size
DOUBLE, FLOAT, FLOAT8, IEEEDOUBLE, NUMBER, NUMERIC	Number, cu FieldSize=Double
TEXT, ALPHANUMERIC, CHAR, CHARACTER, STRING, VARCHAR	Text
LONGTEXT, LONGCHAR, MEMO, NOTE	Memo
LONGBINARY, GENERAL, OLEOBJECT	LOE Object
GUID	Autonumber, cu FieldSize=ReplicationID

Notă: Tipurile aflate pe aceeaşi linie în partea stângă a tabelului de mai sus sunt sinonime.
Putem folosi parametrul *dimensiunea* pentru a specifica numul maxim de caractere al unei coloane de tip text. Dimensiunea implicită este de 255 de caractere. Restul tipurilor de date nu folosesc această opţiune.

Exemplu: Următoarea interogare va crea tabela Titlu1, care are coloanele IdTitlu, Titlu şi Salariu:

```
CREATE TABLE Titlu1 (IdTitlu COUNTER, Titlu TEXT (50),Salariu LONG);
```

Clauza CONSTRAINT
Această clauză este folosită pentru a crea indecşi. Ea poate apărea atât în instrucţiunea CREATE TABLE, cât şi în instrucţiunea ALTER TABLE. Aţi observat că în sintaxa instrucţiunii CREATE TABLE, clauza CONSTRAINT poate apărea ori la declararea fiecărei coloane, ori la sfârşit. Primul caz corespunde creării unui index pe coloana respectivă, iar cel de-al doilea, creării indecşilor pe mai multe coloane ale tabelei.
Sintaxa pentru crearea unui index pe o singură coloană este:

```
CONSTRAINT nume_index [PRIMARY KEY | UNIQUE |
REFERENCES tabela_corelata [(coloana]}
```

Pentru a crea un index pe mai multe coloane ale unei tabele, folosim sintaxa:

```
CONSTRAINT nume_index
{PRIMARY KEY (coloana1 [, coloana2 [,…]]) | UNIQUE |

REFERENCES tabela_corelata [(coloana_corelata1
[, coloana_corelata2 [,…] ] )] }
```

Exemplu: Se creează tabela Titlu şi un index pe cheia primară IdTitlu.Numele indexului estechiar PrimaryKey. (Când lucrăm în modul TableDesign, cheia primară este indexată automat).

```
CREATE TABLE Titlul (IdTitlu COUNTER CONSTRAINT PrimaryKey
PRIMARY KEY, Titlu TEXT(50), Salariu LONG);
```

Pentru a crea tabela Profesori, folosim interogarea:

```
CREATE TABLE Profesori (IdProf COUNTER CONSTRAINT PrimaryKey PRIMARY KEY, Nume
TEXT(50), Catedra TEXT (50), IdTitlu LONG
CONSTRAINT TitluCS REFERENCES Titlu1, [Data Angajarii] DATETIME);
```

In instrucţiunea de mai sus am folosit două clauze CONSTRAINT, prima pentru a crea indexul PrimaryKey pe cheia primară a tabelei Profesori, IdProf, iar a doua pentru a atribui coloanei IdTitlu cheie străină şi a stabili legătura în tabelele Profesori şi Titlu1.
Clauza CONSTRAINT nu permite crearea indecşilor ne-unici. In acest scop, puteţi folosi instrucţiunea CREATE INDEX.

6.2. Instrucţiunea CREATE INDEX

Puteţi folosi această instrucţiune pentru a crea indecşi pe tabelele existente. Sintaxa este:

```
CREATE [UNIQUE] INDEX nume_index
ON tabela (coloana1 [, coloana2 [,…]])
[WITH {PRIMARY | DISSALOW NULL | IGNORE NULL}];
```

Dacă folosim cuvântul cheie UNIQUE, nu vor fi admise deplicate pe coloanele indexate.
Pentru a crea un index pe cheia primară, folosim opţiunea PRIMARY din clauza WITH. Indecşii pe cheia primară sunt în mod implicit unici, deci, în acest caz, nu mai trebuie să folosiţi cuvântul cheie UNIQUE.
Prin opţiunea IGNORE NULL, la crearea indexului vor fi ignorate valorile null ale coloanelor ce urmează a fi indexate. Astfel, dacă o coloană indexată conţine multe valori null, căutările după acea coloană pentru valorile nenul vor fi mult mai rapide. Echivalentul în fereastra Table Design este proprietatea Ignore Nulls.

Folosim opţiunea DISALLOW NULL pentru a împiedica utilizatorul să introducă valoarea null în coloana ce urmează a fi indexată. Similar, în modul Table Design dăm proprietăţii Required a coloanei respective valoarea Yes.

Exemplu: Pentru a crea un index pe coloana Catedra a tabelei Profesori ce nu va permite introducerea valorii null în această coloană:

```
CREATE INDEX Catedra ON Profesori WITH DISSAKIW NULL;
```

6.3. Instrucţiunea ALTER TABLE

Este folosită pentru a modifica structura unei tabele existente. Putem adăuga sau şterge o coloană sau un index.
Pentru a adăuga o coloană la o tabelă, sintaxa este:

```
ALTER TABLE tabela ADD [COLUMN] coloana tipul [(dimensiunea)]
[CONSTRAINT index_pe_acesta_coloana];
```

Exemplu: Pentru a adăuga coloana Avans la tabela Titlu1, scriem

```
ALTER TABLE Titlu1 ADD Avans LONG;
```

Pentru a adăuga un index, folosim sintaxa:

```
ALTER TABLE tabela ADD CONSTRAINT index;
```

Exemplu: Pentru a crea un index unic pe coloana Avans, scriem:

```
ALTER TABLE Titlu1 ADD CONSTRAINT Avans UNIQUE (Avans);
```

Cea de-a treia sintaxă a instrucţiunii ALTER TABLE este cea pentru ştergerea unei coloane:

```
ALTER TABLE tabela DROP [COLUMN] coloana;
```

Exemplu: Pentru a şterge coloana Avans din tabela Titlu1, scriem `ALTER TABLE Titlu1 DROP Avans;`

Notă: Nu putem şterge o coloană indexată dintr-o tabelă fără a şterge, în prealabil, indexul respectiv.
Pentru a şterge un index, sintaxa instrucţiunii ALTER TABLE este următoarea:

```
ALTER TABLE tabela DROP CONSTRAINT index;
```

Exemplu: Pentru a şterge indexul anterior pe coloana Avans: `ALTER TABLE Titlu1 DROP CONSTRAINT Avans;`

Notă: Nu putem şterge un index ce participă la mai multe relaţii fără a şterge, în prealabil, relaţiile respective.

6.4. Instrucţiunea DROP

O puteţi folosi fără a şterge cu totul o tabelă sau un index. In primul caz, sintaxa este:

```
DROP TABLE tabela;
```

Exemplu: Pentru a şterge tabela Titlu1 scriem

```
DROP TABLE Titlu1;
```

Pentru a şterge un index, putem folosi fie instrucţiunea ALTER TABLE, fie instrucţiunea DROP, a cărei sintaxă, în acest caz, este:

```
DROP INDEX index OB tabela;
```

Exemplu: Pentru a şterge indexul Avans creat la secţiunea anterioară, scriem

```
DROP INDEX Avans ON Titlu1;
```