Curs 11. Lucrul cu obiecte (2)
1. Obiecte definite de utilizator
2. Lucrul cu obiecte de tip Recordset
3. Obiecte speciale

1. Obiecte definite de utilizator

In Access putem crea propriile obiecte, cu proprietăţile şi metodele lor, în cadrul modulelor pentru clase.
Pe parcursul acestui curs, creăm un obiect care să poată fi accesat prin VBA care oferă diferite informaţii despre sistem.

Proprietăţile şi metodele obiectului

Primul lucru pe care trebuie să îl facem atunci când creăm un obiect este să stabilim care sunt informaţiile pe care acesta le va oferi; trebuie să determinăm caracteristicile şi comportamentul lui.
Tabelul următor conţine proprietăţile obiectului *Calculator*, pe care îl vom crea.

Proprietate	Descriere
Calculator.Nume	Sir de caractere ce conţine numele calculatorului.
Calculator.PornireWindows	Proprietate de tip Date/Time, ce arată când a fost pornit ultima oară sistemul de operare Window
Calculator.DirAccess	Sir de caractere ce conţine directorul în care se află msaccess.exe

In plus fată de aceste proprietăţi, obiectul Calculator trebuie să-i ofere utilizatorului posibilitatea de a efectua diferite acţiuni, cum ar fi, de exemplu, ştergerea unui director gol sau redarea unui fişier de sunet . Pentru aceasta, vom implementa metodele `Calculator.StergeDir` şi `Calculator.RedareWave`.

Vom defini o clasă (clasa *Calculator*), cu ajutorul unui modul pentru clase (*Class Module*). Apoi, prin VBA, vom putea crea şi folosi obiecte ale acestei clase.

Deschidem fereastra Database şi alegem opţiunea *Class Module* din meniul *Insert* pentru a crea un nou modul pentru clase. Salvăm acest modul cu numele *Calculator* (numele clasei pe care dorim să o implementăm).
Observăm că modulele pentru clase sunt foarte asemănătoare cu modulele standard. Una dintre diferenţe este aceea că ele pot conţine două proceduri speciale, *Class_Initialize( )* şi *Class_Terminate ( )*. Procedura *Class_Initialize ( )* se apelează automat la crearea instanţelor unui obiect al clasei definite în modul respectiv. De aceea, în cadrul acestei proceduri se efectuează, de obicei, iniţializări ale proprietăţilor obiectului. Procedura *Class_Terminate ( )* este apelată ori de câte ori este distrusă o instanţă a unui obiect bazat pe modulul respectiv.

Declarăm întâi la secţiunea (General) (Declarations) a modulului o variabilă globală în cadrul modulului Calculator:

```
Private strDirAccess As String
```

Creăm subrutina *Class_Initialize( )*, selectând din lista combinată *Object*, secţiunea *Class*, iar din lista combinată *Procedure*, evenimentul *Initialize*. In cadrul acestei subroutine vom iniţializa variabila globală declarată anterior. Ea va păstra directorul (cu calea completă) în care este instalată aplicaţia Access (în care se află fişierul Msaccess.exe). Pentru a obţine o informaţie, vom apela funcţia VBA SysCmd. Scriem în cadrul subrutinei *Class_Initialize( )*, următoarea linie de cod:

```
strDirAccess = SysCmd(acSysCmdAccessDir)
```

Funcţia SysCmd() poate fi folosită pentru a efectua diferite acţiuni în Access. Dacă îi dăm ca parametru constanta intrinsecă acSysCmdAccessDir, ea va returna un şir de caractere ce reprezintă directorul în care se află Msaccess.exe. Faptul că am obţinut această informaţie nu este sufficient. Mai întâi ne asigurăm că ea poate fi accesată prin intermediul proprietăţii Calculator.DirAccess. Procedura *PropertyGet* creează o proprietate şi stabileşte modul în care sunt regăsite valorile acestei proprietăţi. Ea are următoarea sintaxă:

```
[Public | Private] [Static] Property Get NumeProprietate_ [(ListaArgumente)] As TipDate
```

unde:
Public – indică faptul că procedura Property Get poate fi apelată de orice altă procedură din orice alt modul;
Private – indică faptul că procedura Property Get poate fi apelată numai de către procedurile din modulul în care a fost declarată;
Static – indică faptul că valorile variabilelor locale ale procedurii se păstrează între apeluri;

NumeProprietate – numele proprietăţii ale cărei valori le regăseşte procedura;
ListaArgumente – lista argumentelor procedurii;
TipDate – tipul de date al proprietăţii.

Procedura *Property Get* ce regăşeşte valoarea proprietăţii DirAccess este:

```
Public Property Get DirAccess() As String
DirAccess = strDirAccess
End Property
```

Proprietatea Calculator.DirAccess este read-only, adică valoarea ei nu poate fi modificată. Totuşi, majoritatea proprietăţilor unui obiect (formular, raport, control etc.) îşi pot schimba valoarea. O astfel de proprietate este şi Calculator.Nume. Pentru a regăsi şi a modifica valoarea acestei proprietăţi, vom folosi două funcţii API, *GetComputerName* şi respective, *SetComputerName*, ale căror declaraţii:

```
Private Declare Function GetComputerName Lib "kerne132" Alias "GetComputerNameA" (ByVal ipBuffer
As String, nSize As Long) As Long
Private Declare Function SetComputerName Lib "kerne132" Alias_
"SetComputerNameA" (ByVal lpComputerName As String) As Long
```

trebuie să le includem la secţiunea (General) (Declarations) a modulului.

Funcţia *SetComputerName* primeşte ca argument un şir de caractere ce reprezintă noul nume pe care dorim să i-l dăm calculatorului. Modificarea nu devine efectivă însă decât după ce am închis şi pornit din nou calculatorul. Astfel, dacă apelăm funcţia SetComputerName cu argumentul "NumeNou" şi apoi apelăm funcţia GetComputerName fără a fi iniţializat între timp calculatorul, aceasta din urmă va returna vechiul nume al calculatorului şi nu şirul de caractere "NumeNou", cum ar fi de aşteptat. De aceea, ne propunem ca proprietatea Calculator.Nume să păstreze ultimul nume pe care l-am dat calculatorului în cadrul unei instanţe a obiectului Calculator şi nu returnat de funcţia GetComputerName.

Deoarece proprietatea Nume poate fi modificată, în plus faţă de procedură Property Get, trebuie să scriem şi o procedură *Property Let* pentru această proprietate, care să ne ajute să-i modificăm valoarea prin intermediul unei simple atribuiri.
Sintaxa instrucţiunii *Property Let* este următoarea:

```
[Public | Private] {Static] Property Let NumeProprietate_(ListaArgumente)
```

Cuvintele cheie Public, Private, Static şi NumeProprietate au aceleaşi semnificaţii ca şi pentru instrucţiunea Property Get, iar ListaArgumente reprezintă argumentele procedurii. Numele şi tipul fiecărui argument (cu excepţia ultimului) trebuie să corespundă exact numelui şi tipului argumentelor procedurii Property Get a aceleiaşi proprietăţi. Ultimul argument reprezintă noua valoare ce va fi atribuită proprietăţii şi deci trebuie să aibă acelaşi tip de date ca şi cel returnat de procedura Property Get.
Proceduri pentru regăsirea şi modificarea valorilor proprietăţii Calculator.Nume:

```
Public Property Get Nume () As String
Nume = strNume
End Property

Public Property Let Nume(strNumeNou As String)
Call SetComputerName(strNumeNou)
strNume = StrNumeNou
```

Variabila strNume este globală în cadrul modulului şi trebuie deci să fie declarată la secţiunea *(General)(Declaration)* a acestuia:

```
Private strNume As String * 255
```

Ea trebuie să fie iniţializată în cadrul funcţiei *Class_initialize()*:

```
Call GetComputerName(strNume, 255)
```

Până acum am vorbit numai despre proprietăţile obiectului Calculator. Metodele unui obiect sunt mult mai simplu de implementat: ele sunt subroutine sau funcţii publice obişnuite. Dăm cele două metode ale obiectului Calculator:

```
Public Function StergeDir(strDir As String)
StergeDir = RemoveDirectory(strDir)
End Function

Public Sub PlayWare(strWare As String)
Call PlaySound(strWare, 0, 0)
End Sub
```

Pentru a apela funcţiile API *RemoveDirectory* şi *PlaySound*, trebuie să includem la secţiunea (General) (Declarations) a modulului declaraţiile lor:

```
Private Declare Function RemoveDirectory Lib "kerne132" Alias_"RemoveDirectoryA" (ByVal
lpPathName As String) As Long
Private Declare Function PlaySound Lib "winmm.dll" Alias_
"PlaySoundA" (ByVal lpszName As String, ByVal hModule As Long, ByVal_ dwFlags As Long)
```

Crearea instanţelor obiectului

Există o diferenţă între crearea unei variabile de tip obiect şi crearea unui obiect. Prin următoarea linie de cod:

```
Dim comp As New Calculator
```

nu se creează o nouă instanţă a obiectului Calculator, ci doar se alocă resursele necesare creării unui obiect de acest tip. Obiectul în sine nu este creat până când una dintre proprietăţile sau metodele sale nu este folosită sau apelată. Cu alte cuvinte, dacă după declaraţia de mai sus am dori să afişăm proprietatea DirAccess şi am scrie:

```
MsgBox comp.DirAccess
```

această instrucţiune este cea care declanşează crearea obiectului, apelarea procedurilor:

`Class_Initialize` şi `Prop Get DirAccess.`

Instrucţiunea folosită pentru alocarea resurselor poate fi înlocuită cu: `Set comp = New Calculator`

In acest caz, instrucţiunea Dim alocă spaţiul necesar creării obiectului, iar instrucţiunea Set determină crearea obiectului şi apelarea procedurii Class_Initialize.
Observăm utilizarea cuvântului cheie *New* în ambele variante folosite pentru crearea instanţelor. Acesta trebuie folosit pentru orice obiect.
Pentru a testa funcţionalitatea unui obiect de tip Calculator, creăm un nou modul standard în care scriem următoarea subrutină:

```
Sub testCalculator()
Dim comp As New Calculator
MsgBox com.DirAccess
MsgBox comp.Nume
comp.Nume = "CalculatorulMeu"
End Sub
```

Durata de viaţă a obiectelor

In situaţia în care avem mai multe variabile care fac referire la acelaşi obiect, acesta nu poate fi distrus şi deci, resursele ocupate de el nu pot fie liberate decât după ce au fost distruse toate variabilele ce fac referire la el. Acest lucru se întâmplă ori când o variabilă îşi încheie durata de viaţă (scopul), ori de când ea primeşte valoarea Nothing:

```
Set comp = Nothing
```

2. Lucrul cu obiecte de tip Recordset

Adevărata putere a unei baze de date constă în modul în care putem să regăsim informaţia, să manipulăm înregistrările şi seturile de înregistrări. Dacă ne decidem să programăm în VBA, vom lucra foarte mult cu obiecte de tip Recordset.

Exemplu: Subrutină ce foloseşte un astfel de obiect pentru a găsi numărul de înregistrări dintr-o tabelă.

```
Public Function NrInregistr(strTabela As String)
Dim db As Database
Dim rst As Recordset
Dim iNrInregistrari
Set db = CurrentDb()
Set rst = db.OpenRecordset(strTabela)
iNrInregistrari = rst.RecordCount
Debug.Print "Tabela " & strTabela & " contine " & iNrInregistrari &_
" inregistrari"
Rst.close
End Function
```

Deschidem fereastra *Immediate* şi scriem: `?NrInregistr("Student")`

Atâta timp cât un set de înregistrări este deschis (între apelul metodei OpenRecordset şi al metodei Close), putem face orice dorim cu înregistrările sale: le putem edita, şterge sau chiar adăuga înregistrări noi. Este important să închidem un set de înregistrări (apelând metoda *Close*) pentru a elibera resursele pe care Access i le-a alocat.

Tipuri de seturi de înregistrări

In VBA există cinci tipuri de obiecte Recordset:

- Obiecte Recordset de tip tabelă;
- Obiecte Recordset de tip dynaset;
- Obiecte Recordset de tip snapshot;
- Obiecte Recordset de tip forward-only;
- Obiecte Recordset de tip dynamic.

In funcţie de următorii factori, decidem pe care dintre aceste tipuri îl vom folosi la un moment dat:

- Dacă dorim să şi actualizăm înregistrările, nu numai să le vedem;
- Dacă tabelele se află într-o bază de date Access sau de alt tip;
- Câte înregistrări conţine setul de înregistrări.

După cum am observat şi din exemplul anterior, pentru a crea un set de înregistrări, folosim expresia:

```
Set rst = db.OpenRecordset(Sursa[, tip[, optiuni[, blocaje]]])
```

sau

```
Set rst = db.OpenRecordset(tip[, optiuni[, blocaje]]])
```

unde:

- Db – este un obiect de tip Database;
- Sursa – este o tabelă, o interogare sau o instrucţiune SQL ce returnează înregistrări;
- Obiect – un obiect al unei baze de date deschis anterior:tabelă, interogare sau un alt set de înregistrări;
- Tip – specifică tipul setului de înregistrări şi poate fi una dintre următoarele constante predefinite:
  - dbOpenTable – set de înregistrări de tip tabelă
  - dbOpenDynaset – dynaset
  - dbOpenSnashot – snapshot
  - dbOpenForwardOnly – set de înregistrări de tip forward-only
  - dbOpenDynamic – set de înregistrări de tip dynamic;
- optiuni – combinatie de constante ce definesc caracteristicile noului set de înregistrări;
- blocaje – constantă ce specifică tipul de blocaj aplicat setului de înregistrări (pentru aplicaţii multiuser).

Notă: Access permite să creăm un fişier de tip recordset printr-o singură linie de cod

Exemplu: `Set rst = Current.Db.OpenRecordset("Profesor", dbOpenTable)`

Metoda compusă din doi paşi este preferată, dacă vom folosi obiectul de tip Database în cadrul procedurii şi pentru alte acţiuni.

Seturi de înregistrări de tip tabelă

Acesta este tipul implicit de set de înregistrări pe care Access îl creează dacă nu specificăm alt tip ca argument al metodei *OpenRecordset* şi dacă sursa de înregistrări este o tabelă din baza de date curentă. Unul dintre avantajele unui astfel de set de înregistrări este acela că putem folosi indecşi pentru a accelera căutările.

Seturi de înregistrări de tip dynaset

Un astfel de set de înregistrări poate conţine date din mai multe tabele locale sau ataşate. Putem edita datele dintr-un dynaset şi modificările vor fi efectuate şi în tabelele de bază . In cazul unei aplicaţii multiuser, Access va actualiza datele dintr-un dynaset deschis pentru a reflecta modificările efectuate de alţi utilizatori asupra tabelelor de bază.
Câteva situaţii în care ar trebui să optăm pentru crearea unui dynaset:

- când dorim să putem modifica înregistrările:
- când obiectul de tip Recordset are dimensiuni foarte mari;
- când obiectul de tip Recordset conţine obiecte OLE.

Acesta este tipul implicit de set de înregistrări pe care Access îl creează dacă sursa de înregistrări este o interogare, o instrucţiune SQL SELECT sau o tabelă care nu se află într-o bază de date Access. Câteva exemple de creare a unui dynaset:

```
Set db = CurrentDb()
Set rst = db.OpenRecordset("IntProfCurs", dbOpenDynaset)
```

unde IntProfCurs este o interogare;

```
Set db = CurrentDb()
Set rst = db.OpenRecordset("SELECT * FROM Student", dbOpenDynaset)
```

unde sursa de înregistrări este o instrucţiune SELECT.

Seturi de înregistrări de tip snapshot

Seturile de înregistrări de tip snapshot nu permit modificarea datelor şi nici nu reflectă modificările efectuate de alţi utilizatori asupra tabelelor de bază. Ele reprezintă instanţe ale datelor la un moment dat. Avantajul lor este acela că sunt mai rapide decât cele dynaset, dar numai atunci când numărul de înregistrări nu este mai mare de 500.

Actualizarea datelor unui set de înregistrări

Pentru a ne asigura că datele unui set de înregistrări sunt actualizate, putem executa din nou interogarea care stă la baza setului, apelând metoda Requery (reinterogare) a obiectului. Nu orice obiect de tip Recordset suportă însă interogarea. De aceea, pentru a afla acest lucru, trebuie ca înainte de a apela la metoda Requery să verificăm dacă proprietatea Restartable a obiectului are valoarea True (altfel, reinterogarea va genera o eroare):

```
If rst.Restartable = True.Then rst.Requery
```

Parcurgerea seturilor de înregistrări

După ce am creat un obiect de tip Recordset, înregistrarea curentă va fi prima înregistrare a setului. Dacă nu am specificat nici o regulă de sortare a înregistărilor, acestea se pot afla în orice ordine.
Access vă pune la dispoziţie cinci metode pentru a parcurge înregistrările unui set de înregistrări:

Metoda	Descriere
`MoveNext`	Înregistrarea următoare devine cea curentă
`MovePrevious`	Înregistrarea precedentă devine cea curentă
`MoveFirst`	Prima înregistrare devine cea curentă
`MoveLast`	Ultima înregistrare devine cea curentă
`Move n[, start]`	Inregistrarea curentă devine cea care se află la n înregistrări distanţă (înainte sau înapoi) de cea curentă sau de o anumită poziţie (start).

Parametrul n al metodei Move indică numărul de înregistrări care trebuie să fie parcurse pentru a ajunge la cea dorită. Dacă n > 0, parcurgerea se face înainte, dacă n < 0, înapoi. Parametrul opţional *start* reprezintă un semn de carte, adică o anumită poziţie de la care începe deplasarea cu n înregistrări. Dacă acest parametru nu este specificat, se porneşte de la înregistrarea curentă.

Regăsirea numărului de înregistrări ale unui set de înregistrări

Intr-un exemplu anterior am folosit proprietatea *RecordCount* a unui obiect de tip Recordset pentru a-i afla numărul de înregistrări. Setul respectiv era de tip tabelă. Pentru acest tip, Access cunoaşte numărul de înregistrări şi dă proprietăţii RecordCount valoarea corespunzătoare. Lucrurile nu sunt însă la fel de simple şi în cazul altor tipuri de seturi sau înregistrări.
Pentru a mări performanţele aplicaţiei, atunci când creează un obiect Recordset de tip dynaset sau snapshot, Access trece la executarea liniei de cod următor celei ce apelează metoda OpenRecorset imediat după ce a fost regăsită prima interogare. De

aceea, Access nu cunoaşte imediat numărul de înregistrări ale unui astfel de set. Pentru ca acest număr să fie calculat corect, trebuie să apelăm metoda MoveLast, pentru a ne asigura că au fost regăsite toate înregistrările şi apoi să folosim proprietatea RecordCount:

```
Set rst = db.OpenRecordset("IntProfCurs", dbOpenDynaset)
rst.MoveLast
Debug.Print rst.RecordCount
```

Notă: Dacă setul de înregistrări conţine măcar o înregistrare, nu mai trebuie şă apelăm metoda MoveLast, pentru că Access nu trece la următoarea linie de cod până când a fost returnată prima înregistrare a setului. De aceea, dacă proprietatea RecordCount are valoarea zero, e clar că setul nu conţine înregistrări.
Dacă adăugăm sau ştergem înregistrări dintr-un dynaset, proprietatea RecordCount a obiectului respectiv se modifică în consecinţă. In cazul unei aplicaţii multiuser, dacă alţi utilizatori adaugă sau şterg înregistrări din tabela din care îşi ia setul datele, aceste modificări nu sunt reflectate imediat, astfel încât va trebui să reinterogaţi şi să apelaţi metoda MoveLast înainte de a folosi proprietatea RecordCount.

Poziţia absolută şi poziţia procentuală

Pentru a parcurge un set de înregistrări sau pentru a afla poziţia înregistrării curente, putem folosi proprietăţile *AbsolutePosition* (poziţia absolută) şi *PercentPosition* (poziţia procentuală) a obiectului de tip Recordset.
Proprietatea *AbsolutePosition* reprezintă poziţia înregistrării curente în cadrul setului (faţă de zero), iar dacă nu există o înregistrare curentă, valoarea ei este -1. Ea e disponibilă numai pentru seturi de tip dynaset sau snapshot.

Exemplu: Pentru ca înregistrarea curentă să devină cea de-a 4-a înregistrare din set, scriem:

```
rst.AbsolutePosition = 3
```

Proprietatea PercentPosition indică poziţia înregistrării curente cu procentaj din numărul total de înregistrări ale setului şi poate avea valori între 0.0 şi 100.00. Astfel, pentru a ne deplasa până la jumătatea setului, putem scrie:

```
rst.AbsolutePosition = 50
```

Pentru a ne asigura că proprietatea PercentPosition returnează o valoare corectă trebuie să reinterogăm setul şi apoi să apelăm metoda MoveLast, din aceleaşi motive prezentate anterior pentru proprietatea RecordCount.

Poziţiile de început şi de sfârşit ale setului de înregistrări (BOF şi EOF)

Orice set de înregistrări are două proprietăţi (BOF şi EOF) care arată dacă înregistrarea curentă se află la începutul setului (BOF) sau la sfârşitul lui (EOF).
Dacă prima înregistrare este cea curentă şi apelăm metoda *MovePrevious*, proprietatea BOF ia valoarea True şi înregistrarea curentă nu mai există. Dacă mai apelăm încă o dată metoda *MovePrevious*, valoarea lui BOF rămâne TRUE, dar va fi generată o eroare.
Dacă ultima înregistrare este cea curentă şi apelăm metoda *MoveNext*, proprietatea EOF ia valoarea True şi nu mai există o înregistrare curentă. Dacă apelăm din nou metoda *MoveNext*, valoarea lui EOF rămâne True, dar va fi generată o eroare.

Regăsirea valorilor câmpurilor unui set de înregistrări

Pentru a afla valorile diferitelor câmpuri ale unui set de înregistrări, corespunzătoare înregistrării curente, putem folosi una dintre următoarele trei variante de sinteză:

Sintaxa	Exemplu
rst!camp	rst!Nume
rst("camp")	rst("Nume")
rst(indice)	rst(1)

Într-un set de înregistrări, primul camp are întotdeauna indicele 0, indiferent de opţiunea *Option Base* pe care am specificat-o. Astfel, rst(1) se referă la cea de-a doua coloană a setului rst.

*Exemplu: D*orim să afişăm numele şi catedrele tuturor profesorilor din tabela Profesor.

```
Public Sub Nume_Catedra()
Dim db As Database
Dim rst As Recordset
```

```
Set db = CurrentDb()
Set rst = db.OpenRecordset ("Profesor")
With rst
      'verificam daca rst contine înregistrari
      If .RecordCount Then
         .MoveLast
         'parcurgem setul de la ultima catre prima înregistrare
         Do Until   .BOF
            Debug.Print  ![Nume],  ![Catedra]
            .MovePrevious
         Loop
       End If
            .Close
End With
End Sub
```

De multe ori vom dori să avem un acces mai rapid la un set de date pe care nu dorim să le modificăm, ci numai să le vizualizăm. Pentru aceasta, putem crea un set de tip snapshot cu opţiunea *dbForwardOnly*, care este foarte rapid, dar nu permite decât parcurgerea înainte a setului. O modalitate de a folosi rapiditatea unui astfel de set de înregistrări şi de a putea, în acelaşi timp, să accesăm în orice ordine datele pe care acesta le conţine, este să copiem datele într-o matrice. Nu trebuie decât să declarăm o variabilă de tip Variant şi să apelăm metoda GetRows() a setului. Access va crea o matrice bidimensională, pe care o va dimensiona corespunzător. Putem deci rescrie procedura Nume_Catedra astfel încât să folosim un snapshot şi o matrice:

```
Public Sub Nume_Catedra_1()
Dim db As Database
Dim rst As Recordset
Dim varDate As Variant
Dim iNrInreg As Integer
Dim I As Integer
Set db = CurrentDb()
Set rst = db.OpenRecordset("Profesor", dbOpenSnapshot,
      dbForwardOnly)
'alegem un numar sufficient de mare de inregistrari
varDate = rst.getRows(1000)
'aflam indicele ultimei linii a matricei
iNrInreg = UBound(varDate,  2)
'tiparim datele coloanei a 3-a (Nume) şi a 4-a (Catedra)
For I = iNrInreg To 0 Step -1
      Debug.Print varDate(2,   i),     varDate(3,   i)
Next i
rst.Close
End Sub
```

Căutarea înregistrărilor

Atunci dacă dorim să regăsim numai o anumită înregistrare (sau un set de înregistrări ce satisfac un anumit criteriu), vom folosi una dintre metodele Seek sau Field ale setului respectiv.

Căutări în seturi de înregistrări de tip tabelă

După ce am creat un obiect Recordset de tip tabelă, cea mai rapidă modalitate de a regăsi anumite înregistrări este apelarea metodei Seek. (Apelarea acestei metode pentru un set de înregistrări care nu este de tip tabelă va genera o eroare). Pentru aceasta, trebuie să efectuaţi doi paşi:

1. Stabiliţi valoarea proprietăţii Index a setului, deoarece căutările cu metoda Seek se fac numai pe indecşi.
2. Apelaţi metoda Seek dându-i ca argumente un operator de comparaţie (<,<=,>= sau >) şi una sau mai multe valori care să reprezinte criteriile de căutare (în funcţie de numărul coloanelor ce intră în componenţa indexului). Valorile trebuie să corespundă tipurilor de date ale coloanelor indexului.

După ce aţi folosite metoda Seek pentru a căuta o înregistrare, trebuie să verificaţi dacă aţi găsit ceva. Pentru aceasta, veţi verifica valoarea proprietăţii NoMatch a setului de instrucţiuni.
Metoda Seek returnează numai prima înregistrare care satisface criteriul specificat, chiar dacă aceasta nu e singura.
Exemplu: dorim să aflăm nota obţinută de un student la o anumită materie. Pentru aceasta, vom crea un set de înregistrări bazat pe tabela *Curs_Student* şi vom apela metoda *Seek* pentru ei. In acest scop, vom folosi drept index cheia primară a tabelei, care e formată din coloanele NrMatricol şi IdCurs.

```
Public Sub SeekStud_Curs (iNrMat As Integer, iIdCurs As Integer)
Dim db As Database
Dim rst As Recordset
Set db = CurrentDb()
Set rst = db.OpenRecordset("Curs_Stud")
'stabilim proprietatea Index a setului
Rst.Index = "PrimaryKey"
'cautam inregistrarea pt. care IdCurs =iIdCurs si NrMatricol=iNrMat
Rst.Seek "=", IdCurs, iNrMat
If rst.NoMatch = True Then
      ' nu a fost gasita o astfel de inregistrare
      Debug.Print "Studentul cu nr.matricol " & iNrMat & vbCrLf & _
      "nu s-a înscris la cursul " & iIdCurs
Else
      Debug.Print "Nota " & rst!Nota
End If
rst.Close
End Sub
```

Căutări în seturi de înregistrări de tip dynaset şi snapshot

Spre deosebire de seturile de tip tabelă, cele dynset şi snapshot nu pot folosi metoda Seek pentru căutări. Atât pentru acestea din urmă, cât şi pentru căutări după coloane neindexate, Access pune la dispoziţie patru metode de căutare a datelor:

Metoda	Descriere
FindFirst	Incepând cu prima înregistrare a setului de înregistrări, caută o înregistrare ce verifică un criteriu dat. Dacă există, înregistrarea găsită devine cea curentă.
FindLast	Incepând cu ultima înregistrare a setului de înregistrări, caută o înregistrare ce verifică un criteriu dat. Dacă există, înregistrarea găsită devine cea curentă.
FindNext	Incepând cu înregistrarea curentă, caută următoarea înregistrare ce verifică un criteriu dat. Dacă există, înregistrarea găsită devine cea curentă.
FindPrevious	Incepând cu înregistrarea curentă, caută precedenta înregistrare ce verifică un criteriu dat. Dacă există, înregistrarea găsită devine cea curentă.

Dacă oricare dintre aceste metode nu reuşeşte să găsească o înregistrare care să verifice criteriul respectiv, setul de înregistrări nu va mai avea o înregistrare curentă şi orice operaţie ulterioară asupra înregistrării curente va genera o eroare.
Sintaxa acestei metode este următoarea:

```
rst.{FindFirst |FindPrevious|FindNext|FindLast}  criteriu
```

unde rst este un obiect de tip dynaset sau snapshot, iar criteriu este un şir de caractere ce conţine o expresie de genul celor din clauza WHERE a unei instrucţiuni SQL:

```
rst.FindFirst "NrMatricol = " & nMatr & " AND Nota >= 5"
```

Exemplu: Subrutina FindCursuriPromovate() caută toate cursurile la care un student a obţinut o notă mai mare sau egală cu 5. Dacă nu există nici o înregistrare care să întrunească acest criteriu, vom afişa un mesaj. Cu ajutorul variabilei nRez ţinem minte dacă au fost găsite 0, una sau mai multe înregistrări, pentru a afişa un mesaj potrivit pentru fiecare situaţie.

```
Public Sub FindCursuriPromovate (nMatr As Integer)
Dim db As Database
Dim rst As Recordset
Dim nPromovate As Integer
Dim strRez As String
Set db = CurrentDb()
Set rst = db.OpenRecordset("Curs_Student", dbOpenSnapshot)
Rst.FindFirst "NrMatricol = " &nMatr & "AND Nota >= 5"
Do While rst.NoMatch = False
      nPromovate = nPromovate + 1
      strRez = strRez & vbCrLf & rst!IdCurs
      rst.FindNext "NrMatricol = " & nMatr & "AND Nota >= 5"
Loop
Select Case nPromovate
```

```
      Case 0
            Debug.Print "Studentul cu nr. matricol " & nMatr & vbCrLf &_
      "nu a promovat nici un curs"
          Case 1
            Debug.Print "Studentul cu nr. matricol " & nMatr & vbCrLf &_
      "nu a promovat nici un curs"
         Case 1
            Debug.Print "Studentul cu nr. matricol " & nMatr & vbCrLf &_
      "a promovat cursul: " & rst!IdCurs
        Case Else
             Debug.Print "Studentul cu nr. matricol " & nMatr & vbCrLf &_
      "a promovat cursurile" & strRez
End Select
rst.Close
End Sub
```

Atunci când dorim să creăm criterii pentru una dintre metodele *Find* sau atunci când creăm şiruri de caractere ce conţin instrucţiuni SQL, va trebui să introducem într-un asemenea şir valoarea unei variabile. Access cere ca această variabilă să fie inclusă în şir între nişte separatori ((") pentru şiruri de caractere şi (#) pentru date).
Exemplu: Presupunem că avem o variabilă, strNume, de tip String, care conţine un nume pe care să-l căutăm într-un set de înregistrări cu ajutorul metodei FindFirst şi, la un moment dat, valoarea ei este "Popescu". Şirul de caractere care constituie criteriul de căutare este:

```
(Nume) = "Popescu"
```

Pentru a introduce caracterul (") într-un şir de caractere, trebuie să îl dublăm. Tinând cont de aceasta, şirul de caractere care ne dă criteriul de mai sus este:

```
"[Nume] = ""Popescu"""
```

Separând şirul "Popescu", care este aici doar un caz particular, vom scrie:

```
"[Nume] = """ & "Popescu" & """"
```

Inlocuind acum valoarea "Popescu" cu variabila strNume, şirul de caractere care ne dă criteriul dorit:

```
"[Nume] = """ & strNume & """"
```

Acelaşi lucru este valabil şi pentru date. Astfel, pentru a specifica următorul criteriu:

```
[Data Angajarii] = varData
```

unde varData este o variabilă de tip Date, vom construe următorul şir de caractere:

```
"[Data Angajarii] = #" & varData & "#"
```

Data pe care o conţine variabila varData trebuie să fie în formularul mm/dd/yy, altfel, nu vom obţine rezultatul dorit. Pentru a converti o dată la formatul mm/dd/yy, puteţi folosi funcţia Format:

```
Format (varData,  "mm/dd/yy")
```

Pentru criterii ce conţin valori numerice, totul este mai simplu, astfel că pentru a avea următorul criteriu: `[IdCurs] = nCurs`

vom scrie şirul de caractere: `"[IdCurs] = " & nCurs`

Semne de carte

In plus faţă de metodele pentru parcurgerea unui set de înregistrări, Access pune la dispoziţie şi proprietatea Bookmark (semn de carte) a unui astfel de obiect. Fiecare set are o singură înregistrare curentă. Un Bookmark este o proprietate a cărei valoare identifică unic înregistrarea curentă a setului la un moment dat. Putem atribui această valoare unei variabile de tip String sau Variant pentru a o folosi ulterior la regăsirea înregistrării respective. Mai exact:

1. Regăsim valoarea proprietăţii Bookmark a înregistrării curente şi o stocăm într-o variabilă de tip String sau Variant.
2. Pentru a ne întoarce ulterior la acea înregistrare, dăm proprietăţii Bookmark valoarea stocată anterior.

Astfel, putem păstra oricâte semne de carte.
Lucrul cu semn de carte este metoda cea mai rapidă de deplasare între înregistrările unui set.

In mare, iată cum vom folosi în cod proprietatea Bookmark a unui set de înregistrări:

```
Dim strBM As String
strBM = rst.Bookmark
'ne deplasam la ultima inregistrare
rst.MoveLast
'…efectuăm diferite operaţii şi apoi revenim de unde am plecat
rst.Bookmark = strBM
```

De multe ori vom dori poate, să comparăm două semne de carte pentru a vedea dacă înregistrarea curentă este aceeaşi cu cea pentru care am salvat un semn de carte. Pentru aceasta, trebuie să ştim că, deşi putem stoca un semn de carte într-o variantă de tip String sau Variant, ea este reprezentată intern ca o matrice de biţi. De aceea, atunci când comparăm două semne de carte, comparaţia trebuie făcută pe biţi. Pentru aceasta, putem folosi funcţia predefinită StrComp, care returnează valoarea 0 dacă cele două variabile comparate sunt egale şi care primeşte un al treilea argument (primele două sunt variabile de comparat), ce specifică modul în care se va face comparaţia: 0 pentru comparaţie pe biţi, 1 pentru comparaţie obişnuită (care nu e senzitivă, adică nu ţine cont de majuscule/minuscule), 2 pentru a folosi tipul de comparaţie specificat de opţiunea Option Compare de la secţiunea (General) (Declarations) a modulului. De exemplu, dacă strBM1 şi strBM2 sunt două semne de carte stocate anterior,

```
nRez = StrComp(strBM1,  strBM2,   0)
```

nRez va avea valoarea 0 dacă cele două semne de carte coincide şi o valoare diferită de zero, altfel.

Seturile de înregistrări bazate pe tabele native Access acceptă proprietatea Bookmark, spre deosebire de cele bazate pe alte baze de date (Paradox, de exemplu). De aceea, înainte de a folosi semne de carte pentru un obiect de tip Recordset, este bine să testaţi dacă el acceptă această proprietate. In acest scop, verificaţi dacă proprietatea Bookmarkable a setului are valoarea True.

Metoda Clone (clonare)

Orice set de înregistrări are o singură înregistrare curentă. Dacă dorim să lucrăm cu două înregistrări curente pe acelaşi set de date, putem folosi metoda Clone pentru a crea o clonă a setului. Astfel, vom avea două obiecte de tip Recordset indicând spre acelaşi set de înregistrări. Această metodă este mult mai rapidă decât dacă am crea un al doilea set bazat pe aceleaşi date. Atunci când lucrăm cu clone, ţinem cont de următoarele două probleme:

1. Un set de înregistrări creat cu metoda Clone nu are de la început o înregistrare curentă. Pentru ca o înregistrare a sa să devină cea curentă, apelăm la una dintre metodele Find sau Move sau dăm proprietăţii Bookmark o valoare regăsită din setul original.
2. Folosirea ulterioară a metodei Clone pentru setul original sau pentru cel clonat nu îl afectează pe cel clonat sau, pe cel original.

Proprietatea RecordsetClone

Dacă metoda Clone este folosită mai ales pentru a lucra cu două înregistrări curente ale aceluiaşi set de date, vom folosi proprietatea RecordsetClone pentru a avea acces la setul de înregistrări al unui formular. Acest lucru este util atunci când dorim să manipulăm setul de date pe care se bazează un formular, fără ca acţiunile noastre să fie vizibile pe formular.
Pentru a ilustra cum se lucrează cu metoda Clone şi cu proprietatea RecordsetClone, vom scrie funcţia ComparaInreg() ce verifică dacă două înregistrări succesive ale unui set de înregistrări au aceeaşi valoare pentru un anumit camp. Funcţia primeşte ca argumente variabila frm de tip Form ce va identifica formularul şi variabila strCamp de tip String (ce conţine numele câmpului respectiv) şi returnează valoarea True dacă înregistrarea curentă şi cea anterioară conţin aceeaşi valoare a câmpului identificat de argumentul strColoana şi False, altfel.

```
Function ComparaInreg(frm As Form, strCamp As String) As Boolean
Dim rst As Recordset
Dim rstClona As Recordset
'obţinem un pointer la setul de înregistrări al formularului
Set rst = frm.RecordsetClone
'sincronizăm setul astfel obţinut cu originalul
rst.Bookmark = frm.Bookmark
```

```
'pentru a putea lucra cu două înregistrări curente,
'clonăm setul obţinut anterior şi le sincronizăm
Set rstClona = rst.Clone
rstClona.Bookmark = rst.Bookmark
'dacă înregistrarea curentă este prima, returnăm valoarea False
rstClona.MovePrevious
If rstClona.BOF  Then
   ComparaInreg = False
Else
      'dacă înregistrarea curentă nu este prima, efectuăm comparaţia cu cea precedentă ei
      'şi returnăm rezultatul
      ComparaInreg = (rst(strCamp) = rstClona(strCamp))
End If
rstClona.Close
End Function
```

Inregistrarea curentă a setului de înregistrări clonat (rstClona) va fi întotdeauna cea precedentă înregistrării curente a setului rst. Aşa putem lucra cu două înregistrări curente pentru a compara valorile câmpurilor unui set, fără a avea nevoie de variabile în care să păstrăm valorile regăsite. Această funcţie poate fi, de exemplu, apelată din cadrul procedurii de tratare a evenimentului Current a unui formular, pentru a afişa un anumit mesaj sau pentru a efectua diferite alte acţiuni în funcţie de valoarea returnată.

Sortarea şi filtrarea unui set de înregistrări

Pentru a sorta un set de înregistrări de tip tabelă trebuie să-i stabilim proprietatea Index. Atunci când creăm un index pe o tabelă, putem specifica ordinea de sortare crescătoare sau descrescătoare pentru indexul respectiv. Setul va fi sortat exact după ordinea specificată de index.

Pentru a crea un set de înregistrări sortat de tip dynaset sau snapshot, putem proceda în două moduri:

1. Să creăm setul pe baza unei instrucţiuni SQL ce conţine clauza ORDER BY:

```
Set rst = db.OpenRecordset ("SELECT " FROM  Profesor ORDER BY Nume;")
```

2. Să stabilim proprietatea Sort pentru dynaset sau snaphot şi apoi să creăm un nou set pe baza celui sortat.

Valoarea proprietăţii Sort este un şir de caractere conţinând coloana după care se va face sortarea şi, opţional, ordinea de sortare (ordinea ascendentă este implicită):

```
rst.Sort = "[Nume]"
rst.Sort = "[Nume] Asc"   'ascendent
rst.Sort = "[Nume] Desc"  'descendent
```

Exemplu: Creăm un set de înregistrări (rst), îl sortăm şi apoi creăm unul nou (rstSortat) pe baza lui:

```
Dim db As Database
Dim rst As Recordset
Dim rstSortat As Recordset
Set db = CurrentDb()
Set rst = db.OpenRecordset("Profesor",   dbOpenDynaset)
rst.Sort = "[Nume]"
Set rstSortat = rst.OpenRecordset()
'…efectuăm diferite operaţii cu rstSortat
rst.Clone
rstSortat.Clone
```

Pentru a filtra un set de înregistrări avem de asemenea două posibilităţi:

1. Să creăm pe baza unei instrucţiuni SQL ce conţine clauza WHERE. Această metodă poate fi folosită numai pentru a crea seturi unui obiect din baza de date.
2. Să stabilim proprietatea *Filter* a unui dynaset sau snapshop pentru a-i restrânge numărul de înregistrări şi apoi să creăm un nou set pe baza acestuia.

Valoarea proprietăţii *Filter* trebuie să fie un şir de caractere asemănătoare unei expresii din clauza WHERE a unei înregistrări SELECT.

Exemplu: Se creează câte un set de înregistrări folosind cele două metode mai sus amintite:

```
Dim db As DataBase
Dim rst As Recordset
Dim rst SQL As Recordset
Dim rstFiltrat As Recordset
```

```
Set db = CurrentDb()
'folosim prima metoda
Set rstSQL = db.OpenRecordset("SELECT FROM Profesor WHERE
 [Catedra]_ = ""Matematici"";")
'folosim a doua metoda
Set rst = db.OpenRecordset("Profesor", dbOpenDynaset)
rst.Filter = "[Catedra] = ""matematici"""
Set rstFiltrat = rst.OpenRecordset()
```

Atnci când folosim proprietatea *Sort* sau *Filter* a unui set de înregistrări, ţineţi cont de următoarele:

- ele nu se aplică seturilor de tip tabelă;
- sortarea sau filtrarea sunt numai pentru setul creat pe baza celui sortat sau filtrate;
- crearea seturilor pe baza instrucţiunilor SQL poate fi mai rapidă decât folosirea proprietăţilor Sort sau Filter.

Editarea înregistrărilor unui set de înregistrări

Pentru a modifica datele unui set de înregistrări trebuie, mai întâi, să avem dreptul să facem asta. Seturile de tip tabelă sau dynaset pot fi modificate numai dacă altcineva nu a blocat tabela respectivă (în cazul aplicaţiilor multiuser). Seturile snapshot şi seturile bazate pe interogări de tip Crosstab sau Union nu permit modificarea datelor. Pentru a vă asigura că puteţi modifica datele unui set de înregistrări, verificaţi dacă proprietatea sa Updatable are valoarea True.
Access vă pune la dispoziţie cinci metode pentru editarea datelor dintr-un set de înregistrări:

Metoda	Descriere
Edit	Copiază înregistrarea curentă într-un buffer pentru a permite editarea
AddNew	Creează o nouă înregistrare într-un buffer
Update	Salvează modificările din buffer
CancelUpdate	Goleşte bufferul fără a salva modificările
Delete	Sterge înregistrarea curentă.

Pentru a vedea dacă în buffer există vreo înregistrare care nu a fost salvată, putem verifica valoarea proprietăţii EditMode, care poate fi:

Valoarea	Constanta intrinsecă	Descriere
0	dbEditNome	Bufferul e gol
1	dbEditInProgress	Bufferul conţine înregistrarea curentă (s-a apelat metoda Edit)
2	dbEditAdd	Bufferul conţine o înregistrare nouă (s-a apelat metoda AddNew)

Modificarea înregistrării curente

Pentru a edita înregistrarea curentă a unui set de înregistrări (pp. că putem face acest lucru), efectuăm următoarele operaţii:

1. Apelăm metoda Edit pentru a copia înregistrarea curentă în buffer.
2. Efectuăm modificările dorite.
3. Apelăm metoda Update pentru a salva modificările.

Exemplu: Să presupunem că în tabela professor s-a introdus greşit numele unui profesor: în loc de "Popescu Marian" s-a introdus "Popescu Marin". Pentru a rectifica din program această eroare, creăm setul rst bazat pe tabela Profesor şi scriem:

```
With rst
'căutăm înregistrarea pe care dorim s-o editäm
.FindFirst "[Nume] = ""Popescu Marin"""
'dacă nu o găsim, afisam un mesaj
If   .NoMatch   Then
    MsgBox "Popescu Marin nu se află în tabela Profesor"
'altfel, o editam
Else   .Edit
      ![Nume] = "Popescu Marian"
      .Update
End If
End With
```

Adăugarea unei noi înregistrări la un set de înregistrări

Pentru a adăuga o nouă înregistrare la un set de (presupunând că puteţi face acest lucru), procedăm astfel:

1. Apelăm metoda AssNew pentru a adăuga o nouă înregistrare (valorile câmpurilor vor fi cele implicite, dacă există valori implicite).

2. Introducem valorile câmpurilor.
3. Apelăm metoda Update pentru a salva înregistrarea.

Inregistrarea curentă, după ce am adăugat o nouă înregistrare, va fi tot cea dinaintea acestei operaţii. De aceea, pentru ca înregistrarea curentă să fie cea nouă, apelăm metoda Move cu valoarea proprietăţii LastModified a setului ca argument.

Exemplu: Se adaugă o nouă înregistrare la un set bazat pe tabela Profesor şi face ca înregistrarea curentă să fie cea nouă.

```
With  .rst
      .AddNew
            ![Nume] = "Toma Dorel"
            ![Catedra] = "Fizica"
            ![IdTitlu] = 2
      .Update
      .Move 0,  .LastModified
End With
```

Stergerea unei înregistrări a unui set de înregistrări

Pentru a şterge înregistrarea curentă a unui set nu trebuie decât să apelăm metoda Delete. După ce am şters o înregistrare, ea continuă să fie cea curentă. Apelaţi, de exemplu, metoda MovePrevious pentru ca înregistrarea precedentă să devină cea curentă.
Pentru a şterge înregistrarea corespunzătoare profesorului "Popescu Marin" dintr-un set bazat pe tabela Profesor, scriem:

```
With  .rst
      .FindFirst "[Nume] = ""Popescu Marin"""
      If  .NoMatch Then
         MsgBox "Popescu Marin nu se află în tabela Profesor"
      Else
         .Delete
         .MovePrevious
      End If
End With
```

3. Obiecte speciale

Există anumite proprietăţi ale unor obiecte care reprezintă, la rândul lor, alte obiecte. Cunoastem deja proprietăţile Me şi RecordsetClone.
Tabelul următor prezintă alte proprietăţi ce reprezintă obiecte.

Proprietatea	A cui este	Ce reprezintă
ActiveControl	Ecran (Screen)	Controlul care are focusul.
ActiveForm	Ecran (Screen)	Formularul care are focusul sau care conţine controlul cu focus.
ActiveReport	Ecran (Screen)	Raportul care are focusul sau care conţine controlul cu focus.
Form	Subformular, formular sau control	Pentru un control de tip Dubform, reprezintă subformatul găzduit. Pentru un formular, reprezintă formularul însăşi.
Me	Formular sau raport	Insuşi formularul sau raportul respectiv.
Module	Formular sau raport	Modulul asociat formularului sau raportului respectiv.
Parent	Control	Formularul sau raportul ce conţine controlul respetiv.
PreviousControl	Formular	Controlul care a avut anterior focusul.
RecordsetClone	Subraport, raport sau control	Setul de înregistrări pe care se bazează formularul.
Report		Pentru un control de tip Dubreport, reprezintă subraportul găzduit. Pentru un raport, reprezintă raportul însuşi.
Section	Control	Secţiunea formularului sau raportului în care se află controlul.