Ngôn ngữ Access Basic và việc thiết kế Module

Giới thiệu chung

- Ngôn ngữ Access Basic là một ngôn ngữ lập trình hoàn thiện được tích hợp trong hệ quản trị CSDL Access 2007, cho phép xây dựng các hàm, thủ tục xử lý dữ liệu và các sự kiện trên các đối tượng trong CSDL Access
- Các thành phần cơ bản:
  - Các kiểu dữ liệu (bao gồm các kiểu cơ bản được định nghĩa sẵn như các kiểu dữ liệu số, chuỗi, Date/Time, Logic, ... và các kiểu dữ liệu do người dùng tự định nghĩa (User-defined Type))
  - Biến, hằng
  - Các toán tử và các cấu trúc điều khiển
  - Các thủ tục/hàm
- Trong Access Basic sử dụng các khái niệm của lập trình hướng đối tượng như các lớp đối tượng, phương thức, thuộc tính

Chương 1: Các khái niệm cơ bản

1. Đặc điểm của chương trình viết bằng ngôn ngữ Access

- Các chương trình viết bằng ngôn ngữ Access Basic được tổ chức và hoạt động trong môi trường của hệ quản trị CSDL MS Access nhằm xây dựng các ứng dụng quản trị dữ liệu phức tạp, hoàn chỉnh hơn
- Chương trình được xây dựng dựa trên nền CSDL Access đã được tạo sẵn với các đối tượng: Table, Query, Report, Form, Macro
- Không tổ chức thành một chương trình thống nhất như các chương trình viết bằng ngôn ngữ C/Pascal mà bao gồm các thủ tục, hàm độc lập nằm rải rác trong các module gắn liền với các đối tượng của CSDL

1. Đặc điểm của chương trình viết bằng ngôn ngữ Access

- **Tổ chức của chương trình:** gồm nhiều module khác nhau
- Các Module dùng chung
- Các Module riêng của mỗi Form/Report
- **Hoạt động của chương trình:** không tuần tự như các chương trình lập trình theo cấu trúc hệ thống mà tuân theo thứ tự hoạt động của các sự kiện (có nhiều loại sự kiện: sự kiện trên Form, sự kiện trên Report, sự kiện trên các ô điều khiển của Form và Report)

→Khi có sự kiện xảy ra thì một chuỗi các hàm, thủ tục xử lý sự kiện sẽ được kích hoạt. Khi không có sự kiện xảy ra thì toàn bộ chương trình ở chế độ nằm chờ

2. Module

- **Module**: tập các hàm/thủ tục xử lý dữ liệu/sự kiện
- **Cấu trúc Module dùng chung**
  - Các tuỳ chọn:
    + Option base 1 (tùy chọn chỉ số mảng từ 1, nếu không có thì từ 0)
    + Option explicit (tùy chọn các biến - bắt buộc phải khai báo khi sử dụng)
    + Option compare text (tùy chọn so sánh văn bản - không phân biệt chữ hoa chữ thường)
    + Option compare binary (tùy chọn so sánh nhị phân - có phân biệt chữ hoa chữ thường)
  - Khai báo các hằng, biến, mảng cấp chương trình (dùng từ khoá Global)
  - Khai báo các hằng, biến, mảng cấp module (dùng từ khoá Dim)
  - Các thủ tục với phạm vi sử dụng cấp chương trình
  - Các thủ tục của riêng module (nếu dùng từ khoá Private)

Lưu ý: Phạm vi sử dụng của các thủ tục trong module dùng chung là toàn chương trình

2. Module

- **Cấu trúc Module của riêng Form/Report**
  - Các tuỳ chọn:
    + Option base 1
    + Option explicit
    + Option compare text
    + Option compare binary
  - Khai báo các hằng, biến, mảng cấp module
  - Các thủ tục xử lý sự kiện
  - Thủ tục tổng quát (chỉ được dùng trong module) được gọi bởi thủ tục xử lý sự kiện hoặc các thủ tục tổng quát khác trong cùng module

Lưu ý: Các thủ tục xây dựng trong module của riêng Form/Report chỉ có phạm vi sử dụng trong riêng Form/Report đó

3. Thủ tục, hàm

- **Cấu trúc của một thủ tục**

```
Sub Tên_thủ_tục(khai báo từng đối số)
    . . .
End Sub
```

- **Lời gọi thủ tục:**

```
Tên_thủ _tục (Danh sách tham số)
```

- **Cấu trúc của một hàm**

```
Function Tên_hàm(khai báo từng đối) [as kiểu]
    . . .
    Tên_hàm = giá trị
End Function
```

4. Quy ước viết lệnh và chú thích

- Mỗi câu lệnh phải viết trên một dòng. Trường hợp ghép nhiều câu lệnh trên cùng một dòng thì cần sử dụng dấu : để ngăn cách
- Để viết chú thích: sử dụng cú pháp REM hoặc thêm dấu nháy đơn ' vào trước phần chú thích
- Ví dụ:

- Thủ tục đảo chuỗi

```
Sub Dao_chuoi(S as string)
   Dim P as string, i as integer, n as integer
   n=len(S) : P=""
   For i=n to 1 step -1
     P=P & Mid(S,i,1)
   Next
   Debug.Print P
End Sub
```


5. Dịch, kiểm tra lỗi và chạy thử các hàm/thủ tục

- Các thủ tục, hàm được biên soạn trong các module của Form/Report hoặc trong các module dùng chung
- Trong quá trình biên soạn các hàm/thủ tục, nếu chuyển con trỏ sang dòng khác thì MS Access sẽ kiểm tra cú pháp của dòng lệnh vừa soạn thảo và đưa ra thông báo sai nếu có
- Để chạy thử một hàm/thủ tục trong chế độ soạn thảo, ta sử dụng chức năng Immediate Window của menu View

Chương 2: Kiểu dữ liệu, Hằng, Biến, Mảng, các Phép toán và các Biểu thức

1. Các kiểu dữ liệu cơ bản

- Trong Access Basic có các kiểu dữ liệu cơ bản sau:
  - Integer
  - Long
  - Single
  - Double
  - Currency
  - String
  - Variant

→ Kiểu Variant có thể giá trị thuộc về một trong số các kiểu số, ngày/giờ, chuỗi hoặc null

2. Biến

- **Quy tắc đặt tên biến (tương tự với hàm/thủ tục)**
  - Bao gồm các chữ cái, chữ số và dấu gạch nối
  - Ký tự đầu phải là ký tự chữ
  - Không được trùng với các từ dành riêng của Access
  - Độ dài không quá 40 ký tự

2. Biến

- **Khai báo ngầm định các biến**

- Nếu trong phần Declaration Section ta có sử dụng tùy chọn **Option explicit** thì không được phép khai báo các biến ngầm định, có nghĩa là các biến phải khai báo tường minh bằng câu lệnh Dim hoặc Global

- Nếu không sử dụng tùy chọn Option explicit, thì có thể sử dụng các biến ngầm định

Cách khai báo các biến ngầm định: Nếu trong chương trình đưa vào một biến mới thì biến đó coi như được khai báo ngầm định. Biến khai báo ngầm định có kiểu Variant và có thể nhận các giá trị kiểu số, kiểu chuỗi, kiểu Date/Time, giá trị null

2. Biến

- **Định kiểu cho các biến ngầm định:** Có 2 cách

*** Cách 1:** Dùng các ký tự định kiểu viết sau tên biến, đó là các ký tự:

% nếu là kiểu Integer
& nếu là kiểu Long
! nếu là kiểu Single
nếu là kiểu Double
@ nếu là kiểu Currency
$ nếu là kiểu String
Kiểu Variant là kiểu mặc định

Ví dụ: stt%, sl&, sm!, ht$, ts#, td (khi đó td có kiểu mặc định là Variant)

2. Biến

*** Cách 2:** Dùng các toán tử định dạng:

Defkiểu Chữ_cái
Defkiểu Chữ_cái-Chữ_cái

để quy định kiểu của biến theo chữ đầu của biến

Kiểu trong toán tử Defkiểu gồm:

Int (Integer)	Lng (Long)	Sng (Single)
Dbl (Double)	Cur (Currency)	Str (String)
Var (Variant)		

Ví dụ:

DefStr T quy định các biến bắt đầu bằng chữ T có kiểu String

DefInt I - L quy định các biến mà chữ đầu từ I đến L có kiểu Integer

2. Biến

- **Khai báo tường minh các biến**

Cú pháp:

Dim Tên_biến [as Kiểu] [, . . .]

Các kiểu dữ liệu có thể dùng trong khai báo Dim:

Integer Single Long Double Currency String

Ví dụ: Dim stt as Integer, ht as String, x

Khi đó, x có kiểu Variant (kiểu mặc định)

Lưu ý:

- Các biến được khai báo được khởi tạo giá trị là 0 hoặc null
- Kiểu dữ liệu trong câu lệnh Dim sẽ được ưu tiên hơn kiểu quy định trong toán tử DefKiểu
- Ngoài các kiểu dữ liệu chuẩn như: Integer, String, ..., còn có các kiểu đối tượng như: Table, Query, Form, ..., và các kiểu tự tạo

2. Biến

- **Phạm vi sử dụng của biến:** tuỳ thuộc vào vị trí khai báo của biến

Dim (trong thủ tục) chỉ sử dụng cục bộ trong thủ tục

Dim (trong Declaration Section của module) sử dụng trong module

Global (trong Declaration Section của module) sử dụng trong toàn bộ chương trình

- **Biến tĩnh**

- Chiếm một vùng nhớ cố định trong suốt thời gian chương trình làm việc
- Cú pháp khai báo:

Static Tên_biến1 [as kiểu], Tên_biến2 [as kiểu], ...

- Nếu muốn chuyển tất cả các biến cục bộ của một thủ tục thành biến tĩnh (dù chúng được khai báo bằng Dim, Static hay ngầm định) ta đặt từ khoá Static vào trước dòng tiêu đề của thủ tục:

Static tên_thủ_tục ( )

3. Hằng

- Cú pháp khai báo hằng:

[Global] Const tên_hằng = biểu_thức [, . . .]

- Phạm vi sử dụng:

Giống như các biến

4. Mảng

- **Cú pháp khai báo:** Tên_mảng(cs1,cs2,...) [as type]
- **Cách xác định miền giá trị của chỉ số:**

Cách 1: Dùng Option base xác định cận dưới, trong khai báo chỉ ra cận trên

Ví dụ: Option base 1

Dim a(6) (chỉ số từ 1 đến 6)

Cách 2: Dùng cách viết: Cận dưới **to** cận trên

Ví dụ: Option base 1

Dim b(5,-3 to 6)

→Khi đó b là mảng 2 chiều kiểu Variant, trong đó chỉ số thứ nhất chạy từ 1 đến 5, chỉ số thứ 2 chạy từ -3 đến 6

4. Mảng

- **Phạm vi sử dụng của mảng:** Giống như biến
- **Cách xác định kiểu mảng:** Các cách xác định kiểu biến đều có thể dùng để xác định kiểu mảng

Ví dụ: Redim b(2 to 8,10) as string

- **Mảng động**

- Là mảng có thể thay đổi số chiều và kích thước mỗi chiều

- Cách khai báo mảng động: Dùng Dim hoặc Global trong phần Declaration section của module:

tên_mảng( ) [as type]

- Trong thủ tục mỗi khi cần tới mảng dùng Redim để xác định chính xác chiều và kích thước mỗi chiều. Kích thước có thể cho bởi các biến hoặc biểu thức

5. Kiểu dữ liệu do người dùng tự định nghĩa (user-defined data type)

- Tương tự như kiểu bản ghi của Pascal hay kiểu cấu trúc của C, bao gồm nhiều thành phần
- Cú pháp định nghĩa kiểu dữ liệu: (đặt trong phần Declaration Section của Module)

Type Tên_kiểu
Tên_thành_phần as Kiểu
[…]
End type

Trong đó:

- Tên_kiểu, Tên_thành_phần được đặt theo quy tắc đặt tên biến
- Kiểu có thể là: Integer, Long, Single, Currency, String, Variant, hoặc một biến tự tạo khác đã được định nghĩa

5. Kiểu dữ liệu do người dùng tự định nghĩa (user-defined data type)

- Khai báo biến, mảng theo kiểu dữ liệu tự định nghĩa: tương tự như đối với các biến, mảng có kiểu chuẩn
- Truy nhập đến các thành phần của biến, mảng có kiểu dữ liệu tự định nghĩa: sử dụng cú pháp

Tên_biến.Tên_thành_phần
Tên_mảng.Tên_thành_phần

6. Các phép toán

- **Các phép toán số học:** ^ , * , / , \ , mod , + , -
- **Các phép toán so sánh:** > , >= , < , <= , = , <>

Các phép toán so sánh cho kết quả là -1 (nếu đúng), là 0 (nếu sai)
Trong Access Basic định nghĩa 2 hằng: True = -1 và False = 0

- **Phép ghép chuỗi**

Cú pháp: chuỗi_1 & chuỗi_2 & . . . & chuỗi_n
chuỗi_1 + chuỗi_2 + . . . + chuỗi_n
Kết quả là một chuỗi

- **Các phép toán Logic**

Các phép toán logic: and, or, not, xor, eqv, imp
Các phép toán trên có thể thực hiện trên các giá trị logic (-1 và 0), hoặc trên các giá trị nguyên

- **Các toán tử truy nhập đối tượng**

. Truy nhập tới đối tượng chuẩn của Access
! Truy nhập tới đối tượng của người sử dụng

7. Các hàm thường dùng

- **Các hàm dùng cho dữ liệu dạng số:** Int(x), Exp(x), Log(x), Sqr(x), Tan(x), Sin(x), Cos(x), ...
- **Các hàm dùng cho dữ liệu dạng chuỗi:** Len(s), Left[$](s,n), Right[$](s,n), Mid[$](s,[m,]n),... Trong đó s là biểu thức chuỗi
- **Các hàm dùng cho dữ liệu dạng ngày và giờ:** Date(), Date$(), Time(), Time$(), Now(), ...
- **Hàm lựa chọn, tìm kiếm theo mẫu**

- **Hàm lựa chọn:** iif(Điều_kiện, giá_trị_1, giá_trị_2)

→ trả về giá_trị_1 nếu Điều kiện đúng, trả về giá_trị_2 nếu Điều_kiện sai

- **Hàm tìm kiếm theo mẫu:** Biểu_thức_chuỗi Like Mẫu

Mẫu là một tập hợp các chuỗi, có thể chứa các ký tự thế sau: *, ?, #, [ký tự đầu - ký tự cuối],[!ký tự đầu - ký tự cuối]

→trả về kết quả là đúng (-1) nếu Biểu_thức_chuỗi thuộc Mẫu, ngược lại cho kết quả là sai (0)

8. Biểu thức

- Biểu thức: kết hợp các biến, hàm, hằng bởi các phép toán
- Biểu thức trả về một giá trị → kiểu của giá trị chính là kiểu dữ liệu của biểu thức. Có các kiểu: biểu thức số (có giá trị nguyên hoặc thực), biểu thức logic (có giá trị -1 hoặc 0), biểu thức ký tự (có giá trị là ký tự hoặc chuỗi ký tự), ...

Chương 3: Vào ra và các toán tử điều khiển

1. Vào ra

- **Hàm Inputbox và Inputbox$:** dùng để nhận thông tin từ người dùng. Hàm Inputbox cho phép nhập dữ liệu kiểu Variant, hàm Inputbox$ cho phép nhập dữ liệu kiểu chuỗi

Cú pháp:

Inputbox[$](Lời_nhắc[,Tiêu_đề][,GT_mặc_định][,X, Y])

Trong đó:

- Lời_nhắc là chuỗi hướng dẫn việc nhập liệu
- Tiêu_đề là chuỗi hiện trên thanh tiêu đề của hộp hội thoại
- GT_mặc_định là biểu thức chuỗi hiện trên hộp văn bản
- X, Y là tọa độ của hộp hội thoại

Hộp văn bản dùng để chứa dữ liệu nhập vào từ bàn phím. Khi chọn nút OK thì hàm trả về giá trị có trong hộp văn bản. Khi chọn nút Cancel thì hàm trả về chuỗi rỗng

1. Vào ra

Ví dụ: R=Inputbox("Nhập bán kính ","Cửa sổ nhập liệu")

Lưu ý: Để ngắt dòng ta phải sử dụng ký tự chr(10)

- **Hàm MSGBOX và thủ tục MSGBOX:** Thủ tục MSGBOX dùng để đưa một biểu thức chuỗi ra hộp thoại. Hàm MSGBOX dùng để đưa một biểu thức chuỗi ra hộp thoại và nhận 1 giá trị số trả về (để biết được người dùng chọn nút nào trong các nút: OK, Cancel, Retry, ...)

Cú pháp:

Hàm: MSGBOX(nd[, kiểu[, tiêu_đề]])

Thủ tục: MSGBOX nd [,kiểu[, tiêu_đề]]

Trong đó:

- nd là biểu thức chuỗi được đưa ra hộp thoại
- tiêu_đề là biểu thức chuỗi được hiện trong thanh tiêu đề của hộp thoại
- kiểu là một số nguyên để xác định: Các nút sẽ hiển thị, các biểu tượng, nút mặc định được chọn

2. Các toán tử điều khiển

- **Toán tử rẽ nhánh If ... Then**

+ Dạng 1: If điều_kiện Then câu_lệnh

+ Dạng 2:
```
If điều_kiện then
    nhóm_lệnh
End if
```

+ Dạng 3:
```
If điều_kiện_1 then
    nhóm_lệnh_1
Else
    If điều_kiện_2 then
        nhóm_lệnh_2
    Else
        nhóm_lệnh_3
    End if
End if
```


2. Các toán tử điều khiển

- **Toán tử lựa chọn Select Case**

Cú pháp:

```
Select case biểu_thức_nguyên
    case danh_sách_giá_trị_1: nhóm_lệnh_1
    case danh_sách_giá_trị_2: nhóm_lệnh_2
        . . .
    case danh_sách_giá_trị_n: nhóm_lệnh_n
   [case else nhóm_lệnh_(n+1)]
End Select
```


2. Các toán tử điều khiển

- **Toán tử For**

Cú pháp:

```
For biến_đếm = giá_trị_đầu to giá_trị_cuối [step bước]
    nhóm_lệnh
   [Exit for]
    . . .
Next
```

Trong đó:

Bước sau từ khóa Step có thể là dương hoặc âm. Nếu bỏ Step thì bước bằng 1

Toán tử Exit for dùng để thoát khỏi chu trình lặp. Khi ra khỏi chu trình biến_đếm giữ nguyên giá trị

2. Các toán tử điều khiển

- **Toán tử Do ... Loop với điều kiện trước**

Dạng 1: Do While điều_kiện_lặp
nhóm_lệnh
[exit do]
. . .
Loop

Dạng 2: Do Until điều_kiện_dừng
nhóm_lệnh
[exit do]
. . .
Loop

2. Các toán tử điều khiển

- **Toán tử Do ... Loop với điều kiện sau**

Dạng 1:
Do
nhóm_lệnh
[exit do]
. . .
Loop While điều_kiện_lặp

Dạng 2:
Do
nhóm_lệnh
[exit do]
. . .
Loop Until điều_kiện_dừng

Ví dụ:
Tính s = 1*2* . . . *n = n!
→Vòng lặp:
s=1:i=1
Do
s=s*i
i=i+1
Loop While i<= n

2. Các toán tử điều khiển

- **Lệnh dịch chuyển điều khiển của chương trình**

- Chuyển điều khiển tới nhãn:
Goto nhãn
- Chuyển tới đoạn chương trình con:
Gosub nhãn
- Dịch chuyển có lựa chọn:
On biểu_thức Goto/Gosub nhãn_1,nhãn_2,...,nhãn_n
- Dịch chuyển khi có lỗi:
On error Goto nhãn

- **Kết thúc chương trình:** sử dụng từ khóa End

Chương 4: Thủ tục và hàm

1. Phạm vi sử dụng của thủ tục và hàm

- Các thủ tục, hàm trong các module chung được chia làm 2 loại:
  - Thủ tục và hàm cấp chương trình (không dùng từ khoá Private): có thể dùng ở bất cứ chỗ nào trong chương trình. Các công cụ của MS Access có thể dùng các hàm này để tạo các điều kiện lựa chọn khi xây dựng các truy vấn
  - Thủ tục và hàm cấp module (dùng từ khoá Private): chỉ có thể được gọi từ các thủ tục/hàm khác trong cùng module
- Các thủ tục/hàm trong các module của Form/Report: chỉ có thể gọi từ các thủ tục/hàm khác trong cùng module của Form/Report

2. Thủ tục (SUB PROCEDURE)

- **Cú pháp tạo thủ tục:**

[Static][Private] Sub Tên_thủ_tục([danh_sách_đối])
    [các câu_lệnh]
    [exit sub]
    [các_câu_lệnh]
End Sub

Trong đó:

- Từ khoá Static: quy định các biến trong thân thủ tục là biến tĩnh
- Từ khoá Private: quy định phạm vi của thủ tục ở cấp module
- Tên_thủ_tục do người lập trình tự đặt theo quy tắc đặt tên
- Các_câu_lệnh bên trong để thực hiện các chức năng của thủ tục
- Câu lệnh exit sub dùng để thoát khỏi thủ tục từ bên trong thủ tục

2. Thủ tục (SUB PROCEDURE)

- Danh sách đối được khai báo theo mẫu:

[By Val] tên_đối[()] [As Type] [, . . .]

Trong đó:

+ Nếu không dùng By Val thì đối được truyền theo tham chiếu, có thể dùng với đối kiểu bất kỳ

+ Nếu dùng By Val thì đối sẽ được truyền theo giá trị, không được dùng với đối kiểu đối tượng hoặc kiểu tự định nghĩa

+ Tên_đối do người dùng tự đặt theo quy tắc như tên biến. Đối sẽ nhận dữ liệu truyền từ các tham số của lời gọi thủ tục. Với đối mảng mảng ta viết thêm dấu () nhưng không chỉ ra số chiều và độ lớn mỗi chiều

+ As Type: dùng khai báo kiểu cho các đối. Kiểu có thể là Integer, Long, Single, Double, Currency, String (chỉ kiểu chuỗi có độ dài thay đổi), Variant, kiểu tự nghĩa hoặc kiểu đối tượng (không dùng mảng đối tượng)

2. Thủ tục (SUB PROCEDURE)

- **Lời gọi thủ tục:**

Tên_thủ_tục([Danh_sách_các_tham_số])

Lưu ý:

- Danh_sách_các_tham_số phải phù hợp với danh sách các đối được khai báo khi xây dựng thủ tục cả về số lượng, vị trí, kiểu dữ liệu
- Có thể thực hiện lời gọi tới một thủ tục từ trong một hàm/thủ tục khác

3. Hàm (Function Procedure)

- **Cú pháp tạo hàm:**

[Static][Private] Function Tên_hàm ([danh_sách_đối]) [As Type]
[các_câu_lệnh]
[Tên_hàm = Biểu_thức]
[exit Function]
[các_câu_lệnh]
[Tên_hàm = Biểu_thức]
End Function

Trong đó:

+ Các từ khoá Static, Private, Tên_hàm, danh_sách_đối: Ý nghĩa tương tự như trong thủ tục

+ Từ khoá As Type dùng để khai báo kiểu hàm (có thể là các kiểu Integer, Long, Single, Double, Currency, String, Variant). Nếu không khai báo hàm sẽ có kiểu Variant

3. Hàm (Function Procedure)

- **Giá trị trả về của hàm:** Khi không có lệnh gán giá trị trả về cho tên hàm thì hàm nhận giá trị mặc định:
  - Là 0 đối với các hàm kiểu số
  - Là chuỗi rỗng đối với các hàm kiểu chuỗi
  - Là Empty đối với hàm kiểu Variant
- **Lời gọi hàm:** (có thể thực hiện trong một hàm/thủ tục khác)

Tên_hàm(Danh_sách_các_tham_số)

- **Hàm đệ quy:** Xét hàm tính giai thừa

```
Function gt(n as integer) as integer
  If n=0 then gt=1
      Else gt=n*gt(n-1)
  End If
End Function
```


Chương 5: Thực hiện Macro trong chương trình

1. Thực hiện Macro

- Để thực hiện một macro, ta dùng câu lệnh DoCmd sau:

DoCmd.Tên_hành_động [danh sách đối]

Có 2 loại đối:

- Số nguyên biểu thị loại đối tượng trong danh sách được tác động theo quy ước
- Tên của đối tượng

Ví dụ: để đóng bảng có tên là HOCSINH, sử dụng câu lệnh:

```
DoCmd.Close 0, "HOCSINH"
```

2. Đóng các đối tượng

- Sử dụng hành động CLOSE để đóng đối tượng, có 2 đối:

- Loại đối tượng: là số nguyên được lấy trong danh sách quy ước:
  - 0 Table
  - 1 Query
  - 2 Form
  - 3 Report
  - 4 Macro
  - 5 Module
- Tên đối tượng

- Ví dụ: Lệnh đóng Form Timkiem:

  DoCmd.Close 2, "Timkiem"

3. Mở các đối tượng

- **Hành động OpenForm: c**ó 6 đối theo thứ tự:
  + Form name: là tên Form cần mở
  + View: chế độ hiển thị, được chọn trong danh sách sau:
    0 Form 1 Design 2 Print Preview 3 Datasheet
  + Filter name: tên của Filter (dùng để sắp xếp hoặc/và lọc ra các thông tin cần thiết)
  + Where condition: điều kiện chọn lọc thông tin
  + Data mode: chế độ làm việc với dữ liệu, chọn trong danh sách sau: 0 Add 1 Edit 2 Read Only
  + Window mode: chế độ cửa sổ, chọn trong danh sách sau:
    0 Nomal 1 Hidden 2 Icon 3 Dialog

Ví dụ: Để mở Form FrmDiem theo dạng Form để cập nhật dữ liệu điểm cho các học sinh, ta có thể dùng câu lệnh sau:

DoCmd.OpenForm "FrmDiem",0, , ,1,0

3. Mở các đối tượng

- **Hành động OpenTable:** có 3 đối theo thứ tự sau:
  + Table name: tên bảng cần mở
  + View: chế độ hiển thị, chọn trong danh sách sau:
    - 0 Datasheet
    - 1 Design
    - 2 Print Preview
  + Data mode: chế độ làm việc với dữ liệu, được chọn trong danh sách sau:
    - 0 Add
    - 1 Edit
    - 2 Read only

Ví dụ: Để mở bảng HOCSINH để cập nhật dữ liệu, dùng lệnh:
DoCmd.OpenTable "HOCSINH",0,1

3. Mở các đối tượng

- **Hành động OpenQuery:** có 3 đối theo thứ tự sau
  + Query name: tên truy vấn cần mở
  + View: được chọn trong danh sách sau:
    - 0 Datasheet
    - 1 Design
    - 2 Print Preview
  + Data mode: được chọn trong danh sách sau:
    - 0 Add
    - 1 Edit
    - 2 Read only

3. Mở các đối tượng

- **Hành động OpenRort:** có 4 đối theo thứ tự sau:

+ Report name
+ View: được chọn trong danh sách sau:
    0 Datasheet
    1 Design
    2 Print Preview
+ Filter name
+ Where condition

3. Mở các đối tượng

- **Hành động OpenModule:** có 2 đối theo thứ tự sau:

    + Module name
    + Procedure name

Ví dụ: Muốn mở module Thuvien để xem, sửa, bổ sung các hàm, thủ tục của nó, bắt đầu từ thủ tục "Nhapdiem", ta dùng câu lệnh sau:

```
DoCmd.OpenModule "Thuvien","Nhapdiem"
```


4. Cập nhật lại đối tượng

- Để đối tượng thể hiện nội dung mới nhất mà nó vừa thay đổi ta dùng hành động REPAINTOBJECT với 2 đối sau:

    + Object type: Được chọn giống như trong hành động CLOSE
    + Object name: Tên đối tượng

Ví dụ: DoCmd.RepaintObject 3, "Timkiem"

5. Làm việc với các ô điều khiển

- **Đặt con trỏ chuột tại một ô điều khiển trên Form:** dùng lệnh

```
DoCmd.GotoControl "tên_ô_điều_khiển"
```

Ví dụ:

```
DoCmd.GotoControl "Hoten"
```

- **Hiện nội dung vừa thay đổi của một ô điều khiển:**

  Nhiều khi nội dung của 1 ô điều khiển đã thay đổi, nhưng trên mẫu biểu ta chỉ quan sát được nội dung cũ của nó. Để có thể hiện tức thời nội dung mới nhất của một ô điều khiển ta dùng câu lệnh:

```
DoCmd.Requery "tên_ô_điều_khiển"
```

Ví dụ:

```
DoCmd.Requery "Hoten"
```

Chương 6: Đối tượng và biến đối tượng

Kiểu đối tượng và tập đối tượng

- Kiểu đối tượng (Object Type): gồm DBEngine, WorkSpace, DataBase, Form, Report, TableDef, QueryDef, RecordSet, Control, Field, Index
- Đối tượng (Object): Mỗi kiểu đối tượng có nhiều đối tượng cụ thể. Chẳng hạn một biểu mẫu đang mở là một đối tượng kiểu Form. Tên biểu mẫu là tên một đối tượng kiểu Form. Tương tự một bảng là một đối tượng kiểu Tabledef và tên bảng là tên một đối tượng kiểu Tabledef
- Biến đối tượng (Object Variable): Biến đối tượng là biến biểu thị một đối tượng, nó được khai báo bằng cách dùng các kiểu đối tượng nêu trên

Ví dụ: Dim DB As Database ' Khai báo biến DB kiểu Database

- Tập đối tượng (Collection): là tập hợp các đối tượng cùng loại
- Để hiển thị một tuyển tập đối tượng (trừ DBEngine) ta chỉ cần thêm chữ S vào sau tên kiểu đối tượng đã nêu ở trên

Ví dụ: Forms là tập gồm các mẫu biểu của một cơ sở dữ liệu. TableDefS là tập gồm các bảng dữ liệu của một cơ sở dữ liệu

2. Các thành phần của đối tượng

- Các thành phần của đối tượng có thể chia thành 3 loại:
  - Đối tượng con (Sub Object)
  - Thuộc tính (Property)
  - Phương thức (Method)
- Đối tượng khởi thuỷ: là đối tượng không nằm trong bất kỳ đối tượng nào khác

  Có 3 đối tượng khởi thuỷ thường dùng: Forms, Reports, DBEngine
  - Các đối tượng con của Forms là các Form đang mở
  - Các đối tượng con của Reports là các Report đang mở
  - Các đối tượng con của DBEngine là các vùng làm việc (đối tượng kiểu WorkSpace)

2. Các thành phần của đối tượng

- **Quy tắc biểu thị thành phần của đối tượng:**

  Bắt đầu từ một đối tượng khởi thuỷ, sau đó là danh sách các đối tượng có quan hệ phụ thuộc (đối tượng sau là thành phần của đối tượng trước), cuối cùng là thành phần cần biểu thị giữa 2 đối tượng trong danh sách liệt kê được kết nối bằng:

  - Dấu chấm than (!) nếu đối tượng sau do người dùng định nghĩa
  - Dấu chấm (.) trong trường hợp còn lại (đối tượng sau là thuộc tính, phương thức hay đối tượng tiền định)

Ví dụ: Để biểu thị thuộc tính Visible của Form "Timkiem" ta có thể dùng một trong 2 cách sau:

```
Forms![Tim kiem].Visible
Forms("Tim kiem").Visible
```

3. Biến đối tượng

- **Cú pháp khai báo:**

  Dim Tên_biến As Kiểu_đối_tượng

Ví dụ: Dim DB As DataBase, FM as Form, FD as Field

- Để gắn một biến với một đối tượng cùng kiểu ta dùng câu lệnh:

  Set Tên_biến = Dạng_biểu_thị_đối_tượng

Ví dụ: Set FM = Forms![Nhansu]

- Sau khi đã gắn một biến với một đối tượng ta có thể dùng biến để biểu thị đối tượng này

Ví dụ:

- Biểu thị thuộc tính Visible của mẫu biểu [nhan su] như sau:

  FM.Visible

- Biểu thị ô điều khiển [Hoten] của biểu mẫu này như sau:

  FM![Hoten]

4. Forms và mối quan hệ phân cấp

- **Sơ đồ quan hệ phân cấp:**

  Mức 1: Đối tượng khởi thuỷ Forms

  Mức 2: Các mẫu biểu đang mở là thành phần của Forms

  Mức 3: Các ô điều khiển (đối tượng kiểu control) là thành phần của Form

  Mức 4: Đối tượng tiền định form là thành phần của ô điều khiển kiểu subform

  Mức 5: Các ô điều khiển của mẫu biểu con là thành phần của Form

Ví dụ: Để hiển thị thuộc tính Visible của ô điều khiển [Hoten]của mẫu biểu [Tim kiem] ta viết như sau:

Forms![Timkiem]![Hoten].Visible

- **Các thuộc tính hay dùng trong mẫu biểu:**
  - Name: tên đối tượng
  - Visible: quy định việc hiển thị hoặc không
  - Count: xác định số đối tượng thành phần

5. Reports và mối quan hệ phân cấp

- **Sơ đồ quan hệ phân cấp**: tương tự như với Forms

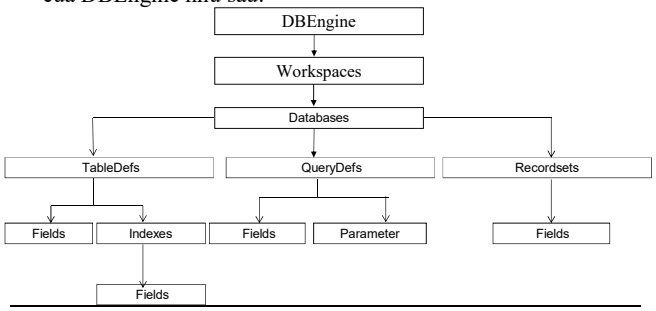
Ví dụ: Thuộc tính Visible của Report [Thongke] có thể biểu diễn theo 3 cách:

- Reports![Thongke].Visible
- Reports("Thongke").Visible
- Reports(0).Visible

- Các thuộc tính hay dùng trong Report: tương tự như mẫu biểu: Name,Visible,Count

6. Mối quan hệ phân cấp trong DBEngine

- Để làm việc với các bảng, truy vấn và tập bản ghi của chúng, ta cần sử dụng các đối tượng TableDefs, QueryDefs và RecordSets. Các đối tượng này đều nằm trong sơ đồ phân cấp của DBEngine như sau:



6. Mối quan hệ phân cấp trong DBEngine

- Ta thường dùng vùng làm việc đầu tiên Workspaces(0), cơ sở dữ liệu hiện hành Databases(0)

Ví dụ: Để biểu thị bảng [Danhsachcanbo] của cơ sở dữ liệu hiện hành ta có thể dùng cách viết sau:

DBEngine.Workspaces(0).Databases(0).Tabledefs("Danhsach canbo")

Chương 7: Tạo mới và thay đổi cấu trúc bảng/truy vấn

1. Tạo một bảng mới

- Cần sử dụng một biến kiểu Database (ví dụ DB), một biến kiểu Tabledef (ví dụ TB), một biến kiểu Field (ví dụ FD). Các biến được khai báo như sau:
  - Dim DB as Database
  - Dim TB as Tabledef
  - Dim FD as Field
- Với đối tượng kiểu Database ta dùng các phương thức:
  - DB.CreateTabledef("Tênbảng") để tạo một bảng mới
  - DB.Tabledefs.Append TB để bổ sung bảng (do biến TB tham chiếu tới) vào tập các bảng của CSDL gắn với biến DB
- Với đối tượng kiểu Tabledef ta dùng các phương thức:
  - TB.CreateField("Têntrường"[,kiểu_trường[,kích_thước_trường]]) để tạo một trường mới
  - TB.Field.Append FD để bổ sung một trường (do biến FD tham trỏ tới) vào tập các trường của bảng gắn với biến TB

1. Tạo một bảng mới

- **Tạo trường:** sử dụng phương thức TB.CreateField với 3 đối:
- **Tên trường:** do người dùng tự đặt theo quy tắc của Access
- **Kiểu_trường:** là một giá trị nguyên (có thể dùng giá trị hay tên hằng) sau đây:

1	DB_BOOLEAN	(kiểu Yes/No)
2	DB_BYTE	(kiểu Byte)
3	DB_INTEGER	(kiểu Integer)
4	DB_LONG	(kiểu Long/Counter)
5	DB_CURENCY	(kiểu Currency)
6	DB_SINGLE	(kiểu Single)
7	DB_DOUBLE	(kiểu Double)
8	DB_DATE	(kiểu Date/Time)
9	DB_TEXT	(kiểu TEXT)
10	DB_OLE	(kiểu OLE)
11	DB_MEMO	(kiểu Memo)

- **Kích thước trường:** tính bằng byte đối với kiểu TEXT

1. Tạo một bảng mới

- **Ví dụ:** Thủ tục sau tạo một bảng mới là [Luongcanbo], gồm các trường [Hoten] kiểu TEXT và trường [Luongchinh] kiểu DOUBLE

```
Sub Tao_tep()
    DIM DB as Database, TB as Tabledef, FD as Field
    Set DB = DBEngine.Workspaces(0).Databases(0)
    Set TB = DB.CreateTabledef("Luongcan bo")
    Set FD = TB.CreateField("Hoten",DB_TEXT,25)
    TB.Fields.Append FD
    Set FD = TB.CreateField("Luongchinh",DB_DOUBLE)
    TB.Fields.Append FD
    DB.TableDefs.Append TB
End Sub
```


2. Liệt kê, xóa bảng khỏi cơ sở dữ liệu

- Xoá bảng: Để xóa một bảng khỏi cơ sở dữ liệu ta dùng phương thức:

```
DB.TableDefs.Delete "Tên_bảng"
```

- Liệt kê các bảng: sử dụng các thuộc tính sau
- Thuộc tính COUNT của đối tượng Tabledefs cho biết số bảng hiện có của một cơ sở dữ liệu
- Thuộc tính NAME của đối tượng kiểu Tabledef cho biết tên bảng

3. Thay đổi cấu trúc của bảng

- Sử dụng các biến để thay đổi cấu trúc của bảng
  + Biến DB kiểu Database, khai báo: Dim DB as Database
  + Biến TB kiểu Tabledef, khai báo: Dim TB as Tabledef
  + Biến FD kiểu Field, khai báo: Dim FD as Field
  + Biến ID kiểu Index, khai báo: Dim ID as Index
- Xoá trường, bổ sung trường và đổi tên trường

+ Xoá trường: dùng phương thức

```
TB.Fields.Delete "Têntrường"
```

+ Bổ sung trường: dùng 2 phương thức sau

```
Set FD = TB.CreateField("Têntrường", kiểu [,Kíchthước])
TB.Fields.Append FD
```

+ Đổi tên trường: Để đổi tên trường ta thay thuộc tính NAME của trường

3. Thay đổi cấu trúc của bảng

- **Ví dụ:** Xoá trường [Namsinh], bổ sung trường [Phucap] kiểu double, đổi tên trường [Luong] thành [Luongchinh] của bảng [DSCB], ta dùng đoạn chương trình sau:

```
Dim DB as Database, TB as Tabledef, FD as Field
Set DB = CurrentDB()
Set TB = DB.Tabledefs ("DSCB")
TB.fields.Delete "Namsinh"
Set FD = TB.CreateField(" Phucap", DB_DOUBLE)
TB.Fields.Append FD
Set FD = TB.Fields ("Luong")
FD.Name = " Luongchinh"
```

3. Thay đổi cấu trúc của bảng

- **Xoá chỉ mục, đổi tên chỉ mục và tạo chỉ mục mới**

+ Xoá chỉ mục: sử dụng phương thức

TB.Index.Delete "Tênchỉmục"

+ Đổi tên chỉ mục: cần thay đổi thuộc tính NAME của chỉ mục

+ Tạo chỉ mục mới: thực hiện các bước

- Bước 1: tạo một chỉ mục mới theo mẫu:

Set ID = TB.CreateInedex("Tênchỉmục")

- Bước 2: chọn các trường đưa vào chỉ mục theo mẫu:

Set FD = ID.CreateField("Têntrường")

FD.Attributes = DBDESCENDING 'Sắp theo chiều giảm

ID.Fields.Append FD

- Bước 3: bổ sung chỉ mục mới tạo vào tập các chỉ mục của bảng, sử dụng lệnh:

TB.Indexes.Append ID

4. Bắt và xử lý lỗi

- **Các lỗi có thể xảy ra khi chạy chương trình:**
  - Xoá một bảng/truy vấn không tồn tại
  - Tạo mới một bảng/truy vấn đã tồn tại
  - Nhận xử lý một giá trị Null từ một ô điều khiển của mẫu báo biểu
  - Chia cho số không
  - …

4. Bắt và xử lý lỗi

- Các câu lệnh cần dùng: Để chương trình chạy thông suốt cần lường trước các lỗi có thể xảy ra, có phương án bắt lỗi, xử lý lỗi. Access cung cấp các câu lệnh sau để tổ chức việc này:
  - Câu lệnh: On Error Goto Xu_ly_loi
  - Khi gặp lỗi sẽ chuyển đến đoạn chương trình xử lý lỗi bắt đầu từ nhãn Xu_ly_loi
  - Các hàm: Error$ cho biết lỗi gì xảy ra

    Err cho biết số hiệu lỗi
  - Câu lệnh: Resume dùng trong cuối đoạn chương trình xử lý lỗi để hướng dẫn sự hoạt động tiếp tục của chương trình, nó thường được viết theo 2 cách sau:

    + Cách 1: Resume Next theo cách này chương trình tiếp tục từ câu lệnh ngay sau câu lệnh gây lỗi

    + Cách 2: Resume Tiep_tuc ở đây Tiep_tuc là một nhãn. Theo cách này chương trình tiếp tục từ câu lệnh có nhãn Tiep_tuc

5. Tạo mới một truy vấn

- Công cụ cần dùng để tạo một truy vấn:

  + Dùng các biến:
    - Biến DB kiểu Database
    - Biến QR kiểu Querydef
    - Biến TS kiểu String

  + Dùng các phương thức:

  Set QR = DB.CreateQuerydef() để tạo một truy vấn mới

  DB.Querydefs.Append QR để bổ sung một truy vấn vào tập các truy vấn của CSDL
- Dùng các phép gán để thay đổi nội dung các thuộc tính NAME và SQL của truy vấn:

  QR.Name = "Têntruyvan"

  QR.SQL = "CâulênhSQL"

6. Liệt kê các truy vấn và xoá truy vấn

- Liệt kê các truy vấn: để liệt kê các truy vấn ta sử dụng các thuộc tính sau:

  - Thuộc tính COUNT của đối tượng QueryDefs: cho biết số truy vấn hiện có của một cơ sở dữ liệu

  - Thuộc tính NAME của đối tượng kiểu QueryDef: cho biết tên truy vấn

- Xoá truy vấn: sử dụng phương thức sau

  DB.QueryDefs.Delete "Têntruyvấn"

7. Thay đổi cấu trúc của truy vấn

- Đổi tên truy vấn: Gán giá trị mới cho thuộc tính NAME

  QR.Name = "Tênmới"

- Đổi nội dung (câu lệnh SQL) của truy vấn: Gán giá trị mới cho thuộc tính SQL của truy vấn theo mẫu:

  QR.SQL = "câu lệnh SQL mới"

Chương 8. Xử lý các bản ghi của bảng và truy vấn

1. Các biến đối tượng để xử lý dữ liệu

- Các đối tượng phục vụ việc xử lý dữ kiện:

  - Đối tượng kiểu Database
  - Đối tượng kiểu Tabledef
  - Đối tượng kiểu Querydef
  - Đối tượng kiểu Recorset

  Để sử dụng các biến đối tượng, các thuộc tính và phương thức của chúng trong việc xử lý dữ liệu, chúng ta cần xem lại sơ đồ quan hệ các đối tượng đã trình bày ở trên

2. Biến biểu thị cơ sở dữ liệu

- **Để làm việc với một cơ sở dữ liệu cần:**
  - Khai báo một biến kiểu Database
  - Cho biến này tham chiếu đến một cơ sở dữ liệu cụ thể trong vùng làm việc
- **Làm việc với cơ sở dữ liệu hiện hành:** Có thể sử dụng 2 cách
  - Cách 1:

```
Dim DB as Database
Set DB = DBEngine.Workspaces(0).Databases(0)
```

  - Cách 2:

```
Dim DB as Database
Set DB = CurrentDB()
```


2. Biến biểu thị cơ sở dữ liệu

- Làm việc với cơ sở dữ liệu khác (sử dụng phương thức OpenDatabase)
- Có thể đồng thời làm việc với nhiều cơ sở dữ liệu khác nhau: xử lý các bảng, truy vấn song chỉ cho phép sử dụng các Form, Report, Macro, Module của CSDL hiện hành
- Để làm việc với một CSDL, cần gắn một biến kiểu Database với nó sử dụng phương thức OpenDatabase

  Ví dụ: gắn biến DB với CSDL "CSDL4.MDB":

```
Dim DB as Database
Set DB =
    DBEngine.Workspaces(0).OpenDatabase("SCDL4.MDB")
```

- Để đóng các đối tượng, sử dụng phương thức Close

3. Biến kiểu Recordset

- Để xử lý dữ kiện (các bản ghi) của bảng, truy vấn, bảng/truy vấn nền của mẫu/báo biểu ta dùng các biến kiểu RecordSet

  Có 3 loại RecordSet là:

Kiểu RecordSet	Số mẫu tin	Dữ kiện
Loại Table	Có thể thay đổi	Có thể thay đổi
Loại DynaSet	Có thể thay đổi	Có thể thay đổi
Loại SnapShot	Không thể thay đổi	Không thể thay đổi

3. Biến kiểu Recordset

- RecordSet loại Table (DB_OPEN_TABLE)
  - Chỉ áp dụng cho bảng của cơ sở dữ liệu hiện hành
  - Có thể thay đổi dữ kiện (thêm, xoá, sửa)
  - Có thể sắp xếp, lập chỉ mục, tìm kiếm theo phương thức Seek
- RecordSet loại DynaSet (DB_OPEN_DYNASET)
  - Có_ thể áp dụng trên: bảng, truy vấn, bảng/truy vấn nền của mẫu/báo biểu của CSDL bất kỳ
  - Có thể thay đổi dữ kiện (thêm, xoá, sửa)
  - Có thể tìm kiếm theo phương thức Find
- RecordSet loại Snapshot (DB_OPEN_SNAPSHOT)
  - Như DynaSet nhưng không cho phép thay đổi dữ kiện, chỉ để xem dữ kiện

4. Tạo biến RecordSet

- Phương thức OpenRecordSet: là phương thức cơ bản dùng để tạo một biến RecordSet. Có thể dùng phương thức này với các đối tượng sau: Database, Tabledef, Querydef

Cú pháp:

Set Biến_RecordSet = Đối_tượng.OpenRecordSet(nguồn,loại)

Trong đó:

- Đối_tượng: Là một biến đối tượng kiểu Database, Tabledef, Querydef
- Nguồn: Là chuỗi ký tự biểu thị tên bảng, truy vấn hoặc câu lệnh SQL
- Loại là một trong các giá trị sau:
  - DB_OPEN_TABLE
  - DB_OPEN_DYNASET
  - DB_OPEN_SNAPSHOT
- Loại mặc định: Đối với đối tượng là bảng của CSDL hiện hành thì loại mặc định là: DB_OPEN_TABLE, đối với đối tượng là bảng/truy vấn của CSDL không hiện hành thì loại mặc định là: DB_OPEN_DYNASET

- Đóng các RecordSet: sử dụng phương thức Close

Tên_biến_RecordSet.Close

5. Tham chiếu đến các trường của RecordSet

- Để lấy dữ liệu từ các trường hoặc thay đổi giá trị các trường, cần tham chiếu đến các trường của RecordSet, sử dụng 1 trong các cách:
  - Tên_biến_RecordSet.Tên_trường
  - Tên_biến_RecordSet("Tên_trường")
  - Tên_biến_RecordSet(Biến_chứa_tên_trường)
- Đếm số mẫu tin của một RecordSet: Dùng thuộc tính RecordCount cho biết tổng số bản ghi của một RecordSet

6. Cập nhật các mẫu tin (thêm, sửa, xoá)

- Thủ tục thay đổi nội dung của một mẫu tin:
  1. Chuyển đến mẫu tin cần sửa
  2. Dùng phương thức EDIT để cho phép sửa mẫu tin hiện hành
  3. Thực hiện các thay đổi cần thiết
  4. Dùng phương thức UPDATE để cập nhật giá trị mới của mẫu tin hiện hành
- Xoá một mẫu tin: sử dụng phương thức DELETE của RecordSet
- Thêm một mẫu tin mới vào RecordSet: thực hiện theo trình tự
  1. Dùng phương thức Addnew để thêm vào cuối RecordSet một bản ghi rỗng
  2. Gán giá trị vào các trường của bản ghi mới thêm
  3. Dùng phương thức Update để cập nhật mẫu tin mới

7. Di chuyển giữa các mẫu tin

- Các phương thức:

MoveFirst	Chuyển về mẫu tin đầu tiên
MoveLast	Chuyển về mẫu tin cuối cùng
MoveNext	Chuyển về mẫu tin ngay sau
MovePrevious	Chuyển về mẫu tin ngay trước

- Các thuộc tính dùng để nhận biết mẫu tin hiện hành:

EOF	End of file
BOF	Begining of file

- Nếu mẫu tin hiện hành là mẫu tin cuối cùng thì phương thức MoveNext sẽ đặt EOF=True
- Nếu mẫu tin hiện hành là mẫu tin đầu tiên thì phương thức MovePrevious sẽ đặt BOF=True

8. Tìm kiếm trong RecordSet loại DynaSet và SnapShot

- Sử dụng các phương thức:

  FindFirst "Điều kiện" Tìm bản ghi đầu tiên

  FindLast "Điều kiện" Tìm bản ghi cuối cùng

  FindNext "Điều kiện" Tìm bản ghi sau bản ghi hiện hành

  FindPrevious "Điều kiện" Tìm bản ghi trước bản ghi hiện hành

- Thuộc tính NoMatch: Sử dụng thuộc tính này để biết kết quả tìm kiếm

  NoMatch = False Nếu phép tìm kiếm có kết quả

  NoMatch = True Nếu phép tìm kiếm không có kết quả

9. Sắp xếp và tìm kiếm theo chỉ mục (RecordSet loại Table)

- Phương thức:

  Rec.ListIndex dùng để liệt kê các chỉ mục của bảng ứng với một RecordSet loại Table

- Để có thể xử lý các mẫu tin của RecordSet theo một chỉ mục, ta cần gán tên chỉ mục này cho thuộc tính Rec.Index:

  Rec.Index = "Tên_chỉ_mục"

- Tìm bản ghi đầu tiên trong RecordSet theo giá trị các trường trong chỉ mục:

  1. Sắp xếp chỉ mục: Rec.Index = "Tên chỉ mục"

  2. Dùng phương thức:

  Rec.Seek "Toán tử so sánh" , v1,v2, . . . ,vm

  Trong đó :

  - Toán tử so sánh có thể là: = , >, >= , < , <=

  - v1,v2, . . . , vm là các giá trị để so sánh với các trường của chỉ mục (giả sử chỉ mục có m trường)

9. Sắp xếp và tìm kiếm theo chỉ mục (RecordSet loại Table)

- Sử dụng thuộc tính BOOKMARK: Một số phần mềm (như Foxpro) dùng số hiệu bản ghi để để nhận diện bản ghi. Access Basic dùng công cụ Bookmark để nhận diện bản ghi của bảng dữ kiện

  - Bookmark là một chuỗi ký tự được Access tạo một cách ngẫu nhiên để nhận diện mỗi bản ghi của một bảng dữ kiện một cách duy nhất

  - Giá trị của thuộc tính Bookmark luôn luôn chỉ đến bản ghi hiện hành. Khi thay đổi giá trị của Bookmark thì cũng thay đổi bản ghi hiện hành

Chương 9: Mẫu biểu và việc sử dụng dữ liệu nhập từ bàn phím

1. Nhận dữ liệu từ bàn phím

- Có thể nhận dữ liệu bằng 2 cách :

Cách 1: Dùng hàm InputBox. Ví dụ:

```
Dim Ten as String, Diem as integer
Ten = Inputbox$("HoTen:")
Diem = InputBox("Diem:")
Form![Hanghoa]![Dongia]=InputBox("Dongia:")
```

Cách 2: Nhận dữ liệu từ các ô điều khiển của mẫu biểu và lưu trữ vào các biến trong chương trình. Ví dụ:

```
Dim DG
DG = Form![Hanghoa]![Dongia]
```

- Đặt một giá trị lên ô điều khiển:

Ví dụ: Form![Hanghoa]![Dongia] = DG

2. Xử lý mẫu biểu trong chương trình

- Để mở một mẫu biểu trong chương trình ta dùng câu lệnh:

  DoCmd OpenForm danh sách các đối

- Các thuộc tính trong mẫu biểu
- Name: tên đối tượng
- Visible: thuộc tính này quy định sự hiển thị hay không hiển thị của mẫu biểu hay ô điều khiển
- Count: xác định số đối tượng
- Enable: xác định ô điều khiển có thể nhận focus hay không trong kiểu hiển thị Form View. Nếu thuộc tính = True thì có thể chuyển con trỏ văn bản đến ô điều khiển để sửa dữ liệu. Thuộc tính = False thì không thể chuyển con trỏ văn bản đến ô điều khiển.
- Locked: xác định cho phép sửa đổi dữ kiện của mẫu biểu hay không trong tình trạng Form View

  Ví dụ: Form![Thong ke].Visible

2. Xử lý mẫu biểu trong chương trình

- Dùng các thuộc tính để điều khiển mẫu biểu trong chương trình: thực hiện gán giá trị thích hợp cho các thuộc tính

Ví dụ: Mở và cho ẩn mẫu biểu "Timkiem":

```
DoCmd OpenForm "Timkiem"
Form![Timkiem].Visible=False
```


3. Kích hoạt mẫu biểu

- Để kích hoạt mẫu biểu, sử dụng cú pháp:

  DoCmd RepaintObject A_Form , "Tên mẫu biểu"

- Để buộc MS Access hiển thị các giá trị mới nhất trên các ô điều khiển ta dùng câu lệnh sau:

  DoCmd Requery "Tên ô điều khiển"

5. Xây dựng bảng giá trị cho List Box và Combo Box trong chương trình

- Sử dụng hàm tạo danh sách (List Function) với 5 đối, đối thứ nhất kiểu Control, còn các đối khác kiểu Variant. Hàm trả về một giá trị kiểu Variant

Cú pháp:

Function Tên_hàm(Ctr as Control, Val, Row, Col, Code)

Trong đó các đối có ý nghĩa sau:

- Ctr: Dùng để tham chiếu đến ô List/Combo Box mà ta muốn điền dữ liệu vào
- Val: Giá trị điền vào List/Combo Box
- Row: Hàng của List/Combo Box (tính từ 0) mà ta cần điền dữ liệu vào
- Col: Cột của List/Combo Box (tính từ 0) mà ta cần điền dữ liệu vào
- Code: Giá trị xác định loại thông tin mà Access cần xem

5. Xây dựng bảng giá trị cho List Box và Combo Box trong chương trình

- **Giá trị của code có ý nghĩa như sau:**
- Code = 0: Khởi tạo. Giá trị của hàm sẽ khác 0 nếu có thể điền dữ liệu vào list/combo box, giá trị của hàm sẽ bằng 0 hoặc null nếu ngược lại
- Code = 1: Mở list/combo box. Giá trị hàm = Timer để nhận biết là list/combo box
- Code = 2: Chưa sử dụng
- Code = 3: Số hàng của danh sách. Giá trị hàm = số hàng của danh sách
- Code = 4: Số cột của danh sách. Giá trị hàm = số cột của danh sách
- Code = 5: Độ rộng cột. Giá trị hàm = độ rộng cột. Nếu hàm = -1 thì cột có độ rộng mặc định

5. Xây dựng bảng giá trị cho List Box và Combo Box trong chương trình

- Code = 6: Giá trị cần điền. Giá trị hàm sẽ được điền vào hàng và cột của danh sách
- Code = 7: Dạng hiển thị. Giá trị của hàm là chuỗi kí tự định dạng hiển thị
- Code = 8: Chưa sử dụng
- Code = 9: Gọi lần cuối. Giá trị của hàm không được sử dụng

5. Xây dựng bảng giá trị cho List Box và Combo Box trong chương trình

Vị trí của List Function:

- Nếu List Function định nghĩa trong một đơn thể sử dụng chung thì nó có thể sử dụng để tạo danh sách giá trị cho bất kỳ List/Combo Box nào trong chương trình
- Nếu List Function định nghĩa trong phần General của một mẫu biểu thì nó có thể sử dụng để tạo danh sách giá trị cho List/Combo Box trong mẫu biểu nói trên

5. Xây dựng bảng giá trị cho List Box và Combo Box trong chương trình

- Cách thực hiện hàn list function
- Access gọi list function 1 lần ứng với code bằng 0, 1, 3 , 4 để:
  Khởi tạo hàm danh sách (code=0)
  Mở list/combo box (code=1)
  Xác định số hàng (code=3), số cột của danh sách (code=4)
- Access gọi list function 2 lần ứng với code = 5 để: một lần để xác định tổng bề rộng các cột, lần thứ 2 xác định bề rộng của các cột chứa giá trị cần điền
- Access gọi list function nhiều lần ứng với code=6 và 7, để xác định giá trị (code=6) và dạng hiển thị tại các dòng của danh sách (code=7)
- Access gọi list function thêm lần cuối nếu code = 9

Chương 10: Thao tác trên các tệp tin

1. Mở tệp, đóng tệp

- Mở Tệp: sử dụng mệnh đề Open
  OPEN Tên_tệp [FOR Kiểu_tệp] [ACCESS Quyền_truy_nhập] AS[#] Số_hiệu_tệp [LEN Reclen]
  (Hai đối bắt buộc là Tên_tệp và Số_hiệu_tệp, còn các đối còn lại tuỳ chọn)

Trong đó:

- Tên_tệp: Là biểu thức chuỗi xác định tên tệp và đường dẫn
- Kiểu_tệp: Xác định cách thức truy nhập trên tệp, có thể là:

+ Random: Đây là kiểu mặc định, cho phép đọc/ghi theo mẫu tin. Kiểu Random cho phép thực hiện cả 3 thao tác: Đọc và ghi, chỉ ghi, chỉ đọc (mặc định là Đọc và ghi)

+ Binary: Kiểu Binary cho phép đọc/ghi từ một vị trí (tính theo byte) bất kỳ trên tệp bằng cách dùng mệnh đề GET(để đọc) và PUT (để ghi). Kiểu Binary cho phép thực hiện cả 3 thao tác: Đọc và ghi, chỉ ghi, chỉ đọc (mặc định là Đọc và ghi)

1. Mở tệp, đóng tệp

+ Input: cho phép thực hiện thao tác đọc trên tệp tuần tự bằng cách sử dụng mệnh đề Input

+ Output: cho phép thực hiện thao tác ghi trên tệp tuần tự bằng cách sử dụng mệnh đề Write

+ Append: Kiểu Append cho phép thực hiện thao tác ghi bổ sung trên tệp tuần tự bằng cách sử dụng mệnh đề Write

- Quyền_truy_nhập: Xác định quyền truy nhập tên tệp, nó là một trong các từ dành riêng: Read, Write, Read Write

Read: Chỉ có quyền đọc

Write: Chỉ có quyền ghi

Read Write: Có quyền đọc và ghi. Quyền này chỉ đúng với các kiểu tệp Random và Binary

1. Mở tệp, đóng tệp

- Số_hiệu_tệp: Là một biểu thức nguyên có giá trị từ 1 đến 255. Sau khi thực hiện mệnh đề Open thì số_hiệu_tệp sẽ gắn với tệp cho đến khi tệp bị đóng. Mọi thao tác trên tệp sẽ thực hiện thông qua số hiệu tệp

Số hiệu tệp trong mệnh đề Open cần không được gắn với một tệp đang mở nào. Để xác định một số hiệu còn tự do, ta dùng hàm FREEFILE, cách viết: FREEFILE[()]

- Reclen: Là một biểu thức nguyên có giá trị từ 1 đến 32767. Đối này thực sự có tác dụng đối với tệp kiểu Random để xác định độ lớn của bản ghi. Giá trị mặc định là 128 byte

Ví dụ: Open "luongcb.sl" FOR Input AS #1
Open "hosocb.sl" FOR Output AS #2

1. Mở tệp, đóng tệp

- Đóng tệp: sử dụng mệnh đề Close
  - Để đóng một số tệp ta liệt kê các số hiệu tệp cần đóng trong mệnh đề CLOSE như sau:
    CLOSE [[#] Số_hiệu_tệp][,[#] Số_hiệu_tệp]. . .
  - Để đóng tất cả các tệp đang mở ta dùng dạng sau: CLOSE
- Mệnh đề RESET: Dùng để đóng tất cả các tệp đang mở, nó được viết dưới dạng: RESET

2. Đọc/ghi trên tệp tuần tự

- **Mệnh đề WRITE:** Dùng để ghi dữ liệu kiểu số, kiểu chuỗi, kiểu Date/time lên từng dòng của tệp tuần tự

Cú pháp:

WRITE # Số_hiệu_tệp, [danh sách biểu thức]

trong đó:

- Số_hiệu_ tệp: Là số hiệu gắn với tệp (xác định trong mệnh đề OPEN). Thường dùng số_hiệu_tệp = freefile để chọn một số hiệu chưa dùng
- Danh sách biểu thức: có thể là các biểu thức kiểu số, kiểu chuỗi và kiểu Date/time, giữa 2 biểu thức phân cách nhau bởi dấu phẩy

Kết quả thực hiện: Mỗi câu lệnh WRITE sẽ ghi 1 dòng văn bản lên tệp, trong một dòng các giá trị được phân cách nhau bởi dấu phẩy

2. Đọc/ghi trên tệp tuần tự

Ví dụ: WRITE #2 "Ha",21,"Mai",25
WRITE #2 "Thu",24,"Lan",35

Kết quả có 2 dòng sau:

"Ha",21,"Mai",25
"Thu",24,"Lan",35

- **Mệnh đề INPUT:** Dùng để đọc dữ liệu kiểu số, kiểu chuỗi, kiểu Date/time từ tệp văn bản và chứa vào các biến tương ứng

Cú pháp: INPUT # Số_hiệu_tệp , Danh_sách_biến

trong đó:

- Số_hiệu_tệp: Là số hiệu gắn với tệp
- Danh_sách_biến: Đó là các biến dùng để nhận dữ liệu đọc từ tệp. Giữa các biến phân cách nhau bởi dấu phẩy

Ví dụ: INPUT #2,TEN,TUOI

3. Đọc/ghi trên tệp ngẫu nhiên/nhị phân (Random/Binary)

- **Mệnh đề PUT:** Dùng để ghi giá trị của một biến (kiểu số, kiểu chuỗi, kiểu date/time) lên một tệp ngẫu nhiên/nhị phân. Kết quả ta nhận được một tệp nhị phân

Cú pháp: PUT # Số_hiệu_tệp, [SH], Tên_biến

- Số_hiệu_tệp: Là số hiệu gắn cho tệp
- Tên_biến: Có thể dùng các biến kiểu số, kiểu chuỗi, kiểu Variant và cả biến kiểu tự tạo
- SH: Là biểu thức nguyên biểu thị số hiệu bản ghi hoặc số hiệu byte đối với tệp nhị phân. Việc ghi sẽ được thực hiện từ bản ghi hoặc byte này. Nếu không có SH thì việc ghi sẽ thực hiện từ bản ghi hoặc byte hiện tại trên tệp

3. Đọc/ghi trên tệp ngẫu nhiên/nhị phân (Random/Binary)

- **Mệnh đề GET:** Mệnh đề này dùng để đọc dữ liệu từ một tệp Radom/Binary và chứa vào một biến.

Cú pháp: GET # Số_hiệu_tệp, [SH], Tên_biến

Trong đó:

- Số_hiệu_tệp: Là số hiệu gắn cho tệp
- Tên_biến: Có thể dùng các biến kiểu số, kiểu chuỗi, kiểu Variant và cả biến kiểu tự tạo
- SH: Là biểu thức nguyên biểu thị số hiệu bản ghi hoặc số hiệu byte đối với tệp nhị phân. Việc đọc sẽ được thực hiện từ bản ghi hoặc byte này. Nếu không có SH thì việc đọc sẽ thực hiện từ bản ghi hoặc byte hiện tại trên tệp

3. Đọc/ghi trên tệp ngẫu nhiên/nhị phân (Random/Binary)

- **Mệnh đề SEEK:** dùng để di chuyển vị trí con trỏ tệp
- **Cú pháp:**

SEEK [#] Số_hiệu_tệp, Vị_trí

Trong đó:

- Số_hiệu_tệp: Là số hiệu gắn cho tệp
- Vị_trí: Là một số nguyên dương từ 1 đến 2147643647 xác định vị trí mà con trỏ tệp sẽ di chuyển tới
- Đối với tệp Radom Vị_trí là số hiệu bản ghi (tính từ 1)
- Đối với tệp Binary Vị_trí là số hiệu byte (tính từ 1)

3. Đọc/ghi trên tệp ngẫu nhiên/nhị phân (Random/Binary)

- **Hàm SEEK:** SEEK(Số hiệu tệp)

→ cho biết vị trí hiện tại của con trỏ tệp. Đối với tệp Radom hàm cho biết số hiệu bản ghi mà con trỏ tệp đang định vị. Đối với tệp Binary, Append, Input, Output hàm cho biết số hiệu byte mà con trỏ tệp đang định vị

- **Hàm LOG:** Hàm này cho biết độ dài tệp tính theo byte. Cách viết:

LOG(Số_hiệu_tệp)

- **Hàm EOF:** Hàm này cho biết còn dữ liệu trên tệp để tiếp tục đọc hay không. Cách viết như sau:

EOF(Số_hiệu_tệp)

Kết quả của hàm: Khi con trỏ đang đặt tại cuối tệp thì hàm EOF có giá trị True, trong các trường hợp khác hàm có giá trị False