【AccessVBAエキスパート】書籍学習用

ノートブック:	210_メモ		
作成:	2015/12/30 17:52	**更新:**	2016/03/04 8:35
作成者:	Tsutsui tomoaki		
URL:	http://detail.chiebukuro.yahoo.co.jp/qa/question_detail/q1113375816		

タイトル：AccessVBAエキスパート

chapter 01【VBAの基礎知識】

01.基本用語

◆プロジェクトとは
データベースファイルに保存されているモジュールを取りまとめて管理するもの。
・プロジェクトの内容はプロジェクトエクスプローラで確認できる
・自由にプロジェクト名を付けることができる
・パスワードを使用して表示をロックすることができる
プロジェクトには次の三種類のモジュールがある。
・フォームモジュール/レポートモジュール
・標準モジュール
・クラスモジュール

モジュールの種類	内容
Microsoft Office Accessクラスオブジェクト	フォームやレポートに関連付けられた、フォームモジュールやレポートモジュールを格納する
標準モジュール	データベース全体で使用する汎用的なモジュールを格納する
クラスモジュール	ユーザーが作成した独自のクラスが定義されたモジュールを格納する

◆モジュールとは
プログラムを構成する単位であるプロシージャを記述・格納するためのオブジェクトです。
自由に作成・削除できる「標準モジュール」「クラスモジュール」
フォームやレポートに関連付けられて作成される「フォームモジュール」「レポートモジュール」
があります。
モジュールの種類によって格納できるプロシージャや使用目的が異なります。
◆プロシージャとは
プロシージャとは、プログラムを構成する最小単位です。プログラムはコンピュータに演算などの処理を行わせる手続き全般を示す。
プロシージャはそれらの処理に対する命令を手順としてまとめ、記述したものです。
つまり、
プログラムは、1つまたは複数のプロシージャから構成されます
プロシージャはすべてモジュールの中身に記述されます。

◆オブジェクトとは
VBAにおいて処理の対象となる、アプリケーションの構成要素を指します。
フォームやレポートのように目に見えるオブジェクトや
ユーザーが独自に作成したオブジェクトがある。
これらのオブジェクトはVBAからその属性を取得・設定したり、動作を指定したりすることができる。
・プロパティとは
・メソッドとは
(--割愛--)

◆演算子とは
プログラムの中で様々な演算処理を行うときに、その演算内容を指示するための記号を指します。

演算子の種類	演算子
算術演算子	+, -, *, /, MOD, ¥, ^
比較演算子	=, <, <=, >, >=, <>, Is, Like
文字列連結演算子	&, +
論理演算子	And, Or, Not, Xor, Eqv, Imp
代入演算子	=

^:数値のべき乗を求める
¥:二つの値の除算を行い、その商を返す。

Is：二つの**オブジェクト参照**が同じであれば、Trueが返る.オブジェクト型出ないと、型不一致のエラーが発生する
myCheck = myObject Is yourObject
Like：Stringの値がPatternに含まれるパターンに一致した場合はTrueを返す
testCheck = "aBBBa" Like "a*a"
instrと同じ事が出来そう。

Like演算子の文字列指定パターン

文字パターン	パターン内容	事例	結果
?	任意の1文字	"ABC" Like "AB?"	True
*	任意の数の文字	"ABCDEFG" Like "AB*Z"	False
#	任意の数値1文字(0～9)	"A1B2C3" Like "A1B#C3"	True
[charlist]	charlistに指定した文字中の任意の1文字	"G" Like "[A-Z]"	True
[!charlist]	charlistに指定した文字以外の任意の1文字	"G" Like "[!A,B,C,D]"	False

文字リストのパターン例

```
[0-9]・・・0～9
[A-Z]・・・A～Z(半角)
[a-z]・・・a～z(半角)
[A-z]・・・A～Z,a-z(半角)
[Ａ-Ｚ]・・・Ａ～Ｚ(全角)
[ａ-ｚ]・・・ａ～ｚ(全角)
[Ａ-ｚ]・・・Ａ～Ｚ,ａ～ｚ(全角)
```

Is演算子

Is演算子はオブジェクト変数が同じオブジェクトを参照しているかどうかを調べます。こちらは、**オブジェクト変数a,b**が同じオブジェクトを参照しているか比較したものでどちらも「Sheets(1)」を参照していますので、「True」を返します。

```
Sub sample()

  Dim a As Object
  Dim b As Object

  Set a = Sheets(1)
  Set b = Sheets(1)

  If a Is b Then

    MsgBox "aとbは同じオブジェクトを参照しています。"

  End If

End Sub
```

続いてこちらの事例ですが、変わった点は参照するオブジェクトがRangeオブジェクトに変更されただけです。こちらを処理すると同じオブジェクトを参照しているはずなのに条件分岐が「False」を返したため、何も処理されません。

Rangeオブジェクトだけは、比較対象できませんので覚えておきましょう。

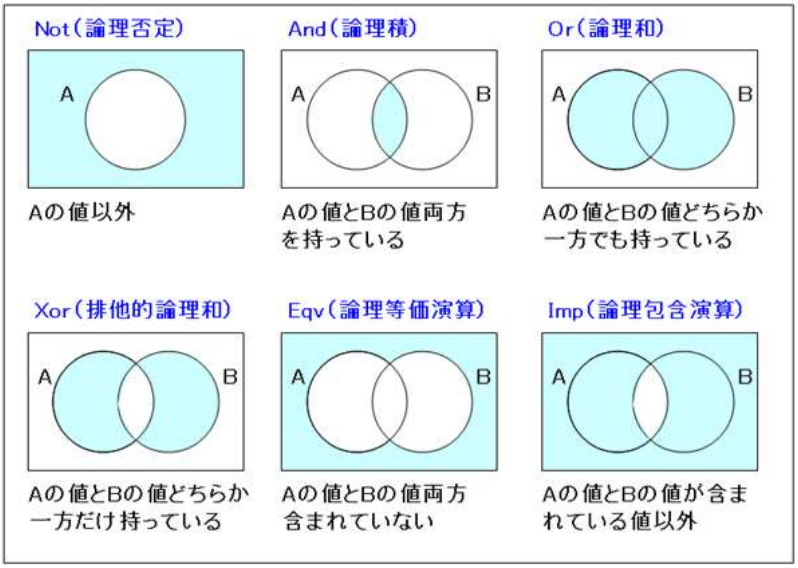
```
Sub sample()

  Set a = Range("A1")
  Set b = Range("A1")

  If a Is b Then

    MsgBox "aとbは同じオブジェクトを参照しています。"

  End If

End Sub
```

論理演算子とは

論理演算子とは複数の条件式を組み合わせて、より複雑な条件式を作時に利用されるのが論理演算子です。

演算子	説明	事例	結果

And	論理積（かつ）	a >= 1 And a < 10	aは1以上かつ10未満
Or	論理和（または）	a = 4 Or a = 1	aは4または1
Not	論理否定（～ではない）	Not a = 4	aは4ではない
Xor	排他的論理和	a >= 1 Xor a <= 10	aは1以上かつ10以下しかし、1以上かつ10以下ではない
Eqv	論理等価演算	a >= 1 Eqv a <= 10	aは1以上かつ10以下または、1以上かつ10以下ではない
Imp	論理包含演算	a <= 10 Imp a Mod 2 = 0	aは偶数、または10以下でも偶数でもない（=5より大きい奇数）



◆キーワードとは
VBAで特別な意味を持つ文字列
◆組み込み定数とは
VBAであらかじめ定義されている定数。
組み込み定数を利用することで、引数の値を直接指定するよりも簡単かつ正確にコードを記述することができる

組み込み定数の種類	内容
Visual Basicの定数	先頭に「vb」のプリフィックス(接頭辞)が付く。 Access以外のVBAでも使用できる
AccessVBAの定数	先頭に「ac」のプリフィックス(接頭辞)が付く。 Access以外のVBAでは使用できない

◆キャリッジリターン/ラインフィード
文字列を改行します。MsgBox関数やDebug.Printで出力させる文字列の中に用いると文章を改行することができる。

定数	内容
vbCr	キャリッジリターン文字
vbLf	ラインフィード文字
vbCrLf	改行文字
vbNewLine	vbCrLfと同じ働きをする

これらの定数の値は、Chr関数を使用した文字コードに対応する文字になります。
つまり「**vbCrLf**」は「**Chr(13)&Chr(10)**」と同じ値になります

◆カラー定数
色を指定します。フォームの背景色やテキストボックスの文字列などの色を、コードの中で指定するときに使用します。

定数	内容
vbBlack	黒色
vbRed	赤色
vbGreen	緑色
vbYellow	黄色
vbBlue	青色
vbMagenta	マゼンタ色
vbCyan	シアン色
vbWhite	白色

実際に使用して背景色を変更してみる！
全色調べる！

色の種類はこのくらいしかないみたい、
他にも便利な定数があったのでリンクだけ貼る。
http://www.red.oit-net.jp/tatsuya/vb/fixed.htm

◆参照設定とは

AccessVBAにない機能を外部のライブラリーから読み込んで利用するための設定
・事前バインディング
・実行時バインディング

02.基本構文

基本構文(ステートメント)
◆分岐処理
・Ifステートメント
・Select Caseステートメント

◆繰り返し処理
・For Nextステートメント
・Do...Loopステートメント
・For Each...Nextステートメント
For Each 要素変数 In 配列またはコレクション

◆その他
・Withステートメント
・Exitステートメント
--Exit Do
--Exit For
--Exit Sub
--Exit Function

03.データベース設計

◆主キー
・Null値が含まれていない(必ず値を持つ)
・他のレコードと値が重複しない

◆インデックス
主キーには自動で設定される
・インデックスが設定されたフィールドでは、並べ替え・検索などの処理時間が大幅に短縮される
・インデックスを固有(重複なし)にすることで、ほかのレコードと値が重複しないように設定できる

◆適切なテーブル分割/正規化
・繰り返し項目を削除するため、データベースのサイズが小さくなる
・テーブルを分離する為、各テーブルの目的が明確になる
・データ更新など、データ管理が容易になる
--正規化のフェーズについて
非正規化：繰り返し項目を含むテーブル
↓第一正規化　繰り返し項目を失くす
第一正規化：主キーの一部によって決まる項目を含むテーブル
↓第二正規化　主キーの一部によって決まる項目を他テーブルに分離する
第二正規化：主キー以外の項目によって決まる項目を含むテーブル
↓第三正規化　主キー以外の項目によって決まる項目を他テーブルに分離する
第三正規化：第三正規化されたテーブル

◆リレーションシップとは
複数のテーブルに共通するフィールドでテーブルを関連付けること。
一対一
一対多
多対多

◆結合の種類
・外部結合
・内部結合

◆参照整合性とは
リレーションシップを設定したテーブル間でデータの整合性を維持するための規則

・関連付ける2つのフィールドのうち、1つが主キーであること
・同じデータ型であること
・数値型フィールドの場合、同じフィールドサイズであること
・関連付ける2つのテーブルが同じデータベース内にあること

規則	説明
多側テーブルへのレコード追加管理	一側テーブルの主キーにない値を、多側テーブルに入力できない
一側テーブルでの関連レコードの削除管理	一側テーブルのレコードを削除するときに、多側テーブルに関連付けられたレコードがある場合、削除できない

一側テーブルでの主キーの更新管理	一側テーブルの主キーの値を変更するとき、多側テーブルに関連付けられたレコードがある場合、変更できない

これらの制限のおかげで参照整合性が保たれるようになる！！
が、実際の運用において不便に感じる…
その場合、、、

・連鎖更新
　一側テーブルの主キーフィールドの更新が許可されます。更新した場合、多側テーブルの外部キーフィールドも自動的に更新されます
・連鎖削除
　一側テーブルのレコードを削除するとき、関連付けられたレコードが多側テーブルにあっても、レコードの削除が許可されます。
削除した場合、多側テーブルの関連レコードもすべて削除される為注意が必要。

上記２つを用いて制限を緩和することが可能

04.特殊なクエリ

SQLクエリの種類	説明
ユニオンクエリ	複数のテーブルやクエリのレコードを1つのクエリに結合して表示する
パススルークエリ	Microsoft SQL Serverなどのデータベースサーバに SQLクエリやコマンドを直接送信する
データ定義クエリ	テーブルの作成、変更、削除、あるいはインデックスやリレーションシップの作成を行うために使用する

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 02【変数・配列・ユーザー定義型・コレクション】

01.変数

◆変数の適用範囲と有効期間
・変数を宣言する場所
・変数を宣言するステートメント
すべての変数はデータベースファイルを閉じるタイミングで初期化されます

・適切な適用範囲とは
For …Nextステートメントのカウンタ変数「i」などは、多くのプロシージャで用いられている。
この変数をその都度、宣言するのが面倒だからと言ってパブリック変数で定義してもいいのか。
プロシージャ終了後も初期化されずに残ってしまうので、もしほかの場所で使用していたら、困ったことになる。
どこからでも、書き換えられるパブリック変数は必要なケースを除いて、使用しないほうが望ましいとも言えます。
変数は出来る限り局所的である(適用範囲が狭い)方が理想的です。
パブリック変数、モジュールレベル変数はどうしても複数のプロシージャで値を共有したいケースを除き、なるべく使用しないようにする！！！

ifステートメントの中で**変数定義すると外では使えない？？**
外でも使える！！使用前に定義していればよい。

```
    If True Then
        Dim ifi As Long
    End If
```

```
    ifi = ifi + 1
    Debug.Print ifi
```
この書き方でも問題無いことが分かった。
Javaのようなスコープ概念ではないみたい

・定数について
◆静的変数とは
staticステートメント
プロシージャ終了時に値を初期化せずに保持していたいときに使用する。
Static MyID As String で変数を宣言する

静的変数は宣言セクションで宣言することはできない
したがって、パブリック変数やモジュールレベル変数は扱えない！！

Static変数もパブリック変数、モジュールレベル変数も
endステートメントを使用すれば初期化される
そのままだと
・ファイルを閉じる
・モジュールに変更を加える
まで初期化されない

```
Option Explicit
Dim i As Long
'Static si As Long
Sub sample()
    Static si As Long
    publicI = publicI + 1
    i = i + 1
    si = si + 1
    Debug.Print i
    Debug.Print publicI
    Debug.Print si
End Sub
```

◆オブジェクト変数とは
・オブジェクト変数の宣言
固有オブジェクト型：具体的なオブジェト型を定義
総称オブジェクト型：Object型で定義
※少しだが、固有オブジェクト型の方が実行速度が向上するというメリットがある。

・Setステートメント
オブジェクト変数にオブジェクト参照を代入するには、Setステートメントを使用
Set オブジェクト変数 = 参照するオブジェクト

Point!!
通常の変数を代入するときはLetを省略している。。。
Let 変数 = 代入する値
オブジェクト変数のSetステートメントは省略することができないので、
Setを必ず記載する！！

・Nothingキーワード
オブジェクト変数に代入したオブジェクトへの参照を解除したいときは、
キーワード「Nothing」を代入する！
【オブジェクトへの参照を解除する】
Set オブジェクト変数 = Nothing
【オブジェクトへの参照が代入されているかどうかを調べる】
オブジェクト変数 Is Nothing
Trueを返すならオブジェクトへの参照が代入されていない！

02.配列

複数の値を変数に格納する。
◆配列とは

多次元配列は最大で**60次元**まで増やすことができる。このような変数を配列変数という。
・配列の宣言
【配列変数を宣言する】
Dim 変数(配列の要素数-1) As データ型
【配列変数に値を代入する】
変数(インデックス番号) = 代入する値

・Option Base ステートメント

配列のインデックス番号の最小値を「0」ではなく、「1」に変更できる
Option Base 最小値

なお！Option Baseステートメントは**必ず宣言セクションに記述**します。また！

Option Baseステートメントで設定した**インデックスの最小値は、同一モジュール内のすべての配列変数**に対して有効になります。

・Toキーワード
モジュール内の配列変数に個別に最小値を設定したい場合に使用する。
Dim 変数(最小値 To 最大値) As データ型

・多次元配列の宣言
【2次元配列の場合】
Dim 変数(1次元の要素数-1, 2次元の要素数-1) As データ型

※注意！！
多次元配列は非常に多くのメモリを使用します。
必要最小限の次元数にとどめて！！

◆動的配列とは
宣言時に要素数を決定していない配列
・動的配列の宣言
Dim 変数() As データ型
Redim 変数(要素数-1)
-1の意味について調査！！
要素数に変数を持ってくることができるので動的な配列宣言が可能！

・Preserveキーワード
Redimするときに格納されていた値の削除を回避することが可能！
ReDim Preserve 変数(新しい要素数-1)

※注意！！
多次元配列に使用した場合、、、
要素数を変更できるのは最後の次元に限られる！！
Toキーワードを使用してインデックスの最小値と最大値を指定している場合、
変更できるのは最大値に限られる！！

ちなみに静的配列はRedimステートメントで
サイズ変更不可能であることがわかった。。。

◆配列の初期化
Eraseステートメント：配列に格納されている値を一気に初期化する！
Erase 配列変数

※Eraseステートメントを使用すると、メモリの解放もしてくれる
静的配列の場合はメモリ解放を行わない
上記を使用すると再度redimしないとだめ？
動的配列の場合、Redimしないと要素自体なくなってしまう

コレクションからすべての項目を削除するには

- Clear メソッド (Collection オブジェクト) を使用します。

  object.Clear()

  Clear メソッドはコレクションを空にし

collectionクラスの要素を一括削除(RemoveAll)したい場合

```
Dim var As New Collection

var.Add(1)
var.Add(2)
var.Add(3)

'↓上記処理で追加した要素を削除して初期状態に戻す
Set var = New Collection
```

上記はメモリの解放もしてくれる？？

03.ユーザー定義型

一つの変数に複数の異なるデータ型の値を格納する
◆ユーザー定義型とは
ユーザーが独自に型の定義を行うことができる機能。
・Typeステートメント
ユーザー定義型の定義は、標準モジュール宣言セクションで行う
Type ユーザー定義型名
　　要素名 1　As データ型 1
　　要素名 2　As データ型 2
　　要素名 3　As データ型 3
End Type

04.コレクション

文字列、数値、オブジェクトなど異なるデータ型のデータを要素とする、独自のオブジェクトを作成することができる
◆コレクションとは
文字列や数値などのデータや、オブジェクトへの参照を要素として格納することができる「Collection」というオブジェクト
Collectionオブジェクトは変数と同じ方法で宣言を行います。
実際に使用するときはSetステートメントを使用。

コレクションオブジェクトにオブジェクトをAddしてみたい

Dim コレクション As Collection
Set コレクション = New Collection
または
Dim コレクション As New Collection

Collectionオブジェクトには、下記のメソッドとプロパティがある。

メソッド	内容
Add	コレクションに要素を追加する
Remove	コレクションから要素を削除する
Item	コレクションから指定した文字列に対応する要素を返す
プロパティ	内容
Count	コレクションの要素数を返す

Point コレクションに格納されている要素を「メンバ」と呼ぶこともあります。

```
Dim MyCol As New Collection

Private Sub Form_Load()
    MyCol.Add 12345
    MyCol.Add Me.txt1
    MyCol.Add Me.txt2
    MyCol.Add Me.txt3
    MyCol.Add "ABCDE"
End Sub

Private Sub btn1_Click()
    Dim i As Long
    Me.lbl1.Caption = ""
    For i = 1 To MyCol.Count
        Me.lbl1.Caption = Me.lbl1.Caption & _
                          MyCol(i) & _
                          vbCrLf
    Next i
End Sub
```

コレクションの値をFor文で回す時の記載方法。！

コレクションの要素を全部取り出す際は、
For i = 1 To MyCol.Count
として要素数の分だけ繰り返す！！

Point!!
コレクションのインデックス番号は必ず「１」から始まります。「Option Base 0」を
指定しても、最小値の「１」は変わりません。

◆コレクションに任意の順番で要素を追加する
コレクションのAddメソッドの引数に「Before」「After」を指定することで、コレクションの任意の順番で要素を追加することができる
MyCol.Add "B", Before:=1
MyCol.Add "C", After:=1

Point!! 名前付き引数で定義！
引数には、メソッドごとに決められている引数の順序に従って値を指定する方法と、
「引数名:=値」のように名前付き引数を使って明示的に値を指定する方法があります。
名前付きではなくてもよさげな！！！たぶん引数が多い時とかわかりにくいときに使ってるのではと推測。。。

◆オブジェクトの集合としてのコレクション
コレクションには、ユーザーが独自に作成できるオブジェクトのほかに
Accessが本来持っているオブジェクトの集合としてのコレクションがある。

主なコレクション	説明
Forms	現在開いているすべてのフォーム
Reports	現在開いているすべてのレポート
Printers	システムで使用可能なすべてのプリンタ
Me.Controls	フォーム・レポート上のすべてのコントロール
CurrentProject.AllForms	データベース内にあるすべてのフォーム
CurrentProject.AllReports	データベース内にあるすべてのレポート

これらのコレクションに対して一括処理を行うには、For Each...Nextステートメントを利用すると便利

```
Private Sub btn1_Click()
    Dim c As Object
    For Each c In Me.Controls
        Select Case c.ControlType
        Case acTextBox, acComboBox
            c.Value = ""
```

```
        Case acCheckBox
            c.Value = False
        Case acOptionGroup
            c.Value = 0
        End Select
    Next
End Sub
```

Point!!
「Forms」「Reports」「Printers」コレクションはApplicaitonオブジェクトいのメンバですので、本来は、「Applicaiton.Forms」のように記述するのですが、
「Applicaiton.」の部分は省略することができるので、記載していない

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 03【プロシージャ・モジュール】

01.プロシージャ

＊プログラムを構成する最小単位＊
◆プロシージャの種類
・標準プロシージャ：汎用的な処理を記述するもの
・イベントプロシージャ：特定のオブジェクトのイベントに関連付けられたもの
・プロパティプロシージャ：クラスモジュールで使用するもの
独自に作成したオブジェクトにプロパティの代入・取得するのに使用するプロシージャ
◆プロシージャの連携
メリット：汎用的な処理をプロシージャにまとめて、複数のプロシージャから呼び出すことによってコスト削減が可能
→ユーザー独自の関数が利用できること！！
＝Subプロシージャの呼び出し＝

```
Sub Test3()
    Call Test4(" 安藤 ")
    Call Test4(" 伊藤 ")
    Call Test4(" 宇野 ")

End Sub
Sub Test4(MyStr As String)
    Debug.Print MyStr & Space(1) & "様"
End Sub
```

＝Functionプロシージャの呼び出し＝
--省略

◆引数と戻り値
--引数の渡し方の説明
!!POINT
引数には、「名前付き引数」として引数の値を指定する方法があります。
引数１:＝値１, 引数２:＝値２, …
と記述します。この時「引数１」「引数２」には呼び出し先のプロシージャで
実際に設定されている引数名を使用します。
また！名前付き引数を使用した場合、呼び出し先のプロシージャの引数の順番通りに、引数を記述する必要はありません！！

◆参照渡しと値渡し
--省略：ExcelVBAで学習済み
ByVal ByRef
◆配列やユーザー定数型を引数で渡す
【呼び出し元のプロシージャ】
Call プロシージャ名(配列変数)
【呼び出し先のプロシージャ】
Sub プロシージャ名(ByRef 配列変数() As データ型)
・配列変数の次元数は、１次元でも多次元でも構いません。
・引数を渡すときは必ず参照渡しになります。
ByValキーワードを使用することはできません。

＃＃　配列引数に値渡しを定義すると…　＃＃
```
Sub ar(ByVal ar() as string)
End Sub
```
定義しようとすると、「配列引数はByRefでないといけません」というエラーがでた！！

＃＃　遊んでみた！！　＃＃
```
Sub count()
    Dim MyStr(10) As String
    Dim i As Long
    For i = 0 To 10
        If Int(Rnd * 10) <> 0 Then
        MyStr(i) = " セットされたよ!! " & i + 1 & "番目の要素"
        End If
    Next i
    Call countSub(MyStr)
End Sub

Sub countSub(ByRef MyStr() As String)
    Dim v As Variant
    For Each v In MyStr
        If v = "" Then
            Debug.Print "セットされなかったよ…"
        Else
        Debug.Print v
        End If
    Next
End Sub
```

INT関数を使用することで切り捨てで整数が取れる
INTの範囲を超えない範囲で、、

!!POINT
配列の中身を値渡ししたい時は、一旦Variant型変数に配列の中身を格納します。引数には配列変数ではなく配列に格納したVariant型変数を使用します。
```
Dim MyStr(1) As String
Dim MyVar As Variant
MyVar = MyStr
Call Test(MyVar)

Sub Test(ByVal MyVar As Variant)
```
→こうすることで値渡しで配列を引数に渡せる！

◆ユーザー定義型の変数を引数として渡す。
--配列の時とほとんど同様！！！
・必ず参照渡しになる。
ユーザー定義型の引数を持ったサンプルプロシージャを作ってみる

```
Sub UT()
    Call cl

End Sub

Sub cl(ByVal user As user)
    user.age = 11
    user.name = bob
    user.live = america


End Sub
```
想定通り落ちることがわかった。。！

＃＃　日付表示について考える　＃＃
```
Sub dateSample()
    Dim dt As Date
    dt = #1/1/2015#
    Debug.Print "日付を表示して見る" & dt
End Sub
```
・IsDate関数
IsDate(日付)
引数が日付であれば、Trueを返却する
2010/3/31
2010-3-31
```
? IsDate("2010/3-1")
? IsDate("2010 3 1")
? IsDate("2010-3/1")
```
これらも日付として扱われるので、
Trueを返す
```
? isdate(#1/2/2000#)
```

True

? isdate(#1 2 2000#)
True
? isdate(#1-2-2000#)
True

? isdate("1 2 2000")
True

? isdate("1/2-2000")
True

? isdate(#3 2 2000#)
True

◆プロシージャの適用範囲
標準モジュールで「Public Sub」または「Public Function」と先頭にPublicキーワードを記述して作成したプロシージャ、
またはPublicキーワードを省略して作成したプロシージャはすべてのモジュールから呼び出すことができる。

Privateキーワードをつけた場合、
同一モジュールにあるプロシージャカラの実、呼び出すことができる。

標準モジュール以外に作成したプロシージャはキーワードの記述に関係なく、他のモジュールから呼び出すことができません。
ココはExcelと違うところ！！！？
→Excelも同様。

02.モジュール

＊プロシージャを記述・格納する為のオブジェクトです＊
◆モジュールの種類
・標準モジュール
汎用的に処理される標準プロシージャを格納します。標準モジュールに記述されたプロシージャはPrivateキーワードを使用しないかぎりどのモジュールからでも参照することができます。
・フォームモジュール、レポートモジュール
フォーム、レポート及びそのコントロールに関連付けられたイベントプロシージャを格納します。
ここに標準プロシージャを作成した場合、他のモジュールからは参照できません。
そのオブジェクトだけで使用する共通した処理がある場合はココに記述するべき。
・クラスモジュール
独自のオブジェクトを作成するためのプロシージャを格納します。クラスモジュールに作成したプロパティプロシージャはそのオブジェクトのプロパティになります。またココに作成した標準プロシージャはそのオブジェクトのメソッドになります。
さらに、「Initialize」イベント「Terminate」イベントというクラスモジュール独自のイベントを利用することができます。
instancingプロパティを設定することができる。

◆モジュールの作成・削除

○モジュールの名前
Accessのナビゲーションウィンドウ またはVBEのプロパティウィンドウから自由に変更可能。
その場合、種類の異なるモジュールであっても、
モジュール名を重複させることはできないので注意してください！

!!POINT
異なる標準モジュールに同じ名前のプロシージャは定義可能！「モジュール名.プロシージャ名」で呼び出す！
特殊なケースを覗いて、同じ名前のプロシージャを複数のモジュールの定義することは推奨されない。。。

◆モジュールのエクスポート・インポート
モジュールをエクスポートすることで、データベースとは異なるファイルとして保存することができます。
エクスポートして保存したファイルは異なるデータベースにインポートして利用することが可能

＞エクスポート形式

モジュールの種類	ファイルの拡張子
標準モジュール	bas
フォームモジュール・レポートモジュール	cls
クラスモジュール	cls

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 04【フォームとレポートの操作】

01.フォーム・レポートの操作

◆フォーム・レポートを参照する構文
【対象となるオブジェクトのモジュールから参照する場合】
Me.メソッドまたはプロパティ
【それ以外のモジュールから参照する場合】
Forms(”フォーム名”).メソッドまたはプロパティ
Reports(”レポート名”).メソッドまたはプロパティ
【対象となるオブジェクトのモジュールから参照する場合】
Me.コントロール名.メソッドまたはプロパティ
【それ以外のモジュールから参照する場合】
Forms(”フォーム名”).コントロール名. メソッドまたはプロパティ
Reports(”レポート名”).コントロール名. メソッドまたはプロパティ

!!POINT
対象となるオブジェクトのモジュールからコントロールを参照する場合は「Meキーワード」を省略することが可能
!!POINT
対象となるオブジェクトのモジュール以外から参照する場合の記述方法

参照対象	構文
フォーム・レポート	Forms! フォーム名
	Forms![フォーム名]
	Forms.フォーム名
コントロール	Forms! フォーム名! コントロール名
	Forms("フォーム名")("コントロール名")
	Forms("フォーム名").Controls("コントロール名")

UNLOAD も、Me キーワードを省略可能？

02.サブフォーム・サブレポートの操作

フォームレポート内に配置されたフォームレポートのことを指す
メインフォーム：元になるフォーム
サブフォーム　：その中に配置されたフォーム

【対象となるオブジェクトのモジュールから参照する場合】
Me.サブフォーム名.Form.メソッドまたはプロパティ
Me.サブレポート名.Report.メソッドまたはプロパティ
【それ以外のモジュールから参照する場合】
Forms(”フォーム名”).サブフォーム名.Form.メソッドまたはプロパティ
Reports(”レポート名”).サブレポート名.Report.メソッドまたはプロパティ

【対象となるオブジェクトのモジュールから参照する場合】
Me.サブフォーム名.Form.コントロール名.メソッドまたはプロパティ
Me.サブレポート名.Report.コントロール名. メソッドまたはプロパティ
【それ以外のモジュールから参照する場合】
Forms(”フォーム名”).サブフォーム名.Form.コントロール名. メソッドまたはプロパティ
Reports(”レポート名”).サブレポート名.Report.コントロール名. メソッドまたはプロパティ

!!注意
ココで言う「サブフォーム名」は、メインフォームに配置されたサブフォームコントロールの名前
サブフォームの名前：[名前]プロパティの設定値
参照サブフォーム名：[ソースオブジェクト]プロパティの設定値

【メインフォーム・メインレポートを参照する場合】
Me.Parent.メソッドまたはプロパティ
【メインフォーム・メインレポートのコントロールを参照する場合】
Me.Parent.コントロール名.メソッドまたはプロパティ

Option Compare Database

```
Option Explicit

Private Sub frm1_AfterUpdate()
    ' 画面を再描画するには、True
    'させないなら、False
    DoCmd.Echo False
    Me.sub1.Form.FilterOn = False
    Select Case Me.frm1.Value
    Case Me.opt1.OptionValue
        Me.sub1.Form.Filter = " 部署コード  = 'B001'"
    Case Me.opt2.OptionValue
        Me.sub1.Form.Filter = " 部署コード  = 'B002'"
    Case Me.opt3.OptionValue
        Me.sub1.Form.Filter = " 部署コード  = 'B003'"
    Case Me.opt4.OptionValue
        Me.sub1.Form.Filter = " 部署コード  = 'B004'"
    Case Me.opt5.OptionValue
        Me.sub1.Form.Filter = " 部署コード  = 'B005'"
    Case Me.opt6.OptionValue
        Me.sub1.Form.Filter = ""
    End Select
    Me.sub1.Form.FilterOn = True
    DoCmd.Echo True
End Sub
```

オプショングループ作ってるっぽい。

03.フォーム間の連携

◆フォームからフォームを開く
DoCmdオブジェクトおOpenFormメソッドを使用する
複数のフォームを開いて処理を行う場合は「Forms("フォーム名")」と明示的に処理の対象となるフォームを指定
ScreenオブジェクトのActiveFormプロパティを使用することで現在のアクティブなフォームを参照することができる

!!POINT
一部のフォームをダイアログ・ボックスとして表示したい場合
DoCmd.OpenForm "フォーム名", , , , , acDialog
６番めの引数に「acDialog」の定数を指定する

```
＃＃　フォーム呼び出し元　＃＃
Option Compare Database
Option Explicit

Private Sub btn1_Click()
    ‘自オブジェクトの社員番号がNULLでなければ
    If Not IsNull(Me.社員番号.Value) Then
        ’フィルタ要素を指定して、ダイアログでウィンドウ表示をしている
        DoCmd.OpenForm "Fフォームを開く2", , , _
            "社員番号 = " & Me.社員番号.Value, , acDialog
    End If
End Sub
＃＃　フォーム呼び出し先　＃＃
Option Compare Database
Option Explicit

Private Sub btn1_Click()
    ‘現在のアクティブなフォームを閉じる！この場合自分のフォーム
    DoCmd.Close acForm, Screen.ActiveForm.Name
End Sub
```

◆フォームを開くときに引数を渡す
フォームを複数のフォームから開く可能性があるシステムで
どのフォームから開かれたのか判断したい場合に使用する。
OpenFormメソッドの７番目の引数「OpenArgs」で渡す。
開かれたフォームのOpenArgsプロパティから取得することができます。Me.OpenArgsと記述

```
＃＃　呼び出し元　＃＃
Option Compare Database
Option Explicit

Private Sub btn1_Click()
    DoCmd.OpenForm "Fフォームを開く4", _
    , , , , acDialog, Screen.ActiveControl.Name
End Sub

Private Sub btn2_Click()
    DoCmd.OpenForm "Fフォームを開く4", _
    , , , , acDialog, Screen.ActiveControl.Name
End Sub

Private Sub btn3_Click()
    DoCmd.OpenForm "Fフォームを開く4", _
    , , , , acDialog, Screen.ActiveControl.Name
End Sub
＃＃　呼び出し先　＃＃
Option Compare Database
Option Explicit
```

```
Private Sub Form_Open(Cancel As Integer)
    Me.txt1.Value = _
        " [" & Me.OpenArgs & "] ボタンより開かれました"
End Sub

Private Sub btn1_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

OpenFormメソッドについて調査

04.フォーム・レポートの応用テクニック

◆KeyPressイベントの応用テクニック
キーボードのANSIキーが押された時に発生します。
＊順番＊
KeyDown → KeyPress → Change → KeyUp
入力された文字の文字コードは「Asc」関数を利用して調べることができる
Asc(文字) → 文字コードを返す
!!POINT
ANSI文字コードとは
米国規格協会(American National Standards Institute)が定めた文字セット

```
Option Compare Database
Option Explicit

Private Sub txt1_KeyPress(KeyAscii As Integer)
    Select Case KeyAscii
    '[BackSpace][Tab][Enter]キーのいづれかが押された場合
    Case 8, 9, 13
        Exit Sub
    Case Else
        If KeyAscii >= Asc("0") And KeyAscii <= Asc("9") Then
            Me.lbl1.Caption = "数字キーが押されました [入力値] : " & Asc(KeyAscii)
        ElseIf KeyAscii >= Asc("a") And KeyAscii <= Asc("z") Then
            Me.lbl1.Caption = "英字キーが押されました [入力値] : " & Asc(KeyAscii)
        Else
            Me.lbl1.Caption = " キーの入力をキャンセルしました "
            KeyAscii = 0
        End If
    End Select
End Sub
```

◆GotFocusイベントの応用テクニック
GotFucusイベントはコントロールにフォーカスが写った時に発生します

```
Private Sub txt2_GotFocus()
    If IsNull(Me.txt2.Value) Then
        Me.txt2.Value = Me.txt1.Value
        Me.txt2.SelStart = Nz(Len(Me.txt1.Value), 0)
    End If
End Sub
```

Nz関数はAccessアプリケーションのメンバなのでExcelでは使用できない。
Excelで実現するには、

```
strName = IIF(IsNull(txtName), "名無し", txtName.Text)
```

◆BeforeUpdateイベントの応用テクニック
コントロールに対して入力されたデータが不正の場合、入力をキャンセルすることができる。

```
＃＃　処理　＃＃　引数がIntegerであることに注目。
Private Sub txt3_BeforeUpdate(Cancel As Integer)
    If IsNumeric(Me.txt3.Value) Then
        If CDbl(Me.txt3.Value) = CLng(Me.txt3.Value) Then
            Me.lbl2.Caption = "整数が入力されています"
            Exit Sub
        Else
            Me.lbl2.Caption = "整数以外の値が入力されています"
        End If
    Else
        Me.lbl2.Caption = "数値以外の値が入力されています"
    End If
    Cancel = True
    Me.txt3.SelStart = 0
    Me.txt3.SelLength = Nz(Len(Me.txt3.Value), 0)
End Sub

＃＃　全角半角判定　＃＃

Private Sub txt4_BeforeUpdate(Cancel As Integer)
    'Unicodeでの文字列バイト数を返します。
    If LenB(Me.txt4.Value) <> _
        'Unicodeからシステム規定のコードページに変換した文字列のバイト数を返します。
        LenB(StrConv(Me.txt4.Value, vbFromUnicode)) Then

        Me.lbl3.Caption = "半角文字が入力されています"
        Cancel = True
        Me.txt4.SelStart = 0
        Me.txt4.SelLength = Nz(Len(Me.txt4.Value), 0)
    Else
```

```
        Me.lbl3.Caption = "全角文字だけが入力されています"
    End If
End Sub
```

◆NotInListイベントの応用テクニック
リストに無いデータがコンボボックスに入力された時に発生するイベント
このイベントを利用することで、リストにないデータが入力された時に
登録用のフォームを開いて新規データを登録することができます。

＃＃　呼び出し元フォーム　＃＃

```
Private Sub cmb1_NotInList(NewData As String, Response As Integer)
    If MsgBox(" 入力されたデータを登録しますか？ ", vbYesNo) = _
        vbYes Then
        DoCmd.OpenForm "F応用2", , , , acFormAdd, acDialog, NewData
    End If
    If DCount("部署コード", "T部署マスタ", "部署コード='" & _
    NewData & "'") <> 0 Then
    Response = acDataErrAdded
    Else
        Response = acDataErrContinue
        Me.cmb1.Undo
    End If
End Sub
```

＃＃　呼び出し先フォーム　＃＃

```
Option Compare Database
Option Explicit

Private Sub Form_Open(Cancel As Integer)
    If IsNull(Me.OpenArgs) Then
        Cancel = True
    End If
End Sub


Private Sub Form_Load()
    Me.部署コード.Value = StrConv(Me.OpenArgs, vbUpperCase)
End Sub

Private Sub Form_BeforeUpdate(Cancel As Integer)
    If IsNull(Me.部署名.Value) Then
        Cancel = True
    End If
End Sub

Private Sub Form_AfterInsert()
    DoCmd.Close acForm, Me.Name
End Sub

Private Sub 部署名_KeyPress(KeyAscii As Integer)
    If KeyAscii = 27 Then
        DoCmd.Close acForm, Me.Name
    End If
End Sub
```

!!POINT
NoInListイベントプロシージャの引数
「NewData」：入力されたリストにない値が格納
「Response」：リストにない値をコンボボックスに追加するかしないかと、
システム メッセージを表示するかしないかを指定することができます。

【弱い…。再チェック必要】

◆Loadイベントの応用テクニック
フォームが読み込まれるときに発生！
Open → Load → Activeの順に発生します
このイベントを活用することで、フォームを開く際に必要な処理を実行させることができます。

```
Option Compare Database
Option Explicit

'＃＃フォームロードプロシージャ＃＃
'使用可能なプリンターをリストボックスに追加します。
Private Sub Form_Load()
    Dim o As Object
    For Each o In Application.Printers
        Me.lst1.AddItem o.DeviceName
    Next
既定のプリンターにリストを合わせている？
    Me.lst1.Value = Application.Printer.DeviceName

End Sub

'＃＃ボタンクリックプロシージャ＃＃
'選択されているプリンタで印刷を行うかどうかのポップアップを出します。
Private Sub btn1_Click()
    Dim MyPrinter As Object
```

```
    If MsgBox("選択されたプリンターで印刷しますか？", vbYesNo) _
        = vbYes Then
        '〇変数に規定プリンターを退避する
        Set MyPrinter = Application.Printer
        'リストよりプリンターを取得して既定プリンターに設定！
        Set Application.Printer = Application.Printers(Me.lst1.Value)
        '★ここで印刷処理を実行★
        DoCmd.OpenReport "R応用3"
        '〇退避させていた既定プリンターを元に戻す
        Set Application.Printer = MyPrinter
        Set MyPrinter = Nothing
    End If
End Sub
```

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 05【応用プログラミング】

01.クラスモジュールの利用

独自オブジェクトを利用する最大のメリットはオブジェクトの再利用！！！
◆クラスモジュールとは
①オブジェクトの動作を設計する手順
②設計したオブジェクトをプログラムの中で利用する手順

クラスモジュール：オブジェクトを設計するためのモジュール
インスタンス生成：オブジェクトをプログラムの中で利用すること

◆メソッドの作成、プロパティの作成
--メソッドの作成

--プロパティの作成
クラスモジュールに宣言されたパブリック変数は、そのオブジェクトのプロパティになる。
プロパティプロシージャを作成することでモジュールレベルの変数をプロパティとして利用することが可能

変数の種類	内容
パブリック変数	オブジェクトのプロパティとして他のモジュールから利用できる
モジュールレベル変数	プロパティプロシージャを使用することで、オブジェクトのプロパティとして他のモジュールから利用できる
プロシージャレベル変数	オブジェクトのプロパティとして利用できない

プロパティに値を設定する為には
・「Property Let」プロシージャ：プロパティに値を設定する為のもの
・「Property Get」プロシージャ：プロパティから値を取得するためのもの

プロシージャの種類	内容
Public Property Let(引数) (「Public」)の記述は省略可	オブジェクトのプロパティに値を設定する

Public Property Get(引数) As 戻り値 (「Public」)の記述は省略可	オブジェクトのプロパティに値を取得する

◆イベントの利用
クラスモジュールでは「Initialize」と「Terminate」の②つのイベントを利用することができます。
Initialize：オブジェクトのインスタンスが生成されるときに発生するイベント
Terminate：オブジェクトのインスタンスが破棄されるときに発生するイベント

◆インスタンスの生成
Dim インスタンス名 As クラスもジール名
Set インスタンス名 = New クラスモジュール名
または、
Dim インスタンス名 As New クラスモジュール名

＃＃　クラスモジュール　＃＃
```
Option Compare Database
Option Explicit

Public プロパティ1 As Long
Dim MyLng As Long

Private Sub Class_Initialize()
    Debug.Print "--- Initializeイベントの実行 ---"
    プロパティ1 = 100
    MyLng = 200
End Sub

Sub 出力()
    Debug.Print プロパティ1 & vbCrLf & MyLng
End Sub

Private Sub Class_Terminate()
    Debug.Print "--- Terminateイベントの実行 ----"
End Sub

Property Let プロパティ2(ByVal MyVal As Long)
    MyLng = MyVal
End Property

Property Get プロパティ2() As Long
    プロパティ2 = MyLng
End Property
```

!!POINT
「プロパティ１」プロパティはプロパティプロシージャがなくても、値の取得・設定を行うことが可能です。
しかし、このようなプロパティの使用方法は推奨されません。
→Javaのローカル変数の隠蔽みたいな、、、

02.コンポーネントの利用

＊コンポーネントとは＊
特定の機能を持つプログラムの部品を指します。
◆事前バインディング
Dim オブジェクト変数 As 特定のオブジェクト型
Set オブジェクト変数 = New 特定のオブジェクト型
または
Dim オブジェクト変数 As New 特定のオブジェクト型
→＃固有オブジェクト型による宣言＃
◆実行時バインディング(遅延バインディングとも呼ばれる)
Dim オブジェクト変数 As Object
Set オブジェクト変数 = CreateObject(オブジェクトのクラス名)
→＃総称オブジェクト型による宣言＃

◆オブジェクトの解放
プロシージャレベルのオブジェクトであれば、プロシージャが終了したタイミングで開放されるが、
エラーなどにより開放されない場合があるので、明示的に「Nothing」キーワードで解放すべき。

03.ファイル操作

◆カレントデータベースのパスと名前
CurrentProjectオブジェクト
Pathプロパティ：データベースファイルが保存されているフォルダのパスを文字列で返します。
Nameプロパティ：データベースファイルの名前を文字列で返します。

CurrentDb.Nameを使用すると、データベースファイルが保存されているフォルダのパスとデータベースファイルの名前を同時に取得できる！！

```
Option Compare Database
Option Explicit

Sub Test()
    Debug.Print CurrentProject.Path
    Debug.Print CurrentProject.Name
    Debug.Print CurrentDb.Name
End Sub
```

＃　出力結果　＃

```
D:¥work¥vba¥AccessVBAスタンダードサンプルデータベース（Access2007用）
S05（完成）.accdb
D:¥work¥vba¥AccessVBAスタンダードサンプルデータベース（Access2007用）¥S05（完成）.accdb
```

!!POINT
Dir関数を利用すると、パスの文字列から簡単にファイル名だけを抜き出すことができます。

◆FileSystemObjectの利用
「Microsoft Scripting Runtimeタイプライブラリー(Scrrun.dll)」
ファイルに格納されたドライブやフォルダ、ファイルなどを操作するためのオブジェクト

```
Option Compare Database
Option Explicit

Sub MyFSO()

    Dim FSO As New FileSystemObject
    Dim DrsCnt As Long
    Dim DrsItm As String
    DrsCnt = FSO.drives.Count
    DrsItm = FSO.drives.Item("C").Path


    Dim MyPath As String
    Dim workDir As String
    workDir = CurrentProject.Path
    MyPath = workDir & "¥cp"

    '##### コピー #####
    FSO.CopyFile MyPath & "¥01.txt", MyPath & "¥CopyFile.txt"
    FSO.CopyFolder MyPath, workDir & "¥CopyFolder¥"

    '##### 新規作成 #####
    On Error Resume Next
    FSO.CreateTextFile MyPath & "¥CreateText.txt"
    FSO.CreateFolder MyPath & "¥CreateFolder"
    On Error GoTo 0
    '##### 削除 #####
    FSO.DeleteFile MyPath & "¥CreateText.txt"
    FSO.DeleteFolder MyPath & "¥CreateFolder"

    '##### 存在確認 #####
    If FSO.DriveExists("C") Then
        log "Cドライブは存在します"
    End If
    If FSO.FileExists(MyPath & "¥01.txt") Then
        log "01.txtは存在します"
    End If
    If FSO.FolderExists(MyPath & "¥test") Then
        log "testは存在します"
    End If

    '##### オブジェクト取得 #####
    Dim fsoDrive As Drive
    Dim fsoFolder As Folder
    Dim fsoFile As File
    Set fsoDrive = FSO.GetDrive("C")
    Set fsoFolder = FSO.GetFolder(workDir & "¥test")
    Set fsoFile = FSO.GetFile(MyPath & "¥01.txt")

    '##### 移動する #####
    FSO.MoveFolder workDir & "¥cp", workDir & "¥test¥"
    FSO.MoveFile workDir & "¥test¥03.txt", workDir & "¥test¥03_move.txt"



End Sub

Sub EditText()
    Dim str As String
    Dim i As Long

    '##### ファイルを開く #####
    Dim FSO As New FileSystemObject
    Dim MyRead As TextStream
    Dim MyWrite As TextStream
    Dim MyAppend As TextStream
    Dim MyPath As String
    MyPath = CurrentProject.Path & "¥fso"
    Set MyRead = FSO.OpenTextFile(MyPath & "¥fsoRead.txt")
    Set MyWrite = FSO.OpenTextFile(MyPath & "¥fsoWrite.txt", ForWriting)
```

```
    Set MyAppend = FSO.OpenTextFile(MyPath & "¥fsoAppend.txt", ForAppending)
    Do While MyRead.AtEndOfStream = False
        str = str & vbCrLf & MyRead.ReadLine
    Loop
    MsgBox str
    For i = 1 To 10
        MyWrite.WriteLine "ForWriting >" & String(i, "#")
    Next
    For i = 1 To 10
        MyAppend.WriteLine "ForAppned >" & String(i, "*")
    Next

    Set MyRead = Nothing
    Set MyWrite = Nothing
    Set MyAppend = Nothing

End Sub


Sub log(log As String)
    Debug.Print Date & Space(1) & Time & Space(1) & log
End Sub
```

◆FileDialogオブジェクトの利用

ファイルまたはフォルダを参照するダイアログボックスを表示し、ユーザーが選択したファイルまたはフォルダへのパスを格納します。
※参照設定

「Microsoft Office XX.X Object Library」

プロパティ	内容
Title	ダイアログボックスのタイトルを指定する
ButtonName	操作ボタンに表示される文字列を指定する
AllowMultiSelect	複数ファイルの選択が可能かどうかを設定する
InitialFileName	初期表示されるファイル名やフォルダパスを指定する
InitialView	ファイルやフォルダの表示方法をMsoFileDialogViewクラスの定数で指定する
Filters	ダイアログボックスで選択できるファイルの種類を設定する
FilterIndex	ダイアログボックスを開いた時に表示されるファイルの種類を設定する
SelectedItems	ユーザーが選択したファイルのパスを取得する

メソッド	内容
Show	ダイアログボックスを表示し、[キャンセル]ボタンが選択されると「０」をそれ以外が選択されると「－１」を返す

InitialViewの定数一覧

定数	内容
msoFileDialogViewDetails	詳細表示
msoFileDialogViewLargeIcons	大きいアイコンで表示
msoFileDialogViewList	一覧表示
msoFileDialogViewPreview	プレビューで表示
msoFileDialogViewProperties	プロパティで表示
msoFileDialogViewSmallIcons	小さいアイコンで表示
msoFileDialogViewThumbnail	縮小表示

```
Sub Test()
    Dim MyDialog As FileDialog
    Set MyDialog = FileDialog(msoFileDialogFilePicker)
    MyDialog.InitialFileName = CurrentProject.Path
    MyDialog.Filters.Clear
    MyDialog.Filters.Add "テキスト", "*.txt", 1
    MyDialog.Filters.Add "エクセル", "*.xls", 2
    MyDialog.Filters.Add "すべてのファイル", "*.*", 3
    MyDialog.FilterIndex = 3                        'リストで３番めに設定したファイルの種類[すべてのファイル(*.*)]が
                                                    'ダイアログボックスを開いた時に表示される
    If MyDialog.Show Then
        Debug.Print MyDialog.SelectedItems(1)
    Else
        Debug.Print "キャンセルされました"
    End If
    Set MyDialog = Nothing
End Sub
```

POINT!!!!
フォルダを選択するダイアログは「Shell.Application」を利用することでも表示することが出来ます。

```
Sub Test2()
    Dim MyShell As Object
    Set MyShell = CreateObject("Shell.Application")
    Set MyShell = _
        MyShell.BrowseForFolder(&H0, "フォルダを選択", &H0)
    If Not MyShell Is Nothing Then
        Debug.Print MyShell.Items.Item.Path
        Set MyShell = Nothing
    End If
End Sub
```

04.OLEオートメーション

OLE：「Objetct Liking and Embedding」の略称で、アプリケーション間でオブジェクトをやり取りするための規格です。

◆Excelとの連携
□事前バインディング
Dim オブジェクト変数 As Excel.Application
set オブジェクト変数 = New Excel.Application
または
Dim オブジェクト変数 As New Excel.Application
□実行時バインディング
Dim オブジェクト変数 As Object
Set オブジェクト変数 = CreateObject("Excel.Applicaiton")

POINT!!!
・オートメーションサーバー　　：オブジェクトを外部に公開する側
・オートメーションクライアント　：それを制御する側

Excelには「Excel.Application」以外にもオートメーションサーバーに使用できるオブジェクトがあります。

オブジェクトのクラス名	オブジェクトの種類
Excel.Application	Excelアプリケーション本体
Excel.Sheet	Excelワークシート
Excel.Chart	Excelグラフ

○参照設定
「Microsoft Excel XX.X Object Library」

```
Option Compare Database
Option Explicit


Sub Test1()
    Dim MyExcel As New Excel.Application
    Dim MyBook As Workbook
    Dim i As Long, j As Long
    MyExcel.Visible = True
    Set MyBook = MyExcel.Workbooks.Add              'ワークブックをAddしている！！
    For i = 1 To 5                                  '行インクリメント
        For j = 1 To 5                              '列インクリメント
            MyBook.ActiveSheet.Cells(i, j).Value _
             = "test(" & i & "," & j & ")"          '指定セルに値の設定
        Next j
    Next i

    MsgBox "Excelを開きました"
    MyBook.SaveAs CurrentProject.Path & "¥sample"   '指定された名称で保存
    MyExcel.Quit
    Set MyBook = Nothing
    Set MyExcel = Nothing

End Sub


Sub Test2()
    Dim MyExcel As New Excel.Application
    Dim MyBook As Workbook
    Dim MyVar As Variant, v As Variant
    Dim MyStr As String, i As Long
    MyExcel.Visible = False
    Set MyBook = _
        MyExcel.Workbooks.Open(CurrentProject.Path & "¥sample")
    MyVar = MyBook.ActiveSheet.UsedRange            'ワークブックで使用している範囲「A1:E5」のセルの値を格納
    For Each v In MyVar
        i = i + 1
        If i Mod 5 <> 0 Then
            MyStr = MyStr & v & Space(2)
        Else
            MyStr = MyStr & v & vbCrLf
        End If

    Next
    MsgBox MyStr
    MyExcel.Quit
    Set MyBook = Nothing
    Set MyExcel = Nothing


End Sub
```

あんまり実用性がある例が浮かばなかったので穴埋め問題を作成して覚えることにする。。。。

使用済みセルの最終行と最終列の取得にはWorksheetオブジェクトの値があるすべてのセルを範囲とする UsedRange プロパティを使用します

```
Sub CellCnt()
    Dim lngYCnt As Long
    Dim intXCnt As Integer

    lngYCnt = Worksheets("Sheet1").UsedRange.Rows.Count
    intXCnt = Worksheets("Sheet1").UsedRange.Columns.Count
    MsgBox "最終行は" & intYCnt & "行、" & _
                        "最終列は" & lngXCnt & "列です"
End Sub
```

◆InternetExplorerとの連携

□事前バインディングによる参照
□実行時バインディングによる参照

```
Option Compare Database
Option Explicit

Sub Test3()

    Dim MyIE As New InternetExplorer
    MyIE.Visible = True
    MyIE.Navigate "http://vbae.odyssey-com.co.jp/"
    MsgBox "InternetExplorerを開きました"
    MyIE.Quit
    Set MyIE = Nothing

End Sub

Sub Test4()
    Dim MyIE As New InternetExplorer
    MyIE.Visible = False
    'MyIE.Navigate "http://vbae.odyssey-com.co.jp/"
    MyIE.Navigate "http://google.com/"
    Do While MyIE.Busy
        DoEvents
    Loop
    Do While MyIE.Document.ReadyState <> "complete"
        DoEvents
    Loop
    MsgBox MyIE.Document.body.InnerText
    MyIE.Quit
    Set MyIE = Nothing

End Sub
```

05.VBAの高速化

より効率的なプログラミングとは何か、無駄な処理を行わないためにはどうすればよいのかについて説明！

◆高速化の考え方
○無駄な処理を行わないヒント

～いろいろ記載が会ったけど略～

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 06【SQL】

01.あいまい検索

SQL「Structured Query Language」

◆パターンマッチングによる条件指定

SELECT * FROM テーブル名 WHERE 抽出条件;
または
SELECT フィールド名1, フィールド名2 , フィールド名3 , フィールド名4... FROM テーブル名
WHERE 抽出条件 ;

この時**Like演算子**を使用すると、パターンマッチングによる検索指定を行える。

フィールド名 Like "パターン文字列"

文字	説明	例	抽出結果
*	任意の数の文字	Like"*本"	本、見本、単行本など
?	任意の1文字	Like"?本"	日本、岡本、見本など
#	任意の1文字の数字	Like"#組"	1組、2組など
[文字リスト]	文字リスト内の1文字	Like"g[eo]t"	get, gotなど
[!文字リスト]	文字リスト以外の1文字	Like"p[!eo]t"	pat, putなど
[文字1-文字2]	文字1~2の範囲の1文字	Like"b[a-c]t"	bat,bbt, bctなど
[!文字1-文字2]	文字1~2の範囲以外の1文字	Like"b[!a-c]t"	bet, bitなど

[a-z]のように昇順で検索する必要がある。
[z-a]のように降順で範囲を指定すると、正しく検索を行いません、、、

※注意 **SQLクエリモードについて**

ANSI-89(**DAO**)	ANSI-92(**ADO**)	説明
*	%	任意の数の文字
?	_	任意の1文字
[!文字リスト]	[^文字リスト]	文字リスト以外の1文字

DoCmd.OpenQuery SQL文字列 でクエリを実行します。

```
Option Compare Database
Option Explicit

Sub Test1()
    Dim StrSQL As String
    StrSQL = "SELECT 社員番号, 社員名 " & _
             "FROM T社員名簿 " & _
             "WHERE 社員名 Like '*藤*' ;"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.OpenQuery "Qクエリ"
End Sub



Sub Test2()
    Dim StrSQL As String
    StrSQL = "SELECT 社員番号, 社員名 " & _
             "FROM T社員名簿 " & _
             "WHERE 社員名 Like '*[藤野]*' ;"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.OpenQuery "Qクエリ"
End Sub
```

02.レコードのグループ化

◆レコードをグループ化する
GROUP BY句を使用する

SELECT … FROM テーブル名 GROUP BY フィールド名１, フィールド名2 …;

◆グループへの条件指定
HAVING句を使用する

SELECT … FROM テーブル名 GROUP BY フィールド名1, フィールド名2 … HAVING 条件式;

!!POINT
WHERE句は、グループ化する前のレコードに対して条件式を指定します。
HAVING句では、グループ化した後のレコードに対して条件式を指定します。
WHERE句は、SQL集計関数を条件式に使用できませんが、
HAVING句では使用することが出来ます。

■よくある課題■
① WHERE句 ▷ GROUP BY ： WHERE句で絞りこまれてから、GROUP BYで集約される
② GROUP BY ▷ HAVING ： GROUP BY で集約してからHAVINGで絞り込む
②の方が高速に動作する
WHERE句で事足りる検索に対して、HAVING句を使用するのは非効率的

```
Option Compare Database
Option Explicit

Sub Test1()
    Dim StrSQL As String
    StrSQL = "SELECT 部署コード, " & _
             "Sum(年齢) AS 年齢合計 " & _
             "FROM T社員名簿 " & _
             "GROUP BY 部署コード " & _
             "HAVING Sum(年齢) >= 90;"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.OpenQuery "Qクエリ"
End Sub

Sub Test2()
    Dim StrSQL As String
    StrSQL = "SELECT 部署コード, " & _
              "Sum(年齢) AS 年齢合計 " & _
              "FROM T社員名簿 " & _
              "WHERE 部署コード <> 'B001' " & _
              "GROUP BY 部署コード " & _
              "HAVING Sum(年齢) >= 90;"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.OpenQuery "Qクエリ"
End Sub
```

03.テーブルの結合

◆内部結合
2つのテーブルの共通するフィールドの値が一致する場合のみ、2つのテーブルを結合します。
INNER JOIN句

!!POINT
結合フィールドは同じフィールド名である必要はありません。
ただし、データ型と種類が同じである必要があります。

```
Sub Test1()
    Dim StrSQL As String
    StrSQL = "SELECT T商品マスタ.商品コード, " & _
             "商品名, 在庫数 " & _
             "FROM T商品マスタ " & _
             "INNER JOIN T在庫マスタ " & _
             "ON T商品マスタ.商品コード = T在庫マスタ.商品コード;"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.OpenQuery "Qクエリ"

End Sub
```

◆外部結合
結合フィールドの値が一致しないレコードを含んだ結果を取得する
左外部結合 LEFT JOIN
右外部結合 RIGHT JOIN

!!POINT
結合したテーブルに値が存在しない場合は、Null値が入る

```
Sub Test2()
    Dim StrSQL As String
    StrSQL = "SELECT T商品マスタ.商品コード, " & _
             "商品名, 在庫数 " & _
             "FROM T商品マスタ " & _
             "LEFT JOIN T在庫マスタ " & _
             "ON T商品マスタ.商品コード = T在庫マスタ.商品コード;"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.OpenQuery "Qクエリ"

End Sub
```

```
Sub Test3()
    Dim StrSQL As String
    StrSQL = "SELECT T在庫マスタ.商品コード, " & _
             "商品名, 在庫数 " & _
             "FROM T商品マスタ " & _
             "RIGHT JOIN T在庫マスタ " & _
             "ON T商品マスタ.商品コード = T在庫マスタ.商品コード;"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.OpenQuery "Qクエリ"

End Sub
```

◆不一致レコードの抽出
外部結合で抽出条件にNull値のレコードを指定することで、不一致レコードを抽出することが出来ます。

```
Sub Test4()
    Dim StrSQL As String
    StrSQL = "SELECT T商品マスタ.商品コード, " & _
             "商品名, 在庫数 " & _
             "FROM T商品マスタ " & _
             "LEFT JOIN T在庫マスタ " & _
             "ON T商品マスタ.商品コード = T在庫マスタ.商品コード " & _
             "WHERE T在庫マスタ.商品コード IS NULL;"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.OpenQuery "Qクエリ"

End Sub
```

◆UNIONによる結合
UNION演算子：　複数のテーブルやクエリの結果を1つにまとめて取得する
重複するレコードは除外され1件のレコードに纏められます。

```
Sub Test5()
    Dim StrSQL As String
    StrSQL = "SELECT * FROM T社員名簿 " & _
             "UNION " & _
             "SELECT * FROM T新入社員;"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.OpenQuery "Qクエリ"

End Sub
```

04.テーブル定義の変更

ALTER TABLEステートメントを使用します。

【新しいフィールドを追加する場合】
ALTER TABLE テーブル名 ADD COLUMN フィールド名 データ型(サイズ);

【既存のフィールドの属性を変更する場合】
ALTER TABLE テーブル名 ALTER COLUMN フィールド名 データ型(サイズ);

【既存のフィールドを削除する場合】
ALTER TABLE テーブル名 DROP COLUMN フィールド名;

DoCmd.RunSQLで実行している！！！
引数にSQL文字列を指定すれば実行できるが、
更新や削除などのアクションクエリのみ！！

```
Option Compare Database
Option Explicit


Sub Test1()
    Dim StrSQL As String
    StrSQL = "ALTER TABLE T在庫マスタ " & _
             "ADD COLUMN 備考 TEXT(4);"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.SetWarnings False
    DoCmd.OpenQuery "Qクエリ"
    DoCmd.SetWarnings True

End Sub

'上記の「Test1」プロシージャはDoCmd.RunSQLメソッドを使用して記述しても同様の動作をします。
Sub Test1()
    Dim StrSQL As String
    StrSQL = "ALTER TABLE T在庫マスタ " & _
```

```
        "ADD COLUMN 備考 TEXT(4);"
    DoCmd.SetWarnings False
    DoCmd.RunSQL StrSQL
    DoCmd.SetWarnings True
End Sub


Sub Test2()
    Dim StrSQL As String
    StrSQL = "ALTER TABLE T在庫マスタ " & _
        "ALTER COLUMN 備考 TEXT(40);"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.SetWarnings False
    DoCmd.OpenQuery "Qクエリ"
    DoCmd.SetWarnings True

End Sub
```

DoCmd オブジェクト (Access)

DoCmd オブジェクトのメソッドを使用すると、Visual Basic から Microsoft Office Access のアクションを実行できます。アクションによって、ウィンドウを閉じる、フォームを開く、コントロールの値を設定する、などのタスクを実行します。

＊DoCmd.SetWarnings メソッド (Access)
Office 2013 and later
SetWarnings メソッドでは、Visual Basic で "SetWarnings/メッセージの設定" アクションを実行します。

05.インデックスとは

◆インデックスとは
インデックスを設定したフィールドでは、
並べ替え・検索などの処理時間が大幅に短縮されます。
インデックスを固有にすることで、他のレコードと値が重複しないように設定することも出来ます。
新しいインデックス作成するには
CREATE INDEXステートメント を使用します
削除するには
DROP INDEXステートメントを使用します

【インデックスを作成する】
CREATE INNDEX インデックス名 ON テーブル名(フィールド名1, フィールド名2 …)

【インデックスを削除する】
DROP INDEX インデックス名 ON テーブル名;

```
Sub Test1()
    Dim StrSQL As String
    StrSQL = "CREATE INDEX idx備考 " & _
        "ON T在庫マスタ(備考);"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.SetWarnings False
    DoCmd.OpenQuery "Qクエリ"
    DoCmd.SetWarnings True

End Sub



Sub Test2()
    Dim StrSQL As String
    StrSQL = "DROP INDEX idx備考 " & _
        "ON T在庫マスタ ;"
    CurrentDb.QueryDefs("Qクエリ").SQL = StrSQL
    DoCmd.SetWarnings False
    DoCmd.OpenQuery "Qクエリ"
    DoCmd.SetWarnings True

End Sub
```

06.SQLの高速化

◆無駄な処理を行わない

```
SELECT * FROM 社員名簿;
```

↓

```
SELECT 社員番号, 社員名 FROM 社員名簿;
```

無駄な列を排除することで高速化が出来ます。

◆より高速な方法を選択する
UNION演算子
「UNION」で結合するよりも
「UNION ALL」で結合する方がより高速に動きます。

・SQL関数の引数に使用するフィールド、
　並べ替えや検索に使用するフィールド　にはインデックスを設定する

・データベースの正規化を適切に行う
・主キーを適切に設定する
・使用していない不要なインデックスを排除する

※※※※インデックスを付与し過ぎると※※※※※※※※
※ 更新処理(INSERT/UPDATE/DELETE)が遅くなる ※※※
※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 07【ADO/DAO】

VBAによる業務システムの開発では、データベースに格納されているデータをVBAから直接操作したいケースが頻繁にある。
ADO/DAOを使用することでデータを直接操作できるようになります。

01.ADO(AcriveX Data Object)とは

データベースを操作する為の様々なオブジェクトを持つオブジェクトライブラリです。

ADOを使用しなくても、Docmdオブジェクトを利用して、ある程度データベース操作を
行うことが出来ます。
しかし、複数のレコードをレコード単位で条件分岐して更新を行う等、細かなデータベース操作を行うにはADOを使用する必要があります。

さらに、、、

ACCESS以外の他のデータベースを共通の方法で操作できるという大きなメリットがあります。

①データベースに接続する
②検索やレコードの追加、更新、削除など、データベースへの操作を行う
③データベースへの接続を解除する

オブジェクトの種類	説明
Connection	データベースへの接続を保持するオブジェクト
Command	データベースへのコマンドを保持するオブジェクト
Recordset	レコードの集まりを保持するオブジェクト
Field	フィールドを保持するオブジェクト
Parameter	パラメータを保持するオブジェクト
Property	プロパティを保持するオブジェクト
Error	エラーを保持するオブジェクト

◆参照設定
「Microsoft ActiveX Data Object X.X Library」にチェックを入れる

##
◆データベースへの接続

＊**Connection**オブジェクト

データベースへの接続を保持するオブジェクトです。

プロパティ	説明
ConnectionString	データベースへの接続情報を返す
State	データベースへの接続状態を返す

メソッド	説明
Open	データベースへの接続を開く
Close	データベースへの接続を閉じる
Execute	コマンドを実行する
BeginTrans	トランザクションを開始する
CommitTrans	変更を保存してトランザクションを終了する
RollbackTrans	変更を取り消してトランザクションを終了する

Stateプロパティ
・接続時　：「adStateOpen」
・未接続時：「adStateClosed」

Dim オブジェクト変数 As ADODB.Connection

ADODBはADOが提供するコンポーネントの名前
ADOを使用する際は、ADODBを使用して明示的にADOのオブジェクトであることを宣言することを推奨します。
接続する際は　；　ConnectionStringプロパティに「接続文字列」を設定
ただしカレントデータベースに接続する場合はCurrentProjectオブジェクトのConnectionプロパティを使用する為、接続文字列は必要ありません。
【カレントデータベースに接続する場合】
Set CN = CurrentProject.Connection

【カレント以外のデータベースに接続する場合】
```
Set CN = New ADODB.Connection
CN.ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;" & _
                      "Data Source=データベースファイルのパス"
CN.Open
```

ここでいうカレントデータベースは、現在開いているデータベースファイルを指します。
それ以外のデータベースとは、Accessで作成されたカレントデータベース以外のデータベースファイルを指します。

!!POINT
ConnectionStringプロパティ　：　データベースの種類を表す「Provider(データプロバイダ)」
　　：　接続先のデータベースを表す「Data Source(データソース)」
Dim CN As New ADODB.Connection
と記述することで、オブジェクト変数の宣言とインスタンスの生成を一度に行うことが出来ます。

▼ポイント
CN.ConnectionString = "接続文字列"
CN.Open
とするか
CN.Open "接続文字列"
とするかで内容は一緒！！

##########
◆レコードの操作
ADOでは主にレコードセットと呼ばれるレコードの集まりを取得
レコードセットに対して追加・更新・削除などの操作を行います。

○Recordsetオブジェクト

プロパティ	定数	説明
BOF		カレントレコードが先頭レコードの前にあるときTrueをそれ以外はFalseを返す
EOF		カレントレコードが最終レコードの後にあるときTrueをそれ以外はFalseを返す
RecordCount		レコード件数を返す
CursorType		カーソルタイプを返す
	adOpenForwardOnly(既定)	前方スクロールカーソル レコードを前方向にのみ移動することができ処理が高速。その他は静的カーソルと同じ動きをする
	adOpenKeyset	キーセットカーソル 他のユーザーによる追加・削除は確認できない。 その他は動的カーソルと同じ動きをする
	adOpenDynamic	動的カーソル レコードをすべての方向に移動することができる。他のユーザーによる追加・更新・削除を確認できる
	adOpenStatic	静的カーソル レコードをすべての方向に移動することができる。他のユーザーによる追加・更新・削除は確認できない
CursorLocation		カーソルの場所を指定する
	adUseServer(既定)	サーバー側のカーソルを使う
	adUseClient	クライアント側のカーソルを使う
LockType		ロックタイプを返す
	adLockReadOnly(既定)	読み取り専用
	adLockPessimistic	レコード単位の排他的ロックを表す
	adLockOptimistic	レコード単位の共有的ロックを表す
	adLockBatchOptimistic	共有的バッチ更新を表す
	adLockUnspecified	ロックタイプを指定しない
Bookmark		レコードを識別するためのブックマークを表す

メソッド	説明	
Open		レコードセットを開く
Close		レコードセットを閉じる
Move/ MoveFirst/ MoveLast/ MoveNext/ MovePrevious		カレントレコードを移動する
Find		レコードを検索する
Clone		レコードセットのコピーを作る
AddNew		レコードを追加する
Update		レコードを更新する
Delete		レコードを削除する

Recordsetオブジェクトのオブジェクト変数を宣言する場合は
Dim オブジェクト変数 As ADODB.Recordset

■方法①：RecordsetオブジェクトのOpenメソッドで開く
Dim RS As ADODB.Recordset
Set RS = New ADODB.Recordset
RS.Open ソース, CN, カーソルタイプ, ロックタイプ

引数	説明
ソース	テーブル名、クエリ名、SQLステートメントなどを指定
CN	Connectionオブジェクトを指定
カーソルタイプ	カーソルタイプを指定。CursorTypeプロパティの定数を指定する
ロックタイプ	ロックタイプを指定。LockTypeプロパティの定数を指定する

Dim RS As New ADODB.Recordset
RS.Open ソース, CN, カーソルタイプ, ロックタイプ
変数の宣言とインスタンスの生成を一度で可能

!!POINT
引数に予めRecordsetオブジェクトのActiveConnectionプロパティ、Sourceプロパティに設定しておく方法もある
RS.Open "T社員名簿", CN

RS.ActiveConnection = CN
RS.Source = "T社員名簿"
RS.Open

上記②つのパターンは同様に動作する

■方法2：ConnectionオブジェクトのExecuteメソッドを使用する
Dim RS As ADODB.Recordset
Set RS = CN.Execute(コマンド)

CN　　：Connectionオブジェクト
コマンド：テーブル名、クエリ名、SQLステートメントなどを指定

!!POINT
ConnectionオブジェクトのExecuteメソッドの結果をRecordsetオブジェクトで取得した場合、レコードセットは常に前方スクロールカーソル読み取り専用となります。
またExecuteメソッドの結果がレコードセットで帰らない場合(アクションクエリを実行した場合)指定したアクションが実行されます。
その場合は、レコードセットのオブジェクト変数は使用しません。
次のように記述します。
CN.Execute コマンド

○カレントレコードの移動
レコードセットを開いた直後は、カレントレコードはレコードセットの先頭レコードに設定されています。
カレントレコードの移動は、Move系メソッドを使用します。

メソッド	説明
MoveFirst	先頭レコードに移動する
MoveLast	最終レコードに移動する
MoveNext	次のレコードに移動する
MovePrevious	前のレコードに移動する
Move番号	引数「番号」に指定した数だけ移動する

※注意※
レコードセットのカーソルタイプに注意
Recordsetのカーソルタイプは既定で前方スクロールカーソル担っています。
前方以外にも移動する場合は、カーソルタイプを変更しないとMove系メソッドを実行した時にエラーが発生します。

	レコードセット内のカーソル位置	BOFプロパティ	EOFプロパティ
①	BOF(先頭レコードより前)	TRUE	FALSE
②	先頭レコード～最終レコード	FALSE	FALSE
③	EOF(最終レコードより後)	FALSE	TRUE

①の時MovePreviousメソッドを実行するとエラー
③の時MoveNextメソッドを実行するとエラー

先頭から順番にレコードを取るコード

```
Do Until RS.EOF
    実行させる処理
    RS.MoveNext
Loop
```

なお、レコードセットにレコードが一件もないときは、
BOF、EOFはともにTrueを返します。
この時Move系メソッドを使用すると、エラーが発生します。

```
Sub Test1()
   Dim CN As ADODB.Connection
   Dim RS As ADODB.Recordset
   Set CN = CurrentProject.Connection
   Set RS = CN.Execute("T社員名簿")
   Do Until RS.EOF
      Debug.Print RS.Fields(0), _
                  RS.Fields(1), _
                  RS.Fields(2)
      RS.MoveNext
   Loop
   RS.Close: CN.Close
   Set RS = Nothing: Set CN = Nothing

End Sub


Sub Test2()
   Dim CN As ADODB.Connection
   Dim RS As ADODB.Recordset
   Set CN = CurrentProject.Connection
   Set RS = New ADODB.Recordset
   RS.Open "T社員名簿", CN, adOpenStatic
   RS.MoveLast
   Do Until RS.BOF
      Debug.Print RS.Fields(0), _
                  RS.Fields(1), _
                  RS.Fields(2)
      RS.MovePrevious
   Loop
   RS.Close: CN.Close
   Set RS = Nothing: Set CN = Nothing

End Sub
```

!!POINT
ステートメントの後ろに「:(コロン)」を入力することで、1行に複数のステートメントを記述することが可能です。
コードの行数を減らしたい時に利用すると便利です。

○レコードの更新
Updateメソッドを使用
◆パターン１
【Updateメソッドの引数として、フィールド名、値を渡す】

```
RS.Update フィールド名１, 値１
RS.Update フィールド名２, 値２
RS.Update フィールド名３, 値３
：
```

◆パターン２
【FieldオブジェクトのValueプロパティに値を代入し、Updateメソッドを実行する】

```
RS("フィールド１").Value = 値１
RS("フィールド２").Value = 値２
RS("フィールド３").Value = 値３
：
RS.Update
```

上記２つのパターンが存在する

＊フィールドを参照する主な記述は下記の通り

```
RS![フィールド名].Value
RS.Fields("フィールド名").Value
RS("フィールド名").Value
RS.Fields(n).Value (nは0から始まるインデックス番号)
RS(n).Value (nは0から始まるインデックス番号)
```

Upadateメソッドの引数として、フィールド名、値を指定した場合、Updateメソッドが実行されると、
直ちに！レコードソースに値の変更が反映されます。

Fieldsオブジェクトに値を代入し最後にUpdateメソッドを実行した場合、Updateメソッドが実行された時点、または
カレントレコードから他のレコードに移動した時点で、レコードソースに値が反映されます。
なおUpadateメソッドを実行してもカレントレコードは移動しません。

!!POINT
レコードセットのレコードに更新・追加・削除の操作を行うには、
レコードセットのLockTypeプロパティに「adLockOptimistic」などの更新可能なロックタイプを指定する必要がある。

```
Option Compare Database
```

```
Option Explicit

Sub Test1()

    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Dim SQL As String
    Set CN = CurrentProject.Connection
    Set RS = New ADODB.Recordset
    SQL = "SELECT * FROM T社員名簿 WHERE 社員番号 = 1002;"
    RS.Open SQL, CN, adOpenKeyset, adLockOptimistic
    RS.Update "部署コード", "B099"
    RS.Update "給与", "500000"
    RS.Close: CN.Close
    Set RS = Nothing: Set CN = Nothing

End Sub

Sub Test2()

    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Set CN = CurrentProject.Connection
    Set RS = New ADODB.Recordset
    RS.Open "T社員名簿", CN, adOpenKeyset, adLockOptimistic
    Do Until RS.EOF
        RS("年齢") = RS("年齢") + 1
        RS.Update
        RS.MoveNext
    Loop
    RS.Close: CN.Close
    Set RS = Nothing: Set CN = Nothing

End Sub
```

○レコードの追加
AddNewメソッドを使用します。

【AddNewメソッドの引数として、フィールドリスト、値リストを渡す】
RS.AddNew フィールドリスト, 値リスト

【AddNewメソッドを実行後、Fieldオブジェクトに値を代入し、Updateメソッドを実行する】

```
RS.AddNew
RS("フィールド１").Value = 値１
RS("フィールド２").Value = 値２
RS("フィールド３").Value = 値３
：
RS.Update
```

フィールドリスト、値リストを作成するには、リストに使用する変数をVariant型で宣言し、
Array関数でリストになる配列を変数に格納します

```
Dim List1 As Variant
Dim List2 As Variant
List1 = Array("F1","F2","F3")
List2 = Array("値１","値２","値３")
RS.AddNew List1, List2
```

AddNewメソッドに引数として、フィールドリスト、値リストを指定した場合、
AddNewメソッドが実行されると、直ちにレコードソースにレコードの追加が反映されます。

AddNewメソッドの実行後、
Fieldのオブジェクトに値を代入し最後にUpdateメソッドを実行した場合、
Updateメソッドが実行された時点で、レコードソースにレコードが追加されます。
AddNewメソッドの実行後は追加したレコードがカレントレコードになり
Updateメソッドを呼び出したあともそのままカレントレコードになります。

○レコードの削除
Deleteメソッドを使用します。
削除対象はレコードセットのカレントレコードなので、
レコードセット全体を削除する場合は、DoLoopなどで
繰り返し処理をしなければならない。

```
Sub Test5()

    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Dim SQL As String
    Set CN = CurrentProject.Connection
    Set RS = New ADODB.Recordset
    SQL = "SELECT * FROM T部署マスタ WHERE 部署コード = 'B088';"
    RS.Open SQL, CN, adOpenKeyset, adLockOptimistic
    RS.Delete
    RS.Close: CN.Close
```

```
    Set RS = Nothing: Set CN = Nothing

End Sub


Sub Test6()

    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Set CN = CurrentProject.Connection
    Set RS = New ADODB.Recordset
    RS.Open "T部署コピー", CN, adOpenKeyset, adLockOptimistic
    Do Until RS.EOF
        RS.Delete
        RS.MoveNext
    Loop
    RS.Close: CN.Close
    Set RS = Nothing: Set CN = Nothing

End Sub
```

○レコードの検索
Findメソッドを使用すると、レコードセットの中から特定のレコードを検索することができます。

RS.Find 検索条件, スキップ行, 検索方向, 開始位置

引数	定数	説明
検索条件		検索条件を指定。複数の条件式は指定できない
スキップ行		検索をスキップするレコード数を指定
検索方向		検索方向を定数で指定
	adSearchForward	最終レコードに向かって検索
	adSearchBackward	先頭レコードに向かって検索
開始位置		検索開始位置を指定

引数「検索条件」には条件式を指定します。
比較演算子やLike演算子が使用できますが、複数条件を指定することはできません、
条件を満たすレコードが見つかった時はそこにカレントレコードが移動します。
複数見つかった時は、最初に見つかったレコードがカレントレコードになります。
レコードが見つからなかった場合、
最終レコードにむかって検索した場合は、EOFプロパティが
先頭レコードにむかって検索した場合は、BOFプロパティが
それぞれTrueになります。

!!POINT

カーソルタイプが前方向スクロールのレコードセットでFindメソッドを使用するとエラーが発生します。
Findメソッドを実行するときは前方向スクロールカーソル以外のカーソルタイプを指定してください。!!!!!!!!

```
Sub Test()

    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Set CN = CurrentProject.Connection
    Set RS = New ADODB.Recordset
    RS.Open "T社員名簿", CN, adOpenStatic  '静的カーソル；すべての方向に移動可能。他のユーザーによる更新・追加・削除は確認できない。
    Do
        RS.Find "住所1 = '愛知県'"
        If Not RS.EOF Then              'レコードが見つかると、EOFプロパティはFalseを返却。最終行までたどり着くとTrueを返します。
            Debug.Print RS("社員名"), _   'レコードセットの中にフィールド名を入れると、暗黙でValueプロパティが返却。値が返る
                    RS("住所1")
            RS.MoveNext
        Else
            Exit Do
        End If
    Loop
    RS.Close: CN.Close
    Set RS = Nothing: Set CN = Nothing

End Sub
```

!!!POINT　：　Seekメソッドについて

Seekメソッドも使用可能
プロバイダがインデックスとSeekメソッドをサポートしている必要がある。

!!!POINT　：　Null値の検索について

FindメソッドでNull値を検索する場合

RS.Find "フィールド名 = Null"

と記述します。SQLのWhere句では
フィールド名 IS NULL

と記述しましたが、**Findメソッドでは=演算子を使用するので注意！！**
なお**SeekメソッドではNull値の検索は不可能**

○レコードの並べ替え

Sortプロパティを使用すると、レコードセットを昇順または降順に並べ替えることができます。
昇順：ASC
降順：DESC
ASCまたはDESCを省略するとASCになります
複数のフィールドで並べ替える場合は、カンマで区切って記述します。

RS.Sort = "フィールド１ ASC またはDESC, フィールド２ ASCまたはDESC"

!!POINT

並べ替えを解除するには！**Sortプロパティに「""(長さ0の文字列)」を設定**します。
また！！並べ替えを行うには、**CursorLocationプロパティに「adUseClient」が設定されている必要がある**

```
Sub Test()

    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Set CN = CurrentProject.Connection
    Set RS = New ADODB.Recordset
    RS.CursorLocation = adUseClient
    RS.Open "T社員名簿", CN
    RS.Sort = "部署コード ASC, 年齢 DESC"
    Do Until RS.EOF
        Debug.Print RS("社員名"), _
                    RS("部署コード"), _
                    RS("年齢")
        RS.MoveNext
    Loop
    RS.Close: CN.Close
    Set RS = Nothing: Set CN = Nothing

End Sub
```

◯レコードの抽出

Filterプロパティを使用　：　レコードセットの中から特定のレコードを抽出することができる
カレントレコードは抽出されたレコードの先頭に移動する

RS.Filter = "フィールド１ ＝ 条件１ And またはOr フィールド２ ＝ 条件２ …"

若干Sortプロパティの時と違う！
CursorLocationにadUseClientを指定しなくても大丈夫

!!!POINT

抽出を解除するにはFilterプロパティに「""(長さ0の文字列)」を設定します。
解除すると！！カレントレコードはレコードセットの先頭レコードに移動します。

```
Sub Test()
    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Set CN = CurrentProject.Connection
    Set RS = New ADODB.Recordset
    RS.Open "T社員名簿", CN
    RS.Filter = "社員番号 = 1002 Or 社員番号 = 1005"
    Do Until RS.EOF
        Debug.Print RS("社員番号"), _
                    RS("社員名")
        RS.MoveNext
    Loop
    RS.Close: CN.Close
    Set RS = Nothing: Set CN = Nothing
End Sub
```

◯レコードセットを利用する

Recordsetオブジェクトをフォームに表示したり、リストボックスやコンボボックスに表示することができます
この場合、**CursorLocationプロパティに「adUseClient」が設定されている必要があります　Sortプロパティを設定する時も必要であった！！**

```
Option Compare Database
Option Explicit

Private Sub btn1_Click()
    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Set CN = CurrentProject.Connection
    Set RS = New ADODB.Recordset
    RS.CursorLocation = adUseClient             'RecordsetのCursorLocationにadUserClientを設定している
    RS.Open "T社員名簿", CN
    Set Me.lst1.Recordset = RS
    RS.Close: CN.Close
    Set RS = Nothing: Set CN = Nothing
End Sub

Private Sub btn2_Click()
    Set Me.lst1.Recordset = Nothing
    Me.lst1.Requery                         'Requeryで画面を更新している
End Sub

Private Sub btn3_Click()
    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
```

```
    Set CN = CurrentProject.Connection
    CN.CursorLocation = adUseClient             'ConnectionのCursorLocationに設定している
    Set RS = CN.Execute("T社員名簿")
    Set Me.lst1.Recordset = RS
    RS.Close: CN.Close
    Set RS = Nothing: Set CN = Nothing
End Sub
```

◆トランザクション
トランザクションとは：データベースに対する一連の作業を１つの処理に纏めたもの

○BeginTrans / CommitTrans / RollbackTrans メソッド
BeginTrans　　：トランザクション開始
CommitTrans　　：トランザクションにおける変更を保存し、終了
RollbackTrans　：トランザクションにおける変更を破棄し、終了

```
Option Compare Database
Option Explicit

Sub Test()
    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Set CN = CurrentProject.Connection
    Set RS = New ADODB.Recordset
    RS.Open "T社員名簿", CN, adOpenKeyset, adLockOptimistic
    On Error GoTo ErrExit
    CN.BeginTrans
    Do Until RS.EOF
        If RS("社員番号") = 1003 Then
            RS("給与") = RS("給与") & "A"
        Else
            RS("給与") = RS("給与") * 1.1
        End If
        RS.Update
        RS.MoveNext
    Loop
    CN.CommitTrans
    MsgBox "トランザクションを確定しました"
    Exit Sub
ErrExit:
    CN.RollbackTrans
    MsgBox "トランザクションを取り消しました"
End Sub
```

◆外部データベースの利用
ADOを利用して、CSVファイルやExcelブックに接続し、レコードセットを取得する
○CSVファイルへの接続

```
Sub Test1()

    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Dim MyPath As String
    MyPath = CurrentProject.Path & "¥"
    Set CN = New ADODB.Connection
'    CN.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
'            "Data Source=" & MyPath & ";" & _
'            "Extended Properties='Text;HDR=NO'"
'この書き方でも可能＊どちらかを使用
    CN.Provider = "Microsoft.Jet.OLEDB.4.0"
    CN.Properties("Extended Properties") = "Text;HDR=NO"
    CN.ConnectionString = MyPath
    CN.Open
    'Set RS = CN.Execute("SELECT * FROM test.csv")  拡張子の区切り文字は # でも可能
    Set RS = CN.Execute("SELECT * FROM test#csv")
    Do Until RS.EOF
        Debug.Print RS.Fields(0), _
                    RS.Fields(1), _
                    RS.Fields(2), _
                    RS.Fields(3)
        RS.MoveNext
    Loop
    RS.Close: CN.Close
    Set RS = Nothing: Set CN = Nothing

End Sub
```

!!!POINT
接続文字列の「Extended Properties」で「HDR」の設定を「NO」にした場合
CSVファイルの１行目はデータとして認識されます。
「YES」にした場合、１行目はフィールド名として扱われます。

カンマ区切りなら、拡張子が「txt」のテキストファイルでの
同様に外部データベースとして利用することができる

○Excelブックへの接続

```
Sub Test2()
```

```
    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Dim MyPath As String
    MyPath = CurrentProject.Path & "¥"
    Set CN = New ADODB.Connection
    CN.Open "Provider=Microsoft.ACE.OLEDB.12.0;" & _
            "Data Source=" & MyPath & "test.xlsx;" & _
            "Extended Properties='Excel 12.0;HDR=YES'"
    Set RS = _
    CN.Execute("SELECT * FROM [Sheet1$] WHERE 購入品区分 = '本体'")
    Do Until RS.EOF
        Debug.Print RS("購入品コード"), _
                    RS("購入品名"), _
                    RS("単価"), _
                    RS("購入品区分")
        RS.MoveNext
    Loop
    RS.Close: CN.Close
    Set RS = Nothing: Set CN = Nothing

End Sub
```

!!POINT

レコードセットを取得するとき、WHERE句を使用して、レコードを抽出することができる
接続文字列の「Extended Properties」で「HDR」の設定を「YES」にしている場合

SELECT * FROM [シート名$] WHERE フィールド名 = 値

とフィールド名を使って抽出条件を指定します。CSVファイルなら

SELECT * FROM ファイル名.拡張子 WHERE フィールド名 = 値

と指定します。「HDR」の設定を「NO」にしている場合はフィールド名を使って抽出条件を指定することができませんので、

SELECT * FROM [シート名$] WHERE [シート名$].列番号 = 値

と指定します。「列番号」には、抽出条件に指定するフィールドを「F+番号」の形式で記述します

SELECT * FROM [ファイル名.拡張子] WHERE [ファイル名#拡張子].列番号 = 値

◆例外処理
ADOでは、データベースプロバイダで発生したエラーに関する情報はErrorオブジェクトに格納される
データアクセスに関するエラーが発生すると、Errorsコレクションに１つ以上のErrorオブジェクトが作成されます。
Errorオブジェクトの主なプロパティは次の通り

Number　　：　エラー番号を返す
Description　：　エラーに関する情報を返す
Source　　：　エラーを起こしたオブジェクトを返す

Errorコレクションは、ConnectionオブジェクトのErrorsプロパティで参照します。

```
Sub Test()

    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Dim MyErr As ADODB.Error
    Set CN = CurrentProject.Connection
    Set RS = New ADODB.Recordset
    RS.Open "T部署マスタ", CN, adOpenKeyset, adLockOptimistic
    On Error GoTo ErrExit
    RS.AddNew
    RS("部署コード") = "B001"
    RS("部署名") = "経理部"
    RS.Update '重複値登録でエラーを発生させる
    Exit Sub

ErrExit:
    For Each MyErr In CN.Errors
        MsgBox "発生したエラー情報は次の通りです" & vbCrLf & _
               MyErr.Number & vbCrLf & _
               MyErr.Source & vbCrLf & _
               MyErr.Description
    Next

End Sub
```

例外処理の中で例外を起こすと、キャッチされないことを確認

```
Sub Test()
    Dim CN As ADODB.Connection
    Dim RS As ADODB.Recordset
    Dim MyErr As ADODB.Error
    Set CN = CurrentProject.Connection
    Set RS = New ADODB.Recordset
    RS.Open "T部署マスタ", CN, adOpenKeyset, adLockOptimistic
    On Error GoTo ErrExit1
    RS.AddNew
    RS("部署コード") = "B001"
    RS("部署名") = "経理部"
```

```
    RS.Update
    Exit Sub
ErrExit1:
    On Error GoTo ErrExit2
    Dim i As Integer
    i = 40000000
    For Each MyErr In CN.Errors
        MsgBox "発生したエラー情報は次の通りです" & vbCrLf & _
               MyErr.Number & vbCrLf & _
               MyErr.Source & vbCrLf & _
               MyErr.Description
    Next
    Exit Sub
ErrExit2:
    For Each MyErr In CN.Errors
        MsgBox "発生したエラー情報は次の通りです" & vbCrLf & _
               MyErr.Number & vbCrLf & _
               MyErr.Source & vbCrLf & _
               MyErr.Description
    Next
End Sub
```

02.DAO(Data Access Object)とは

Accessのデータベースエンジンである
Jetデータベースエンジンに直接接続してデータを操作することのできるオブジェクトライブラリー

◆ADOとの違い

DAOはJetデータベースエンジンに直接接続するため、単体のAccessシステムを操作するケースに適している

ADOはAccessにかぎらず、MicrosoftSQLServerなどのデータベースに接続できる汎用性のあるオブジェクトを提供
データベースの種類が異なっても、同じ手法でレコードを操作できるという利点があります。

DAOにはテーブルやクエリを作成する機能があります
ADOにはテーブルやクエリを作成する機能はなく、ADOXというADOの拡張機能を使用する必要があります。
また、DAOはレコードセットの操作やトランザクションの管理など、ADOで行うことのできる処理の多くを実行することが可能

◆DAOを使用するには
※参照設定

「Microsoft Office 12.0 Access database engine Object Library」にチェック！！！

DAOとADOは同じ名前のオブジェクトが多く存在するので、
コンポーネント名を記述して明示的にしてあげる。

```
Dim RS As ADODB.Recordset
Dim RS As DAO.Recordset
```

コンポーネント名を省略すると、参照設定で優先度の高いほうが使用されるので注意！！

```
Dim RS As Recordset
```

◆データベースへの接続

Databaseオブジェクトを使用します。

```
Dim オブジェクト変数 As DAO.Database
```

カレントデータベースに接続する場合は、CurrentDbメソッドを使用します。
カレント以外のデータベースに接続する場合は、OpenDatabaseメソッドを使用します。

【カレントデータベースに接続する場合】

```
Set DB = CurrentDb
```

【カレント以外のデータベースに接続する場合】

```
Set DB = OpenDatabase("データベースファイルのパス")
```

◆SQLの実行

DatabaseオブジェクトのExecuteメソッドを使用することでSQLを実行することができます。
この時実行できるSQLはアクションクエリのみです。
レコードセットを返すSQLは実行することができません。

```
DB.Execute  クエリ
```

```
Sub Test()

    Dim DB As DAO.Database
    Set DB = CurrentDb
    DB.Execute "SELECT * " & _
               "INTO T社員コピー " & _
               "FROM T社員名簿"
    Set DB = Nothing
```

```
End Sub
```

◆テーブル・クエリの作成

CreateTableDefメソッド　：　新規テーブル作成
CreateQueryDefメソッド　：　新規クエリ作成

○CreateTableDefメソッド(**テーブル作成**)
テーブルを作成するにはCreateTableDefメソッドを使用して、
TableDefオブジェクトを作成します。

第二引数に指定するデータ型の主な定数

定数	データ型
dbBoolean	Yes / No型
dbLong	長整数型
dbDouble	倍精度浮動小数点型
dbDate	日付/時刻型
dbText	テキスト型

dbTextの場合は第三引数でフィールドサイズを指定します。

!!!POINT
新しいテーブルを作成するには１つ以上のフィールドを作成する必要があります。
作成したテーブルは、TableDefs.Deleteメソッドで削除することができます。

※注意※
作成するテーブルと同名のテーブルがすでにデータベースに存在する場合は
実行時エラーが発生するので注意してください

```
Sub Test1()

    Dim DB As DAO.Database
    Dim TD As DAO.TableDef
    Set DB = CurrentDb
    Set TD = DB.CreateTableDef("T在庫マスタ")
    TD.Fields.Append TD.CreateField("商品番号", dbText, 4)
    TD.Fields.Append TD.CreateField("在庫数", dbLong)
    TD.Fields.Append TD.CreateField("在庫区分", dbText, 2)
    DB.TableDefs.Append TD
    Set TD = Nothing: Set DB = Nothing

End Sub
```

○CreateQueryDefメソッド(**クエリを作成**)
CreateQueryDefメソッドを使用して、QueryDefオブジェクトを作成します。
クエリ名を指定してQueryDefオブジェクトを作成した場合、作成と同時にデータベースにクエリが追加されます。
クエリ名を指定せずにQueryDefオブジェクトを作成した場合、一時的なクエリとして作成されます。

クエリを作成する
```
Sub Test2()

    Dim DB As DAO.Database
    Dim QD As DAO.QueryDef
    Dim SQL As String
    Set DB = CurrentDb
    SQL = "SELECT * FROM T社員名簿 WHERE 年齢 >= 50;"
    Set QD = DB.CreateQueryDef("Q50以上社員", SQL)
    Set QD = Nothing: Set DB = Nothing

End Sub
```

一時的なクエリを作成して実行する
```
Sub Test3()

    Dim DB As DAO.Database
    Dim QD As DAO.QueryDef
    Dim SQL As String
    Set DB = CurrentDb
    SQL = "INSERT INTO T在庫マスタ " & _
          "VALUES('S001',5,'Z1');"
    Set QD = DB.CreateQueryDef("", SQL)
    QD.Execute
    Set QD = Nothing: Set DB = Nothing

End Sub
```

一時的なクエリの実行結果をレコードセットにいれることが可能
```
Sub Test4()

    Dim DB As DAO.Database
    Dim QD As DAO.QueryDef
```

```
    Dim RS As DAO.Recordset
    Dim SQL As String
    Set DB = CurrentDb
    SQL = "SELECT * FROM T社員名簿 WHERE 年齢 >= 50;"
    Set QD = DB.CreateQueryDef("", SQL)
    Set RS = QD.OpenRecordset
    Do Until RS.EOF
        Debug.Print RS("社員番号"), _
                RS("社員名"), _
                RS("年齢")
        RS.MoveNext
    Loop
    Set QD = Nothing: Set RS = Nothing: Set DB = Nothing
End Sub
```

!!!POINT
作成したクエリは、QueryDefs.Deleteメソッドで削除することができる
また一時的に作成したクエリはデータベースに追加されません。

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 08【Visual Basic Editorの操作とエラーへの対応】

VBEの機能
エラーの種類
エラーへの対応　について取り上げる

01.Visual Basic Editor(VBE)

VBAによる開発を行うために用意されたAccessとは別のアプリケーション！！

◆VBEの画面構成
①プロジェクトエクスプローラ
②プロパティウィンドウ
③コードウィンドウ
④イミディエイトウィンドウ

⑤ローカルウィンドウ
⑥ウォッチウィンドウ

◆イミディエイトウィンドウの利用
○Debug.Print

○演算・関数・ステートメントの実行

!!POINT
イミディエイトウィンドウは大文字小文字を区別しません。
また、コードウィンドウのようにキーワードを入力しても自動的に大文字小文字を変換しません。

○プロシージャを呼び出す
イミディエイトウィンドウからプロシージャを呼び出すことができる。
Subプロシージャを呼び出す場合は、Subプロシージャは結果を返すことができないので、
プロシージャ名をそのままイミディエイトウィンドウに記載します。
mysub1
mysub2 100
mysub2(100)　これでもOK!!

Functionプロシージャの場合
?myfunc1
?myfunc2(100)
×××××× ?myfunc2 100 ××××××　Functionの場合この書き方はNG!!

◆その他の機能の利用

○ウォッチウィンドウの使い方
変数の内容が特定の値になった場合、プロシージャの実行をストップすることができる

○ローカルウィンドウの使い方
プロシージャ内のすべての変数の変化について確認することができます。

○呼び出し履歴の使い方
Stopステートメント等で実行が中断した時に、プロシージャの呼び出し履歴を確認することができます。
上にある程最新で下に表示されているものが最初に呼び出されたプロシージャという形で表示される。

プロシージャ名.モジュール名の形式で表示

!!POINT
プロシージャの実行が一時停止して中断モードになった時だけ表示することができます。

○オブジェクトブラウザの使い方
VBAのオブエジェクトについて検索したり、
メソッドやプロパティを参照することのできるツール

!!POINT
検索文字列ボックスは大文字小文字を区別しないため、
「locktype」と入力しても同様に検索を行います。

02.エラーへの対応

プログラムを開発する上でエラーの発生は避けて通れない問題です。
エラーの種類は下記の通り

エラーの種類	内容
コンパイルエラー	構文や文法に誤りがあり、コードの実行ができないエラー
実行時エラー	コードの実行時に、処理が継続できなくなり発生するエラー
論理エラー	プログラムが、目的どおりの動作を行わないエラー

◆コンパイルエラー
・Ifステートメントに対するEnd If ステートメントの記述がない
・ステートメントの途中で改行

＊プログラムの記述中に発生するコンパイルエラー
＊プログラムの実行中に発生するコンパイルエラー

○プログラムの記述中に発生するコンパイルエラー
If i = 0　ここで改行をするとエラー

○プログラムの実行時に発生するコンパイルエラー
Ifステートメントに対するEnd Ifがないなど

!!POINT
プログラム実行時の構文チェックは
実際にプログラムを実行しなくても行うことができます。
[デバッグ]メニュー → [○○のコンパイル](○○はプロジェクト名)
エラーが見つかった時は先程と同様にエラー文言が表示される。

◆実行時エラー
VBAの文法は間違っていないが、
変数に格納するデータのデータ型が間違っていたり、実行不可能な処理を実行しようとして
処理の継続ができなくなった場合に発生します。
例)
Integer型 -32768 - 32768
の範囲を超える演算の結果オーバーフローが発生した時

◆論理エラー
プログラムが意図したとおりに動作しないエラーのこと

```
Sub Test()
    Dim 税込単価 As Long
    税込単価 = 5
    税込単価 = 税込単価 * 1.05
    Debug.Print 税込単価
End Sub
```

Double型だと小数点まで拾ってくれる

◆エラー処理
前もってプログラムに組み込まれたエラーが発生した時に行う処理のこと

○On Errorステートメント
On Errorステートメントはエラーが発生した時に実行する処理を指定するステートメントです。
前もってエラーに対処しておくことをエラートラップと呼びます。

On Errorステートメントの種類	説明
On Error Goto	エラーが発生した時行ラベルの場所に移動して以降の処理の実行する
On Error Resume Next	エラーが発生してもエラーを無視して処理を継続する
On Error Goto 0	エラートラップを無効にする

○On Error Goto ステートメント
行ラベルで指定した場所に移動して以降の処理を実施するステートメント
行ラベル以降に記述する、エラーが発生したとき実行される処理を、エラー処理ルーチンと呼びます。
エラー処理ルーチンの直前には、
エラーが発生しなかった時に実行されないように、Exit Subを記載しておきます。

```
Sub Test1()

    Dim MyNumber As Long
    On Error GoTo ErrExit
    MyNumber = InputBox("数値を入力してください")
    MsgBox MyNumber & "の値が入力されました"
    Exit Sub
ErrExit:
    MsgBox "数値が入力されませんでした"
End Sub


Sub Test2()

    Dim MyNumber As Long
    Dim MyDate As Date
    On Error GoTo ErrExit1
    MyNumber = InputBox("数値を入力してください")
    On Error GoTo ErrExit2
    MyDate = InputBox("日付を入力してください")
    MsgBox MyNumber & "の値と" & _
          MyDate & "の日付が入力されました"
    Exit Sub
ErrExit1:
    MsgBox "数値が入力されませんでした"
    Exit Sub
ErrExit2:
    MsgBox "日付が入力されませんでした"
    Exit Sub

End Sub
```

○On Error Resume Nextステートメント
エラーが発生してもエラーメッセージを表示することなくエラーを無視してそのまま処理を継続します。

※注意※
On Error Resume Nextステートメントはエラーを無視するので場合によってはプログラムが正常に動作しない可能性がある。
プログラムの動作に支障がない部分に限定してその部分の処理を終えたらOn Error Gotoステートメントや
On Error Goto 0ステートメントを使用して、エラートラップを解除することを勧められている

○On Error Goto 0ステートメント
エラートラップを無効にするステートメント

○Resumeステートメント
エラーが発生したケースで、処理を再実行するときに使用します。
Resumeステートメントとは、On Error Goto ステートメントによるエラートラップが有効なとき、エラー処理ルーチンでエラー処理を行った後、
指定した場所から処理を再実行するステートメント

Resumeステートメントの種類	説明
Resume	エラーが発生した行から処理を再実行する
Resume Next	エラーが発生した行の次の行から処理を再実行する
Resume 行ラベル	指定した行ラベルから処理を再実行する

```
Sub Test5()

    Dim MyNumber As Long
    On Error GoTo ErrExit
    MyNumber = InputBox("数値を入力してください")
    MsgBox MyNumber & "の値が入力されました"
ErrRetry:
    Exit Sub
ErrExit:
    Select Case MsgBox("もう一度入力しますか？", vbYesNoCancel)
    Case vbYes
        Resume
    Case vbNo
        Resume Next
    Case vbCancel
        Resume ErrRetry
    End Select
End Sub
```

○Errオブジェクト

```
Sub Test6()

    Dim MyNumber As Integer
    On Error Resume Next
    MyNumber = InputBox("数値を入力してください")
    Select Case Err.Number
    Case 0
        MsgBox MyNumber & "の値が入力されました"
    Case 6
        MsgBox Err.Description & vbCrLf & _
            "数値は-32768～32768の範囲で入力してください"
    Case 13
        MsgBox Err.Description & vbCrLf & _
            "数値以外のデータを入力しないでください"
    End Select

End Sub
```

◆よくあるエラーへの対処

○無限ループが発生するエラーの例
○レコードが１件もないときに発生するエラーの例
○インデックス番号が有効でない時に発生するエラーの例
○InputBox関数を使用した時に発生するエラーの例

> 03.XXXXX

> 04.XXXXX

> 05.XXXXX

> 06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 09【XXXXXXXXXXXXX】

01.XXXXX

02.XXXXX

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

DateAdd関数

構文
DateAdd(Interval,Num,Date)
引数Intervalには、追加する日付や時間の間隔を指定します。
引数Numには、Dateに対して増加させる日付や時間を指定します。
引数Dateには、Numを増加させる元になる日付や時間を指定します。

解説

DateAdd関数は、任意の日付や時間に特定の間隔を追加してその結果を返します。
引数Intervalに指定できる間隔は次のとおりです。

設定値	内容
yyyy	年
q	四半期
m	月
y	年間通算日
d	日
w	週日
ww	週
h	時
n	分
s	秒

DateAdd関数は無効な日付を返しません。たとえば次の例は"97/01/31"に1ヶ月を加えていますが、DateAdd関数が返すのは"97/02/28"であり"97/02/31"ではありません。

DateAdd("m", 1, "97/01/31")

サンプル
次の例は、現在の日付を起点にして「1日後」「1週間後」「1ヶ月後」を表示します。

```
Sub Sample()
    Dim tmp As String, CL As String
    CL = Chr(13) & Chr(10)
    tmp = "明日は" & DateAdd("d", 1, Date) & "です" & CL
    tmp = tmp & "1週間後は" & DateAdd("ww", 1, Date) & "です" & CL
    tmp = tmp & "1ヶ月後は" & DateAdd("m", 1, Date) & "です"
End Sub
```

DatePart関数

構文
DatePart(interval,date[,firstdayofweek[,firstweekofyear]])
引数intervalには、時間間隔の単位を指定します。
引数dateには、評価の対象になる日付を指定します。
引数firstdayofweekは省略可能です。週の始まりの曜日を指定します。
引数firstweekofyearは省略可能です。年度の第1週を指定します。

解説
DatePart関数は、引数dateで指定した日付を引数intervalで指定した間隔で評価します。
引数Intervalに指定できる間隔は次のとおりです。

設定値	内容
yyyy	年
q	四半期
m	月
y	年間通算日
d	日
w	週日
ww	週
h	時
n	分
s	秒

引数firstdayofweekに指定できる設定値は次のとおりです。

定数	値	内容
vbUseSystem	0	各国語対応APIの設定値を使います
vbSunday	1	日曜(既定値)
vbMonday	2	月曜
vbTuesday	3	火曜
vbWednesday	4	水曜
vbThursday	5	木曜
vbFriday	6	金曜
vbSaturday	7	土曜

引数firstweekofyearに設定できる設定値は次のとおりです。

定数	値	内容
vbUseSystem	0	各国語対応APIの設定値を使います
vbFirstJan1	1	1 月 1 日を含む週を年度の第 1 週として扱います(既定値)
vbFirstFourDays	2	7 日のうち少なくとも 4 日が新年度に含まれる週を年度の第 1 週として扱います
vbFirstFullWeek	3	全体が新年度に含まれる最初の週を年度の第 1 週として扱います

サンプル
次の例は、ユーザーが入力した任意の日付が「第何週」で「第何四半期」に該当するかを表示します。

```
Sub Sample()
    Dim myDate As String, msg As String, CL As String
    CL = Chr(13) & Chr(10)
    myDate = InputBox("日付を入力してください")
    If IsDate(myDate) Then
        msg = myDate & "は、" & CL
        msg = msg & DatePart("ww", myDate) & "週め" & CL
        msg = msg & "第" & DatePart("q", myDate) & "四半期に属します"
        MsgBox msg
    End If
End Sub
```

【多次元配列について検証】

```
Sub Test1()
    Dim i As Long
    Dim j As Long
    Dim k As Long
    Dim MyArray(1 To 5, 1 To 5, 1 To 5) As String
    For i = 1 To 5
        For k = 1 To 5
            For j = 1 To 5
                MyArray(i, k, j) = i & "-" & k & "-" & j & "番目の要素"
            Next j
        Next k
    Next i
    For i = 1 To 5
        For k = 1 To 5
            For j = 1 To 5
                Debug.Print i & "-" & k & "-" & j & "番目の要素"
            Next j
        Next k
    Next i
End Sub
```