EBook Gratis

# APRENDIZAJE visual-foxpro

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#visual-foxpro

# Tabla de contenido

# Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: visual-foxpro

It is an unofficial and free visual-foxpro ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official visual-foxpro.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Capítulo 1: Empezando con visual-foxpro

## Observaciones

Foxpro fue creado a principios de los 80 (originalmente como FoxBase - 1984?) Por el software de Fox y es compatible con las plataformas Mac OS, Unix, DOS y Windows. Era conocido como el motor de base de datos más rápido para PC en ese momento. Más tarde, en 1992, *desafortunadamente* , fue adquirido por Microsoft. Después de que Microsoft se hizo cargo, en 1994 se lanzaron Foxpro para DOS (FPD) y Foxpro para Windows (FPW) 2.6. A fines de 1995, Foxpro recibió el nombre de "Visual" y el soporte de la plataforma solo se limitó a Windows. También fue la primera versión de Foxpro donde el lenguaje resultó ser orientado a objetos.

El sitio oficial de Microsoft Visual Foxpro (comúnmente denominado solo VFP) lo describe como:

> El sistema de desarrollo de bases de datos Microsoft® Visual FoxPro® es una herramienta poderosa para crear rápidamente aplicaciones de bases de datos de escritorio, cliente enriquecido, cliente distribuido, cliente / servidor y web de alto rendimiento.

Aunque es un idioma antiguo, todavía se considera que es el lenguaje más fácil para crear una aplicación centrada en datos rápidamente para el escritorio de Windows. **Si** lo que necesita es crear ***una aplicación basada en datos para el escritorio de Windows y*** luego elegir VFP, lo haría de manera fácil y rápida.

## Versiones

| Versión | Publicado |
|---|---|
| FPW 2.6a | 1994-10-28 |
| Visual Foxpro 3.0 | 1995-12-16 |
| Visual Foxpro 5.0 | 1997-01-24 |
| Visual Foxpro 6.0 | 2000-08-18 |
| Visual Foxpro 7.0 | 2002-01-04 |
| Visual Foxpro 8.0 | 2003-10-25 |
| Visual Foxpro 9.0 | 2004-12-13 |
| Visual Foxpro 9.0 SP2 | 2007-10-21 |

## Examples

## Instalación o configuración

Instrucciones detalladas sobre cómo configurar o instalar Visual-FoxPro.

## Hola Mundo

Tradicionalmente, en todos los idiomas, el primer ejemplo es imprimir "Hello World". Probablemente hacerlo sea más fácil en VFP:

```
? "Hello World"
```

## Añadir Global Error Handler

Una forma sencilla de detectar errores no manejados (excepciones) en una aplicación VFP es usar el comando ON ERROR cerca del comienzo de su programa principal.

El siguiente comando ON ERROR llama a un método en el programa actual llamado "errorHandler". Los valores devueltos por ERROR (el Número de error de VFP), MENSAJE (el Mensaje de error de VFP), PROGRAMA (nombre del programa que se está ejecutando actualmente) y LINENO (el número de línea del error) se pasan al método errorHandler.

```
ON ERROR DO errorHandler WITH ERROR(), MESSAGE(), PROGRAM(), LINENO()
```

Un método simple errorHandler puede parecerse a lo siguiente.

```
PROCEDURE errorHandler
    LPARAMETERS tnVFPErrorNumber, tcVFPErrorMessage, tcProcWithError, tnLineNumber

    STORE 'Error message: ' + tcVFPErrorMessage + CHR(13) + ;
        'Error number: ' + TRANSFORM(tnVFPErrorNumber) + CHR(13) + ;
        'Procedure with error: ' + tcProcWithError + CHR(13) + ;
        'Line number of error: ' + TRANSFORM(tnLineNumber) TO lcDetails

    MESSAGEBOX(lcDetails, 16, "Unhandled Exception")

    ON ERROR *
    ON SHUTDOWN
    CLEAR EVENTS

    QUIT
ENDPROC
```

También puede cambiar y restaurar el controlador de errores en el medio. Por ejemplo, en un momento dado desea abrir todas las tablas en una carpeta exclusivamente, y si no puede, no desea continuar:

```
procedure DoSomethingWithExclusiveLock(tcFolder)
local lcOldError, llInUse, ix && by default these variables have a value of .F.
lcError = on('error') && save current handler
on error llInUse = .T.  && new handler
local array laTables[1]
```

```
for ix=1 to adir(laTables, addbs(m.tcFolder) + '*.dbf'))
   use (addbs(m.tcFolder)+laTables[m.ix,1]) in 0 exclusive
endfor
on error &lcError && restore old handler
if m.llInUse && couldn't get exclusive lock on all tables
   close databases all
   return
endif
* do whatever
endproc
```

Sugerencia: a veces, especialmente durante la depuración, desearía restaurar el controlador de errores predeterminado, que le permite interrumpir y entrar en el código donde ocurrió el error, luego en cualquier lugar donde obtuvo el error, agregando temporalmente:

```
on error
```

haría esto

Lea Empezando con visual-foxpro en línea: https://riptutorial.com/es/visual-foxpro/topic/7391/empezando-con-visual-foxpro

# Capítulo 2: Los operadores

## Observaciones

En VFP, los operadores se agrupan en aquellos:

- Operadores Numéricos
- Operadores logicos
- Operadores de personajes
- Operadores de fecha y hora
- Operadores relacionales

También hay operadores, implementados como funciones (como operaciones bitwise, comparación de objetos ...).

Vamos a ver en cada uno por ejemplo.

## Examples

### Operadores Numéricos

Los operadores numéricos son los más fáciles y casi los mismos que en otros idiomas.

- +, -, * y /. Operadores de suma, resta, multiplicación y división (en VFP no hay división entera, puede convertir un resultado a entero con las funciones INT (), CEILING () y FLOOR ()).
- Operador% módulo.
- ^ y **. Potencia del operador (s). Ambos hacen la misma cosa.
- (). Agrupación de operadores.
- Los operadores tienen prioridad. El orden es:

```
( )
^ (or **)
/ and *
- and +
```

```
? 10 / 5 + 2 && Outputs 4
? 2 + 10 / 5 && Outputs 4 as well. Division has precedence.

* Both multiplication and division have same precedence
* They would be interpreted from left to right.
? 4 * 5 / 2 + 5 && Outputs 15
* Use parentheses whenever you are in doubt or want to be explicit
? ( (4 * 5) / 2 ) + 5 && Outputs 15. Explicit grouping of operations

? 4 * 5^2 && ^ has precedence, this is same as 4 * (5^2) = 100.
? (4 + 5)^2 && Using parentheses we say add 5 to 4 (9) and then square = 81.
```

## Operadores logicos

Los operadores lógicos en VFP, en su orden de precedencia son:

| Operador | Descripción |
|---|---|
| () | Paréntesis, expresiones de grupos. |
| NO, | Lógicamente niega la expresión. NO o! no tiene diferencia |
| Y | Lógicamente y las expresiones. |
| O | Lógicamente O las expresiones |
| <>,! =, # | Compruebe si hay desigualdad. Así lo mismo que la lógica exclusiva O - XOR |

> Históricamente, NOT, AND, OR se escriben como .NOT., .AND., .OR. Aún puede usarlos si lo desea, pero AND, OR, NOT es más simple y limpio.
>
> Para falso y verdadero, tienes que usar .F. y T. literales respectivamente. No puedes elegir usar F y T en su lugar.

```
* Some logical variables
local llOld, llEmail  && any variable declaration implicitly initializes the variable as .F. - false
? m.llOld, m.llEmail && Prints .F. .F.

llOld   = .T.
llEmail = .F.

if ( m.llOld AND m.llEmail )
   ? 'Old AND should be emailed to'
endif
if ( m.llOld OR m.llEmail )
   ? 'Old OR should be emailed to'
endif
if ( m.llOld AND !m.llEmail ) && Same as (m.llOld AND NOT m.llEmail)
   ? 'Old BUT should NOT be emailed to'
endif

* Above code outputs
Old OR should be emailed to
Old BUT should NOT be emailed to
```

> En VFP, las expresiones lógicas se evalúan de forma abreviada. Es decir, si la primera parte de la verificación satisface todo el resultado, el resto de la expresión **ni** siquiera se interpreta. Una muestra sigue:

```
? 1 = '2' && An obvious error. It would complain operator/operand type mismatch.

* However we could use such an expression in an if and get no error
* because it is not interpreted at all
* (VFP is dynamic and there is no compile time check)
```

```
local llProcess
llProcess = .T.

if (m.llProcess OR (1='2'))
   ? 'Should do processing'
endif

* Would output

Should do processing

* without any error because m.llProcess true means
* the whole expression would be true, thus the expression after OR
* is not interpreted at all.
```

Un error que atrapa a los novatos es que, a veces, es posible que necesite varias comprobaciones, por ejemplo, en una consulta SQL, que están conectadas con los operadores AND, OR. Cuando hay muchos de ellos, uno puede ignorar el hecho de que los operadores tienen una precedencia (en orden (), NO, Y, O) y pensar que la interpretación se haría de izquierda a derecha en una cadena. Considere una muestra:

```
select * from myTable where !isCustomer AND debit > 5000 OR discount > 5
```

¿Cuál es la intención de esta consulta? Si lo hacemos explícito usando paréntesis de agrupación dice:

```
((NOT isCustomer) AND debit > 5000) OR discount > 5
```

simplificado, se parece a "primera expresión" O (descuento> 5). Cualquiera que fuera la intención, debido a esto O seleccionaría:

> todas las filas que tienen (descuento> 5), y también aquellas donde es un cliente con más de 5000 débitos.

Probablemente la intención era "darme aquellos donde NO es un cliente Y (el débito es más de 5000 O el descuento es más de 5)". Quedaría claro desde el principio si usáramos paréntesis:

```
select * from myTable where !isCustomer AND (debit > 5000 OR discount >
5)
```

Puede usar, pero no vale la pena, tener paréntesis para el operador NO inicial, cuando su operando es una sola expresión, es lo suficientemente legible con su prioridad -! IsCustomer se lee claramente como (NOT isCustomer).

## Operadores de personajes

Solo hay 4 operadores de caracteres, en orden de precedencia:

| Operador | Descripción |
|---|---|
| () | Paréntesis para la agrupación. Nota: la documentación de VFP, que tengo, se pierde esta. Sin esto, el operador es casi siempre inútil. |
| + | Concatena (une) cadenas de lado a lado. |
| - | Concatena cadenas **moviendo** los espacios finales desde la cadena izquierda hasta el final de la cadena derecha. |
| PS | Comprueba si la primera cadena está contenida en la segunda. |

+ es el más sencillo y también se utiliza para concatenar cadenas en muchos otros idiomas.

```
local firstName, lastName
firstName = "John"
lastName  = "Smith"

? m.firstName + " " + m.lastName
```

Salidas: John Smith

- Es un poco complicado, y no es muy conocido. Toma espacios finales de la cadena izquierda, agrega esos espacios a la cadena de la derecha. Supongamos que tiene una tabla con el nombre y el apellido y cada uno de ellos tiene 20 caracteres. Queremos concatenar nombres y apellidos para crear un nombre completo, **y** también queremos que el tamaño resultante sea fijo (en este caso, 20 + 20 + 1 espacio = 41). Hagamos que también tengas una columna de segundo nombre y queremos que el nombre completo se vea como "lastName, firstName middleName_____". Es más fácil hacer esto con un operador, pero debe tener en cuenta el truco de usar paréntesis aquí para agrupar, de modo que obtengamos exactamente lo que queremos:

```
* Create a cursor for our sample and fill in a few names
Create Cursor Names (firstName c(20), midName c(20), lastName c(20))

Insert Into Names (firstName, midName, lastName) Values ('Cetin','', 'Basoz')
Insert Into Names (firstName, midName, lastName) Values ('John', 'M', 'Smith')
Insert Into Names (firstName, midName, lastName) Values ('John', 'F', 'Kennedy')
Insert Into Names (firstName, midName, lastName) Values ('Tom', '', 'Hanks')

* Select with tricky - operator
Select *, ;
    lastName - (', '+firstName-(' '+midName)) As FullName ;
    from Names ;
    INTO Cursor crsNames ;
    nofilter

Browse
```

Y la salida es así:

| nombre de pila | medio nombre | apellido | nombre completo |
|---|---|---|---|
| Boletín | | Basoz | Basoz, Cetin |

| nombre de pila | medio nombre | apellido | nombre completo |
|---|---|---|---|
| Juan | METRO | Herrero | Smith, John M |
| Juan | F | Kennedy | Kennedy, John F |
| Tom | | Hanks | Hanks, Tom |

En la columna fullName, todos los espacios finales se empujan hasta el final muy bien. Si verifica la estructura, la columna fullName tiene 63 caracteres de ancho (3 * 20 + 3 caracteres que agregamos).

> Tenga en cuenta la importancia de agrupar paréntesis (intente eliminar paréntesis o organizarlos de otra forma).

Aunque el operador puede ser tentador de usar en tales casos, hay otra cara de la moneda. Este operador es específico de VFP y, por lo tanto, el SQL no es portátil. En su lugar, podría obtener el mismo resultado con este SQL compatible con ANSI:

```
Select *, ;
    CAST(RTRIM(lastName) +', '+ RTRIM(firstName) +' '+ midName as char(63)) As FullName ;
    from Names ;
    INTO Cursor crsNames ;
    nofilter
```

El último operador es $. Simplemente verifica si la cadena izquierda es parte de la cadena derecha.

```
local upcased, digits, hexDigits
upcased = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
digits  = '0123456789'
hexDigits  = m.digits + 'ABCDEF'

? 'A' $ m.upcased && .T.
? 'a' $ m.upcased && .F.
? '1' $ m.digits && .T.
? 'F' $ m.digits && .F.
? 'F' $ m.hexDigits && .T.
```

> **Importante:** en VFP, aunque puede escribir su código en cualquier caso (superior, inferior o mixto), las cadenas siempre distinguen entre mayúsculas y minúsculas. Por ejemplo: "Smith" y "Smith" son dos valores distintos. O en su tabla, si hay una columna de país, no encontrará "EE. UU." Si la busca con "EE. UU.". Lo mismo ocurre con el operador $, "GE" $ "Alemania" es falso.
>
> Nota personal: aunque le gustaría $ por su simplicidad y puede encontrarlo a menudo en los códigos fuente de Microsoft, en mi humilde opinión tiene muy poco valor. Pensando en muchos miles de líneas que he escrito en mi soporte, creo que encontraría muy pocas apariciones en mi propio código. Casi siempre hay una mejor alternativa (especialmente cuando el operando de la izquierda no es un solo carácter y / o la sensibilidad de mayúsculas y minúsculas).

## Operadores de fecha y hora

Básicamente hay dos operadores para los valores de fecha y hora. + y - están sobrecargados (probablemente un término de C) para hacer matematicas de fecha / fecha:

| Operador | Descripción |
|---|---|
| + | Agrega días (fecha) o segundos (fecha y hora) a un valor de fecha / fecha y hora. |
| - | Obtiene la diferencia de dos valores de fecha / fecha / hora. Resta días (fecha) o segundos (fecha y hora) de los valores de fecha y hora. |

+ es el más fácil. Tiene dos operandos, uno es una fecha o un valor de fecha y hora y el otro es numérico (aunque puede usar cualquier numérico, es un número entero para todos los propósitos prácticos).

Cuando uno de los operandos es una fecha, el operando numérico se toma como "día":

```
? Date() + 10 && Get the date 10 days later
* VFP is leap year aware when doing date math
? Date(2016, 2, 28) + 1 && Add 1 day to Feb 28, 2016. Returns Feb 29, 2016.
? Date(2017, 2, 28) + 1 && Add 1 day to Feb 28, 2017. Returns Mar 1, 2017.
```

Cuando uno de los operandos es datetime, el operando numérico se toma como "segundo":

> Hay 24 * 60 * 60 = 86400 segundos en un día

```
? Datetime() + 86400 && Add 1 day to current datetime.
```

Agregue 4 horas y 15 minutos al 1 de enero de 2016 a las 2:20 p.m.

```
? Datetime(2016, 1, 1, 14, 20, 0) + (4 * 3600 + 15 * 60)
```

Salidas viernes, 1 de enero de 2016, 6:35:00 PM.

> Con una impresión simple usando?, Lo que ve en su pantalla depende de la configuración de la fecha. Por ejemplo, si no ha cambiado nada, sus ajustes de fecha son de estilo americano (MDY), el formato de 12 horas (AM / PM) y el siglo se muestran solo con los últimos 2 dígitos.
>
> > Hay un símbolo especial ^ para la fecha y las fechas que obligan a una cadena a interpretarse 'estrictamente' como formato aaaa / MM / dd [HH: mm: ss | hh: mm: ss tt]. Por lo tanto, ^ también podría considerarse como operador de fecha / fecha. Por ejemplo, considere que algunos datos llegan de una fuente en un formato como 201610082230 (aaaaMMddHHmm). Para obtener ese valor como un Datetime válido:

```
Local cSample, tSample
cSample = '201610082230'
tSample = Ctot(Transform(m.cSample, '@R ^9999/99/99 99:99'))
? Transform(m.tSample, '@YL')
```

Salidas (según la configuración de fecha larga de su sistema):

Sábado, 8 de octubre de 2016, 10:30:00 PM

- Se usa para restar. Sus operandos son ambos valores de fecha / fecha y hora O uno es fecha / fecha y el otro es numérico.

Comencemos con los operandos de fecha / fecha y fecha más simples (como con el operador +):

Cuando uno de los operandos es una fecha, el operando numérico se toma como "día":

```
? Date() - 10 && What was the date 10 days ago?
? Date(2016, 3, 1) - 1 && Returns Feb 29, 2016.
? Date(2017, 3, 1) - 1 && Returns Feb 28, 2017.
```

Cuando uno de los operandos es datetime, el operando numérico se toma como "segundo":

```
? Datetime() - 86400 && Go back exactly one day
```

Obtenga 1 hora y 30 minutos antes de "ahora":

```
? Datetime() - (1 * 3600 + 30 * 60)
```

La segunda forma es obtener la diferencia entre dos valores de fecha / fecha / hora. Los operandos son tanto de fecha como de fecha y hora, no puede usar la fecha y la fecha y hora al mismo tiempo (escriba la conversión según sea necesario, VFP no lo hace por usted). Las reglas son como en + y -, los operandos son fecha, luego la diferencia está en **días** , los operandos son fecha y hora y la diferencia está en **segundos** .

¿Cuántos días para la víspera de año nuevo (para el año 2016)?

```
? Date(2016, 12, 31) - Date()
```

¿Cuántos segundos faltan para la medianoche?

```
? Dtot(Date()+1) - Datetime()
```

En la última muestra usamos una función de Fecha / Fecha, DTOT - DateToTime para obtener el valor de la medianoche de mañana. Hay muchas funciones útiles de fecha / fecha / hora disponibles, las omitimos todas ya que técnicamente no se consideran operadores (aunque operan en fecha / fecha / fecha). Lo mismo ocurre con otros operadores.

La fecha / fecha resta se **firma** . Es decir, si utiliza la fecha / fecha más pequeña como

primer operando, el resultado sería negativo. Puede usar la función abs () si necesita obtener un resultado positivo independientemente del orden de fecha / fecha y hora.

## Operadores relacionales

De todos los operadores, los operadores relacionales son los más complejos, por eso los dejamos hasta el final.

> Los operadores relacionales también se conocen como operadores de comparación, se usan para comparar cosas.
>
> > El resultado de la comparación es booleano falso o verdadero.
>
> Sin embargo, es interesante que, si lo verifica en VFP, le ayude a ver solo las operaciones de una lista corta y algunas líneas más como si se tratara de esos operadores.

Bueno, la complejidad proviene del hecho de que operan en ***cualquier tipo,*** ya sea numérico, fecha, fecha, lógica o cadena, e incluso en objetos. Además, el comportamiento puede parecer incómodo, no obtienes lo que esperas a *menos* que sepas qué efectos tienen los resultados.

Empecemos con una lista de operadores relacionales:

| Operador | Descripción | Muestra mas básica |
|---|---|---|
| > | Mas grande que | ? 1> 2 && .F. |
| < | Menos que | ? 1 <2 && .T. |
| > = | Mayor qué o igual a | ? 1> = 2 && .F. |
| <= | Menos que o igual a | ? 1 <= 2 && .T. |
| = | Igual a | ? 1 = 1 && .T. |
| == | Es exactamente igual a (tiene sentido para cadenas) | ? '1' = '1' && .T. |
| ! =, #, <> | No es igual a (los 3 operadores actúan de la misma manera, elige tu favorito) | ? 1! = 1 && .F. |

> Aunque puede utilizarlos con todos los tipos de datos, debe haber una compatibilidad de tipo entre los operandos. Por ejemplo, obtendrías un error si intentas comparar una Fecha con un Entero.
>
> La fecha y la fecha y hora se pueden comparar, aunque son tipos diferentes, VFP realiza la conversión implícitamente por usted.

```
? Date() > DateTime() && .F.
```

```
? Date() <= DateTime() && .T.
? Date() < DateTime() && .T. if it is not midnight
```

Cuando los operandos son numéricos, todos estos operadores son simples y directos, funcionan como lo harían en expresiones matemáticas.

Con los operandos lógicos, .F. se considera que es menor que .T.

Con los objetos lo que estamos comparando es la referencia del objeto en la memoria. Por lo tanto, la comparación más utilizada es determinar si dos variables de objeto apuntan al mismo objeto. es decir:

```
local o1, o2
o1 = createobject('Label')
o2 = createobject('Label')
? m.o1 = m.o2 && is o1 and o2 the same object?
? m.o1 > m.o2 && this would work too but likely you would never use

* remember we are comparing their references in memory
*
* They are different objects, but do they have any difference in their properties?
? CompObj(m.o1, m.o2) && .T. They are identical properties wise
```

La comparación de tipo de datos de caracteres, también conocida como comparación de cadenas, es la más confusa en VFP. No funciona como en otros idiomas y / o bases de datos y es exclusivo de VFP (y quizás de algún otro idioma de xBase).

> Hace muchos años, incluso he visto algunos miembros muy avanzados en la comunidad que aún no sabían cómo trabajan estos operadores en VFP. Así que es bastante comprensible que los pequeños matices confundan fácilmente a los novatos.

La comparación es básicamente sobre ser iguales o no. Si no son iguales, entonces podríamos pensar en los operadores>, <,> =, <=, ¿verdad? Con las cadenas es confuso cuando *dos cadenas se consideran iguales* .

**Importante: las** cadenas VFP distinguen entre mayúsculas y minúsculas. 'A' y 'a' son dos cadenas distintas. Este no es el caso con muchas bases de datos donde el valor predeterminado es utilizar una intercalación que no distinga mayúsculas y minúsculas. Por ejemplo, en postgreSQL o MS SQL Server en una tabla creada con intercalación de mayúsculas y minúsculas (CI):

```
select * from myTable where Country = 'TURKEY'

select * from myTable where Country = 'Turkey'
```

Daría el mismo resultado. En VFP, solo se obtienen aquellos en los que coincida la carcasa. Sin embargo, VFP tiene cierto soporte de intercalación y hace una comparación que no distingue entre mayúsculas y minúsculas. (No confíes, ver más abajo)

- Si dos cadenas no son iguales, hasta ahora tan buenas, **siempre que no haya cambiado ningún valor predeterminado,** entonces se comparan en *función de sus valores ASCII* .

```
? 'Basoz' < 'Cetin' && is true.
? 'basoz' < 'Cetin' && is false.
? 'Cetin' < 'David' && is true.
? 'Çetin' < 'David' && is false.
```

El valor predeterminado para la intercalación es 'máquina' y esto es lo que obtienes entonces. Cuando cambia la colación a otra cosa, obtiene la comparación basada en el orden de clasificación de esa colación. Con una configuración de intercalación distinta de la **máquina** predeterminada **,** también implica una insensibilidad de mayúsculas en la comparación (NO confíe en esto para la igualdad):

```
set collate to 'GENERAL'
? 'Basoz' < 'Cetin'
? 'basoz' < 'Cetin'
? 'Cetin' < 'David'
? 'Çetin' < 'David'
```

Ahora todas estas expresiones son VERDADERAS.

> Asesoramiento **personal** : las colaciones en VFP nunca han sido lo suficientemente confiables. Le sugiero que no use intercalaciones y se quede con la MÁQUINA predeterminada. Si usaría colaciones, tenga en cuenta que primero debe verificarlo cuando experimente algo inesperado con respecto a los datos de caracteres. He visto y demostrado que falla en muchos casos, pero luego dejé de intentar usarlo mucho antes de la versión VFP9, podría ser consistente ahora, realmente no lo sé.

Teniendo en cuenta que cubrimos los casos de desigualdad con cadenas, lo difícil es el caso de la igualdad. En VFP básicamente dos ajustes efectúan la comparación:

1. SET EXACT (el valor predeterminado es OFF y realiza comparaciones regulares, excepto SQL)
2. SET ANSI (el valor predeterminado es OFF y las comparaciones de efectos solo en SQL. SET EXACT **no tiene ningún efecto** en las comparaciones realizadas dentro de las consultas SQL).

Con SET EXACT OFF, lea la comparación como "¿la cadena en la derecha comienza con la cadena en la izquierda"? Se comparan hasta la longitud de la cadena correcta.

```
? "Bobby" = "B" && Bobby starts with B, so TRUE
? "Bobby" = "Bob" && Bobby starts with Bob, so TRUE
? "Bobby" = "Bob " && Bobby starts with Bob but there is a trailing space there, FALSE
? "Bobby" = "bob" && would be true with collation set to GENERAL
```

> Tenga en cuenta que con la comparación regular, "Bobby" = "B" es VERDADERO, pero "B" = "Bobby" es FALSO. En otras palabras, el lugar de los operandos es importante.

Con SET EXACT ON, las cadenas deben coincidir por completo, pero sus espacios finales se ignoran (ignoramos el conjunto de compilaciones aquí, lo que también haría insensibilidad a las mayúsculas):

```
? "BOBBY" = "BOB" && FALSE
? "BOBBY" = "BOBBY" && TRUE
? "BOBBY" = "BOBBY     " && TRUE
? "BOBBY     " = "BOBBY" && TRUE
```

Ahora, con los comandos SQL, SET EXACT no tiene ningún efecto y se comportaría como lo hace SET EXACT OFF.

```
Select * from Customers where Country = 'U'
```

Seleccionaría a los clientes de EE. UU., Reino Unido cualquier país que comience con 'U'.

Sin embargo, en SQL, por definición, cambiar el orden de los operandos debería producir el mismo resultado. Así:

```
Select * from Customers where 'U' = Country
```

También funcionaría de la misma manera (tenga en cuenta la diferencia con los comandos que no son de SQL).

Cuando quiere implicar coincidencias exactas, una opción es activar ANSI:

```
SET ANSI ON
Select * from Customers where Country = 'USA'
```

Devuelve todos esos clientes de USA. Tenga en cuenta que, los espacios al final en el campo del país O en la expresión correcta se ignoran. No importa cuántos seguimientos de ambos lados tengas. Se obtiene la comparación como si se hiciera como: RTRIM (País) = RTRIM ('EE. UU.').

> Aunque no se menciona en Operadores en VFP, un *operador de* SQL es LIKE. Cuando usa LIKE, obtiene una comparación de coincidencia exacta sin importar la configuración de ANSI SET (usando las fuerzas de LIKE y el caso ANSI implícito implícito - es un operador ANSI después de todo). Sin embargo, ten cuidado, hay una ligera diferencia en el comportamiento. No ignoraría los espacios finales, a **menos que** el tamaño total de los remolques sea igual o menor que el tamaño del campo. Por ejemplo, si el campo País es C (10), entonces País = 'USA' o País = 'USA__' funcionaría, pero País = 'USA_____' fallaría (los *guiones bajos* denotan un espacio y el último tiene más de 7 espacios finales).

Por fin estamos hasta el último operador, ==. Eso significa exactamente igual y hace que se use con cuerdas. Una ventaja es que usar == siempre significa que desea una coincidencia exacta **independientemente de los** ajustes **de** SET EXACT o SET ANSI. Sin embargo, tenga cuidado nuevamente, su comportamiento es diferente cuando se trata de un comando SQL o un comando regular que no es SQL.

Con SQL:

```
Select * from Customers where Country == 'USA'
```

*cualquiera que sea* la configuración ANSI y EXACTA, queremos que todos los clientes solo de EE. UU. Se ignoran los espacios finales en cualquier lado.

Con no-SQL:

```
? m.lcString1 == m.lcString2
```

sería cierto ***solo si*** son exactamente iguales, con respecto a su revestimiento y longitud (los espacios al final NO se ignoran). No se efectúa desde los ajustes SET ANSI, EXACT o COLLATE.

Lea Los operadores en línea: https://riptutorial.com/es/visual-foxpro/topic/7625/los-operadores

# Capítulo 3: VFP Interop con .NET

## Introducción

Este tema cubrirá la interoperabilidad entre VFP y .NET.

## Examples

### Usando wwDotNetBridge para ejecutar código .NET

Con la ayuda de wwDotNetBridge de West Wind , puede tener acceso fácilmente al código .NET dentro de un programa VFP.

El documento técnico tiene todos los detalles, pero este ejemplo conciso ayudará a ilustrar los pasos básicos para ejecutar un método en un ensamblado .NET.

Tenga en cuenta que wwDotNetBridge puede acceder directamente a propiedades simples como cadenas, ints, etc. Para acceder a estructuras más complicadas como listas, primero debe usar la función wArtNetBridge CreateArray para convertir la estructura .NET a una matriz COM de VFP (como se muestra en la parte inferior de este ejemplo).

```
*!* Load WestWind .NET wrapper library (wwdotnetbridge.prg assumed to be in the search path)
IF (!wwDotNetBridge())
    RETURN .F.
ENDIF

lowwDotNetBridge = CREATEOBJECT("wwDotNetBridge","V4")

*!* Load .NET Assembly (include full or relative path if necessary)
IF !lowwDotNetBridge.LoadAssembly("SomeDotNetAssembly.dll")
    lcAssemblyLoadError = "LoadAssembly error: " + lowwDotNetBridge.cErrorMsg
    =MESSAGEBOX(lcAssemblyLoadError, MB_ICONSTOP, "Error")
    RETURN .F.
ENDIF

*!* Parameters to pass to class constructor
*!* You can pass up to 5 paramenters to the constructor
lcParameter1 = "StringParameter1"
lcParameter2 = "StringParameter2"
lnParameter3 = 3
lcParameter4 = .NULL.

*!* Get an instance of the assembly class
loAssemblyReference = lowwDotNetBridge.CreateInstance("MyDotNetProject.MyDotNetClass", ;
    lcParameter1, lcParameter2, lnParameter3, lcParameter4)
IF lowwDotNetBridge.lError
    lcAssemblyLoadError = "An error occurred loading the class: " + lowwDotNetBridge.cErrorMsg
    RETURN .F.
ENDIF

*!* Usage Example
```

```
*!* This example runs a method that return a boolean
*!* and populates a List<string> (SomeStringList).
*!*
*!* The assembly has a public property named "LastErrorMessage"
*!* with details about any handled exceptions/problems.

IF (!loAssemblyReference.SomePublicMethod())
    msg = "There was a problem executing the method:" + CRLF + ;
        loAssemblyReference.LastErrorMessage
    =MESSAGEBOX(msg, MB_ICONSTOP, "Error")
    RETURN .F.
ENDIF

*!* At this point the string list (SomeStringList) should be populated
*!* wwDotNetBridge can convert that list to a VFP COM array (0-based)

laVFPArrayOfStrings = lowwDotNetBridge.CreateArray()
laVFPArrayOfStrings.FromEnumerable(loAssemblyReference.SomeStringList)

FOR x = 0 TO laVFPArrayOfStrings.Count-1
    ? laVFPArrayOfStrings.Item(x)
ENDFOR
```

Lea VFP Interop con .NET en línea: https://riptutorial.com/es/visual-foxpro/topic/9390/vfp-interop-con--net

# Creditos

| S. No | Capítulos | Contributors |
|---|---|---|
| 1 | Empezando con visual-foxpro | Cetin Basoz, Community, Steve |
| 2 | Los operadores | Cetin Basoz |
| 3 | VFP Interop con .NET | Steve |