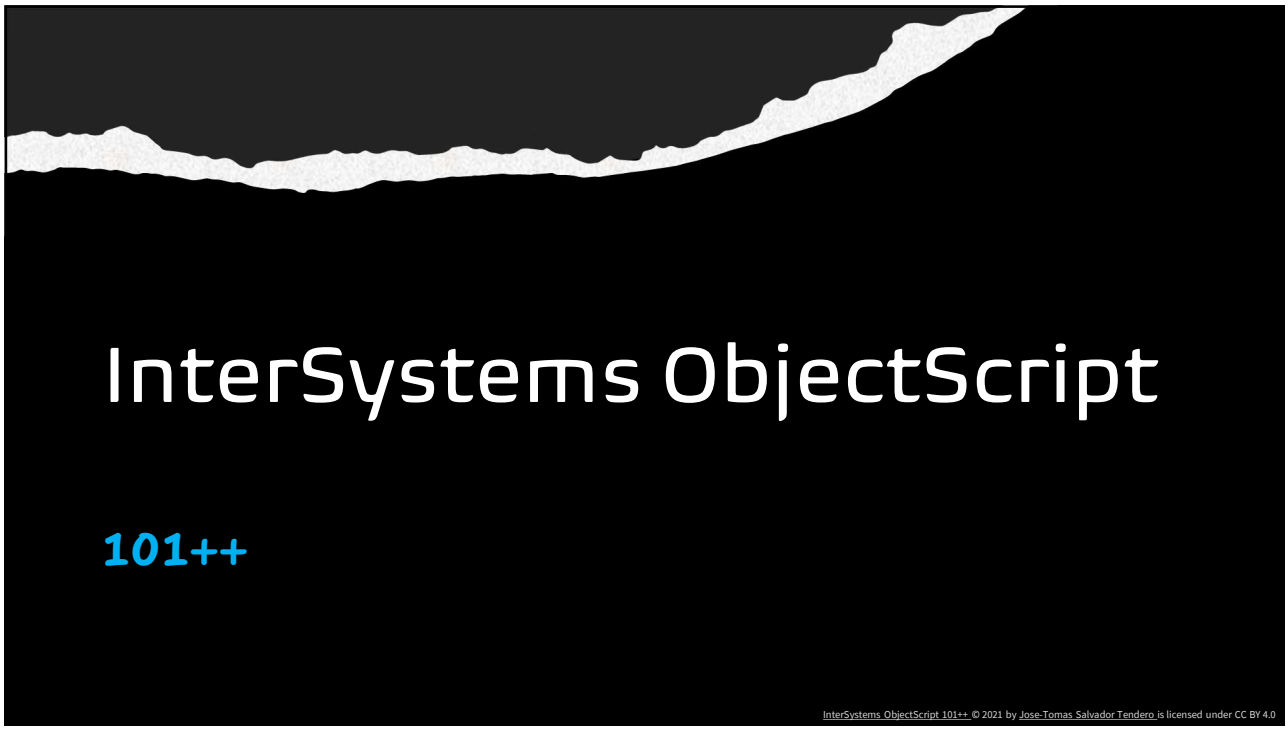
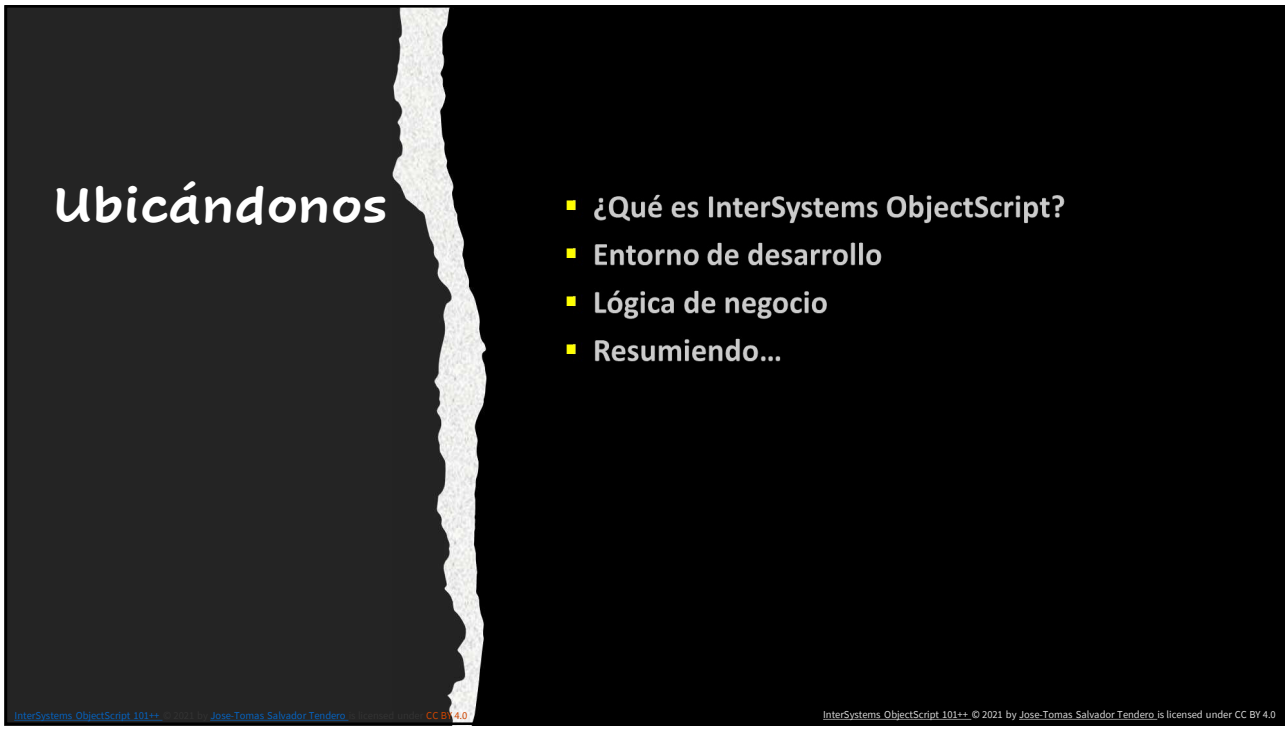
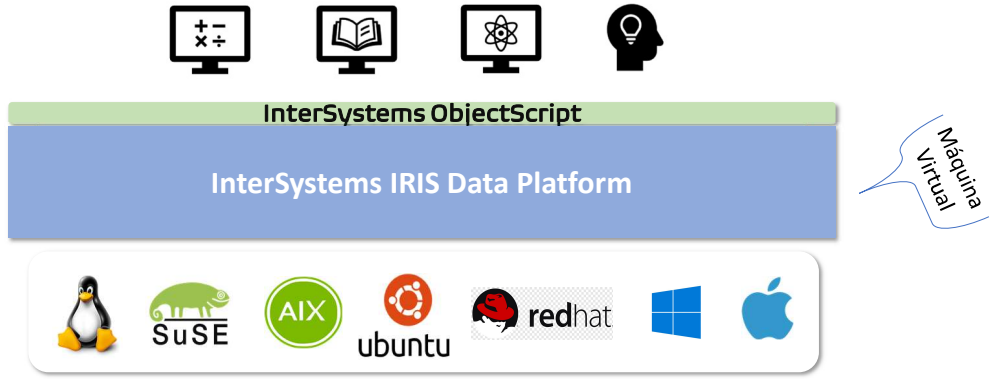


1



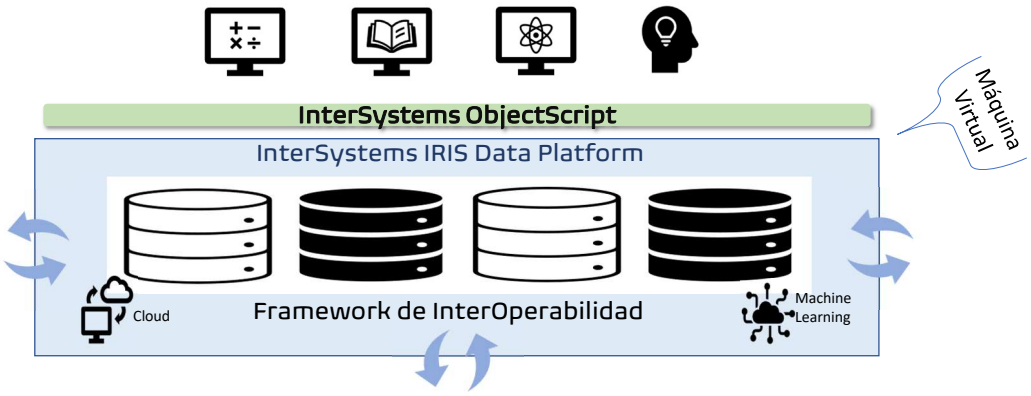
2

¿Qué es InterSystems ObjectScript?



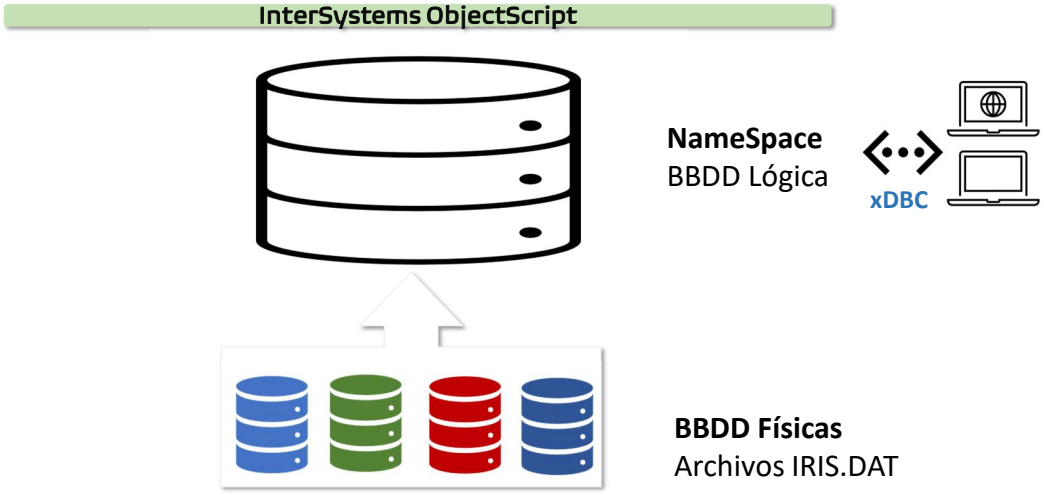
3

¿Qué es InterSystems ObjectScript?



4

Arquitectura: Namespace y Database



5

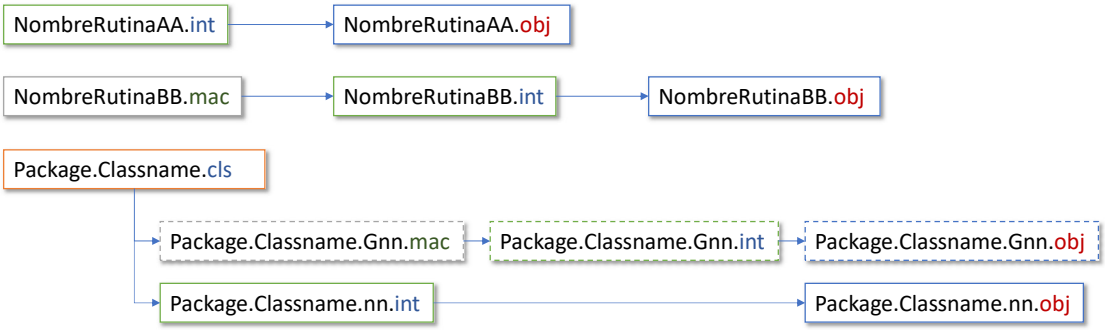
Entorno

- Terminal/Consola
- Portal de Gestión del Sistema
- IDE's:
  - InterSystems Studio
  - Visual Studio Code

6

Componentes de lógica de negocio

- Rutinas (*.MAC, *.INT, *.OBJ)
- Clases (*.CLS)
- Macros (*.INC)



7

Resumiendo...

- ObjectScript se ejecuta sobre InterSystems IRIS Data Platform:
  - IRIS: Máquina virtual + Plataforma de Datos + Plataforma de Interoperabilidad
- Siempre en el contexto de un *Namespace* o BBDD lógica
  - Acceso a varias BBDD físicas, transparente al desarrollador
  - Un programa puede saltar entre distintos *Namespaces*
- Accesible en modo consola/shell
  - Terminal en Windows
  - Comando **`iris session`** en Linux/Mac
- Clases, Rutinas y Macros son los elementos en los que podremos implementar lógica de negocio en ObjectScript

8

Aprender

Iterando

- Identificadores en ObjectScript
- Tipos de datos
- Tipos de variables
- Comandos básicos
- Operadores
- Bloques de control de flujo
- Ejemplos

InterSystems ObjectScript 101++ © 2021 by Jose-Tomas Salvador Tendero is licensed under CC BY 4.0

9

Identificadores en ObjectScript

- Un identificador o nombre es un alfanumérico que puede identificar a una:
  - Rutina [`do ^nombreRutina`]
  - Global [`set ^nombreglobal(1,"Nombre") = "Eva"`]
  - Variable [`set miContador = 1`]
  - Etiqueta de subrutina (`set x = suma^Math(2,2)`]
- Comienza con un % o un carácter alfabético
  - Seguido de caracteres alfanuméricos
  - Sólo los primeros 31 caracteres son significativos
  - Si se utiliza el %, sólo puede ser como primer caracter
- Distingue entre mayúsculas y minúsculas

Convención de nombres de identificadores:
https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.cls?KEY=GORIENT_appx_identifiers

10

Tipos de Datos

- String
- Número
  - Puede mezclarse cualquier tipo de número (enteros y de coma flotante)
  - Puede utilizarse notación exponencial: 5.35E3 = 5300
  - Los strings se evalúan a número si se utilizan operadores numéricos
- Boolean
  - 0 → false
  - <> 0 → true

11

Tipos de variables

- **Locales**
  - Existen sólo mientras dure el proceso en que se definen
  - Escalares (1 valor) o vectores (arrays)
    - **x = 1 y = "hola" h(1,2) = "Capitulo 2"**
  - %[z|Z]varname → pública al proceso
  - i%propName → variable instancia
- **Globales** [Ej.: `^nombreVarGlobal`]
  - Persisten más allá de la vida del proceso que las define o modifica
    - Mostrar con: `zw ^nombreVarGlobal` o do `^%G`
- **Globales privadas** (PPG – Process Private Global) [Ej.: `^||nombreVarGlbPrivado`]
  - Sólo existen dentro del proceso en que se definen. Sin limitación de tamaño
    - Mostrar con: do `^GETPPGINFO("*")`
- **Sistema** – precedidas por
  - **$** para variables generales de sistema, como $JOB, $HOROLOG,…
  - **^$** para variables estructuradas de sistema (SSVN), como ^$GLOBAL, ^$JOB,…

- https://docs.intersystems.com/irisforhealthlatest/csp/docbook/DocBook.UI.Page.cls?KEY=GCOS_variables
- SSVN: Structured System Variables - ObjectScript Reference - InterSystems IRIS for Health 2020.3 (https://docs.intersystems.com/irisforhealthlatest/csp/docbook/DocBook.UI.Page.cls?KEY=RCOS_SVARIABLES)

12

Comandos básicos

- **S**ET <var|(var,…)> = <valor>
- **K**ILL [[var,…]|(var,…)]
- **D**O [rutina|classmethod]
- **W**RITE [<var>|expresión]
- **R**EAD <var>
- **H**ANG <seconds>
- **H**ALT

Postcondicional (**:**)

```
set:condicion=1 variableX = "la condición es cierta"
Equivalente:
     if condicion=1 {set variableX = "la condición es cierta"}
write:verbose "Información en línea…"
     if verbose {write "Información en línea…"}
```


13

Operadores

- Aritméticos: **+ - * ** ** (potencia) **/ \ # ** (mod) [2**3=8 , 5\2 = 2, 5#2 = 1]
- Unarios: **'** (not) **+ -**
- Numéricos: **> < =**
- Lógicos: **&** (and) **!** (or) **&&** (and) **||** (or)
- De cadena: **_** (concatena) **?**(patrón) **]** (sigue) **]]** (se ordena tras) **[** (contiene)

IMPORTANTE

En ObjectScript, el resultado de una expresión puede ser:

- Un "valor Verdadero/True", que se expresa como un número: 0 (falso) o 1 (verdadero)
- Un número
- Un literal alfanumérico (string)

ObjectScript evalúa los operadores ESTRICTAMENTE DE IZQUIERDA A DERECHA, **sin precedencia de operadores** (con la excepción de los operadores unarios que sí tienen precedencia sobre el resto).

14

Operadores. Ejemplos...
Mucha atención aquí

- 2 + 2 * 2 = 8
- si x = 15
  x<11!x=1 → true(1) o false(0) 1 (true)
- "2 manzanas" + "2 peras" = 4
- +"019abc" = 19
- 2*"400años" = 800
- +"4/2" = 4
- +"texto" = 0
- -"-300litros" = 300 (positivo)
- -"19000,23291" = -19000

15

Operadores. Ejemplos...

- Concatenación de strings _ (guión bajo):
  ```
  set x = "ejemplo"
  "esto es un "_x = esto es un ejemplo
  x_" "_3 = ejemplo 3
  3_2 = 32
  ```
- Comilla simple y dobles comillas en strings:
  ```
  x_" es 'example' en inglés" = ejemplo es 'example' en inglés
  ```
  Si queremos incluir " en un string, añadimos otras " antes:
  ```
  "Dobles comillas ""_" " = Dobles comillas "
  "2 dobles comillas """"_" " = 2 dobles comillas ""
  ```
- Comparación (x = y) → Si x o y es un string, se comparan como texto
  ```
  x = 2, y = "2cents", z=2, j="abc", h="abc "
  x = y →0 (false) / x = +y → 1 (true) / x = z → 1 (true) / j = h → 0 (false)
  ```
- Mayor o igual, Menor o igual
  - '< → no menor que… ergo… mayor o igual que
  - '> → no mayor que… ergo… menor o igual que

16

Operadores. Pattern …

- <expresion>?<repetido desde X>.<hasta Y veces><patrón>

  <patron> = <repetido desde X>.<hasta Y veces>[A,C,E,L,N,P,U|"string"]

Alfabético, Carácter de control, CualquiEr carácter, Lowercase (minúsculas), Números, símbolo de Puntuación, Uppercase (mayúsculas)

Ej.

```
texto = "236-4567.pdf"
write texto?1.3N1"-"4N1".pdf" ➔ 1 (true)
```

1.3N1"-"4N1".pdf"
Se lee: de 1 a 3 números seguidos de 1 guión seguido de 4 números y la cadena .pdf
.3N1"-"4N1".pdf"
Se lee: ningún o hasta 3 números, seguidos de 1 guión seguido de 4 números y la cadena .pdf
3.N1"-"4N1".pdf"
Se lee: 3 o más números, seguidos de 1 guión seguido de 4 números y la cadena .pdf

https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.cls?KEY=GCOS_operators#GCOS_operators_pattern

Ejercicio. Patrones y precedencia

1. Pattern

Escribe una o más sentencias de pattern que acepten los siguientes formatos de números telefónicos de EEUU y rechacen cualquier otro:

621-0600
617-621-0600
(617)-621-0600
(617) 621-0600
(1) 617-621-0600
(1) (617)-621-0600
(1) (617) 621-0600

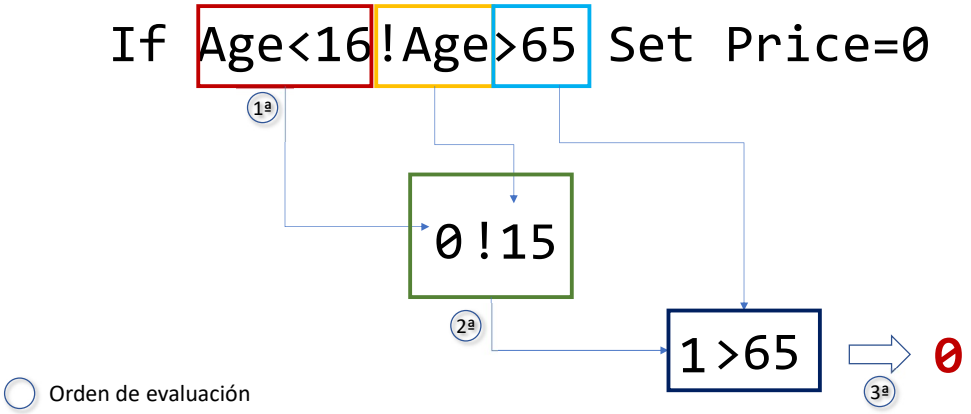
2. Expresiones

¿Qué es incorrecto con el siguiente comando?

```
If Age<16!Age>65 Set Price=0 ; gratis para niños y pensionistas
```

¿Cómo lo corregirías?

Ejercicio



Bloques de control de flujo...

If <condicion> {}
elseif <condicion> {}
else {}

for [<var>=<desde>:<incremento>[:<hasta>]] {}
for <var>=<valor1,valor2,…,valorN> {}

while <condicion = 1> {}

do {} while <condicion = 1> {}

Comandos de salida...

- **Q**UIT: Termina la ejecución del bucle o rutina/método en que se encuentra
- **RET**URN: Termina la ejecución de la rutina/método en que se encuentra
- **H**ALT: Termina la ejecución del proceso en que se encuentra

21

Ejercicio. Cálculo de primos

Rutina para calcular si un número es o no primo

Premisa:

Un número es primo si no es divisible por ningún número aparte de 1 o él mismo.

Idea de implementación:

- Buscamos el primer divisor que dé resto 0 (empezando por 2).
- Si Numero#divisor <> 0 , aumentamos el divisor en 1 y probamos otra vez… así hasta encontrar un divisor…
  - Si llegamos a un punto en que el divisor es ya mayor que el cociente podemos dejar de buscar…

22

24

Variables de Sistema

- **$H**OROLOG = <días desde 31/12/1840>,<segundos desde las 00:00 horas>
- **$ZTS**|$ZTIMESTAMP = $HOROLOG pero en UTC y con precisión de centésimas de segundo
- **$J**OB = número del proceso en ejecución
- $USERNAME = usuario actual
- $ROLES = lista de roles asociados al proceso, separados por coma
- **$ZV**ERSION = versión de InterSystems IRIS
- **$S**TORAGE = # de bytes en la tabla de símbolos del proceso en ejecución
- **$SY**STEM = nombre del sistema (host) e instancia de IRIS
- **$X**, **$Y** = posición columna/fila (X/Y) del cursor en modo terminal

25

Algunos comandos y funciones útiles (I)

- HANG:pc hangarg
- ZNSPACE:pc nspace
- ZWRITE:pc expression,...
- ZZDUMP:pc expression,...
- $EXTRACT(string,from,to) / set $EXTRACT(string,from,to)=value
- $FIND(string,substring,position)
- $LENGTH(expression,delimiter)
- $PIECE(string,delimiter,from,to)
  - SET $PIECE(string,delimiter,from,to)=value
- $REPLACE(string,searchstr,replacestr,start,count,case)
- $TRANSLATE(string,identifier,associator)
- $ZCONVERT/$ZCVT(string,mode,trantable,handle)
- $ZSTRIP(string,action,remchar,keepchar)
- $RANDOM(range)

26

Algunos comandos y funciones útiles (II)

- $ZDATE
  - $zdate(+$h,3) → 2021-06-03
- $ZDATETIME/$ZDT
  - $zdatetime($h,3) → 2021-06-03 20:08:45
- $ZTIME
  - $ztime($piece($h,",",2),1) → 20:11:33
- $ZDATEH/$ZDH
  - $zdh("2021-06-03",3) → 65898
- $ZDATETIMEH/$ZDTH
  - $zdth("2021-06-03 20:30:21",3) → 65898,73821
- $ZTIMEH/$ZTH
  - $zth("20:39:01",2) → 74341
- $NOW(tzmins)
  - $now() → 65898,73712.587234
  - $now(300) → 65898, 25202.678012 // hora en la costa este de EEUU
  - $now(-120) → 65898, 77325.251004 // hora en Israel
- %PosixTime class
  - $system.SQL.TOPOSIXTIME("2021-06-03 20:55:00",3) → 1154530963806846976
  - ##class(%PosixTime).CurrentTimeStamp() → 1154530963806847976

27

Algunos comandos y funciones útiles (III)

- $SELECT(expression:value,...,1:"default")
  - Devuelve el valor de la primera expresión que evalue a true

```
set X=5
set TEST=$SELECT(X<10:"A",X<20:"B",1:"C")
TEST = "A"
```

- $CASE(target,case:value,case:value,...,:default)
  - Compara con *target* y devuelve el valor asociado al primer *case* que coincida con ella o default

```
set diaDeSemana=$ZDATE($HOROLOG,10)
write $CASE(diaDeSemana,
            1:“Lunes",2:“Martes",3:“Miércoles",
            4:“Jueves",5:“Viernes",
            6:“Sábado",0:“Domingo",:“error")
```


28

Algunos comandos y funciones útiles (IV)

- $LB|$LISTBUILD(element,…)/$LISTBUILD(var1,var2,...)=list
  - Construye una lista de elementos a partir de los valores/expresiones indicadas

```
set listaA = $ListBuild("azul","rojo","verde","amarillo")
set listaB = $ListBuild("gris","negro")
set lista = listaA_listaB

set $ListBuild(colorAgua,colorPeligro,colorEsperanza,colorSol) = lista
```

- $LIST(list,position,end) / $LIST(list,position,end)=
  - Devuelve o reemplaza elementos de una lista

- $LISTLENGTH, $LISTFROMSTRING, $LISTTOSTRING, $LISTVALID, $LISTNEXT,…

https://docs.intersystems.com/irisforhealthlatest/csp/docbook/Doc.View.cls?KEY=RCOS_fchar#RCOS_fchar_list

29

Utilidades de Sistema

- %SYSTEM package y $SYTEM.* referencias
  - $system.OBJ.*
    - $system.OBJ.Help()
    - $system.OBJ.Load()
    - $system.OBJ.Compile()
    - $system.OBJ.DisplayError()
  - $system.SQL.*
    - $system.SQL.Help()
    - $system.SQL.Shell()
    - $system.SQL.Functions.*
- ^RESJOB (desde namespace %SYS)
  do ^RESJOB para ver y actuar sobre procesos abiertos

Ejercicio. Calcular la potencia

Construye un programa que calcule la potencia de un numero, escogido aleatoriamente de 1 a 10, elevado a una potencia de 0 a 6, también escogida aleatoriamente.

Pistas:

- Usa la función $random()
- Llama a las funciones (subrutinas que retornan un valor) con **$$labelSubrutina**(param,…)
- Si queréis utilizar procedimientos:

```
Routine …
Start
  //tu código
  quit
Proc1() public|private(default)
{
   //tu código
   //
}
Proc2()
{
   //tu código
}
```

Ejercicio. Generador de Nombres

Escribe un programa que pregunte al usuario el número de nombres a generar y el género. Se podrá escoger un máximo de 100 nombres (mínimo de 1) y, en cuanto al genero: H/h para hombre, M/m para mujer, X/x mezclados.

El programa responderá escribiendo en pantalla varias líneas con los nombres y apellidos generados.

Pistas:

- `$zconvert()` permite convertir a mayúsculas o minúsculas
- `$listnext()` puede venirte muy bien si tienes que recorrer alguna lista

32

Rizando el rizo con la indirección

- Operador @
- Comando XECUTE

34

Operador @

- Indirección a nivel de:
  - Nombre:
    ```
    set var = “X”
    set @var = “Valor de X”
    set glb = “^MiGlobal(123)”
    set @glb = “Valor del nodo 123”
    ```
  - Argumento
    ```
    set arg = “Z = 1”
    set @arg
    ```
  - Subscript
    ```
    set id = 88392
    if tipo = 1 {set glb=“^paciente(id)”} else {set glb=“^doctor(id)”}
    set nombre = @glb@(“nom”)
    ```
  - Pattern
    ```
    set pattern = ^valida(“PacienteID”)  // por ejemplo = “1u6n2u”
    if varID’?@pattern write !,”Invalid”
    ```


35

XECUTE

XECUTE:pc xecutearg,...

donde xecutearg puede ser:

"cmdline":pc

("(fparams) cmdline",params):pc

Ejemplo:

```
xecute "set diahora = $zdatetime($horolog,3)“_
       " write !,diahora"
xecute ("(dh,format) set diahora = $zdatetime(dh,format)",$horolog,3),
       " write !, diahora"
```


36

GLOBALS
Persistencia Multidimensional out-of-the-box

- ¿Qué es un global?
- Manejo de globals
- Ejemplo
- Transacciones

37

¿Qué es un Global?

- Variable Global (persistente)
  - Definida por el símbolo ^ como primer carácter del nombre de la variable
  - Automáticamente almacenada en BBDD en el momento de su creación
  - Disponible para todos los procesos con acceso a la BBDD en que se creó
- Estructura jerárquica
  - Compuesta por 1 o más nodos
  - Cada nodo se identifica por 1 o más subíndices (*subscripts*)
  - Un *subscript* puede ser cualquier cadena alfanumérica <> “”
  - No se predefine su tamaño/dimensión, aumenta o disminuye dinámicamente
  - Los nodos sin valor no ocupan espacio en BBDD
- Variable multi-dimensional
  - Sin ^ , misma estructura y capacidades, pero mantenida en memoria (no persistente)

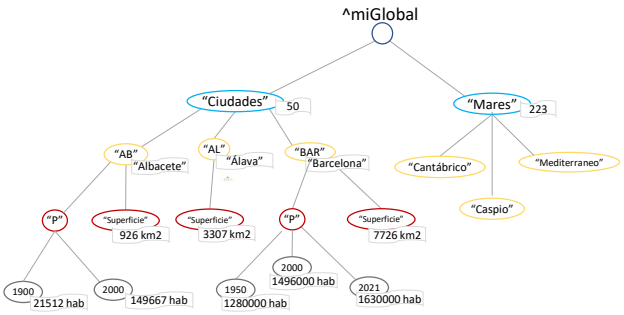
38

Manejo de Globals (I)

- **S**ET : crea y asigna/modifica un valor a un global o uno de sus nodos
```
set ^miGlobal=“2021-06-27 20:12:29”
set ^miGlobal(“Ciudades”)=2
set ^miGlobal(“Ciudades”,”AL”)=“Álava”
set ^ciudades(“Ciudades”,”AB”)=“Albacete”
set ^ciudades(“Ciudades”,”AB”,”Población”)=187000
```
- **K**ILL : elimina un global completo o un nodo y sus subnodos
```
kill ^miGlobal(“Ciudades”,”AB”)
kill ^miGlobal
```
- **ZK**ILL: elimina un nodo sin eliminar sus descendentes
```
zk ^miGlobal(“Ciudades”)
```
- **ZW**RITE : muestra el contenido del global completo o un nodo y sus subnodos.
```
zw ^miGlobal(“Ciudades”)
```


39

Manejo de Globals (I)



40

Manejo de Globals (II)

- **$G**ET(<global(nodo,…)>,<valor por defecto>)

```
Set valor = $get(^miGlobal("Ciudades","AL"),"n/d")
Write valor
"Alava"
```

- **$D**ATA(<global(nodo,…)>) : examina el nodo del global y nos indica si contiene datos y/o subnodos
  - 00: El nodo no existe
  - 01: Del nodo no cuelgan subnodos pero contiene datos (aunque sea "")
  - 10: Del nodo cuelgan subnodos, pero no contiene datos
  - 11: Del nodo cuelgan subnodos y, además, contiene datos

```
$data(^miGlobal) → 10
$data(^miGlobal("Ciudades")) → 11
$data(^miGlobal("Ciudades","AL","Superficie")) → 1
$data(^miGlobal("Mares","Egeo")) → 1
$data(^miGlobal("Planetas")) → 0
```


Manejo de Globals (III)

- **M**ERGE: copia el global o variable origen en el global o variable destino

```
USER> merge ^target = ^source
USER> merge a = b
```

- **$O**RDER(*^global(nodo,…,),dirección,valorDestino*)

  Devuelve el siguiente subscript en ese nivel, a derecha o izquierda (si dirección = -1)

```
USER> set idx = ""
USER> set idx = $order(^miGlobal("Ciudades",idx))
USER> write idx
"AB"
```

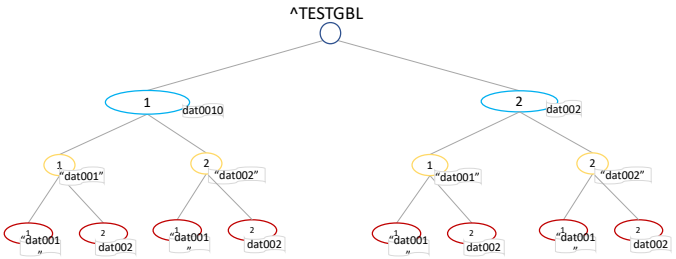
- **$Q**UERY(*^global(nodo,…),dirección,valorDestino*)

  Devuelve la referencia completa del siguiente nodo con datos

  - $NAME, $QLENGTH, $QSUBSCRIPT para crear referencias, contar niveles y obtener el valor de un subscript

```
USER> set glbRef = $query(^miGlobal("Ciudades","AL"))
USER> write glbRef
^miGlobal("Ciudades","AL","Superficie")
USER> write $name(@glbRef,2)," ** ", $qlength(glbRef)," ** ", $qsubscript(glbRef,3)
^miGlobal("Ciudades","AL") ** 3 ** Superficie
```

Ejemplo. Global de N niveles



Transacciones

- **TS**TART:pc
  - $TLEVEL nos indica el nivel de transacción en que estamos y se incrementa con cada TSTART empezando por 0
- **TC**OMMIT:pc
- **TRO**LLBACK:pc [1]
  - Decrementa $TLEVEL en una unidad (con argumento 1), o lo pone a 0 (sin argumentos)
  - Vuelve a los valores de inicio o, con argumento 1, del nivel de transacción anterior
  - No aplica a variables locales ni a Process Private Globals

Bloqueos

- LOCK:pc [+|-]lockname#locktype:timeout,...
  [+|-](lockname#locktype,...):timeout,..

  donde:
  + aplica lock y – lo quita
  locktype: "S" compartido / "E" escalating / "D" Deferred unlock / "I" Inmediate unlock
  timeout: lo que espera (en segundos) a obtener el lock o hacer el unlock
  - **$TEST** es 1 si se ha conseguido el bloqueo antes del timeout… 0 en caso contrario o no se estableció ningún timeout
  - Por defecto, los locks son exclusivos no *escalating*
  - El lockname puede ser cualquier cosa, cualquier nombre de variable válido:

    lock asvd99123#"S"
    lock +wwer
    lock +^miGlobal(1)

48

$Increment y $Sequence

- $INCREMENT(variable,num)
  - Incrementa o decrementa el valor de una variable en 1 o *num*
  - Ej:
    - `set x = $Increment(cont)`
    - `do $Increment(^datos("ESP")`
- $SEQUENCE(gvar)
  - Autoasigna rangos de incremento por proceso según necesidad
  - Incrementos de 1
  - Útil cuando múltiples procesos incrementan un mismo global
  - Ej:
    - Set x = $Sequence(^datos)
    - Set $Sequence(^datos)="" // Elimina el nodo raíz (sin tocar el resto)

  Ambas:
  - Si la variable no existe, la crea
  - Pueden utilizarse variables locales, globales o multidimensionales
  - Se considera una operación atómica. No se revierte en los rollbacks.
  - No le afectan los locks ni intenta hacer locks

49

Ejercicio. Registro de Clientes

Queremos crear una estructura que almacene información de clientes de forma persistente. Vamos a almacenar nuestros clientes en un global llamado ^clientes, con la siguiente estructura:

```
^clientes(sede,cliNum) = "flags^nombre^telefono^email^fechaUltimoPago^balance"
```

donde:

- *sede*: coódigo de la sede en la que se registró el cliente (6 alftanuméricos)
- *cliNum*: número de clienbte (6 numéricos)
- *flags*: tipo de cliente (numérico de 1 a 4) y estado (numérico de 1 a 4)
- *nombre*: nombre del cliente, con formato "apellidos, Nombre"
- *teléfono*: número de teléfono de contacto
- *email:* dirección email de contacto
- *fechaUltimoPago*: fecha, en formato interno, del ultimo pago realizado
- *balance*: balance de la cuenta del cliente decimal (positive o negative) con hasta 2 decimales de precision

Los códigos de tipo de cliente y estado están almacenados en otro global y su código numérico se corresponde con la posición de su descripción:

```
^codigos("tipo") = "Distribuidor,Mayorista,Minorista,Usuario"
^codigos("estado") = "Activo,Cerrado,Incobrable,Lista Negra"
```

Tarea: Escribe un pequeño programa para generar 100 registros aleatorios de clientes en tu base de datos y con funciones que nos permitan:

- Obtener los datos de clientes de una sede por su ID, pero también por su primer apellido.
- Obtener la descripción del tipo de cliente según el número asociado a cada tipo
- Obtener la descripción del estado del cliente según el número asociado a cada estado

Ejercicio. CRUD sobre global

Crearemos una API básica para añadir, leer, modificar o borrar información en un global.

Asumimos:

- Nombre global: ^empleados
- Nodos:
  - 1 sólo nivel, con valor de nodo numérico creciente empezando por 1
  - 1 nodo por registro
- Registro: Apellidos,Nombre,FechaNacimiento,Telefono,Email
- El acceso para creación o borrado es exclusivo
- El acceso para modificación o consulta es compartido

Resumen

Persistencia Multidimensional Out-of-the-box

- ¿Qué es un global?
- Manejo de globals
- Ejemplo
- Transacciones

52

Objetos en ObjectScript

- Definición de clases
- Macros
- Gestión de errores
- Ejemplos

53

Esto es una clase...

```
Include (ficheroMacros1, ficheroMacros2,…)
///Documentación de <class>MiPaquete.NombreDeClase</class>
///Ejemplo: <example> set obj = ..%New() </example>
Class MiPaquete.NombreDeClase extends ([Clase1, Clase2,…])[Cualificadores]
{
  ///Documentación de <parameter>nombreParametro1</parameter>
  Parameter nombreParametro1 [= Valor];
  ///Documentación de <property>nombrePropiedad1</propiedad>
  Property nombrePropiedad1 as TipoDeDatos [Cualificadores];
  ///Documentación de <method>metodoInstancia</method>
  Method metodoInstancia(par1,..,parN) [as TipoDeDatos][Cualificadores]
    { //código }
  ClassMethod metodoDeClase(par1,.., parNN) [as TipoDeDatos][Cualificadores]
    { //código }
  Storage
    {
    <Type>[%Storage.Persistent|%Storage.SQL|%Storage.Shard|CustomStorageClass]</Type>
    }
}
```


54

Ejemplo: Lo más básico, 1 clase y 1 método

- Creación de clase de tipo %RegisteredObject
- Propiedades básicas
- Método de Instancia
- Instanciación de objeto en memoria

55

Generalidades(I)

- Se distingue entre mayúsculas y minúsculas (Case Sensitivity)
  - Casi todo es sensible a mayúsculas y minúsculas
    - Variables, métodos, rutinas, paquetes, clases
    - Los commandos y funciones de ObjectScript no son *case-sensitive*
    - ##class no es *case-sensitive*.
  - Ante la duda, assume que todo es *case-sensitive*
- Nombres de Classes
  - <paquete>.<subpaquete1>.*.<subpaqueteN>.<nombreClase>
  - Los paquetes se crean, si no existen ya, al definir una clase que los incluya
  - Los paquetes %* están disponibles en todos los namespaces
    - EJEMPLO: El paquete %Library
- Importación de paquetes
  - Permiten referirnos a métodos de clase sin utilizar la referencia completa
  - Se puede declarar un `import` a nivel de clase o de método:
    - `Import (paquete1,…)`
    - `#import paquete` → a nivel de método – una directiva por cada paquete importado
    - AVISO: Atención a la ambiguedad en los nombres cortos de clases… si se repiten en distintos paquetes debemos usar el nombre completo.

56

Generalidades (II)

- **Herencia**
  - La herencia permite a una clase (subclase) utilizar propiedades y métodos de otra clase (superclase) como si fueran propios
  - ObjectScript soporta herencia múltiple
    - Precedencia de izquierda a derecha
    - Puede cambiarse la precedencia con el cualificador (class keyword): [Inheritance = right]
  - Una subclase puede definir sus propiedades o métodos particulares y sobrescribir los de su(s) superclase(s) (si no son “finales”)

57

Tipos de clase más comunes

- Registrada(%[Library.]RegisteredObject)
  - Funcionalidad básica para gestionar objetos en memoria)
- Persistente (%[Library.]Persistent)
  - Funcionalidad para almacenamiento de OO en Base de Datos
- Serie (%[Library.]SerialObject)
  - Funcionalidad para almacenamiento de objetos embebidos
- Tipo de datos (%[Library].DataType)
  - Superclase de los tipos de datos básicos/preconstruidos

Entrando en detalle...

```
Include (ficheroMacros1, ficheroMacros2,…)
///Documentación de <class>MiPaquete.NombreDeClase</class>
///Ejemplo: <example> set obj = ..%New() </example>
Class MiPaquete.NombreDeClase extends ([Clase1, Clase2,…])[Cualificadores]
{
  ///Documentación de <parameter>nombreParametro1</parameter>
  Parameter nombreParametro1 [= Valor];
  ///Documentación de <property>nombrePropiedad1</propiedad>
  Property nombrePropiedad1 as TipoDeDatos [Cualificadores];
  ///Documentación de <method>metodoInstancia</method>
  Method metodoInstancia(par1,..,parN) [as TipoDeDatos][Cualificadores]
    { //código }
  ClassMethod metodoDeClase(par1,.., parNN) [as TipoDeDatos][Cualificadores]
    { //código }
  Storage
    {
   <Type>[%Storage.Persistent|%Storage.SQL|%Storage.Shard|CustomStorageClass]</Type>
    }
}
```

Métodos y argumentos

```
(Class)Method nombreMetodo(par1,..,parN) [as TipoDeDatos][Cualificadores]
    { //código }
```

- **De instancia**

```
do obj.metodoInstancia()  // set resul = ..metodoInstancia()
do $method($this,"metodoInstancia")
```

- **De clase**

```
do ##class(Clase).metodoClase() // set resul = ##class(Clase).metodoClase()
do $classmethod("clase","metodoClase") // set resul = $classmethod("Clase","metodoClase")
```

  - Argumentos
    - Por valor (por defecto)
    - Por referencia
  - Definición de argumentos
    - [Output|ByRef] nombreArgumento [as TipoDato | …]

```
Method MiMetodo(ByRef arg1 as %String, arg2 as %Integer=0, Output arg3 as %Date, arg4…)
```

      - arg4… indica que recibiremos tantos argumentos como se quiera y que el método los almacenará en una variable multidimensional…
- **$this** y **..**
  - Para referirse al objeto o clase en curso
- **#dim** para declarar variables (informativo)

60

Métodos y funciones básicas

- %New()

```
Set objRef = ##class(Paquete.NombreClase).%New()
```

- %Save()

```
Set objID = ..%Save()
```

- %OpenId()

```
Set objRef = ##class(Paquete.ClasePersistente).%OpenId(IDdelObjeto)
```

- %DeleteId()

```
Do ##class(Paquete.ClasePersistente).%DeleteId(IDdelObjeto)
```

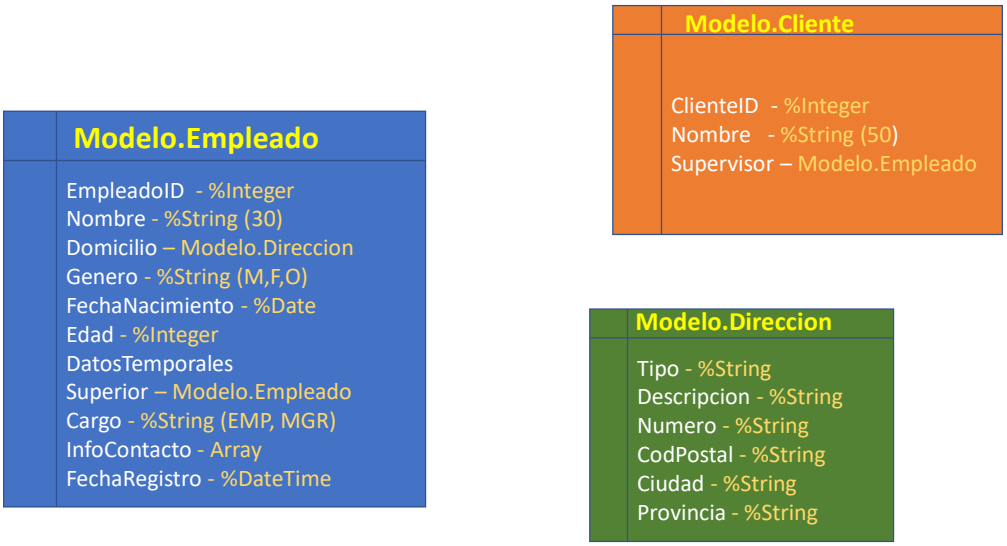
- %KillExtent()

```
Do ##class(Paquete.ClasePersistente).%KillExtent()
```

- $isobject(oref)
  - Devuelve 1 o 0 si oref es una referencia o no a un objeto
- $classname(oref)
  - Devuelve el nombre de la clase a la que pertenece oref

61

Ejemplo: Un pequeño Modelo de Clases



62

Indirección en Objetos OS

- $classmethod(clase, método,arg1, arg2, arg3,…)
  - Ejecuta el método de clase de la clase, pasándole argumentos si procede
- $method(oref, método,arg1, arg2, arg3,…)
  - Ejecuta el método de instancia de oref, pasándole argumentos si procede
- $property(oref, propiedad)
  - Devuelve el valor de la propiedad de oref
- $parameter(clase, parámetro)
  - Devuelve el valor del parámetro de la clase

63

Macros

- `#Define Macro[(Args)] [Value]`
  - Sustitución de texto (Value) antes de compilar
  - El nombre de cada variable/argumento de una macro debe comenzar por **%**
  - Los valores de los argumentos no pueden incluir comas
- `##Expression/##SafeExpression`
  - Evalúan una expression ObjectScript en tiempo de compilación
- Normalmente se agrupan en archivos include (*.inc)
  - Cabecera en VSCode: `ROUTINE <nombrefichero> [Type=INC]`
  - Aunque también puedes definir macros dentro de un método
- **Intellisense:** Incluir la descripción con **///** en la línea anterior al #define para que ofrezca la macro
- Uso:
  - **Include (**<fichero>,…**)** para clases
  - **#Include** para rutinas, métodos y otros ficheros de macros
  - **:#Include** para procedimientos almacenados

64

Algunas macros comunes

- **$$$OK**
  - devuelve el Código de Status que el Sistema interpreta como OK/Éxito/NoError
- **$$$ISERR(**sc**)**
  - "True" (1) si el estado *sc* representa un código de error. "False" (0) en caso contrario
- **$$$ISOK(**sc**)**
- **$$$GETERRORCODE(**sc**)**
  - Nos da el código de error del estado indicado por parámetro
- **$$$ERROR(**errorcode, arg1, arg2, ...**)**
  - Genera un código de estado de error
- **$$$THROWONERROR(**sc,expr**)** / **$$$TOE(**sc,expr**)**
  - Evalúa la expresión ObjectScript: expr, guarda el estado en: sc. Si es error, lanza una excepción con un **THROW**
  - No confundir con **$$$ThrowOnError(**sc**)** que lanza un throw si el estado sc indica error.

65

JSON Nativo en ObjectScript

- %DynamicObject
  - %FromJSON(str) | %ToJSON() | %GetIterator() | %Set(key,value,type) | %Get(key,default,type)
- %DynamicArray
  - %Pop() | %Push(value, type) | %Remove(pos)
- %Iterator.Object / %Iterator.Array
  - %GetNext(.key,.value)
- Ejemplo sintaxis:

```
Set docJSON = {"nom":"Daniel","edad":38,"activo":true,"fecha":($zdate(+$h,3))}
Set arrJSON=[{"signal":1,"value":10.34},{"signal":203,"value":0.112}]
```


Gestión de Errores

- Mecanismo tradicional
  - **$ZTRAP**

    ```
    Set $ZTRAP = "[^rutina | subrutina^rutina | label]"
    ```
  - **$ZERROR**
    - $ZERROR → <CLASS DOES NOT EXIST>PrintResult+3^MyTest *%SYSTEM.XXQL
- **Try { /* code */ } catch** [exceptionHandler] **{** [//code]**}**
  - %Exception.SystemException
    - Propiedades: **Name** del error, **Code** del error, **Location** donde se produjo y **Data** adicional
  - %Exception.AbstractException

https://docs.intersystems.com/irislatest/csp/docbook/Doc.View.cls?KEY=GCOS_errors

Otras funcionalidades out-of-the-box...

- %Collection
- %Stream
- %Net
  - HTTP, LDAP, POP3, SMTP, Oauth, MQTT, WebSocket,..
- %REST
  - %CSP.REST
  - %CSP.Session | %CSP.Request | %CSP.Response
- %UnitTest

68

¿Y cómo trabajo con SQL?

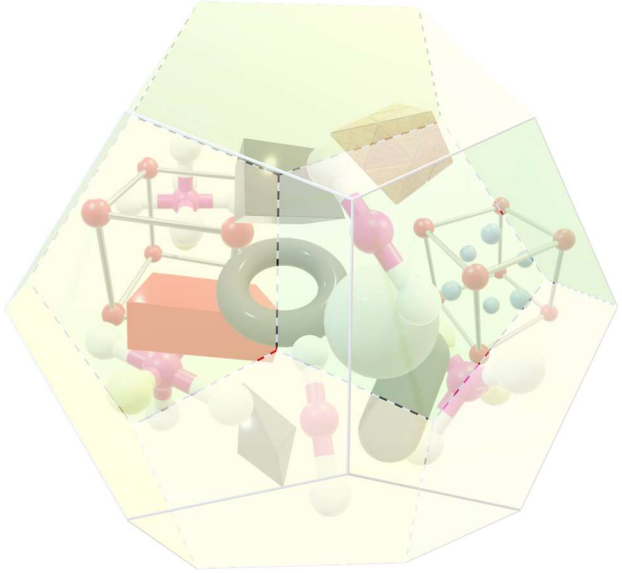
- InterSystems IRIS Multi-modelo
  - Relacional vs OO.OO. vs no-SQL vs Document
- Paradigma Relacional
- SQL Embebido y dinámico
  - Paquete %SQL.*
  - SQLCODE

69

Multi-Modelo... ¿Qué es eso?

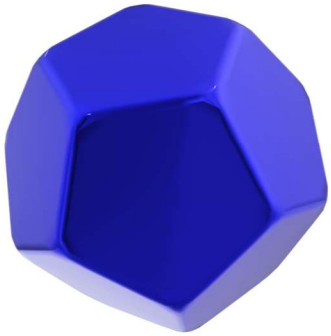
71

Multi-Modelo... ¿Qué es eso?



72

Multi-Modelo... ¿Qué es eso?



73

Multi-Modelo... ¿Qué es eso?

74

Ejemplo: Un pequeño ¿Modelo de Clases?

Modelo.Empleado

EmpleadoID - %Integer
Nombre - %String (30)
Domicilio – Modelo.Direccion
Genero - %String (M,F,O)
FechaNacimiento - %Date
Edad - %Integer
DatosTemporales
Superior – Modelo.Empleado
Cargo - %String (EMP, MGR)
InfoContacto - Array
FechaRegistro - %DateTime

Modelo.Cliente

ClienteID - %Integer
Nombre - %String (50)
Supervisor – Modelo.Empleado

Modelo.Direccion

Tipo - %String
Descripcion - %String
Numero - %String
CodPostal - %String
Ciudad - %String
Provincia - %String

75

Ejemplo: Un pequeño modelo E/R

Modelo.Empleado

EmpleadoID - %Integer
Nombre - %String (30)
Domicilio_Tipo – %String
Domicilio_Descripcion – %String
Domicilio_Numero – %String
Domicilio_CodPostal – %String
Domicilio_Ciudad – %String
Domicilio_Provincia – %String
Genero - %String (M,F,O)
FechaNacimiento - %Date
Edad - %Integer
DatosTemporales
Superior – Modelo.Empleado
Cargo - %String (EMP, MGR)
FechaRegistro - %DateTime

Modelo.Cliente

ClienteID - %Integer
Nombre - %String (50)
Supervisor – Modelo.Empleado

Modelo.Empleado_InfoContacto

ID - %String
Empleado- %String
InfoContacto - %String
element_key - %String

76

Paradigma Relacional (I)

```
CREATE TABLE Pruebas.Nombre_De_Tabla
(
   nom  VARCHAR(30),
   ape_1 VARCHAR(50),
   edad INT DEFAULT 0
)

CREATE METHOD nombre_apellidos(IN nom VARCHAR(20),IN ape VARCHAR(50))
  RETURNS VARCHAR(100)
  PROCEDURE
  FOR Pruebas.NombreDeTabla
LANGUAGE OBJECTSCRIPT
{
   set nomape = nom_" "_ape
   set %sqlcontext.SQLCODE = 0
   return nomape
}
```


77

Proyección ¿Relacional? (II)

```
Class Pruebas.NombreDeTabla extends %Persistent [SqlTableName = Nombre_De_Tabla]
{
   Property nom as %String (MAXLEN=30) [SqlColumnNumber = 2];
   Property ape1 as %String (MAXLEN=50) [SqlColumnNumber = 3, SqlFieldName = ape_1];
   Property edad as %Integer [InitialExpression = 0, SqlColumnNumber = 4];

ClassMethod nombreapellidos(
               nom as %String(MAXLEN=20),
               ape as %String(MAXLEN=50))
    as %String (MAXLEN = 100) [SqlName = nombre_apellidos, SqlProc]
    {
       set nomape = nom_" "_ape
       set %sqlcontext.SQLCODE = 0
       return nomape
    }
}
```


78

SQL Embebido

- **&sql() | ##sql()**
- **Variables host** – permiten intercambiar valores entre el SQL y el ObjectScript
  - :miVariable, :objRef.propA, :miVarMultidimensional(), :miVarMulti("nom"),…

```
ClassMethod testSQLBasico()
{
   &SQL(select count(*) into :filas from SQLUser.nombre_de_tabla)
   if SQLCODE = 0 { write !,"Filas encontradas: "_filas}
   else {write !,"Algo ha podido ir mal. SQLCODE = "_SQLCODE}

   &SQL(select * into :filas() from SQLUser.nombre_de_tabla)
   if SQLCODE = 0 { zwrite filas}
   else {write !,"Algo ha podido ir mal. SQLCODE = "_SQLCODE}

   &SQL(select nom into :filas("nombre") from SQLUser.nombre_de_tabla)
   if SQLCODE = 0 { write !,"Nombre: "_filas("nombre")}
   else {write !,"Algo ha podido ir mal. SQLCODE = "_SQLCODE}
   quit
}
```


79

SQL Embebido (II)

- SQLCODE: <0 Error / 0 Éxito / 100 Éxito pero no hay filas o se ha alcanzado el final de los datos
- %msg: mensaje de error. Sólo cambia si SQLCODE < 0
- %ROWID: último RowID afectado por operaciones insert, update, delete o fetch (en cursores)
- %ROWCOUNT: Número de filas afectadas por insert, update, delete, truncate, select,…

```
ClassMethod testSQLCursor()
{
  &sql(DECLARE miCursor CURSOR FOR
            SELECT nom, ape_1 into :nombre,:apellidos FROM SQLUser.nombre_de_tabla)
  &sql(OPEN miCursor)
  &sql(FETCH miCursor)
  while (SQLCODE = 0)
  {
    write:('SQLCODE) !,"[ "_%ROWCOUNT_"] Nombre y apellidos: "_nombre_" "_apellidos
    &sql(FETCH miCursor)
  }
  &sql(CLOSE miCursor)
  quit
}
```

80

SQL Dinámico: El paquete %SQL.*

- **%SQL.Statement**
  - %SelectMode: 0 – Logical / 1 – ODBC / 2 - Display
  - %New() + %Prepare() + %Execute() ➔ %SQL.StatementResult
  - %ExecDirect() ➔ %SQL.StatementResult
- **%SQL.StatementResult**
  - %Next()
  - %Get(<nombreColumna>) | %GetData(<posicionColumna>) | rset.<nombreColumna>
  - %SQLCODE
  - %Message
  - More methods…:
    - %Print(), %Display, %DisplayFormatted(), %GetRow(), %GetImplementationDetails(),…

SQL Dinámico: El paquete %SQL.* (II)

```
ClassMethod testSQLDinamico()
{
  set tQuery = "SELECT nom, ape_1 FROM SQLUser.nombre_de_tabla WHERE cod = ?”
  set tStmt = ##class(%SQL.Statement).%New()
  do tStmt.%Prepare(tQuery)
  set tRS = tStmt.%Execute(23)
  return:tRS.%SQLCODE'=0

  while tRS.%Next()
  {
    write:'tRS.%SQLCODE) !,"[ "_tRS.%ROWCOUNT_"] Nombre y apellidos: "_tRS.%Get("nom")_" "_tRS.%Get("ape_1")
  }
  return
}
```

Práctica

Terminamos ¿Y ahora qué?

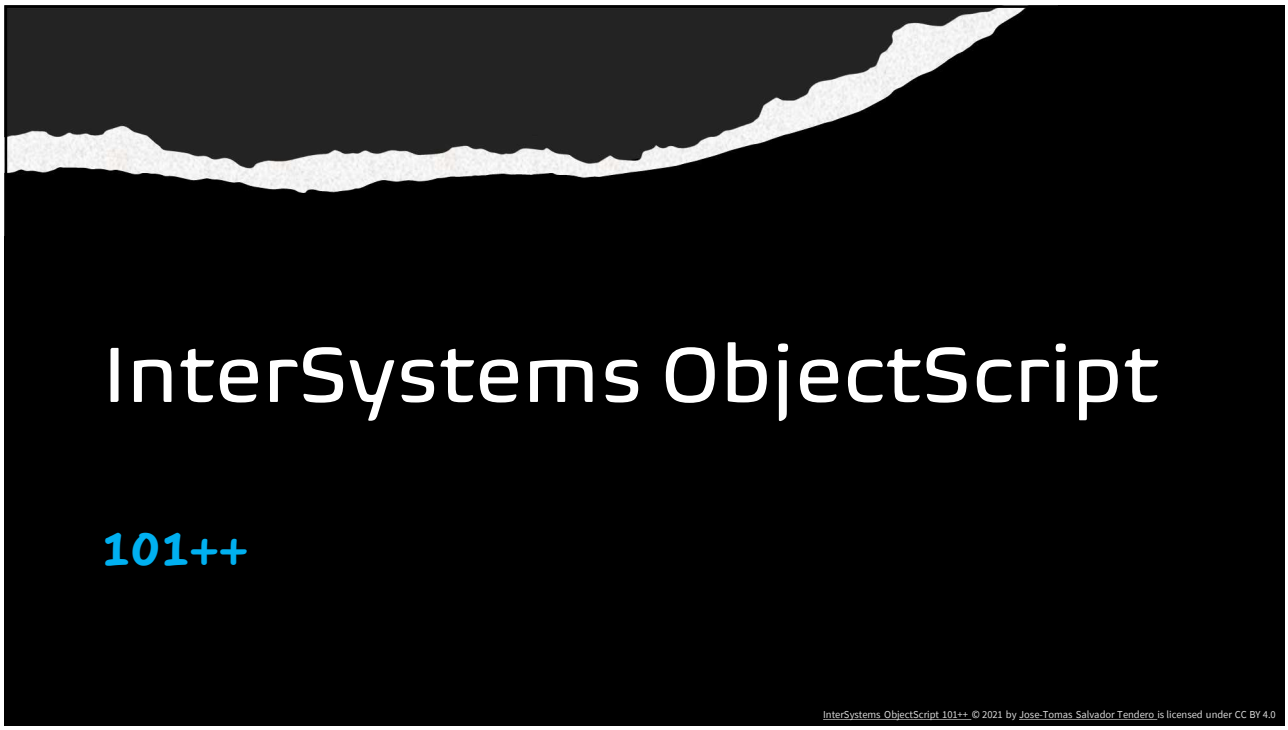
- Comandos OPEN, USE, CLOSE, …
- Funciones $ASCII, $CHAR,
- Colecciones
- Relationships (OO.OO) vs Foreign keys (SQL)
- Código generado
- %JSON.Adaptor y %JSON.Formatter
- Native APIs (Python, Java, .NET, C+,…)
- DocDB
- Definición de Triggers
- Definición de índices: tradicionales o bitmap
- Permisos y privilegios
- Procedimientos almacenados
- Campos calculados
- Gestión de BLOBs y CLOBs
- Gestión de outliers
- Importación de scripts SQL
- Optimización de Rendimiento
- …

83

Terminamos ¿Y ahora qué?

InterSystems ObjectScript 101++ © 2021 by Jose-Tomas Salvador Tendero is licensed under CC BY 4.0.
To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/

84



85