Artículo
Nancy Martínez · 13 mar, 2020 Lectura de 9 min

Cómo realizar un seguimiento de los cambios en los datos. El control de cambios (Parte 1 de 2)

Introducción

Un requisito frecuente en muchas aplicaciones es registrar en una base de datos los cambios que se realizan en los datos- qué datos se modificaron, quién los modificó y cuándo (control de cambios). Hay muchos artículos relacionados con el tema y existen diferentes métodos sobre cómo hacer esto en Caché.

Por ello, comparto un procedimiento que puede ayudar con la implementación de una estructura para seguir y registrar los cambios en los datos. Este procedimiento crea un trigger mediante un método "objectgenarator" cuando su clase persistente la hereda de "Audit Abstract Class" (Sample.AuditBase). Debido a que su clase persistente hereda Sample.AuditBase, cuando sea compilada, el activador generará automáticamente el control de las modificaciones.

Audit Class

Esta es la clase en la que se registrarán las modificaciones.

```
 Class Sample.Audit Extends %Persistent
{
  Property Date As %Date;

  Property UserName As %String(MAXLEN = "");

  Property ClassName As %String(MAXLEN = "");

  Property Id As %Integer;

  Property Field As %String(MAXLEN = "");

  Property OldValue As %String(MAXLEN = "");

  Property NewValue As %String(MAXLEN = "");
}
```

Audit Abstract Class

Esta es la clase abstracta de la que heredará su clase persistente. Esta clase contiene el método trigger (objectgenerator), que sabe cómo identificar cuáles fueron los campos que se modificaron, quién los modificó, cuáles son los antiguos y los nuevos valores, etc.; además escribirá las modificaciones en la tabla de auditoría (Sample.Audit).

```
 Class Sample.AuditBase [ Abstract ]
{

Trigger SaveAuditAfter [ CodeMode = objectgenerator, Event = INSERT/UPDATE, Foreach =
row/object, Order = 99999, Time = AFTER ]
{
  #dim %compiledclass As %Dictionary.CompiledClass
  #dim tProperty As %Dictionary.CompiledProperty
```

```
  #dim tAudit As Sample.Audit

  Do %code.WriteLine($Char(9)_"; get username and ip adress")
  Do %code.WriteLine($Char(9)_"Set tSC = $$$OK")
  Do %code.WriteLine($Char(9)_"Set tUsername = $USERNAME")

  Set tKey = ""
  Set tProperty = %compiledclass.Properties.GetNext(.tKey)
  Set tClassName = %compiledclass.Name

  Do %code.WriteLine($Char(9)_"Try {")
  Do %code.WriteLine($Char(9,9)_"; Check if the operation is an update - %oper = UPDATE")
  Do %code.WriteLine($Char(9,9)_"if %oper = ""UPDATE"" { ")

  While tKey '= "" {
    set tColumnNbr = $Get($$$EXTPROPsqlcolumnnumber($$$pEXT,%classname,tProperty.Name))
    Set tColumnName = $Get($$$EXTPROPsqlcolumnname($$$pEXT,%classname,tProperty.Name))

    If tColumnNbr '= "" {

      Do %code.WriteLine($Char(9,9,9)_";")
      Do %code.WriteLine($Char(9,9,9)_";")
      Do %code.WriteLine($Char(9,9,9)_"; Audit Field: "_tProperty.SqlFieldName)
      Do %code.WriteLine($Char(9,9,9)_"if {" _ tProperty.SqlFieldName _ "*C} {")
      Do %code.WriteLine($Char(9,9,9,9)_"Set tAudit = ##class(Sample.Audit).%New()")
      Do %code.WriteLine($Char(9,9,9,9)_"Set tAudit.ClassName = """_tClassName_"""")
      Do %code.WriteLine($Char(9,9,9,9)_"Set tAudit.Id = {id}")
      Do %code.WriteLine($Char(9,9,9,9)_"Set tAudit.UserName = tUsername")
      Do %code.WriteLine($Char(9,9,9,9)_"Set tAudit.Field = """_tColumnName_"""")
      Do %code.WriteLine($Char(9,9,9,9)_"Set tAudit.Date = +$Horolog")

      Do %code.WriteLine($Char(9,9,9,9)_"Set tAudit.OldValue = {"_tProperty.SqlFieldName_"*O}")
      Do %code.WriteLine($Char(9,9,9,9)_"Set tAudit.NewValue = {"_tProperty.SqlFieldName_"*N}")

      Do %code.WriteLine($Char(9,9,9,9)_"Set tSC = tAudit.%Save()")
      do %code.WriteLine($Char(9,9,9,9)_"If $$$ISERR(tSC) $$$ThrowStatus(tSC)")

      Do %code.WriteLine($Char(9,9,9)_"}")
    }
    Set tProperty = %compiledclass.Properties.GetNext(.tKey)
  }

  Do %code.WriteLine($Char(9,9)_"}")

  Do %code.WriteLine($Char(9)_"} Catch (tException) {")

  Do %code.WriteLine($Char(9,9)_"Set %msg = tException.AsStatus()")
  Do %code.WriteLine($Char(9,9)_"Set %ok = 0")
  Do %code.WriteLine($Char(9)_"}")

  Set %ok = 1
}

}
```

Data Class (Persistent Class)

Esta es la clase de datos del usuario, en la que el usuario (la aplicación) realiza modificaciones, genera y elimina

historiales, o hace cualquier cosa que se le permita hacer :)

En resumen, esta generalmente es su clase %Persistent.

Para iniciar el seguimiento y registro de los cambios, necesitará que la clase persistente se herede de la clase abstracta (Sample.AuditBase).

```
Class Sample.Person Extends (%Persistent, %Populate, Sample.AuditBase)
{
  Property Name As %String [ Required ];

  Property Age As %String [ Required ];

  Index NameIDX On Name [ Data = Name ];
}
```

Prueba

Como se ha heredado la clase de datos (Sample.Person) de la clase de Audit Abstract Class (Sample.AuditBase) se podrá insertar datos, realizar modificaciones y analizar los cambios que se registraron en Audit Class (Sample. Audit).

Para comprobar esto, es necesario crear un método Test() para la clase, en la clase Sample.Person o en cualquier otra clase se elija.

```
ClassMethod Test(pKillExtent = 0)
{
   If pKillExtent '= 0 {
  Do ##class(Sample.Person).%KillExtent()
   Do ##class(Sample.Audit).%KillExtent()
   }

   &SQL(INSERT INTO Sample.Person (Name, Age) VALUES ('TESTE', '01'))
   Write "INSERT INTO Sample.Person (Name, Age) VALUES ('TESTE', '01')",!
   Write "SQLCODE: ",SQLCODE,!!!

   Set tRS = $SYSTEM.SQL.Execute("SELECT * FROM Sample.Person")
   Do tRS.%Display()

   &SQL(UPDATE Sample.Person SET Name = 'TESTE 2' WHERE Name = 'TESTE')
   Write !!!
   Write "UPDATE Sample.Person SET Name = 'TESTE 2' WHERE Name = 'TESTE'",!
   Write "SQLCODE:",SQLCODE,!!!

   Set tRS = $SYSTEM.SQL.Execute("SELECT * FROM Sample.Person")
   Do tRS.%Display()

   Quit
}
```

Ejecutar el método Test():

```
 d ##class(Sample.Person).Test(1)
```

Parameter 1 will kill extent from Sample.Person and Sample.Audit classes.

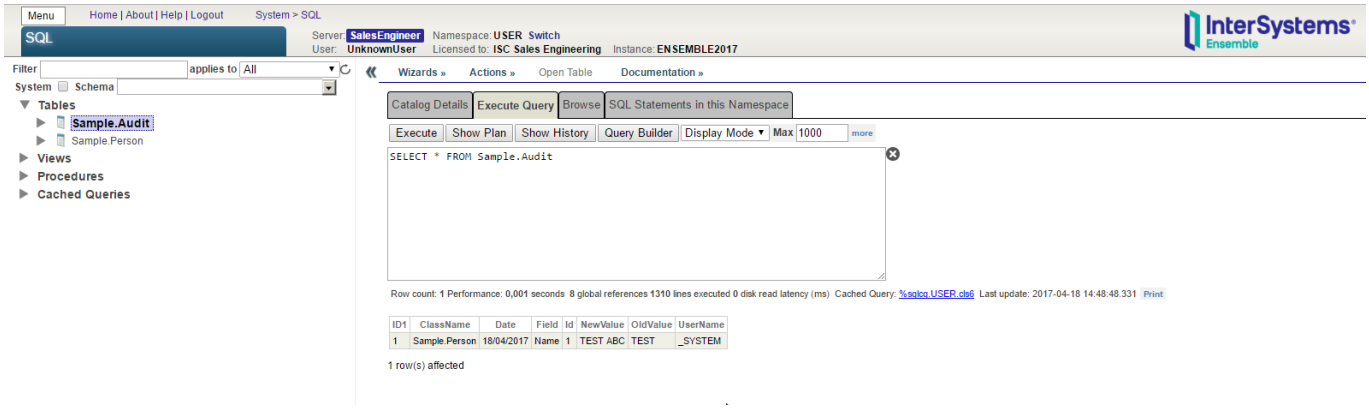
```
d ##class(Sample.Person).Test(1)
INSERT INTO Sample.Person (Name, Age) VALUES ('TEST', '01')
SQLCODE: 0

ID  Age Name
1   01  TEST

1 Rows(s) Affected


UPDATE Sample.Person SET Name = 'TEST ABC' WHERE Name = 'TEST'
SQLCODE:0


ID  Age Name
1   01  TEST ABC

1 Rows(s) Affected
```

El método Test para clase realiza lo siguiente:

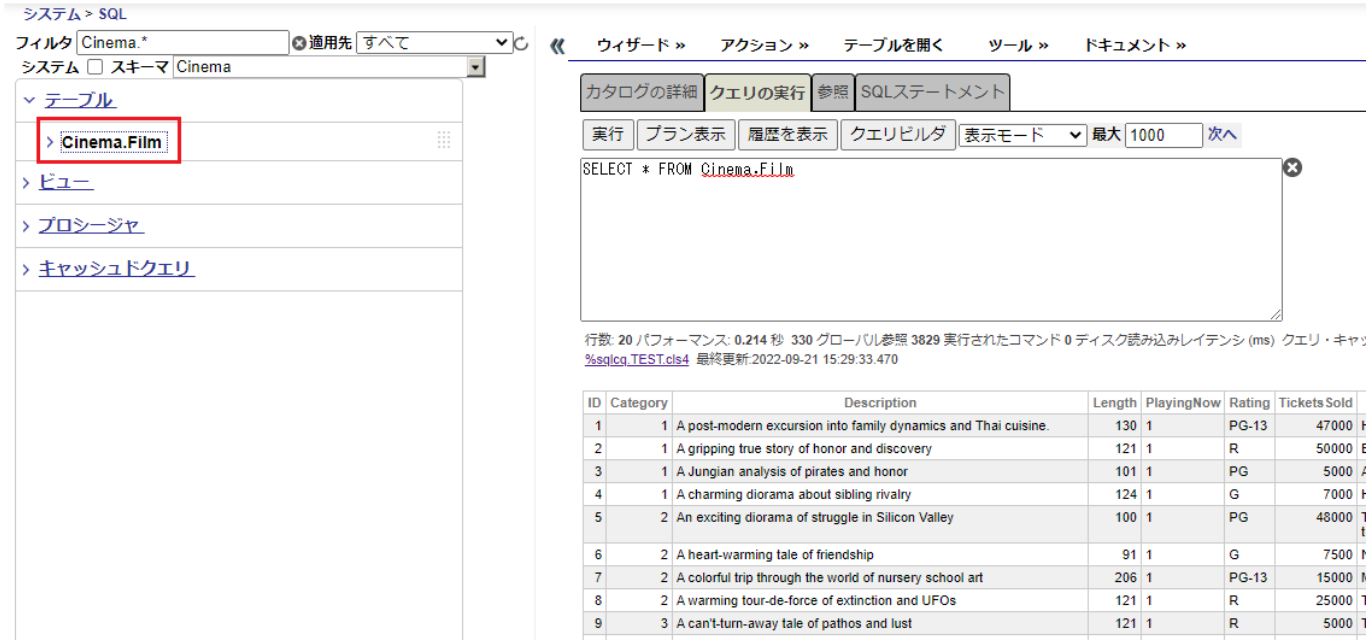
- Introduce una nueva persona con el nombre "TEST";
- Muestra el resultado de la introducción;
- Actualiza la persona "TEST" a "TEST ABC";
- Muestra el resultado de la actualización;

Ahora se podrá verificar la tabla de control de cambios. Para ello, entra en el Portal de Administración del Sistema -> Explorador del Sistema -> SQL. (No olvides cambiarse a su namespace)

Ejecuta el siguiente comando en SQL y verifica los resultados:
SELECT * FROM Sample.Audit



Debes Tener en cuenta que la variable OldValue es "TEST" y la NewValue es "TEST ABC". A partir de ahora podrás realizar tus propias pruebas al cambiar el nombre "TEST ABC" por "Su propio nombre" y/o cambiar, por ejemplo, los valores de Age. Consulte:
UPDATE Sample.Person SET Name = 'Fabio Goncalves' WHERE Name = 'TEST ABC'



ID	Category	Description	Length	PlayingNow	Rating	Tickets Sold
1	1	A post-modern excursion into family dynamics and Thai cuisine.	130	1	PG-13	47000
2	1	A gripping true story of honor and discovery	121	1	R	50000
3	1	A Jungian analysis of pirates and honor	101	1	PG	5000
4	1	A charming diorama about sibling rivalry	124	1	G	7000
5	2	An exciting diorama of struggle in Silicon Valley	100	1	PG	48000
6	2	A heart-warming tale of friendship	91	1	G	7500
7	2	A colorful trip through the world of nursery school art	206	1	PG-13	15000
8	2	A warming tour-de-force of extinction and UFOs	121	1	R	25000
9	3	A can't-turn-away tale of pathos and lust	121	1	R	5000

Generación del Código

Teniendo en cuenta que hemos implementado el procedimiento de control de cambios que se muestra a continuación, inicia Studio (o Atelier) en tu equipo, abre la clase persistente (Sample.Person) y analiza el código intermedio que se generó después de compilar la clase Sample.Person. Para hacerlo, presiona Ctrl + Shift + V (Consultar otro código fuente), y analiza .INT. Desplázate hacia abajo hasta la etiqueta zSaveAuditAfterExecute y échele un vistazo al código que se generó:

```
zSaveAuditAfterExecute(%oper=0,pNew,pOld,pChanged,%ok,%msg)
    ; get username and ip address
    Set tSC = 1
    Set tUsername = $USERNAME
    Try {
        ; Check if the operation is an update - %oper =  UPDATE
        if %oper = "UPDATE" {
            ;
            ;
            ; Audit Field: Age
            if pChanged(2) {
                Set tAudit = ##class(Sample.Audit).%New()
                Set tAudit.ClassName = "Sample.Person"
                Set tAudit.Id = pNew(1)
                Set tAudit.UserName = tUsername
                Set tAudit.Field = "Age"
                Set tAudit.Date = +$Horolog
                Set tAudit.OldValue = pOld(2)
                Set tAudit.NewValue = pNew(2)
                Set tSC = tAudit.%Save()
                If ('tSC) Throw ##class(%Exception.StatusException).ThrowIfInterrupt(tSC)
            }
            ;
            ;
            ; Audit Field: Name
            if pChanged(3) {
                Set tAudit = ##class(Sample.Audit).%New()
                Set tAudit.ClassName = "Sample.Person"
                Set tAudit.Id = pNew(1)
                Set tAudit.UserName = tUsername
                Set tAudit.Field = "Name"
                Set tAudit.Date = +$Horolog
                Set tAudit.OldValue = pOld(3)
                Set tAudit.NewValue = pNew(3)
                Set tSC = tAudit.%Save()
                If ('tSC) Throw ##class(%Exception.StatusException).ThrowIfInterrupt(tSC)
            }
        }
```

Ventajas

Es muy sencillo implementar el control de cambios basado en el despliegue de datos antiguos. No es necesario tablas adicionales. El mantenimiento también es sencillo. Si decides eliminar datos antiguos, entonces debes utilizar un SQL.

Si se necesita implementar el control de cambios en más tablas, solo es necesario heredarlo desde la clase abstracta (Sample.AuditBase)

Realiza los cambios conforme a tus necesidades, por ejemplo, registra los cambios en Streams.

Registra solamente los campos que se han modificado. No guarda todo el historial de cambios.

Desventajas

Un problema puede ser que cuando se modifican los datos, también se copia todo el historial, es decir, también se copian los datos que no se modificaron.

Si la tabla persona tiene una columna "foto" con los datos binarios (stream) que contiene la fotografía, entonces cada vez que el usuario cambie la imagen también se registrará la función stream (la cual consume espacio en el disco).

Otra desventaja es que también aumenta la complejidad con cada tabla complementaria que añade en el control de cambios. Deberas tener en cuenta, todo el tiempo, que no es tan sencillo recuperar los historiales. Siempre debes utilizar la cláusula SELECT con el condicional: "...WHERE Status = active" o considere algún "DATE INTERVAL"

Todas las modificaciones que se realicen a los datos se registran en una tabla común.

Piensa en las translocaciones y los rollbacks.

El control de cambios es un requisito importante para que algunas aplicaciones sean eficientes. Por lo general, para determinar si hubo alguna modificación en los datos, los desarrolladores deben implementar un método de seguimiento personalizado en sus aplicaciones, mediante una combinación de triggers, columnas para registrar la hora y tablas adicionales. Crear estos procedimientos normalmente requiere mucho trabajo para su implementación, conduce a actualizaciones de esquemas y con frecuencia, implica una disminución en el rendimiento. Este es un ejemplo sencillo que te podría ayudar a establecer tu propia estructura.

#Modelo de datos de objetos #ObjectScript #Caché

URL de fuente:https://es.community.intersystems.com/post/c%C3%B3mo-realizar-un-seguimiento-de-los-cambios-en-los-datos-el-control-de-cambios-parte-1-de-2