記事
Toshihiko Minamoto · 2021年1月5日 9m read

データ変更の追跡 - 監査ログ（1/2）

はじめに

多くのアプリケーションに共通する要件は、データベース内のデータ変更のログ記録です。どのデータが変更されたか、誰がいつ変更したかをログに記録する必要があります（監査ログ）。
このような質問について書かれた記事は多く存在し、Caché で行う方法の切り口もさまざまです。

そこで、データ変更を追跡して記録するためのフレームワークを実装しやすくする仕組みを説明することにします。 これは、永続クラスが「監査抽象クラス」（Sample.AuditBase）から継承すると「objectgenarator」メソッドを介してトリガーを作成する仕組みです。 永続クラスは Sample.AuditBase
から継承されるため、永続クラスをコンパイルすると、変更を監査するためのトリガーが自動的に生成されます。

監査クラス

次は、変更が記録されるクラスです。

```
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 = "");
}
```

監査抽象クラス

これは、永続クラスの継承元となる抽象クラスです。 このクラスには、監査テーブル（Sample.Audit）に変更を書き込むほか、どのフィールドが変更されたのか、誰が変更したのか、変更前と後の値は何であるかなどを識別する方法を知っているトリガーメソッド（objectgenerator）が含まれています。

```
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
}

}
```

データクラス（永続クラス）

これは、ユーザー（アプリケーション）が変更を加えたり、レコードを作成したり、レコードを削除したりなど、ユーザーに許可したことをすべて実行するユーザーデータクラスです。 :)。 つまり、通常は %Persistent クラスということです。

変更を追跡して記録するには、この永続クラスを抽象クラス（Sample.AuditBase）から継承する必要があります

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

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

テスト

監査抽象クラス（Sample.AuditBase）からデータクラス（Sample.Person）を継承しているため、データの挿入、変更の追加、および変更内容の確認を監査クラス（Sample. Audit）で行うことができます。

これをテストするには、Sample.Person クラスまたは他の任意のクラスに Test()
クラスメソッドを作成する必要があります。
```
 ClassMethod Test(pKillExtent = )
{
  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
}
```

Test() メソッドを実行しましょう。
```
 d ##class(Sample.Person).Test(1)
```

パラメータの「1」は、Sample.Person と Sample.Audit クラスからエクステントを削除します。

```
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
```

このテストクラスメソッドは次の内容を実行します。

- 「TEST」という名前で新しい人を挿入する
- 挿入の結果を表示する
- 「TEST」という人を「TEST ABC」に更新する
- 更新の結果を表示する

ここで、監査ログテーブルを確認してみましょう。 これを行うには、システム管理ポータル -> システムエクスプローラ -> SQL を開きます。 （ネームスペースを忘れずに切り替えてください）

次の SQL コマンドを実行して結果を確認します。

```
SELECT * FROM Sample.Audit
```

OldValue が「TEST」、NewValue が「TEST ABC」であることに注意してください。これ以降は、「TEST ABC」を自分の名前に変更したり、年齢の値を変更したりして、独自のテストを行ってみると良いでしょう。次に例を示します。

```
UPDATE Sample.Person SET Name = 'Fabio Goncalves' WHERE Name = 'TEST ABC'
```

生成されるコード

以下の監査メカニズムを実装しているとした上で、コンピュータで Studio（または Atelier）を起動し、永続クラス（Sample.Person）を開いて、Sample.Person クラスをコンパイルした後に生成される中間コードを調べてみましょう。 これを行うには、Ctrl + Shift + V（ほかのソースコードを表示）を押して、.INT を検査します。 zSaveAuditAfterExecute ラベルまでスクロールし、生成されたコードを確認します。

メリット

古いデータのロールアウトに基づいて監査ログ機能を実装するのは簡単です。
追加のテーブルは必要ありません。 メンテナンスも簡単で、 古いデータを削除するのであれば、1 つの SQL で済みます。
ほかのテーブルでも監査ログ機能を実装する必要がある場合は、抽象クラス（Sample.AuditBase）から継承するだけです。
必要に応じて変更してください。 例: ストリームの変更を記録する。
変更されたフィールドのみを記録し、 変更したレコード全体を保存しないでください。

デメリット

データが変更されると、レコード全体がコピーされるため、変更されていないデータもコピーされてしまうことが問題となる場合があります。
テーブル Person に写真が含まれるバイナリデータ（stream）を持つ「photo」という列がある場合、ユーザーが写真を変更するたびに、ストリーム全体が記録されてしまいます（ディスクスペースが消費されてしまいます）。
もう 1 つの難点は、監査ログをサポートする各テーブルの複雑さが増すところにあります。
レコードの取得は容易にはいかないことを肝に銘じておきましょう。 SELECT 句は必ず条件「...WHERE Status = active」とともに使用するか、「DATE INTERVAL」などを検討するようにしてください。
すべてのデータ変更は共通テーブルにログされます。
トランザクションをロールバックとして考えるようにしましょう。

アプリケーションの効率化には、監査は重要な要件です。 通常、データ変更を判別するには、アプリケーション開発者が、トリガー、タイムスタンプ列、およびその他のテーブルを組み合わせてアプリケーションにカスタムの追跡メソッドを実装することが必要です。 こういった仕組みを作成するには大抵、多くの作業を実装する必要があり、スキーマの更新や高パフォーマンスのオーバーヘッドが生じることがよくあります。
この記事は簡単な例として、独自のフレームワークを作成し始める際に役立ててください。

#ObjectScript #オブジェクトデータモデル #ベストプラクティス #Caché

ソースURL:
https://jp.community.intersystems.com/post/%E3%83%87%E3%83%BC%E3%82%BF%E5%A4%89%E6%9B%B4%E3%81%AE%E8%BF%BD%E8%B7%A1-%E7%9B%A3%E6%9F%BB%E3%83%AD%E3%82%B0%EF%BC%8812%EF%BC%89