記事
Toshihiko Minamoto · 2021年3月1日 11m read

グロ-バルをクラスにマッピングする技 (4/3)

グロ-バルをクラスにマッピングする技 (4/3)

三連載のはずが 4
記事目に突入してしまいました。『銀河ヒッチハイクガイド』のファンという方はいませんか？

古くなった MUMPS アプリケーションに新たな生命を吹き込みたいとお考えですか？
以下にご紹介するステップを実行すれば、グロ-バルをクラスにマッピングし、美しいデ-タを Object や SQL に公開できます。

上の内容に馴染みが無い方は、以下の記事を初めからお読みください。

グロ-バルをクラスにマッピングする技 (1/3)

グロ-バルをクラスにマッピングする技 (2/3)

グロ-バルをクラスにマッピングする技 (3/3)

この記事は Joel、あなたのために書きます！
前回の例で定義した親子関係を土台に、今度は孫クラスを作成、^ParentChild グロ-バルに追加された季節情報を処理したいと思います。

前回と同じ免責事項: これらの記事を読んでもグロ-バルがよく理解できないという方は、WRC (Support@InterSystems.com) までメ-ルでお問い合わせください。喜んでサポ-トさせていただきます。

グロ-バルをクラスにマッピングするステップ。

1. グロ-バルデ-タが繰り返し使用されるパタ-ンを特定する。
2. 固有キ-の構成を特定する。
3. プロパティそれぞれの型を特定する。
4. クラス内のプロパティを定義する (数数の添え字をお忘れなく)。
5. IdKey のインデックスを定義する。
6. Storage Definition を以下の手順で定義する。

   a. 添え字を IdKey まで (IdKey を含む) 定義する。
   b. Data セクションを定義する。
   c. Row ID セクションには触れない。 デフォルトが 99% の割合で適切なので、これはシステムに任せます。

7. クラス / テ-ブルをコンパイルし、テストします。

^ParentChild(1)="Brendan^45956"

^ParentChild(1,"Hobbies",1)="Pit Crew"

^ParentChild(1,"Hobbies",1,"Seasons")="Fall*Winter"

^ParentChild(1,"Hobbies",2)="Kayaking"

^ParentChild(1,"Hobbies",2,"Seasons")="Spring*Summer*Fall"

^ParentChild(1,"Hobbies",3)="Skiing"

^ParentChild(1,"Hobbies",3,"Seasons")="Summer*Winter"

^ParentChild(2)="Sharon^46647"

^ParentChild(2,"Hobbies",1)="Yoga"

^ParentChild(2,"Hobbies",1,"Seasons")="Spring*Summer*Fall*Winter"

^ParentChild(2,"Hobbies",2)="Scrap booking"

^ParentChild(2,"Hobbies",2,"Seasons")="Spring*Summer*Fall*Winter"

^ParentChild(3)="Kaitlin^56009"

^ParentChild(3,"Hobbies",1)="Lighting Design"

^ParentChild(3,"Hobbies",1,"Seasons")="Spring*Summer*Fall*Winter"

^ParentChild(3,"Hobbies",2)="pets"

^ParentChild(3,"Hobbies",2,"Seasons")="Spring*Summer*Fall*Winter"

^ParentChild(4)="Melissa^56894"

^ParentChild(4,"Hobbies",1)="Marching Band"

^ParentChild(4,"Hobbies",1,"Seasons")="Fall"

^ParentChild(4,"Hobbies",2)="Pep Band"

^ParentChild(4,"Hobbies",2,"Seasons")="Winter"

^ParentChild(4,"Hobbies",3)="Concert Band"

^ParentChild(4,"Hobbies",3,"Seasons")="Spring*Summer*Fall*Winter"

^ParentChild(5)="Robin^57079"

^ParentChild(5,"Hobbies",1)="Baking"

^ParentChild(5,"Hobbies",1,"Seasons")="Spring*Summer*Fall*Winter"

^ParentChild(5,"Hobbies",2)="Reading"

^ParentChild(5,"Hobbies",2,"Seasons")="Spring*Summer*Fall*Winter"

^ParentChild(6)="Kieran^58210"

^ParentChild(6,"Hobbies",1)="SUBA"

^ParentChild(6,"Hobbies",1,"Seasons")="Summer"

^ParentChild(6,"Hobbies",2)="Marching Band"

^ParentChild(6,"Hobbies",2,"Seasons")="Fall"

^ParentChild(6,"Hobbies",3)="Rock Climbing"

^ParentChild(6,"Hobbies",3,"Seasons")="Spring*Summer*Fall"

^ParentChild(6,"Hobbies",4)="Ice Climbing"

^ParentChild(6,"Hobbies",4,"Seasons")="Winter"

ステップ 1:

この新しいクラスも繰り返し使用されるデータを見つけるのは簡単ですね、そう Season サブノードです。
“Spring*Summer*Fall” をすべて同じ行に並べる代わりに、“Spring”、“Summer”、“Fall” という個別の行を 3 つ作成する、という場合に少しややこしくなります。

^ParentChild(1)="Brendan^45956"

^ParentChild(1,"Hobbies",1)="Pit Crew"

^ParentChild(1,"Hobbies",1,"Seasons")="Fall*Winter"

^ParentChild(1,"Hobbies",2)="Kayaking"

^ParentChild(1,"Hobbies",2,"Seasons")="Spring*Summer*Fall"

^ParentChild(1,"Hobbies",3)="Skiing"

^ParentChild(1,"Hobbies",3,"Seasons")="Summer*Winter"

趣味 (Hobby) には、季節を最大 4 つ割り当てることができます。 Example3Child クラスにプロパティをあと 4 つ作成しないことはないですが、誰かが新しい季節を勝手に作ってしまったらどうなるでしょう。
そこで、もっと柔軟な解決策として、孫テーブルを作り、季節の数を動的に割り当てられるようにします。

ステップ 2:

添え字に 注 目すれば、IdKey に含まれる部分が分かるので、添え字 1 と 3 は IdKey に含まれるということが分かります。ところが、それぞれの季節を一意に識別するにはもう 1 つ添え字が必要なのですが、使える添え字が残っていません!

マッピングに入力している情報は、クエリを実行するためのコードを生成するために使用されます。 どのような COS
コマンドを使えばこの情報を取得できるのかと考えることで、マッピングを定義しやすくなるかもしれません。
ここで、実行する必要のある重要なコマンドが 3 つあります。

SET sub1=$ORDER(^Parentchild(sub1))

SET sub2=$ORDER(^Parentchild(sub1,"Hobbies",sub2))

SET season=$PIECE(^ParentChild(sub1,"Hobbies",sub2","Seasons"),"*",PC)

マッピングの Subscripts セクションでも同じことを行えます。 Caché SQL Storage は、4 種類の添え字をサポートしています (Piece、Global、Sub、Other)。 ここまでは、デフォルトの Sub を使っています。必需的な存在として活躍する $ORDER() ループを使えるのもそのおかげです。

この例では、Piece オプションをご紹介します。 このレベルで使用するプロパティは、Piece Counter (上の例で PC として表示) として使用されます。 デフォルトの動作として、文字列の最後に達するまでこれを 1

ずつ増加させます。

ステップ 3:

3 つのプロパティ: Data は単純な Season、そして Relationship プロパティ HobbyRef があり、最後に childsub として PieceCount が必要になります。

ステップ 4:

Property Season As %String;

Property PieceCounter As %Integer;

Relationship HobbyRef As Mapping.Example3Child [Cardinality = parent, Inverse = Seasons];

ステップ 5:

Subscripts のマッピングを見ると、変数のレベルが 3 つありますが、IdKey のインデックスでは、2 つのプロパティ HobbyRef と PieceCounter だけを参照しています。

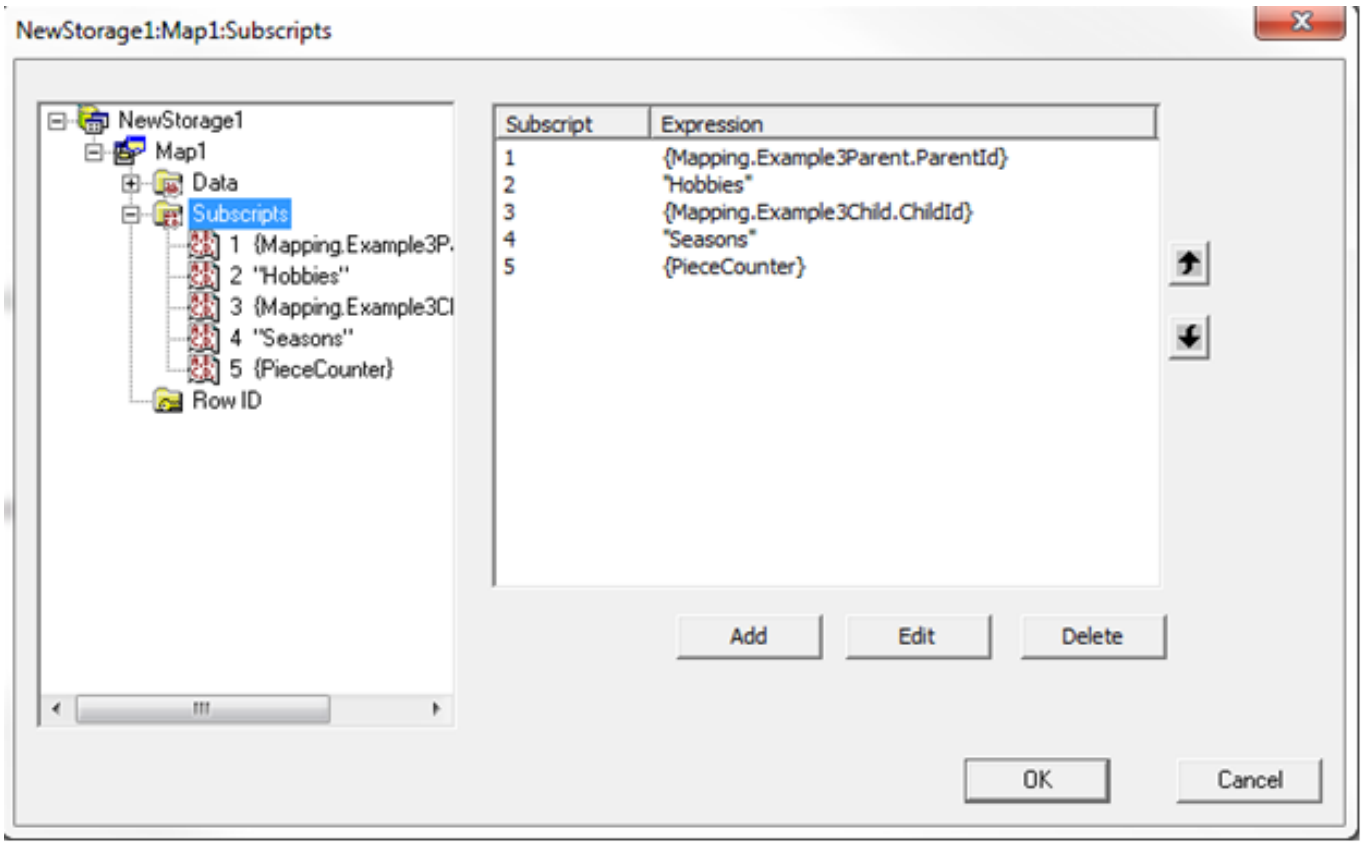
Index Master On (HobbyRef, PieceCounter) [IdKey];

ステップ 6:

これまで使用してきたお馴染みの 3 つのセクションです。 Row ID は引き続きそのままにしておきます。
このステップでは、Subscripts についてもう少し詳しく説明して、Access Type を定義する必要があります。
このクラスでは、'Sub' と 'Piece' を使います。 'Global' と 'Other' を使った例をご覧になりたい方は、私が例をまとめた zip ファイルをダウンロードしてください。

ステップ 6a:

Subscripts のメインページはいつも同じですが、レベルが 2 つ追加されています。 Parent 参照の 2 つの部分については、先ほど参照した 2 つのクラス {Mapping.Example3Parent} と {Mapping.Example3Child} を参照し直す必要があることを覚えておきましょう。 ウィンドウの左側にある Subscripts Levels の 1 つをクリックすると、違いが表示されます。



この画像では、Subscripts Level で行える様々なアクションをご覧いただけます。 'Sub' の Access Type では、「""」からはじめて、「""」に到達するまで $ORDER() を実行しようとお考えではないでしょうか。 そうでない場合は、Start Value または Stop Value またはその両方を指定できます。

'Data Access' を使うと、確認する内容を前のレベルから変更できます (イテレーションするグローバルを変更するなど)。

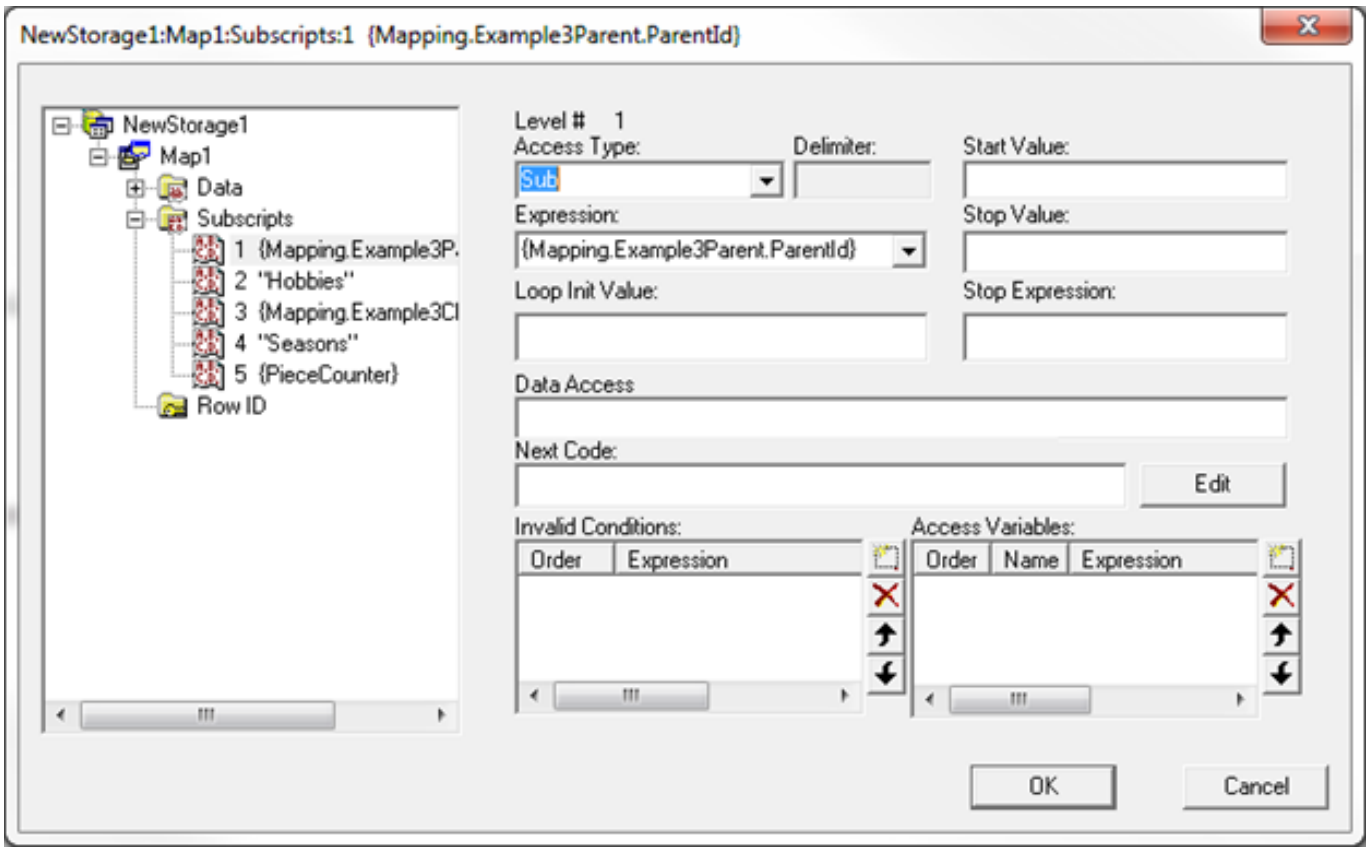
'Next Code' と 'Invalid Conditions' は一緒に使います。このウィンドウで一番頻繁に使うことになるでしょう。 シンプルな $ORDER() では有効な値を順に取得できないという場合は、代わりに独自のコードを書いてください。

'Next Code' は、$ORDER() と同様、1 つの有効な値からその次の有効な値に移動するために使用されます。

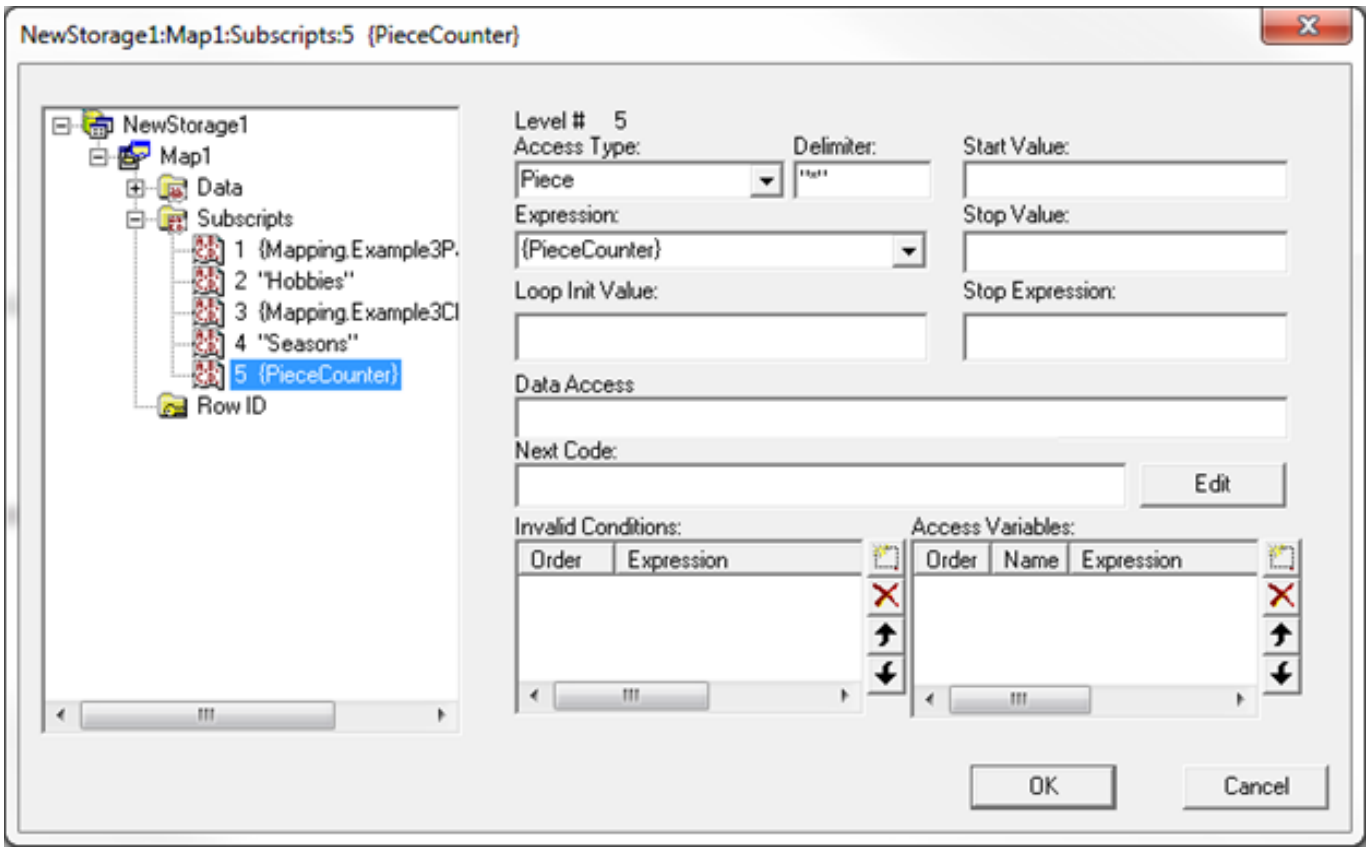
'Invalid Condition' は特定の値を評価するのに使用されます。 'subscriptX' の値を指定した場合は、それを見つけるためにわざわざ 'Next' を呼び出す必要がなくなります。 必要なのは、その値が有効なのかどうかを見極めるコードです。

長い間にわたってお約束してきた zip ファイルには、Next Code' と 'Invalid Conditions' を使うクラスがたくさん入っています。

ページの最後に登場する 'Access Variables' ですが、使うことは滅多にありません。 簡単に言うと、変数を設定し、1 つの Subscript Level で値を割り当て、それを上位の Subscript Level で使用するためのものです。 スコーピングは生成されるテーブルのコードが代わりにやってくれます。

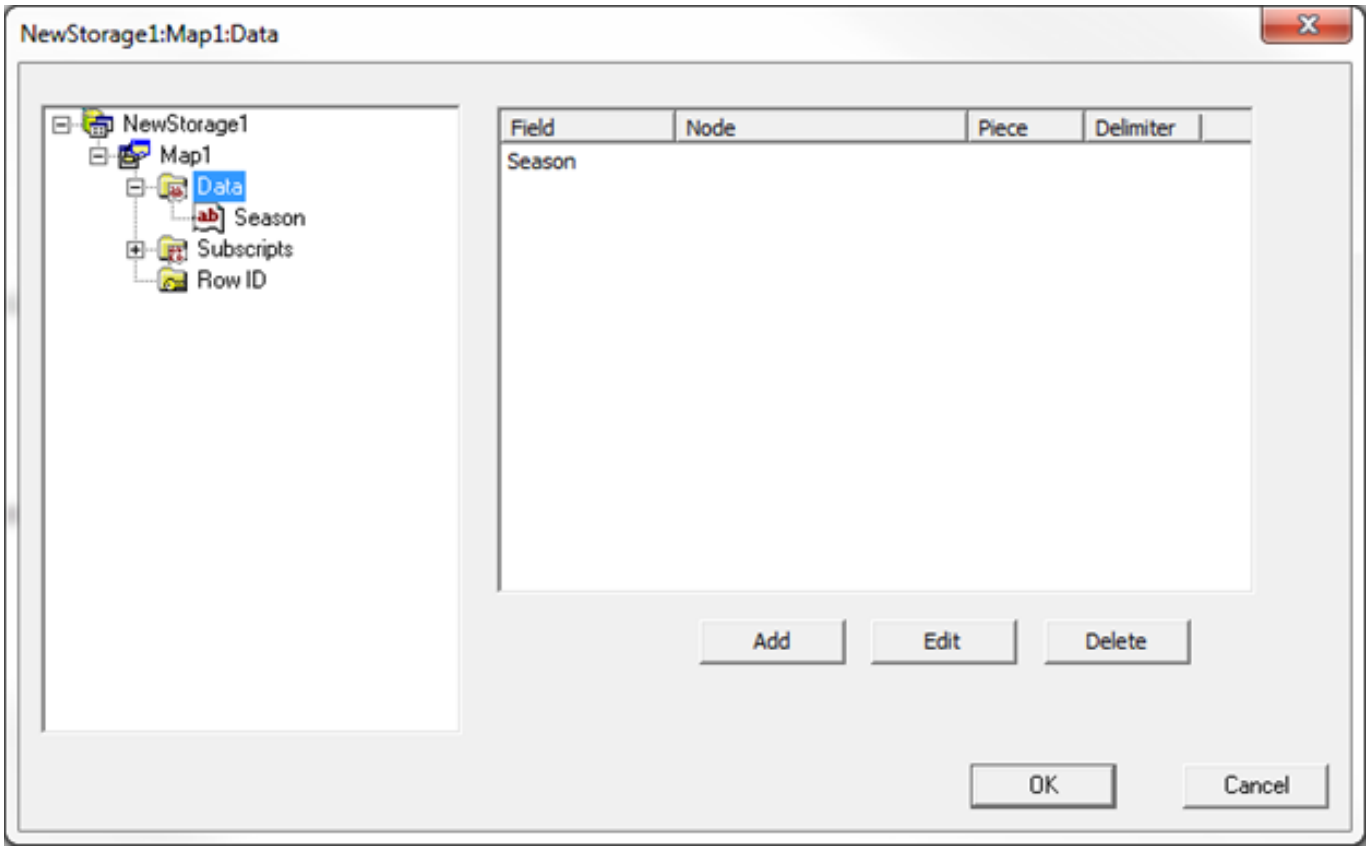


Subscript Level 5 では、'Access Type' は 'Piece'、'Delimiter' は "*" になります。 生成されるコードは、Piece 1 からスタートし、$PIECE の値がなくなるまで 1 ずつ増加していきます。 ここでも、Start Value または Stop Value またはその両方を指定すれば、これを制御できます。



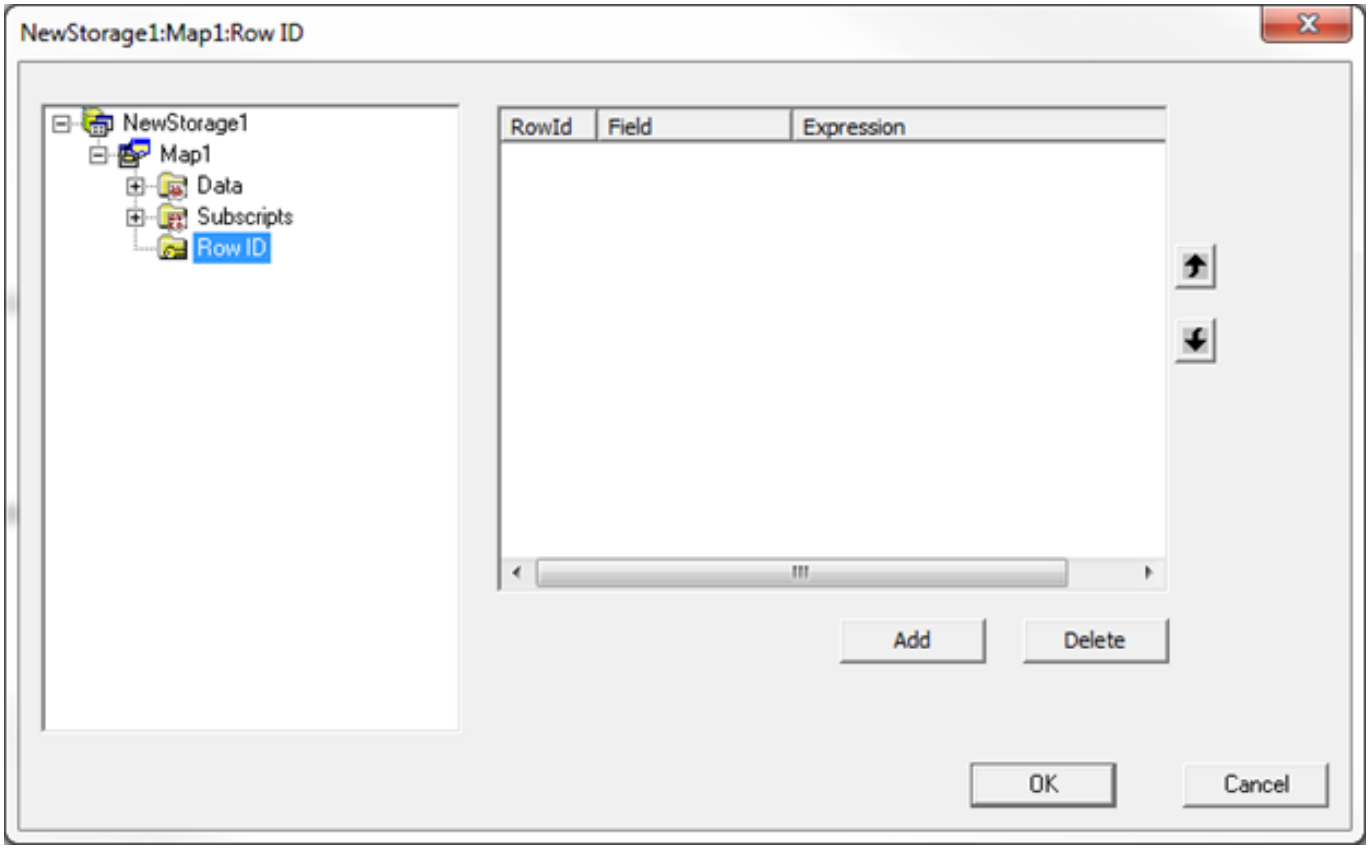
ステップ 6b:

Data セクションにはプロパティが 1 つしかないので、'Piece' も 'Delimiter' も必要ありません。
このテーブルにもっとフィールドがあれば、'Piece' と 'Delimiter'
を指定することになると思いますが、それはそれで問題ありません。



ｽﾃｯﾌﾟ 6c:

まだ空白のままにしておきます。



ｽﾃｯﾌﾟ 7:

すべて正常にｺﾝﾊﾟｲﾙします。

```
Compilation started on 11/30/2016 08:17:42 with qualifiers 'uk/importselectivity=1 /checkuptodate=expandedonly'
Compiling 2 classes, using 2 worker jobs
Compiling class Mapping.Example3Child
Compiling class Mapping.Example3GrandChild
Compiling table Mapping.Example3GrandChild
Compiling table Mapping.Example3Child
Compiling routine Mapping.Example3Child.1
Compiling routine Mapping.Example3GrandChild.1
Compilation finished successfully in 1.021s.
```

3 つのﾃｰﾌﾞﾙの結合

```
SELECT P.ID, P.Name, P.DateOfBirth,

C.ID, C.Hobby, G.ID, G.Season

FROM Mapping.Example3Parent P

JOIN Mapping.Example3Child C ON P.ID = C.ParentRef

JOIN Mapping.Example3Grandchild G ON C.ID = G.HobbyRef
```

WHERE P.Name = 'Kieran'

この結果、以下が出力されます。

ID	Name	DateOfBirth	ID	Hobby	ID	Season
6	Kieran	05/16/2000	6||1	SUBA	6||1||1	Summer
6	Kieran	05/16/2000	6||2	Marching Band	6||2||1	Fall
6	Kieran	05/16/2000	6||3	Rock Climbing	6||3||1	Spring
6	Kieran	05/16/2000	6||3	Rock Climbing	6||3||2	Summer
6	Kieran	05/16/2000	6||3	Rock Climbing	6||3||3	Fall
6	Kieran	05/16/2000	6||4	Ice Climbing	6||4||1	Winter

子テーブルの IdKey は、常に Parent 参照と Childsub の 2
つで構成されると説明したことを覚えておいてください。

最初の行は、Example3Child ID が 6||1、Parent 参照が 6、Childsub が 1 となっています。

Example3GrandChild の IdKey は、3 つの部分 (6||1||1) で構成されていますが、それでも Parent 参照と Childsub
を表しています。 Parent 参照は少し複雑な感じに 6||1 となって 、Childsub は 1 となっています。

Example3Child では、Subscripts のプロパティの数が IdKey のプロパティの数に一致しています。
親子構造の入れ子状態が深まると、IdKey は複合化され、添え字の数は増えていきます。

今回の例で使用した 3 つのクラスはこちらにエクスポートしておきました: MappingExample4.zip。

#マッピング #SQL #オブジェクトデータモデル #グローバル #Caché

ソースURL: https://jp.community.intersystems.com/post/%E3%82%B0%E3%83%AD%E3%83%BC%E3%83%90%E3%83%AB%E3%82%92%E3%82%AF%E3%83%A9%E3%82%B9%E3%81%AB%E3%83%9E%E3%83%83%E3%83%94%E3%83%B3%E3%82%B0%E3%81%99%E3%82%8B%E6%8A%80%E8%A1%93-43