記事
Toshihiko Minamoto · 2021年9月14日 10m read

Open Exchange

リレーショナルデータベースにおけるEntity-Attribute-Valueモデル。 グローバル変数はテーブルでエミュレートする必要がありますか？ パート2.

より産業向けのグローバルストレージスキーム

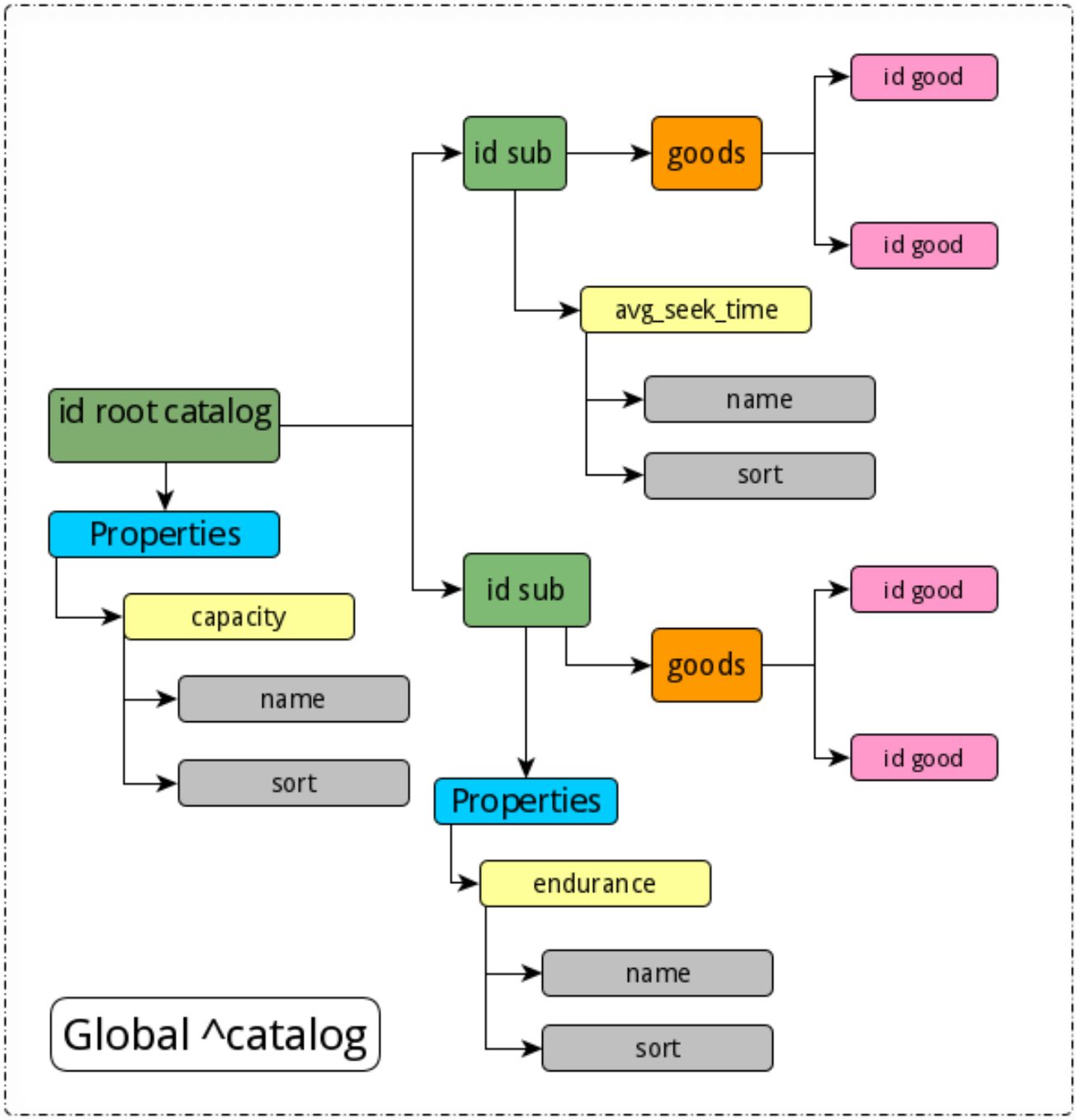
この連載の第1回では、リレーショナルデータベースにおけるEAV（Entity-Attribute-Value）モデルを取り上げ、テーブルにエンティティ、属性、および値を保存することのメリットとデメリットについて確認しました。 このアプローチには柔軟性という点でメリットがあるにもかかわらず、特にデータの論理構造と物理ストレージの基本的な不一致などによりさまざまな問題が引き起こされるという深刻なデメリットがあります。

こういった問題を解決するために、階層情報の保存向けに最適化されたグローバル変数を、EAVアプローチが通常処理するタスクに使用できるかどうかを確認することにしました。

パート1
では、オンラインストア向けのカタログをテーブルを使って作成し、その後で1つのグローバル変数のみで作成しました。 それでは、複数のグローバル変数で同じ構造を実装してみることにしましょう。

最初のグローバル変数^catalogには、ディレクトリ構造を保存します。
2つ目のグローバル変数^goodには、店の商品を保存します。
^indexグローバルには、店のインデックスを保存します。 プロパティは階層的なカタログに関連付けられているため、プロパティ用の個別のグローバル変数は作成しません。

このアプローチでは、エンティティごとに（プロパティを除く）、個別のグローバル変数を使用しているため、論理の観点では優れています。 グローバルカタログ構造は次のようになります。




```
Set ^?atalog(root_id, "Properties", "capacity", "name") = "Capacity, GB"
Set ^?atalog(root_id, "Properties", "capacity", "sort") = 1

Set ^?atalog(root_id, sub1_id, "Properties", "endurance", "name") = "Endurance, TBW"
Set ^?atalog(root_id, sub1_id, "Properties", "endurance", "sort") = 2

Set ^?atalog(root_id, sub1_id, "goods", id_good1) = 1
Set ^?atalog(root_id, sub1_id, "goods", id_good2) = 1

Set ^?atalog(root_id, sub2_id, "Properties", "avg_seek_time", "name") = "Rotate speed
, ms"
Set ^?atalog(root_id, sub2_id, "Properties", "avg_seek_time", "sort") = 3

Set ^?atalog(root_id, sub2_id, "goods", id_good3) = 1
```

```
Set ^?atalog(root_id, sub2_id, "goods", id_good4) = 1
```

商品のグローバル変数は、次のようになります。

```
Set ^good(id_good, property1) = value1
Set ^good(id_good, property2) = value2
Set ^good(id_good, property3) = value3
Set ^good(id_good, "catalog") = catalog_id
```

もちろん、商品のあるすべてのカタログセクションで、必要なプロパティで並べ替えを行えるようにインデックスが必要となります。 インデックスグローバルは、次のような構造になります。

```
Set ^index(id_catalog, property1, id_good) = 1
; To quickly get the full path to concrete sub-catalog
Set ^index("path", id_catalog) = "^catalog(root_id, sub1_id)"
```

したがって、カタログのすべてのセクションで、リストを並べ替えることができます。
インデックスグローバルはオプションです。 カタログのこのセクションの商品数が多い場合にのみ役立ちます。

デモデータを操作するためのObjectScriptコード

では、データを操作するために、ObjectScriptを使用しましょう。
まず、特定の商品のプロパティを取得することから始めます。
特定の商品のIDがあり、そのプロパティを並べ替えの値で指定された順序で表示する必要があります。
そのためのコードは次のようになります。

```
get_sorted_properties(path, boolTable)
{
  ; remember all the properties in the temporary global
  While $QLENGTH(@path) > 0 {
    if ($DATA(@path("Properties"))) {
      set ln=""
      for {
        Set ln = $order(@path("Properties", ln))
        Quit: ln = ""

        IF boolTable & @path("Properties", ln, "table_view") = 1 {
          Set ^tmp(@path("Properties", ln, "sort"), ln) = @path("Properties", ln, "na
me")
        }
      ELSE {
        Set ^tmp(@path("Properties", ln, "sort"), ln) = @path("Properties", ln, "name
")
      }
    }
  }
}

print_sorted_properties_of_good(id_good)
{
  Set id_catalog = ^good(id_good, "catalog")
  Set path = ^index("path", id_catalog)
```

```
  Do get_sorted_properties(path, 0)

  set ln =""
  for {
   Set ln = $order(^tmp(ln))
   Quit: ln = ""
   Set fn = ""
   for {
    Set fn = $order(^tmp(ln, fn))
    Quit: fn = ""
    Write ^tmp(ln, fn), " ", ^good(id_good, fn),!
   }
  }
}
```

次に、idcatalogに基づいて、カタログセクションの商品を表形式で取得します。

```
print_goods_table_of_catalog(id_catalog)
{
  Set path = ^index("path", id_catalog)
  Do get_sorted_properties(path, 1)

  set id=""
  for {
    Set id = $order(@path("goods"), id)
    Quit: id = ""

    Write id," ", ^good(id, "price"), " "

    set ln =""
    for {
      Set ln = $order(^tmp(ln))
      Quit: ln = ""
      Set fn = ""
      for {
        Set fn = $order(^tmp(ln, fn))
        Quit: fn = ""
        Write ^tmp(ln, fn), " ", ^good(id, fn)
      }
      Write !
    }
  }
}
```

可読性: EAV SQLとグローバル変数

では、EAVとSQLの使用をグローバル変数の使用と比較してみましょう。
コードの明確さについては、これが主観的なパラメーターであることは明らかです。
しかし、例として新しい商品の作成方法を見てみましょう。

SQLを使用したEAVアプローチから確認します。
まず、オブジェクトのプロパティリストを取得する必要があります。
これは別のタスクであり、非常に時間がかかります。

capacity、weight、およびenduranceという3つのプロパティのIDがすでに分かっているとします。

```
START TRANSACTION
INSERT INTO good (name, price, item_count, catalog_id) VALUES ('F320 3.2TB AIC SSD',
700, 10, 15);

SET @last_id = LAST_INSERT_ID ();

INSERT INTO NumberValues ??Values??(@last_id, @id_capacity, 3200);
INSERT INTO NumberValues ??Values??(@last_id, @id_weight, 0.4);
INSERT INTO NumberValues ??Values??(@last_id, @id_endurance, 29000);
COMMIT
```

この例ではプロパティが3つしかないため、例にはそれほど圧倒されません。
一般的なケースでは、トランザクション内のテキストテーブルにいくつかの挿入があります。

```
INSERT INTO TextValues ??Values??(@last_id, @ id_text_prop1, 'Text value of property
1');
INSERT INTO TextValues ??Values??(@last_id, @ id_text_prop2, 'Text value of property
2');
...
INSERT INTO TextValues Values (@last_id, @id_text_propN, 'Text value of property N');
```

もちろん、数値の代わりに「capacity」を使うというように、IDプロパティの代わりにテキスト表記を使用すれば、SQLバージョンをもう少し簡略することも可能ですが、 SQLの世界では、これは受け入れられません。
エンティティのインスタンスを列挙するには、数値IDを使用するのが慣例です。 このため、インデックス処理が高速化し（インデックス処理のバイトが少なくなるため）、一意性を追跡しやすくなり、新しいIDを自動的に作成しやすくなります。 この場合、挿入フラグメントは次のようになります。

```
INSERT INTO NumberValues ??Values??(@last_id, 'capacity', 3200);
INSERT INTO NumberValues ??Values??(@last_id, 'weight', 0.4);
INSERT INTO NumberValues ??Values??(@last_id, 'endurance', 29000);
```

次は、同じ例をグローバル変数を使用した場合のコードです。

```
TSTART
Set ^good(id, "name") = "F320 3.2TB AIC SSD"
Set ^("price") = 700, ^("item_count") = 10, ^("reserved_count") = 0, ^("catalog") = i
d_catalog
Set ^("capacity") = 3200, ^("weight") = 0.4, ^("endurance") = 29000
TCOMMIT
```

では、EAVアプローチで商品を削除してみましょう。

```
START TRANSACTION
DELETE FROM good WHERE id = @ good_id;
DELETE FROM NumberValues ??WHERE good_id = @ good_id;
DELETE FROM TextValues ??WHERE good_id = @ good_id;
COMMIT
```

そして、グローバル変数でも同じことを行います。

```
Kill ^good(id_good)
```

2つのアプローチをコードの長さの観点から比較することもできます。
上記の例からわかるように、グローバル変数を使用した方が、コードは短くなります。 これはメリットです。
コードが短くなるほど、エラーの数も減り、コードを理解して管理するのも容易になります。

一般に、コードが短いほど処理が高速化します。 そして、この場合には、グローバル変数はリレーショナルテーブルよりも低位データ構造であるため、確かにそのとおりです。

EAVとグローバル変数におけるデータのスケーリング

次に、水平方向のスケーリングを見てみましょう。 EAVアプローチでは、少なくとも3つの最も大きなテーブル（Good、NumberValues、TextValues）を複数のサーバーに分散する必要があります。 エンティティと属性のあるテーブルにはほとんど情報がないため、これらのテーブルは単純にすべてのサーバーに丸ごとコピーすることができます。

各サーバーでは、水平方向のスケーリングにより、さまざまな商品がGood、NumberValues、およびTextValuesテーブルに保存されます。
異なる商品でIDが重複しないように、各サーバーの商品に対して特定のIDブロックを割り当てる必要があります。

グローバルを使って水平方向のスケーリングを行う場合、グローバルでID範囲を構成し、グローバル範囲を各サーバーに割り当てる必要があります。

複雑さは、EAVとグローバルであまり変わりませんが、EAVアプローチの場合は、3つのテーブルにID範囲を構成しなければなりません。 グローバルの場合は、1つのグローバル変数のみにIDを構成するだけで済みます。
つまり、水平方向のスケーリングを調整するには、グローバル変数の方が簡単と言えます。

EAVとグローバル変数におけるデータ損失

最後に、データベースファイルの破損によるデータ損失のリスクを検討してみましょう。 5つのテーブルか3つのグローバル（インデックスグローバルを含む）のどちらにすべてのデータを保存する方が簡単でしょうか。

3つのグローバルの方が簡単だと思います。 EAVアプローチでは、さまざまな商品のデータがテーブルに混在しているのに対し、グローバルでは情報がより全体的に保存されています。
基盤のブランチは、保存されて順に並べ替えられています。 そのため、データがパスタが絡み合うように保存されるEAVアプローチに比べれば、グローバルの一部の破損によってダメージにつながる可能性は低くなります。

データ回復におけるもう1つの悩みの種は、情報の表示方法です。 EAVアプローチでは、情報は複数のテーブルに分割されているため、1つにまとめるには特別なスクリプトが必要です。 グローバルの場合は、ZWRITEコマンドを使用するだけで、ノードのすべての値と基盤のブランチを表示することができます。

InterSystems IRISのグローバル: より優れたアプローチ？

EAVアプローチは、階層データを保存するためのトリックとして出現しました。
テーブルは元々、ネストされたデータを保存するようには設計されてはいなかったため、
テーブルでグローバルをエミュレーションするのがEAVの事実上のアプローチです。 テーブルがグローバルよりも高位で低速のデータストレージ構造であることを考えると、EAVアプローチは、グローバルと比較した場合に失敗となります。

個人的な意見を言えば、階層データ構造の場合、グローバルの方がプログラミングの点でより利便性が高く理解しやすいと思います。また、より高速でもあります。

プロジェクトでEAVアプローチを計画している場合は、InterSystems
IRISのグローバルを使用して階層データを保存することを検討するようお勧めします。

#SQL #グローバル #データベース #パフォーマンス #ヒントとコツ #リレーショナルテーブル #非構造化データ
#Caché #InterSystems IRIS #InterSystems IRIS for Health
InterSystems Open Exchangeで関連アプリケーションを確認してください

ソースURL:
https://jp.community.intersystems.com/post/%E3%83%AA%E3%83%AC%E3%83%BC%E3%82%B7%E3%83%A7%E3%83%8A%E3%83%AB%E3%83%87%E3%83%BC%E3%82%BF%E3%83%99%E3%83%BC%E3%82%B9%E3%81%AB%E3%81%8A%E3%81%91%E3%82%8Bentity-attribute-value%E3%83%A2%E3%83%87%E3%83%AB%E3%80%82-%E3%82%B0%E3%83%AD%E3%83%BC%E3%83%90%E3%83%AB%E5%A4%89%E6%95%B0%E3%81%AF%E3%83%86%E3%83%BC%E3%83%96%E3%83%AB%E3%81%A7%E3%82%A8%E3%83%9F%E3%83%A5%E3%83%AC%E3%83%BC%E3%83%88%E3%81%99%E3%82%8B%E5%BF%85%E8%A6%81%E3%81%8C%E3%81%82%E3%82%8A%E3%81%BE%E3%81%99%E3%81%8B%EF%BC%9F-%E3%83%91%E3%83%BC%E3%83%882