Article
Lorenzo Scalese · Juin 1, 2022 9m de lecture

Open Exchange

Modèle entité-attribut-valeur dans les bases de données relationnelles. Faut-il émuler les globales dans les tables ? Partie 2

Un système de stockage global d'aspect plus industriel

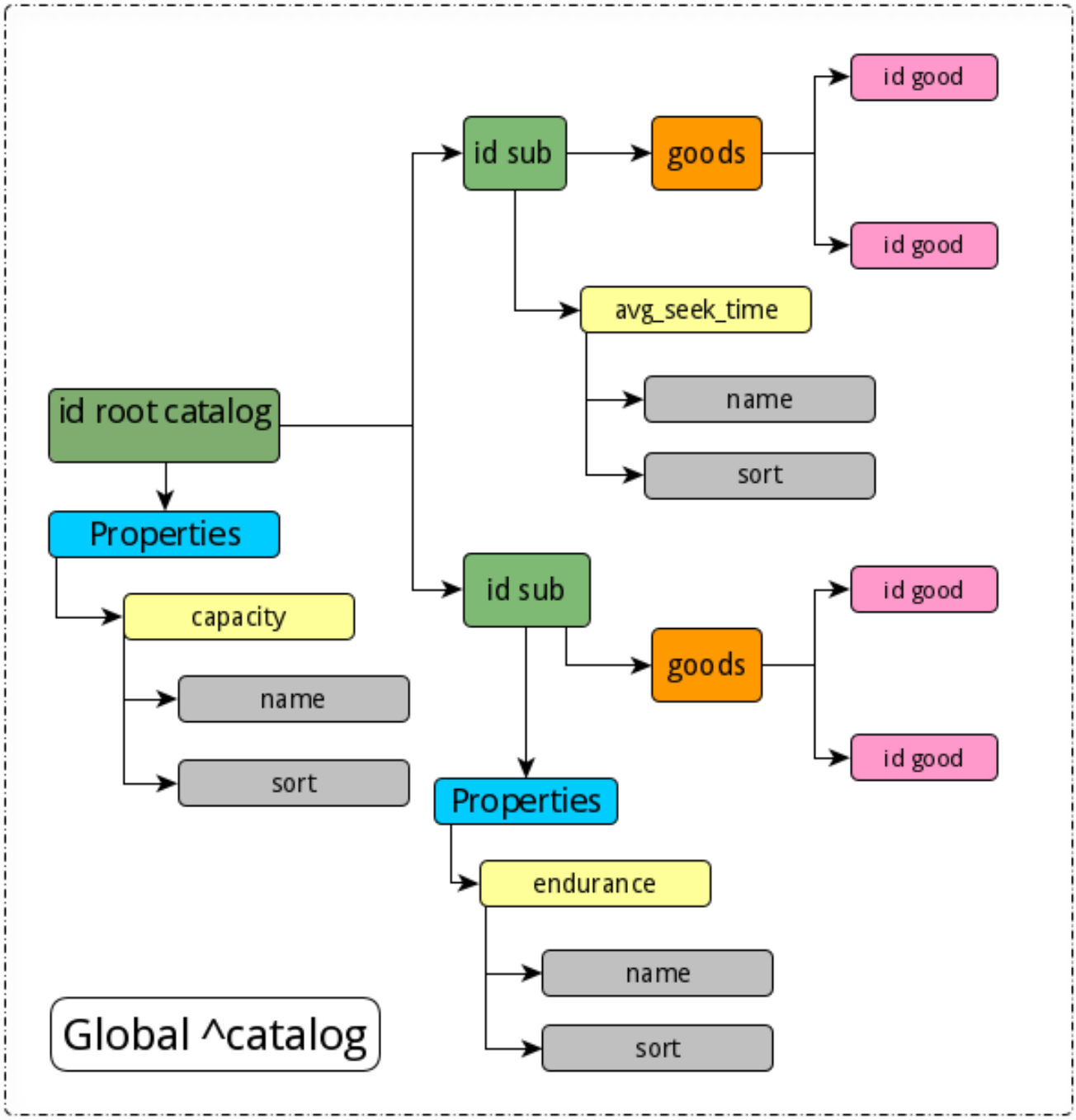
Dans le premier article de cette série, nous avons étudié le modèle entité-attribut-valeur (EAV) dans les bases de données relationnelles, et nous avons examiné les avantages et les inconvénients du stockage de ces entités, attributs et valeurs dans des tables. Nous avons appris que, malgré les avantages de cette approche en termes de flexibilité, elle présente de réels inconvénients, notamment une inadéquation fondamentale entre la structure logique des données et leur stockage physique, qui entraîne diverses difficultés.

Pour résoudre ces problèmes, nous avons décidé de voir si l'utilisation de globales - qui sont optimisées pour le stockage d'informations hiérarchiques - serait efficace pour les tâches que l'approche EAV traite habituellement.

Dans la Partie 1, nous avons créé un catalogue pour une boutique en ligne, d'abord en utilisant des tables, puis en utilisant une seule globale. Maintenant, essayons d'implémenter la même structure pour quelques globales.

Dans la première globale, ^catalog, nous allons stocker la structure du répertoire. Dans la deuxième globale, ^good, nous allons stocker les marchandises. Et dans la globale ^index, nous allons stocker les index. Puisque nos propriétés sont liées à un catalogue hiérarchique, nous ne créerons pas de globale séparée pour elles.

Avec cette approche, pour chaque entité (à l'exception des propriétés), nous avons une globale séparée, ce qui est bon du point de vue de la logique. Voici la structure du catalogue global :




```
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
```

Une globale avec des marchandises ressemblera à quelque chose comme ceci :

```
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
```

Bien sûr, nous avons besoin d'index afin que pour toute section du catalogue contenant des marchandises, nous puissions trier par les propriétés dont nous avons besoin. Une globale d'index aura une structure semblable à quelque chose comme ceci :

```
Configurer ^index(id_catalog, property1, id_good) = 1
; Pour obtenir rapidement le chemin complet du sous-catalogue concret
Configurer ^index("path", id_catalog) = "^catalog(root_id, sub1_id)"
```

Ainsi, dans n'importe quelle section du catalogue, on peut obtenir une liste triée. Une globale d'index est facultative. Il n'est utile que si le nombre de produits dans cette section du catalogue est important.

Code ObjectScript pour travailler avec des données de démonstration Demo Data

Maintenant, nous allons utiliser ObjectScript pour travailler avec nos données. Pour commencer, nous allons obtenir les propriétés d'une marchandise spécifique. Nous avons l'ID d'une marchandise particulière et nous devons afficher ses propriétés dans l'ordre donné par la valeur de tri. Voici le code pour cela :

```
get_sorted_properties(path, boolTable)
{
  ; mémoriser toutes les propriétés dans la globale temporaire
  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),!
   }
  }
}
```

Ensuite, nous voulons récupérer les produits de la section catalogue sous la forme de la table, basé sur idcatalog :

```
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 !
    }
  }
}
```

Lisibilité : EAV SQL contre les globales

Comparons maintenant l'utilisation d'EAV et de SQL par rapport à l'utilisation de globales. En ce qui concerne la clarté du code, il est évident qu'il s'agit d'un paramètre subjectif. Mais regardons, par exemple, la création d'un nouveau produit.

Nous allons commencer par l'approche EAV, en utilisant SQL. Tout d'abord, nous devons obtenir une liste des propriétés de l'objet. Il s'agit d'une tâche distincte qui prend beaucoup de temps. Supposons que nous

connaissions déjà les IDs de ces trois propriétés : capacité, poids, et endurance.

```
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
```

Dans cet exemple, nous n'avons que trois propriétés, et l'exemple ne semble donc pas si inquiétant. Dans le cas général, nous aurions toujours quelques insertions dans la table de texte à l'intérieur de la transaction :

```
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');
```

Bien sûr, nous pourrions simplifier un peu la version SQL si nous utilisions la notation textuelle à la place des propriétés ID, par exemple "capacité" au lieu d'un nombre. Mais dans le monde SQL, ce n'est pas acceptable. Il est plutôt d'usage d'utiliser un ID numérique pour énumérer les instances d'entités. Cela permet d'obtenir des index plus rapides (il faut indexer moins d'octets), il est plus facile de suivre l'unicité et il est plus facile de créer automatiquement un nouvel ID. Dans ce cas, le fragment d'insertion aurait l'apparence suivante :

```
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);
```

Voici le même exemple en utilisant des globales :

```
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
```

Supprimons maintenant une marchandise en utilisant l'approche EAV :

```
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
```

Et ensuite, faisons la même chose avec les globales :

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

Nous pouvons également comparer les deux approches en termes de longueur de code. Comme vous pouvez le constater dans les exemples précédents, lorsque vous utilisez des globales, le code est plus court. C'est une bonne chose. Plus le code est court, moins il y a d'erreurs et plus il est facile à comprendre et à gérer.

En général, un code plus court est aussi plus rapide. Et, dans ce cas, c'est certainement vrai, puisque les globales constituent une structure de données de niveau inférieur aux tables relationnelles.

Mise à l'échelle des données avec EAV et Globales

Ensuite, examinons la mise à l'échelle horizontale. Avec l'approche EAV, nous devons au moins distribuer les trois plus grandes tables sur les serveurs : Good, NumberValues, et TextValues. Les tables contenant des entités et des attributs peuvent simplement être entièrement copiés sur tous les serveurs, car ils contiennent peu d'informations.

Dans chaque serveur, avec une mise à l'échelle horizontale, des produits différents seraient stockés dans les tables Good, NumberValues et TextValues. Nous devrions allouer certains blocs d'identification pour les produits sur chaque serveur afin d'éviter la duplication des identifiants pour des produits différents.

Pour une mise à l'échelle horizontale avec des globales, il faudrait configurer des plages d'ID dans la globale et attribuer une plage de globale à chaque serveur.

La complexité est à peu près la même pour EAV et pour les globales, sauf que pour l'approche EAV, nous devrions configurer des plages d'ID pour trois tables. Avec les globales, nous configurons les ID pour une seule globale. C'est-à-dire qu'il est plus facile d'organiser la mise à l'échelle horizontale pour les globales.

Perte de données avec EAV et avec Globales

Enfin, considérons le risque de perte de données dû à des fichiers de base de données corrompus. Où est-il plus facile de sauvegarder toutes les données : dans cinq tables ou dans trois globales (y compris une globale d'index) ?

Je pense que c'est plus facile dans trois globales. Avec l'approche EAV, les données des marchandises différentes sont mélangées dans des tables, alors que pour les globales, les informations sont stockées de manière plus holistique. Les branches sous-jacentes sont stockées et triées séquentiellement. Par conséquent, la corruption d'une partie de la globale est moins susceptible d'entraîner des dommages que la corruption de l'une des tables dans l'approche EAV, où les données sont stockées comme des pâtes entremêlées.

Un autre casse-tête dans la récupération des données est l'affichage des informations. Avec l'approche EAV, les informations sont réparties entre plusieures tables et des scripts spéciaux sont nécessaires pour les assembler en un seul ensemble. Dans le cas des globales, vous pouvez simplement utiliser la commande ZWRITE pour afficher toutes les valeurs et les branches sous-jacentes du nœud.

Les Globales d'InterSystems IRIS : Une meilleure approche ?

L'approche EAV est apparue comme une astuce pour stocker des données hiérarchiques. Les tables n'ont pas été conçus à l'origine pour stocker des données imbriquées. L'approche EAV de facto est l'émulation des globales dans les tables. Étant donné que les tables représentent une structure de stockage de données de plus haut niveau et plus lente que les globales, l'approche EAV échoue par rapport aux globales.

À mon avis, pour les structures de données hiérarchiques, les globales sont plus pratiques et plus

compréhensibles en termes de programmation, tout en étant plus rapides.

Si vous avez prévu une approche EAV pour votre projet, je vous suggère d'envisager d'utiliser les globales d'InterSystems IRIS pour stocker les données hiérarchiques.

#Bases de données #Conseils et astuces #Données non structurées #Globals #Performances #SQL #Tables relationnelles #Caché #InterSystems IRIS #InterSystems IRIS for Health
Voir l'application sur InterSystems Open Exchange