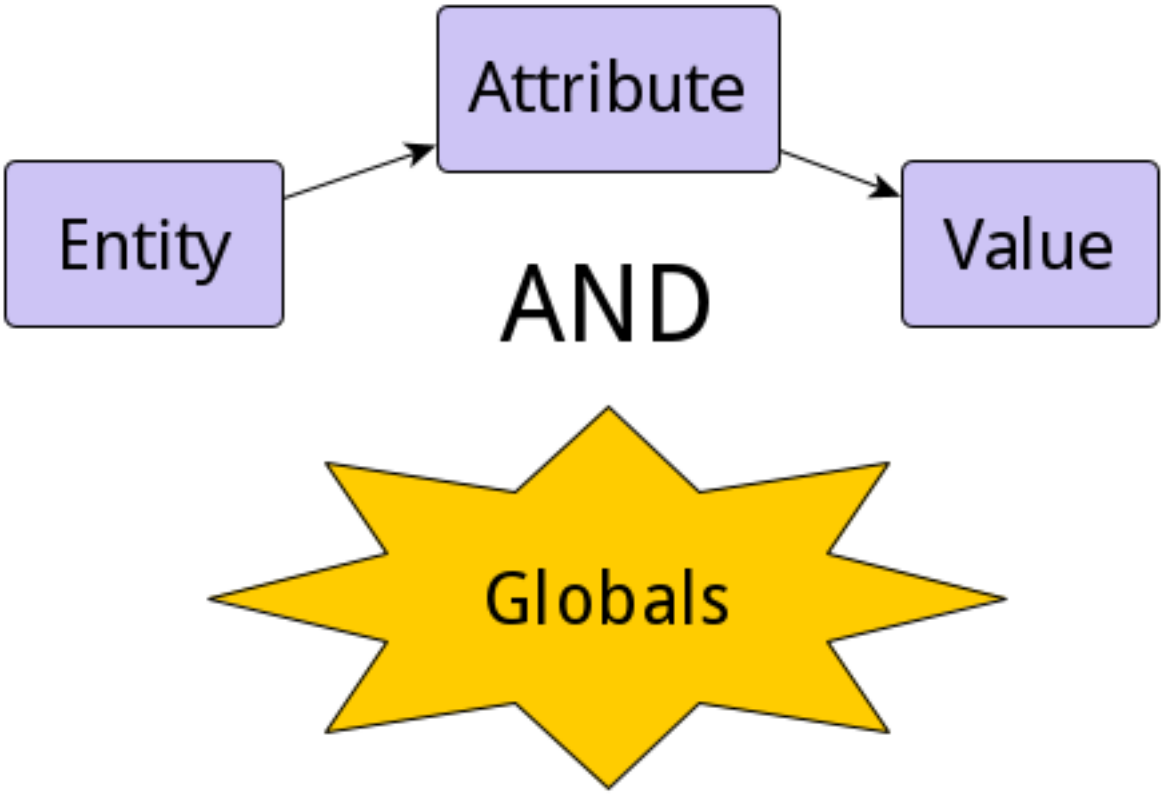
Article
Lorenzo Scalese · Mai 30, 2022 9m de lecture

Open Exchange

Modèle entité-attribut-valeur dans les bases de données relationnelles. Faut-il émuler les globales dans les tables ? Partie 1



Introduction

Dans le premier article de cette série, nous examinerons le modèle entité-attribut-valeur (EAV) dans les bases de données relationnelles pour voir comment il est utilisé et à quoi il sert. Ensuite, nous comparerons les concepts du modèle EAV aux globales.

Parfois, on dispose d'objets comportant un nombre inconnu de champs, ou peut-être des champs hiérarchiquement imbriqués, pour lesquels, en règle générale, il faut effectuer une recherche.

Par exemple, voici une boutique en ligne avec divers groupes de produits. Chaque groupe de produits a son propre ensemble de propriétés uniques et a également des propriétés communes. Par exemple, les disques SSD et les disques durs ont la propriété commune "capacité", mais tous deux ont également des propriétés uniques, "Endurance, TBW" pour les SSD et "temps moyen de positionnement de la tête" pour les disques durs.

Dans certaines situations, le même produit, fabriqué par différents fabricants, possède des propriétés uniques.

Ainsi, imaginons que nous ayons une boutique en ligne qui vend 50 groupes de marchandises différents. Chaque groupe de produits a ses cinq propriétés uniques, qui peuvent être numériques ou textuelles.

Si nous créons une table dans lequel chaque produit possède 250 propriétés, alors que seules cinq d'entre elles sont réellement utilisées, non seulement nous augmentons considérablement (50 fois !) les exigences en matière

d'espace disque, mais nous réduisons aussi considérablement les caractéristiques de vitesse de la base de données, puisque le cache sera encombré de propriétés inutiles et vides.

Mais ce n'est pas tout. Chaque fois que nous ajoutons une nouvelle famille de produits avec ses propriétés propres, nous devons modifier la structure du tableau à l'aide de la commande **ALTER TABLE**. Sur les tables de grande taille, cette opération peut prendre des heures ou des jours, ce qui est inacceptable pour les entreprises.

"Oui", remarquera le lecteur attentif, "mais nous pouvons utiliser une table différente pour chaque groupe de produits." Bien sûr, vous avez raison, mais cette approche nous donne une base de données avec des dizaines de milliers de tables pour un grand magasin, ce qui est difficile à administrer. De plus, le code, qui doit être pris en charge, devient de plus en plus complexe.

D'autre part, il n'est pas nécessaire de modifier la structure de la base de données lors de l'ajout d'un nouveau groupe de produits. Il suffit d'ajouter une nouvelle table pour un nouveau groupe de produits.

Dans tous les cas, les utilisateurs doivent être capables de rechercher facilement les produits dans un magasin, d'obtenir une table pratique des marchandises indiquant leurs propriétés actuelles et de comparer les produits.

Comme vous pouvez l'imaginer, un formulaire de recherche comportant 250 champs serait extrêmement gênant pour l'utilisateur, tout comme le fait de voir 250 colonnes de propriétés diverses dans la table des produits alors que seulement cinq propriétés pour le groupe sont nécessaires. Il en va de même pour les comparaisons de produits.

Une base de données marketing pourrait également servir comme un autre exemple utile. Pour chaque personne stockée dans la base, de nombreuses propriétés (souvent imbriquées) doivent être ajoutées, modifiées ou supprimées en permanence. Dans le passé, une personne peut avoir acheté quelque chose pour un certain coût, ou avoir acheté certains groupes de produits, avoir participé à un événement, avoir travaillé quelque part, avoir de la famille, vivre dans une certaine ville, appartenir à une certaine classe sociale, et ainsi de suite. Il pourrait y avoir des milliers de champs possibles, en constante évolution. Les spécialistes du marketing réfléchissent sans cesse à la manière de distinguer différents groupes de clients et de leur proposer des offres spéciales convaincantes.

Pour résoudre ces problèmes et disposer en même temps d'une structure de base de données précise et définie, l'approche entité-attribut-valeur a été développée.

Approche EAV

L'essence de l'approche EAV est le stockage séparé des entités, des attributs et des valeurs d'attributs. En général, pour illustrer l'approche EAV, on utilise seulement trois tables, appelés Entité, Attribut et Valeur :

La structure des données de démonstration que nous allons stocker.

Implémentation de l'approche EAV à l'aide de tables

Considérons maintenant un exemple plus complexe utilisant cinq tables (quatre si vous choisissez de consolider les deux derniers tables pour en faire un seul).

La première table est Catalog:

```
CREATE TABLE Catalog (
id INT,
name VARCHAR (128),
parent INT
);
```

Cette table correspond en fait à l'Entité dans l'approche EAV. Elle permettra de stocker les sections du catalogue

hiérarchique des marchandises.

La deuxième table est ****Field :

```
CREATE TABLE Field (
id INT,
name VARCHAR (128),
typeOf INT,
searchable INT,
catalog_id INT,
table_view INT,
sort INT
);
```

Dans cette table, nous spécifions le nom de l'attribut, son type, et si l'attribut est recherchable. Nous indiquons également la section du catalogue qui contient les marchandises auxquelles ces propriétés appartiennent. Tous les produits de la section du catalogue de catalogid ou inférieur peuvent avoir des propriétés différentes qui sont stockées dans cette table.

La troisième table est Good.EIle est conçue pour stocker les marchandises, avec leurs prix, la quantité totale des marchandises, la quantité réservée des marchandises, et le nom des marchandises. En principe, vous n'avez pas vraiment besoin de cette table mais, à mon avis, il est utile d'avoir une table séparée pour les marchandises.

```
CREATE TABLE Good (
id INT,
name VARCHAR (128),
price FLOAT,
item_count INT,
reserved_count,
catalog_id INT
);
```

La quatrième table (TextValues) et la cinquième table (NumberValues) sont conçues pour stocker les valeurs du texte et les attributs numériques des marchandises, et elles ont une structure similaire.

```
CREATE TABLE TextValues ??(
good_id INT,
field_id INT,
fValue TEXT
);

CREATE TABLE NumberValues ??(
good_id INT,
field_id INT,
fValue INT
);
```

Au lieu des tables de valeurs textuelles et numériques, vous pouvez utiliser une seule table CustomValues avec une structure de ce type :

```
CREATE TABLE CustomValues ??(
good_id INT,
```

```
field_id INT,
text_value TEXT,
number_value INT
);
```

Je préfère stocker les différents types de données séparément car cela augmente la vitesse et économise de l'espace.

Accès aux données à l'aide de l'approche EAV

Commençons par afficher le mappage de la structure du catalogue à l'aide de SQL :

```
SELECT * FROM Catalog ORDER BY id;
```

Afin de former un arbre à partir de ces valeurs, un code distinct est nécessaire. En PHP, cela ressemblerait à quelque chose comme ceci :

```
$stmt = $ pdo-> query ('SELECT * FROM Catalog ORDER BY id');
$aTree = [];
$idRoot = NULL;

while ($row = $ stmt->fetch())
{
    $aTree [$row ['id']] = ['name' => $ row ['name']];

    if (! $row['parent'])
      $idRoot = $row ['id'];
    else
      $aTree [$row['parent']] ['sub'] [] = $row['id'];
}
```

À l'avenir, nous pourrons simplement dessiner l'arbre si nous partons du nœud racine $aTree[$ idRoot].

Maintenant, nous allons obtenir les propriétés d'un produit spécifique.

Tout d'abord, nous allons obtenir une liste de propriétés spécifiques à ce produit, puis y attacher les propriétés qui sont dans la base de données. Dans la vie réelle, toutes les propriétés indiquées ne sont pas renseignées et nous sommes donc obligés d'utiliser LEFT JOIN :

```
SELECT * FROM
(
SELECT g. *, F.name, f.type_of, val.fValue, f.sort FROM Good as g
INNER JOIN Field as f ON f.catalog_id = g.catalog_id
LEFT JOIN TextValues ??as val ON tv.good = g.id AND f.id = val.field_id
WHERE g.id = $ nGood AND f.type_of = 'text'
UNION
SELECT g. *, F.name, f.type_of, val.fValue, f.sort FROM Good as g
INNER JOIN Field as f ON f.catalog_id = g.catalog_id
LEFT JOIN NumberValues ??as val ON val.good = g.id AND f.id = val.field_id
WHERE g.id = $nGood AND f.type_of = 'number'
) t
ORDER BY t.sort;
```

Si nous utilisons une seule table pour stocker les valeurs numériques et textuelles, la requête est considérablement simplifiée :

```
SELECT g. *, F.name, f.type_of, val.text_value, val.number_value, f.sort FROM Good as
 g
INNER JOIN Field as f ON f.catalog = g.catalog
LEFT JOIN CustomValues ??as val ON tv.good = g.id AND f.id = val.field_id
WHERE g.id = $nGood
ORDER BY f.sort;
```

Maintenant, nous allons obtenir les produits sous la forme de table contenue dans la section du catalogue $nCatalog. Tout d'abord, nous obtenons une liste de propriétés qui doivent être reflétées dans la vue de la table pour cette section du catalogue :

```
SELECT f.id, f.name, f.type_of FROM Catalog as c
INNER JOIN Field as f ON f.catalog_id = c.id
WHERE c.id = $nCatalog AND f.table_view = 1
ORDER BY f.sort;
```

Ensuite, nous construisons la requête pour créer la table. Supposons que pour une vue tabulaire, nous ayons besoin de trois propriétés supplémentaires (sans compter celles de la table Good). Pour simplifier la requête, nous supposons que :

```
SELECT g.if, g.name, g.price,
            f1.fValue as f1_val,
            f2.fValue as f2_val,
            f3.fValue as f3_val,
FROM Good
LEFT JOIN TextValue as f1 ON f1.good_id = g.id
LEFT JOIN NumberValue as f2 ON f2.good_id = g.id
LEFT JOIN NumberValue as f3 ON f3.good_id = g.id
WHERE g.catalog_id = $nCatalog;
```

Les avantages et les inconvénients de l'approche EAV

L'avantage évident de l'approche EAV est sa flexibilité. Avec des structures de données fixes telles que les tables, nous pouvons nous permettre de stocker une grande variété d'ensembles de propriétés pour les objets. Et nous pouvons stocker différentes structures de données sans modifier le schéma de la base de données.

Nous pouvons également utiliser SQL, qui est familier à un grand nombre de développeurs.

Le défaut le plus évident est l'inadéquation entre la structure logique des données et leur stockage physique, qui entraîne diverses difficultés.

En outre, la programmation implique souvent des requêtes SQL très complexes. Le débogage peut être difficile car vous devez créer des outils non-standards pour visualiser les données EAV. Enfin, vous pouvez être amené à utiliser des requêtes LEFT JOIN, qui ralentissent la base de données.

Globales : Une alternative à EAV

Comme je suis familier à la fois du monde SQL et du monde des globales, j'ai eu l'idée que l'utilisation des globales pour les tâches résolues par l'approche EAV serait beaucoup plus intéressante.

Les globales sont des structures de données qui vous permettent de stocker des informations dispersées et hiérarchiques. Un point très important est que les globales sont soigneusement optimisées pour le stockage d'informations hiérarchiques. Les globales sont elles-mêmes des structures de niveau inférieur aux tables, ce qui leur permet de travailler beaucoup plus rapidement que ces derniers.

Dans le même temps, la structure de globale elle-même peut être sélectionnée en fonction de la structure des données, ce qui rend le code très simple et clair.

Structure de globale pour le stockage des données démographiques

Une globale représente une structure tellement flexible et élégante pour le stockage des données que nous pourrions nous débrouiller avec une seule globale pour le stockage des données dans les sections du catalogue, les propriétés et les produits, par exemple, de la manière suivante :

Remarquez à quel point la structure de globale est similaire à la structure de données. Cette conformité simplifie grandement le codage et le débogage.

En pratique, il est préférable d'utiliser plusieurs globales, bien que la tentation de stocker toutes les informations dans une seule globale soit assez forte. Il est judicieux de créer des globales distinctes pour les indices. Vous pouvez également séparer le stockage de la structure de la partition du répertoire des marchandises.

Quelle est la suite ?

Dans le deuxième article de cette série, nous aborderons les détails et les avantages du stockage des données dans des globales InterSystems Iris au lieu de suivre le modèle EAV.

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