文章
姚 鑫 · 六月 20, 2022 阅读大约需 4 分钟

第五章 操作位和位串

第五章 操作位和位串

有时可能希望在基于数据平台的应用程序中存储一系列相关的布尔值。可以创建许多布尔变量，也可以将它们存储在数组或列表中。或者可以使用称为 “ 位串 ” 的概念，它可以定义为位序列，首先呈现最低有效位。位串允许您以非常有效的方式存储此类数据，无论是在存储空间还是处理速度方面。

位串可以以两种方式之一存储，作为压缩字符串或整数。如果在没有上下文的情况下听到术语 “ 位串 ”，则表示位序列存储为压缩字符串。本文向介绍了这两种类型的位串，然后介绍了一些可用于操作它们的技术。

将位序列存储为位串

存储位序列的最常见方式是在位串中，这是一种特殊的压缩字符串。除了节省存储空间外，还可以使用 ObjectScript 系统函数有效地操作位串。

这样的系统函数是 $factor，它将整数转换为位串。我们可以通过执行以下语句将整数 11744 转换为位串：

```
set bitstring = $factor(11744)
```

要查看位串内容的表示，可以使用 zwrite 命令：

```
zwrite bitstring
bitstring=$zwc(128,4)_$c(224,45,0,0)/*$bit(6..9,11,12,14)*/
```

起初它看起来很神秘，但在输出的末尾，会看到一条注释，其中显示了已设置的实际位的列表：6、7、8、9、11、12 和 14。位串中的位 1表示 2^0，位 2 表示 2^1，依此类推。将所有位加在一起，我们得到 2^5 +2^6 + 2^7+ 2^8 + 2^10 + 2^11 + 2^13 = 11744。

要获得更令人愉悦的视觉表示，可以使用另一个系统函数 $bit：

```
for i=1:1:14 {write $bit(bitstring, i)}
00000111101101
```

在此示例中，$bit(bitstring, i) 返回位串的位 i 的值。

注意：要更深入地了解此位序列是如何在内部存储的，请仔细查看 zwrite 命令的输出：

```
bitstring=$zwc(128,4)_$c(224,45,0,0)/*$bit(6..9,11,12,14)*/
```

该位串存储在四个块中，每个块 8 位。分别计算每个块会给你 224、45、0、0。

如果它有助于将位串视为一个字符串，可以将每个块视为一个 8 位字符。

位串的一个常见应用是位图索引的存储。位图索引是一种特殊类型的索引，它使用一系列位串来表示对应于特定属性的给定值的对象集。位图中的每个位代表类中的一个对象。

例如，假设创建一个动物及其特征的数据库，并按如下方式定义一个类：

```
Class User.Animal Extends %RegisteredObject
{

Property Name As %String [ Required ];

Property Classification As %String;

Property Diet As %String;

Property Swims As %Boolean;

Index ClassificationIDX On Classification [ Type = bitmap ];

Index DietIDX On Diet [ Type = bitmap ];

Index SwimsIDX On Swims [ Type = bitmap ];

}
```

用一些动物填充数据库后，它可能如下所示：

ID	Name	Classification	Diet	Swims
1	Penguin	Bird	Carnivore	1
2	Giraffe	Mammal	Herbivore	0
3	Cheetah	Mammal	Carnivore	0
4	Bullfrog	Amphibian	Carnivore	1
5	Shark	Fish	Carnivore	1
6	Fruit Bat	Mammal	Herbivore	0
7	Snapping Turtle	Reptile	Carnivore	1
8	Manatee	Mammal	Herbivore	1
9	Ant	Insect	Omnivore	0
10	Rattlesnake	Reptile	Carnivore	0

位图索引 DietIDX 跟踪具有特定饮食属性值的动物。索引的每一行可以存储一个代表多达 64,000 只动物的块。

Diet属性的位图索引

Diet	Chunk	Bitmap
Carnivore	1	1011101001
Herbivore	1	0100010100
Omnivore	1	0000000010

在内部，位图索引存储在全局 ^User.AnimalI 的以下节点中：

```
^User.AnimalI("DietIDX"," CARNIVORE",1)
^User.AnimalI("DietIDX"," HERBIVORE",1)
^User.AnimalI("DietIDX"," OMNIVORE",1)
```

第一个下标是索引的名称
(DietIDX)，第二个下标是被索引的属性的值（例如，CARNIVORE），第三个下标是块编号（在本例中为 1）。

同样，位图索引 SwimsIDX 跟踪具有特定 Swims 属性值的动物。

Swims 属性的位图索引

Swims	Chunk	Bitmap
True	1	1001101100
False	1	0110010011

此位图索引存储在 ^User.AnimalI 的以下节点中：

```
^User.AnimalI("SwimsIDX",1,1)
^User.AnimalI("SwimsIDX",0,1)
```

为了了解位串的威力，可以通过计算位图中的CARNIVORE食肉动物数量非常轻松地计算数据库中的食肉动物数量，而无需检查实际数据。只需使用系统函数 $bitcount：

```
set c = ^User.AnimalI("DietIDX"," CARNIVORE",1)

write $bitcount(c,1)
6
```

同样，可以计算 swim 动物数量：

```
set s = ^User.AnimalI("SwimsIDX",1,1)

write $bitcount(s,1)
5
```

要计算游动的食肉动物的数量，请使用 $bitlogic 函数查找两组的交集：

```
set cs = $bitlogic(c&s)

write $bitcount(cs,1)
4
```

注意：再次使用 zwrite 检查肉食动物的位图是如何在内部存储的：

```
zwrite ^User.AnimalI("DietIDX"," CARNIVORE",1)
^User.AnimalI("DietIDX"," CARNIVORE",1)=$zwc(413,2,0,2,6,8,9)/*$bit(2,4..6,8,11)*/
```

在这里，可以看到饮食属性为 CARNIVORE 的所有动物对应的位。如所知，位图索引被分成 64,000 位的块。为具有给定 ID 的动物存储的位存储在块 (ID\64000) + 1，位置 (ID#64000) + 1 中。因此，表示具有 ID 1 的动物的位存储在块 1，位置 2 中。所以，在这个位串中，位 2 代表企鹅，而不是长颈鹿。

SQL 引擎包括许多可以利用位图索引的特殊优化，因此可以在编写 SQL 查询时获得好处。

#SQL #Caché

源

URL:
https://cn.community.intersystems.com/post/%E7%AC%AC%E4%BA%94%E7%AB%A0-%E6%93%8D%E4%BD%9C%E4%BD%8D%E5%92%8C%E4%BD%8D%E4%B8%B2