文章
Weiwei Gu · 七月 12 阅读大约需 9 分钟

Globals - 存储数据的魔剑-树：第二部分

开始 - 请拉到页面底部查看该系列文章第一部分 .

3. 使用globals时结构的变体

一个结构，比如说一个有序排列的“ 树 ”，有各种特殊的情况。让我们来看看那些对使用globals有实际价值的情况。

3.1 特殊情况1. 一个没有分支的节点

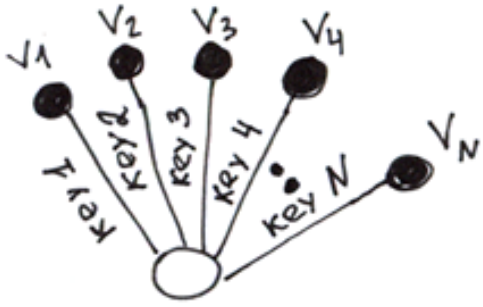
Globals不仅可以像数组一样使用，而且可以像普通变量一样使用。例如，用于创建一个计数器:

```
Set ^counter = 0  ; setting counter
Set id=$Increment(^counter) ;  atomic incrementation
```

同时，一个global除了值以外，还可以有分支。一个并不排斥另一个。

3.2 特殊情况2. 一个节点和多个分支

事实上，这是一个典型的键值库。而如果我们把健和值都存下来而不是仅仅是存值的话，那我们会得到一个有主键的普通表。



Key-Value DB on the Global

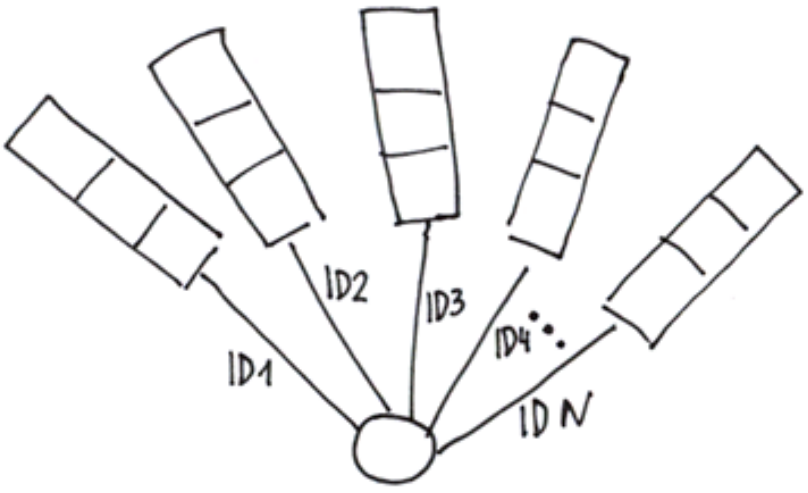


Table on the Global.
Storing row as Value.

为了实现一个基于globals的表，我们将不得不从列值中形成字符串，然后通过主键将它们保存到global中。为了能够在读取过程中把字符串分割成列，我们可以使用以下方法。

1. 分隔符

```
Set ^t(id1) = "col11/col21/col31"
Set ^t(id2) = "col12/col22/col32"
```

2. 一个固定的方案，即每个字段占据特定数量的字节。在关系型数据库中通常就是这样做的。
   一个特殊的$LB
3. 一个特殊的 $LB函数（从Caché开始引入的），可以从值中组成一个字符串。

```
Set ^t(id1) = $LB("col11", "col21", "col31")
Set ^t(id2) = $LB("col12", "col22", "col32")
```

有趣的是，使用globals做一些类似于关系型数据库中外键的事情并不难。我们把这种结构称为index globals。Index globals是一个补充"树"，用于快速搜索那些不属于主Global主键组成部分的字段。你需要编写额外的代码来填充和使用它。

下面，让我们在第一列的基础上创建一个Index global.

```
Set ^i("col11", id1) = 1
Set ^i("col12", id2) = 1
```

要想通过第一列快速搜索，你需要查看^i global，并找到与第一列中必要值对应的主键（id）。

当插入一个值时，我们可以同时为必要的字段创建值和Index
global。为了保证可靠性，让我们把它包装成一个事务(transaction)。

```
TSTART
Set ^t(id1) = $LB("col11", "col21", "col31")
Set ^i("col11", id1) = 1
TCOMMIT
```

更多的信息可以从这里查看 making tables in M using globals and emulation of secondary keys.

如果用COS/M编写插入/更新/删除函数并进行编译，这些表的工作速度将与传统DB一样快（甚至更快）。

我通过对一个单一的双列表进行大量的INSERT和SELECT操作，同时使用TSTART和TCOMMIT命令（transactions事务）来验证这个声明。

我没有测试更复杂的并发访问和并行事务的情况。

在不使用transactions事务的情况下，一百万个值的插入速度为778,361次/秒。

对于3亿个值，速度是422,141次/秒。

当使用transactions交易时，对于5000万个值，速度达到572,082次插入/秒。所有的操作都是通过编译的M代码运行的。我使用了普通的硬盘，而不是SSD。RAID5有回写功能。所有运行在Phenom II 1100T CPU上。

为了对SQL数据库进行同样的测试，我们需要写一个存储过程，在一个循环中进行插入。当使用同样的方法测试MySQL 5.5（InnoDB存储）时，我从来没有得到超过每秒11K次的插入。

确实，用globals实现表比在关系型数据库中做同样的事情要复杂。这就是为什么基于globals的工业数据库会有SQL访问，以来简化表格数据的工作。

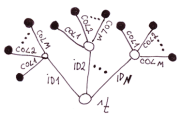
一般来说，如果数据模式不会经常改变，插入的速度不是很关键，而且整个数据库可以很容易地用规范化的表来表示，那么使用SQL就比较容易，因为它提供了一个更高的抽象层次。

在这种情况下，我想表明globals可以被用作创建其他DB的构造函数。就像汇编语言可以用来创建其他语言一样。而这里有一些使用globals来创建对应的 键值key-values, 列表lists, 集合sets, 表格-tabular, 文档数据库-document-oriented DB 的例子。

如果你需要以最小的努力创建一个非标准的数据库，你应该考虑使用globals。

3.3 特殊情况 3.一个有两个层级的“ 树 ”，每个二级节点都有固定数量的分支

你可能已经猜到了：这是一个使用globals的表格的可选实现形式。我们把它与之前的那个进行比较。



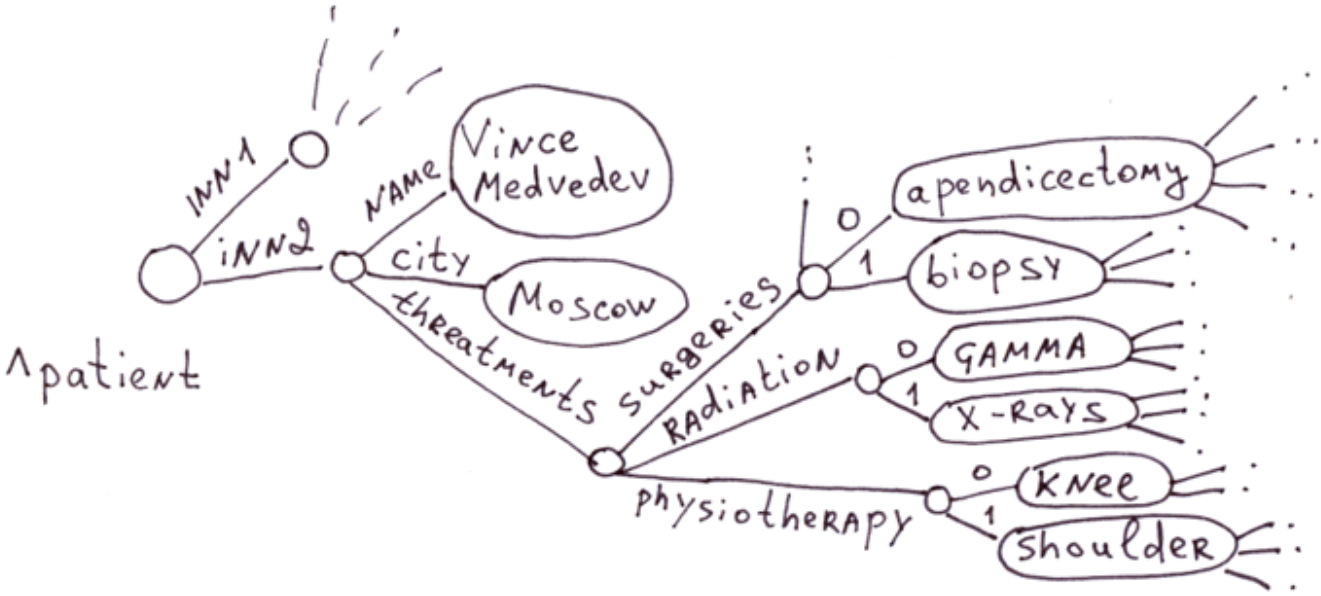
两层树中的表 VS .一层树中的表	
缺点	优点
1.插入速度慢，因为节点的数量必须设置为与列的数量相等。 2 更高的硬盘空间消耗，因为带有列名的全局索引（如数组索引）占用了硬盘空间，并且每一行都是重复的。	1.对特定列的值的访问速度更快，因为你不需要解析字符串。根据我的测试，对于2个列来说，它的速度要快11.5%，对于更多的列来说，速度甚至更快。 2. 更容易改变数据模式 3. 更容易阅读代码

结论:
没什么可写的。由于性能是globals的关键优势之一，使用这种方法几乎没有任何意义，因为它不可能比关系型数据库中的普通表工作得更快。

3.4 一般情况。"树"和有序键

任何可以被表示为"树"的数据结构都能完美地适合globals。

3.4.1 有子对象的对象



这就是传统上使用 globals 的领域。在医疗领域有无数的疾病、药物、症状和治疗方法。为每个病人创建一个有一百万个字段的表是不合理的，尤其是99%的字段都是空白的。

想象一下，一个由以下表格组成的SQL数据库。"病人"10万个字段，"药物 "10万个字段，"治疗 "10万个字段，"并发症 "10万个字段，等等。作为一个替代方案，你可以创建一个有数千个表的数据库，每个表都代表一个特定的病人类型（它们也可以重叠！）、治疗、药物，以及这些表之间关系的数千个表。

Globals就像一只手套一样适合医疗行业，因为它使每个病人都有完整的病例记录、治疗方法列表、使用的药物及其效果--所有这些都以"树"的形式存在，而不会像关系型数据库那样在空的列上浪费太多的磁盘空间。

当任务是最大限度地积累和系统化关于客户的各种个人数据时，**Globals用于记录个人各种细节的数据库非常有效**。这对于医疗、银行、营销、档案和其他领域来说尤其重要。

不言而喻，SQL也能让你只用几个表(EAV, 1,2,3,4,5,6, 7,8)来模拟一棵树， 但它要复杂得多，工作速度也慢。从本质上讲，我们必须写一个基于表的Global，并将所有与表有关的routines隐藏在一个抽象层下。用高层技术（SQL）来模拟底层技术（globals）是不正确的

改变巨大的表的数据模式（ALTER TABLE）可能需要相当长的时间，这并不是什么秘密。例如，MySQL在执行ALTER TABLE ADD|DROP COLUMN操作时，会将所有数据从旧表复制到新表（我在MyISAM和InnoDB上测试过）。这可能会使一个有数十亿条记录的生产数据库停滞几天，甚至几周。

如果我们使用globals，改变数据结构对我们来说是没有成本的。我们可以在任何时候向层次结构中任何级别的任何对象添加任何新的属性。需要对分支进行重命名的改变可以在后台模式下应用，同时数据库也会启动并运行。

因此，当涉及到存储具有大量可选属性的对象时，globals工作得非常好！

我也提醒一下各位，对任何一个属性的访问都是即时的，因为在global中，所有的路径都是一个B-tree。

在一般情况下，基于globals的数据库也是一种面向文档的数据库，支持存储分层信息。因此，在存储医疗卡的领域，面向文档的数据库可以有效地与globals竞争。

但是，现在还不完全是这样。

让我们以MongoDB为例。在这个领域，它输给了globals，原因如下：

1.文档大小

存储单元是一个JSON格式的文本（确切地说，是BSON），最大尺寸为16MB左右。引入这个限制的目的是为了确保JSON数据库在解析过程中不会变得太慢，当一个巨大的JSON文档被保存到数据库中时，需要处理特定的字段值。这个文件应该有关于病人的完整信息。我们都知道病人卡可以有多“厚”。如果卡的最大大小被限制在16MB，它就会立即过滤掉卡中包含核磁共振扫描、X光扫描和和其他材料的病人。Global的一个分支可以有数千兆字节和数万兆字节的数据。这算是说明了一切，但我还可以告诉你更多。

2. 创建/改变/删除病人卡上的新属性所需的时间

这样一个数据库需要将整个卡片复制到内存中（大量的数据！），解析BSON数据，添加/改变/删除新的节点，更新索引，将其全部打包回BSON并保存到磁盘。而一个Global只需要寻址必要的属性并执行必要的操作。

3.对特定属性的访问速度

如果文档有许多属性和多级结构，对特定属性的访问会更快，因为Global中的每个路径都是B-Tree。在BSON中，你需要对文档进行线性解析以找到必要的属性。

3.3.2 关联数组

关联数组（即使是嵌套数组）可以完美地与globals一起工作。例如，这个PHP数组将看起来像3.3.1中的第一个插图。

```
$a = array(
  "name" => "Vince Medvedev",
  "city" => "Moscow",
  "threatments" => array(
    "surgeries" => array("apedicectomy", "biopsy"),
    "radiation" => array("gamma", "x-rays"),
    "physiotherapy" => array("knee", "shoulder")
```

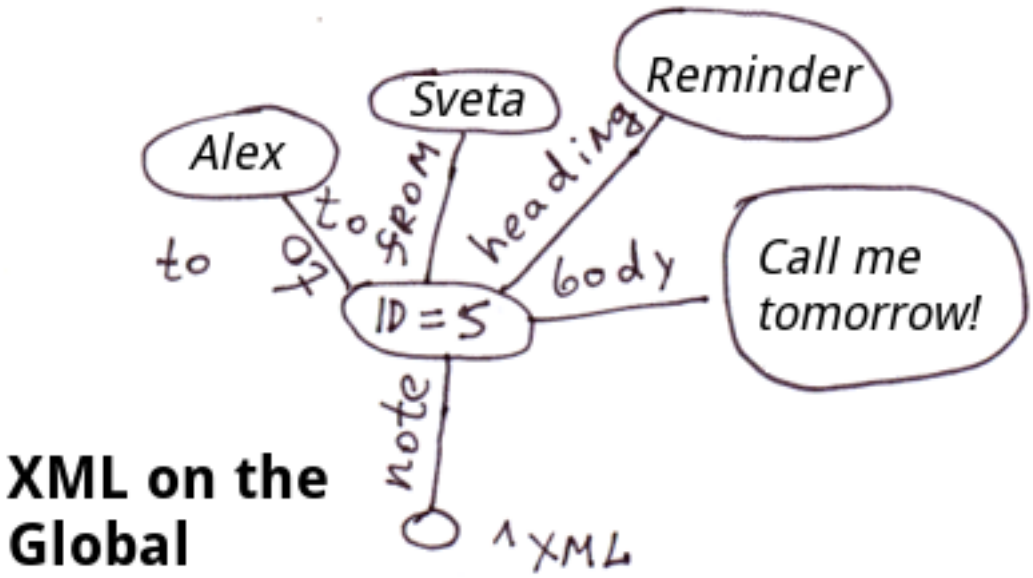
```
  )
);
```

3.3.3 层次化的文件。XML、JSON

也可以很容易地存储在globals中并以不同的方式进行分解。

XML

将XML分解成globals的最简单方法是将标签属性存储在节点中。而如果你需要快速访问标签属性，我们可以把它们放在单独的分支中。

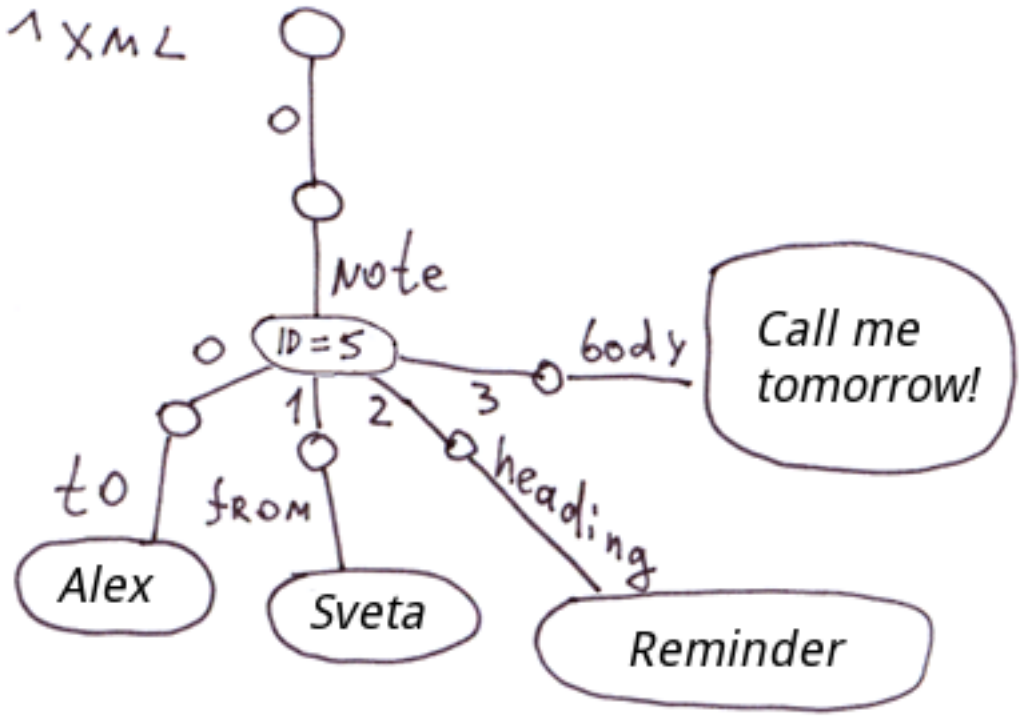



```
<note id=5>
<to>Alex</to>
<from>Sveta</from>
<heading>Reminder</heading>
<body>Call me tomorrow!</body>
</note>
```

在COS中，代码将看起来像这样。

```
Set ^xml("note")="id=5"
Set ^xml("note","to")="Alex"
Set ^xml("note","from")="Sveta"
Set ^xml("note","heading")="Reminder"
Set ^xml("note","body")="Call me tomorrow!"
```

注意
：对于XML、JSON和关联数组，你可以想出很多方法来在globals中显示它们。在这个特殊的例子中，我们没有在"note "标签中反映嵌套标签的顺序。在^xml
global中，嵌套标签将按字母顺序显示。为了精确地显示顺序，你可以使用下面的模式，比如:



JSON

这个JSON文档的内容显示在第3.3.1节的第一个插图中

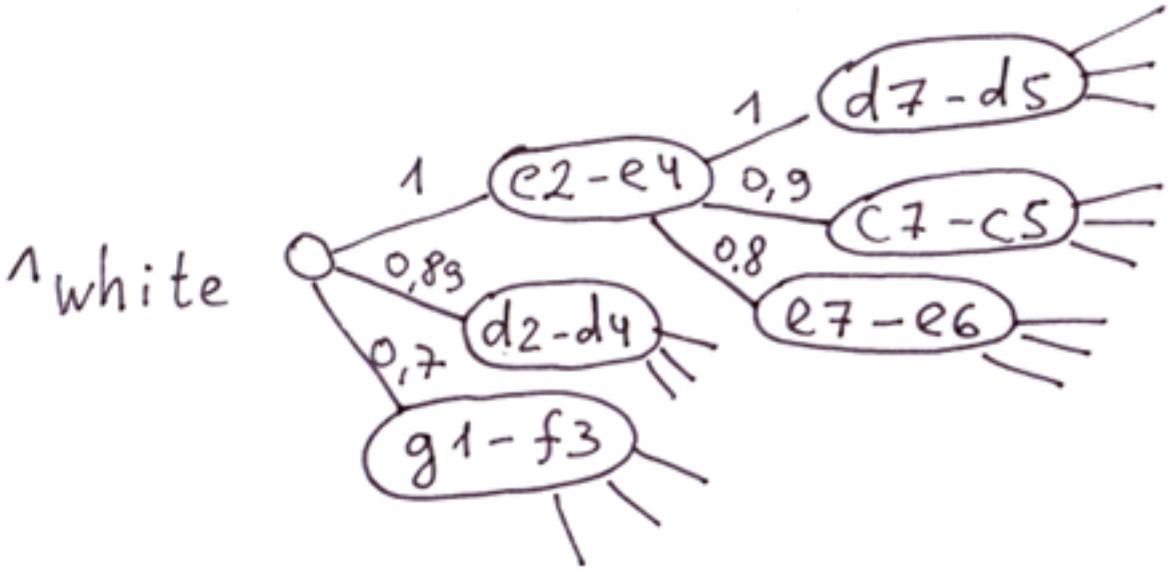
```
var document = {
  "name": "Vince Medvedev",
  "city": "Moscow",
  "threatments": {
    "surgeries": ["apedicectomy", "biopsy"],
    "radiation": ["gamma", "x-rays"],
    "physiotherapy": ["knee", "shoulder"]
  },
};
```

3.3.4 由等级关系约束的相同结构

例子：销售办公室的结构组成，传销组织结构中人的位置，国际象棋的首秀。

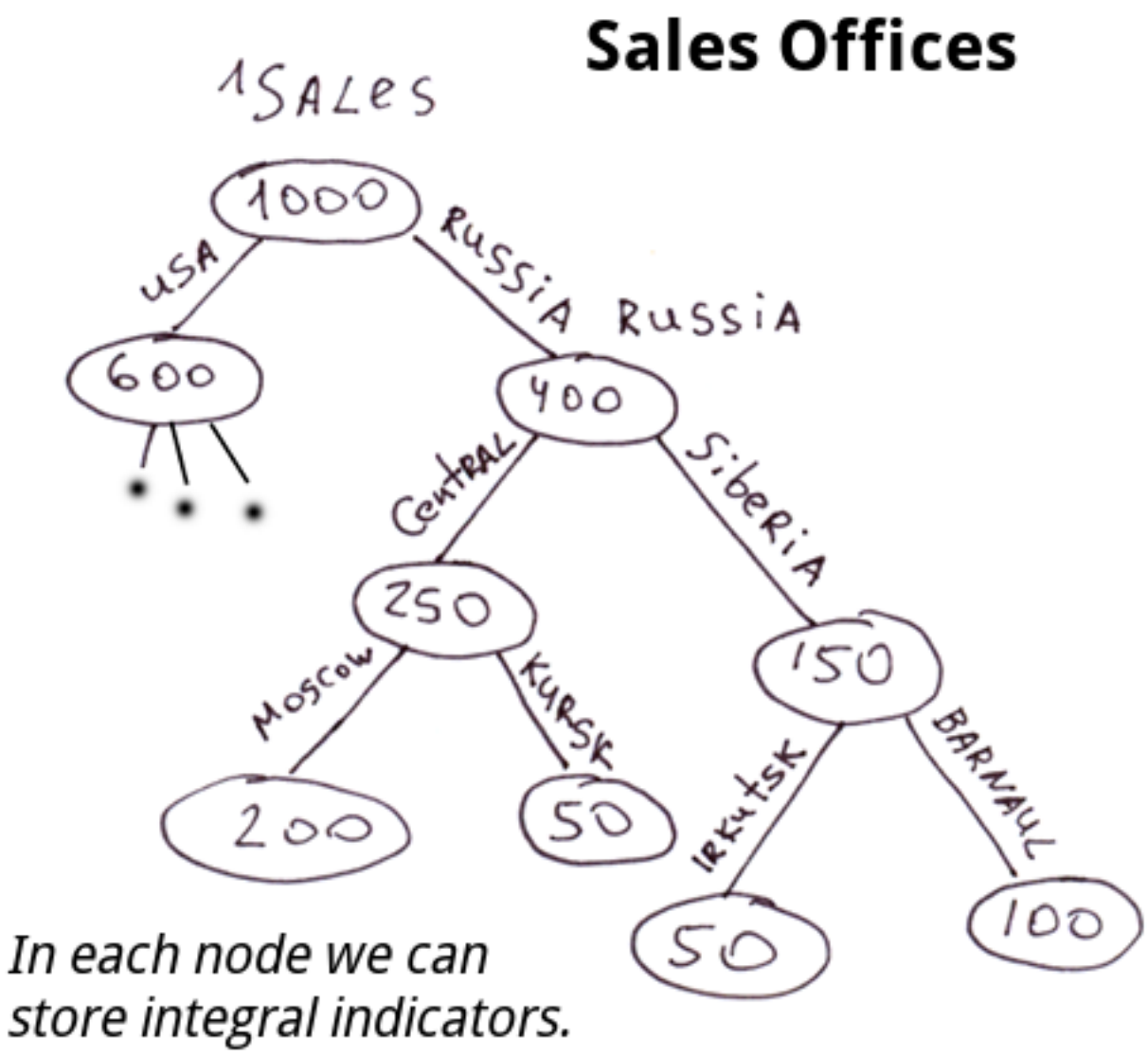
关于首秀的数据库。

你可以使用棋力评估作为Global的节点索引的值。在这种情况下，你需要选择一个具有最高权重的分支来确定最佳棋步。在Global中，每一层的所有分支都将按棋力进行排序。



An example of debuts DB on the Global. Select the branch with the highest weight and make a move.

销售办公室的结构，传销公司的人。
节点可以存储一些反映整个子树特征的缓存值。例如，这个特定子树的销售人员情况。我们可以在任何时候获得关于任何分支的销售成果的确切信息。



4. 使用globals有好处的情况

第一栏包含了使用globals会在性能方面给你带来相当大的优势的情况列表，第二栏则包含了使用globals会简化开发或数据模型的情况列表。

Speed	数据处理/呈现的便利性
1. 插入[每层都有自动排序]，[通过主键建立索引]。 2. 移除子树 3. 具有大量嵌套属性的对象，你需要对其进行单独访问 4. 分层结构，可以从任何一个分支开始，甚至是不存在的分支，进行子分支的遍历。 5.深入的树形遍历	1.具有大量非必需[和/或嵌套]属性/物质的对象/物质 2.无模式的数据--经常可以添加新的属性和删除旧的属性。 3.你需要创建一个非标准的DB。 4.路径数据库和解决方案树。当路径可以方便地表示为一棵"树 " 的时候。 5.在不使用递归的情况下删除层次结构

下一章继续第三篇，未完待续！（待翻译） "Globals - Magic swords for Storing Data. Sparse Arrays. Part 3"

Disclaimer: *this article and my comments on it reflect my opinion only and have nothing to do with the official position of the InterSystems Corporation.*

#Node.js #Globals #关系表 #性能 #数据模型 #新手 #Caché #InterSystems IRIS

源URL: https://cn.community.intersystems.com/post/globals-%E5%AD%98%E5%82%A8%E6%95%B0%E6%8D%AE%E7%9A%84%E9%AD%94%E5%89%91-%E6%A0%91%EF%BC%9A%E7%AC%AC%E4%BA%8C%E9%83%A8%E5%88%86