Artículo
Minoru Horita · 2 jun, 2020 Lectura de 14 min

グローバルはデータを保存するための魔法の剣ですパート2 - ツリー

最初の記事については、パート1を参照してください。

3. グローバルを使用する場合のさまざまな構造

順序付きツリーなどの構造には、さまざまな特殊ケースがあります。
グローバルを使用する上で実用的な価値があるものを見てみましょう。

3.1 特殊ケース1 - 枝のない1つのノード

グローバルは配列のようにも、通常の変数のようにも使用できます。
例えば、カウンターを作成する場合を考えてみましょう。

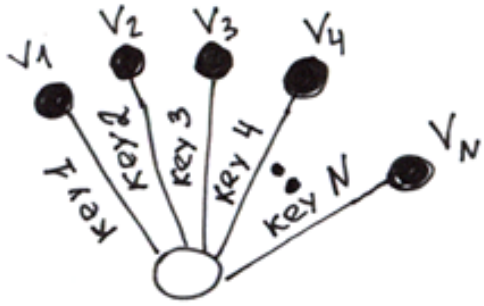
```
Set ^counter = 0  ; ????????

Set id=$Increment(^counter) ;  ??????????????
```

また、グローバルには値に加えて枝を持たせることができます。 一方が他方を除外することはありません。

3.2 特殊ケース2 - 1つのノードと複数の枝

実際、これは典型的なキー・バリューベースのデータ構造です。
また、値の代わりに値のタプルを保存すると、主キーを持つ通常のテーブルが得られます。



Key-Value DB on the Global

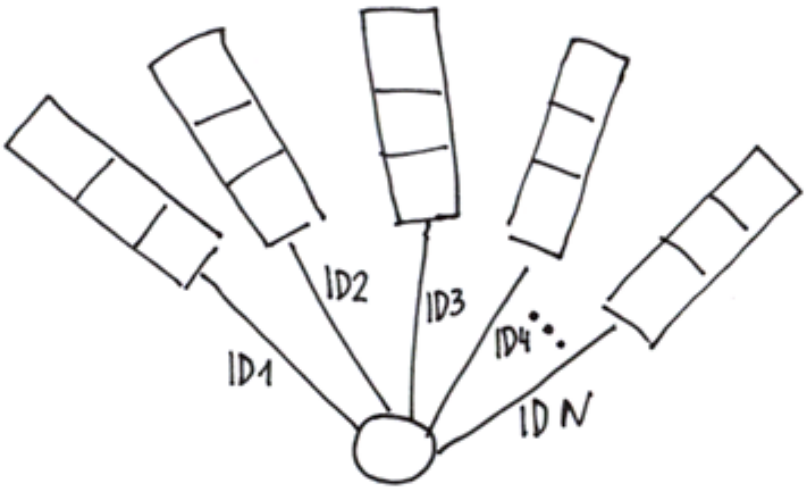


Table on the Global.
Storing row as Value.

グローバルに基づくテーブルを実装するには、カラムの値から文字列を作成し、主キー別にそれをグローバルに保存する必要があります。
読み取りの時に文字列をカラムに分割できるようにするため、以下のいずれかを使用することができます。

1. 区切り文字

```
Set ^t(id1) = "col11/col21/col31"

Set ^t(id2) = "col12/col22/col32"
```

2. 各フィールドが特定のバイト数を占める、固定のスキーマ。
   これは、リレーショナルデータベースで通常行われる方法です。

3. 値から文字列を作る専用の $LB 関数（Cachéで導入されます）。

```
Set ^t(id1) = $LB("col11", "col21", "col31")

Set ^t(id2) = $LB("col12", "col22", "col32")
```

興味深いことに、グローバルを使用すればリレーショナルDBの外部キーと同様のことを行うのは難しくありません。 このような構造をインデックスグローバルと呼びましょう。 インデックスグローバルは、メイングローバルの主キーを構成しないフィールドですばやく検索するための補助的なツリーです。
インデックスグローバルにデータを入れて使用するには、追加のコードを記述する必要があります。

最初のカラムに基づいてグローバルインデックスを作成しましょう。

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

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

最初のカラムですばやく検索するには、^i
グローバルを調べ、最初のカラムで必要な値に対応する主キー（id）を見つける必要があります。

値を挿入する際には、必要なフィールドの値とインデックスグローバルの両方を作成できます。
信頼性を確保するため、トランザクションで囲みましょう。

```
TSTART

Set ^t(id1) = $LB("col11", "col21", "col31")

Set ^i("col11", id1) = 1

TCOMMIT
```

詳細については、グローバルとセカンダリキーのエミュレーションを使用してMでテーブルを作成する方法
をご覧ください。

挿入/更新/削除関数がCOS/Mで記述されてコンパイルされている場合、これらのテーブルは従来のDBと同じくらい高速に（またはさらに高速に）機能します。

単一の2カラム構成のテーブルにINSERTおよびSELECT操作の大部分を適用し、TSTARTおよびTCOMMITコマンド（トランザクション）も併用することにより、このことをを検証しました。 同時アクセスと並列トランザクションが発生する、

より複雑なシナリオはテストしていません。
トランザクションを使用しない場合、100万件の値を挿入したときの速度は毎秒778,361レコードでした。

3億件の場合、挿入速度は毎秒422,141レコードでした。

トランザクションが使用された場合、挿入速度は5,000万件の値で毎秒572,082レコードでした。
すべての操作はコンパイル済みのMコードから実行されました。
SSDではなく、通常のハードドライブを使用しました。
ライトバックキャッシュ付きのRAID5を構成していました。 すべての処理は、Phenom II 1100T CPUで実行されました。 SQLデータベースに対して同じテストを実行するには、ループで挿入を行うストアドプロシージャを作成する必要があります。

同じ方法を使用してMySQL 5.5（InnoDBストレージ）をテストしたときは、1秒あたりの挿入件数が11,000を超えることはありませんでした。

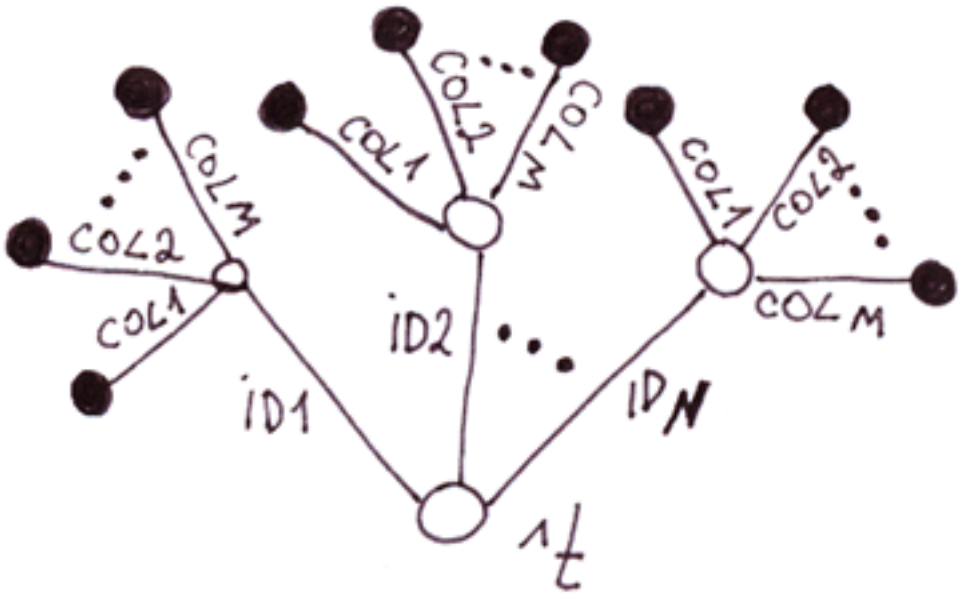
そうです。グローバルを使ったテーブルの実装は、リレーショナルデータベースで同じ実装をするよりも複雑です。 そのため、グローバルに基づく実務用DBはSQLデータにアクセスし、表形式データでの作業を簡略化しているのです。

一般的に、データスキーマが頻繁に変更されない場合、挿入速度は重要ではなく、データベース全体を正規化されたテーブルで簡単に表現することができます。これにより、より高度な抽象化が行われるため、SQLでの作業が容易になります。

このケースでは、グローバルを他の種類のDBを作成するための構成要素として使用できることを示したいと思いました。 これは、他の言語の作成に使用できるアセンブリ言語と同じです。
また、グローバルを使用してキー/値、リスト、セット、表形式、ドキュメント指向のDB
に相当するものを作成する例がいくつか存在します。

最小限の労力で標準的ではないDBを作成する必要がある場合は、グローバルの使用を検討すると良いでしょう。

3.3 特殊ケース3 - 第2階層に枝の数が固定されたノードを持つ2階層のツリー



ご想像のとおり、これはグローバルを使用したテーブルの代わりに実装したものです。
以前のものと比較してみましょう。

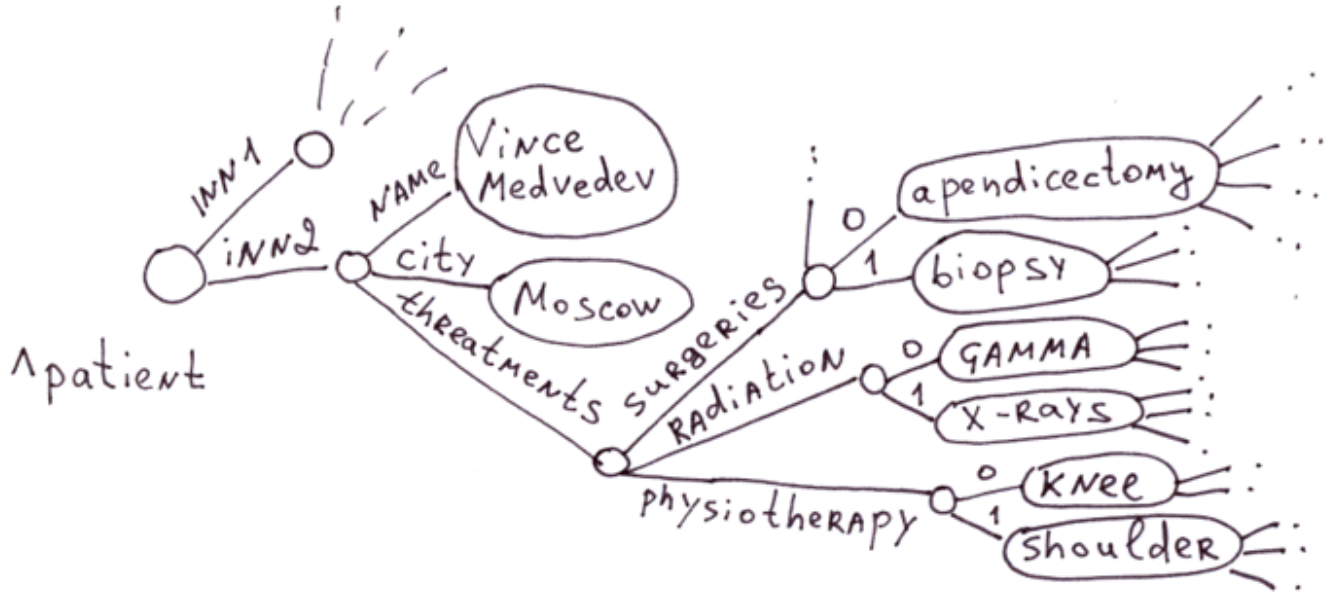
2階層ツリー内のテーブルと 1階層ツリー内のテーブルの比較。	
短所	長所
1. ノード数とカラム数が等しくなるように設定する必要があるため、挿入が遅くなります。 2. カラム名を含むグローバルインデックス（配列インデックスなど）がハードドライブの容量を占有し、各レコードで複製されるため、ハードドライブの容量の消費が多くなります。	1. 文字列を解析する必要がないため、特定カラムの値に高速にアクセスできます。 私がテストした結果によれば、2カラムの場合は11.5%高速で、カラム数が増えるとさらに高速になりました。 2. データスキーマの変更が簡単 3. コードが読みやすい

結論： 特筆すべきことはありません。 パフォーマンスはグローバルの主なメリットの1つであるため、このアプローチを使用しても実質的には意味がありません。リレーショナルデータベースの通常のテーブルよりも高速に動作することはほとんどないからです。

3.4 一般的なケース - ツリーと順序キー

ツリーとして表現できるすべてのデータ構造は、完全にグローバルに適合します。

3.4.1 サブオブジェクトを持つオブジェクト



これは、グローバルが昔から使用されている分野です。
医療分野には数多くの病気、薬剤、症状、治療法があります。 これらのフィールドの99%は空になるため、患者ごとに100万フィールドのテーブルを作成するのは不合理です。

「患者」～100,000フィールド、「投薬」100,000フィールド、「治療」100,000フィールド、「合併症」100,000フィールドなどのテーブルで構成されたSQL DBを想像してください。 別の方法として、特定の患者タイプ（これも重複する可能性があります！）、治療、投薬ごとの数千テーブルと、これらのテーブル間のリレーション用の数千テーブルを含むDBを作成できます。

グローバルは手袋のようにヘルスケアに適合します。これは、各患者に対して完全な症例記録、治療法のリスト、投与薬とその効果をすべて、空のカラムに無駄なディスク領域を浪費することなくツリーの形で持たせることができるためです。リレーショナルデータベースではそうはいきません。

グローバルは、個人の詳細情報を含むデータベースに適しています。取引先に関するさまざまな個人データを最大限に蓄積し、システム化することが課題である場合が想定されます。
これは、医療、銀行、マーケティング、アーカイブなどの分野で特に重要です。

EAV、1、2、3、4、5、6、7、8
）、その場合はかなり複雑になり、処理が遅くなります。 つまりテーブルに基づいてグローバルを記述し、すべてのテーブル関連ルーチンを抽象化レイヤーの下に隠す必要があるでしょう。
上位の技術（SQL）を利用して下位の技術（グローバル）をエミュレートすることは正しくありません。
それは単純に言って不当です。

巨大なテーブルのデータスキーマを変更する処理（ALTER
TABLE）には、かなりの時間がかかる場合があるのは明らかです。
例えば、MySQLの場合はすべてのデータを古いテーブルから新しいテーブルにコピーすることにより、ALTER TABLE ADD|DROP COLUMN操作を実行します（MyISAMとInnoDBでテスト済みです）。 このような処理は、数十億件のレコードを含む本番データベースを数週間とは言わないまでも、数日間ハングアップさせる可能性があり

ます。

グローバルを使用している場合、データ構造の変更にコストはかかりません。いつでも任意の階層の任意のオブジェクトにいくらでも新しいプロパティを追加できます。
枝の名前変更が必要な場合は、DBがバックグラウンドモードで稼働している状態で適用できます。

そのため、グローバルは省略可能なプロパティを多数含むオブジェクトを格納する場合に最適です。

グローバルではすべてのパスがBツリーであるため、プロパティへのアクセスが瞬時に行われることを思い出してください。

一般的なケースでは、グローバルに基づくデータベースは階層情報の格納に対応した一種のドキュメント指向データベースであると言えます。
したがって、ドキュメント指向のデータベースはカルテ保存の分野でグローバルと効果的に競合できます。

しかし、それではまだ不十分です。
MongoDBを例にとってみましょう。**この分野では**、以下の理由でグローバルに劣っています。

1. ドキュメントサイズ。保存単位はJSON形式（正確にはBSON形式）のテキストで、最大サイズは約16 MBです。 この制限は、解析中に巨大なJSONドキュメントが保存され、特定フィールド値のアドレスが指定された場合にJSONデータベースが遅くなりすぎないようにするために導入されました。
このドキュメントには、患者に関する完全な情報が含まれていなければなりません。
私たちは皆、カルテがどれほど厚いかを知っています。 カードの最大サイズが16 MBに制限されているのであれば、カードにMRIスキャン画像、X線スキャン画像などの資料が含まれている患者はすぐに除外されてしまうでしょう。 グローバルの単一ブランチには、ギガバイトや数千テラバイトのデータを持たせることができます。
それがすべてを物語っていますが、もう少し話を続けましょう。

2.
患者カードから新しいプロパティを作成/変更/削除するのに要する時間
。このようなデータベースでは、カルテ全体（大量のデータ！）をメモリにコピーし、BSONデータを解析し、新しいノードを追加/変更/削除し、インデックスを更新し、すべての情報をBSONに圧縮してディスクに保存する必要があります。
グローバルの場合は必要なプロパティのアドレスを指定し、必要な操作を実行するだけで済みます。

3. 特定プロパティへのアクセス速度。ドキュメントに多くのプロパティと複数階層構造がある場合、各パスがBツリーになっているグローバルのほうが特定プロパティへのアクセスが高速になります。
BSONでは、必要なプロパティを見つけるためにドキュメントを順番に解析する必要があります。

3.3.2 連想配列

連想配列（入れ子になった配列でさえも）は、グローバルで完璧に使用できます。
例えば、次のPHP配列は3.3.1の最初の図のようになります。

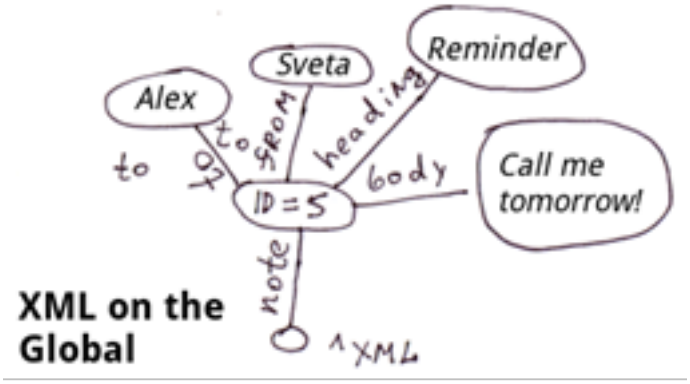
```
$a = array(
  "name" => "Vince Medvedev",
  "city" => "Moscow",
  "threatments" => array(
    "surgeries" => array("apedicectomy", "biopsy"),
    "radiation" => array("gamma", "x-rays"),
    "physiotherapy" => array("knee", "shoulder")
  )
);
```

3.3.3 階層ドキュメント：XML、JSON

これらも簡単にグローバルに格納し、さまざまな方法で分解できます。

XML

XMLをグローバルに分解するには、ノードにタグ属性を格納するのが最も簡単です。 タグの属性に素早くアクセスする必要がある場合は、それらを別々の枝に配置できます。

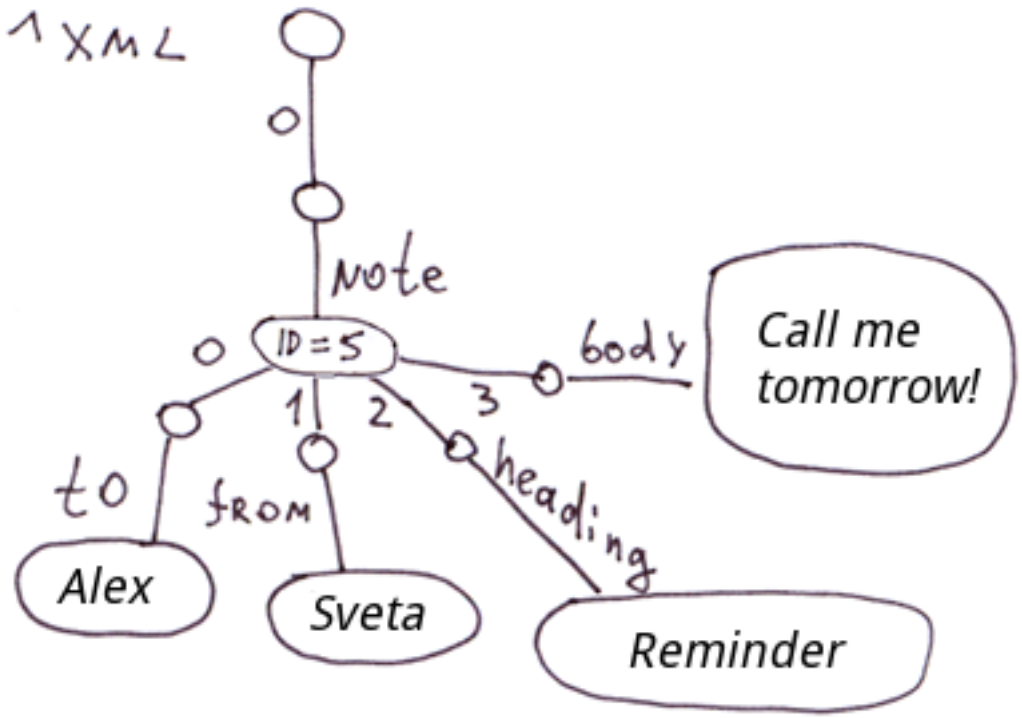



```
<note id=5>
<to>Alex</to>
<from>Sveta</from>
<heading>Reminder</heading>
<body>Call me tomorrow!</body>
</note>
```

COSでは、次のようなコードになります。

```
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：XML、JSON、および連想配列の場合、それらをグローバルに表示する方法はいくつか見つけることができます。 この特定のケースでは、「note」タグ内で入れ子になったタグの順序は反映されていません。 ^xml グローバルでは、入れ子になったタグはアルファベット順に表示されます。
順序を正確に表すには、次のようなモデルを使用できます。



JSON

このJSONドキュメントの内容は、セクション3.3.1の最初の図に示されています。

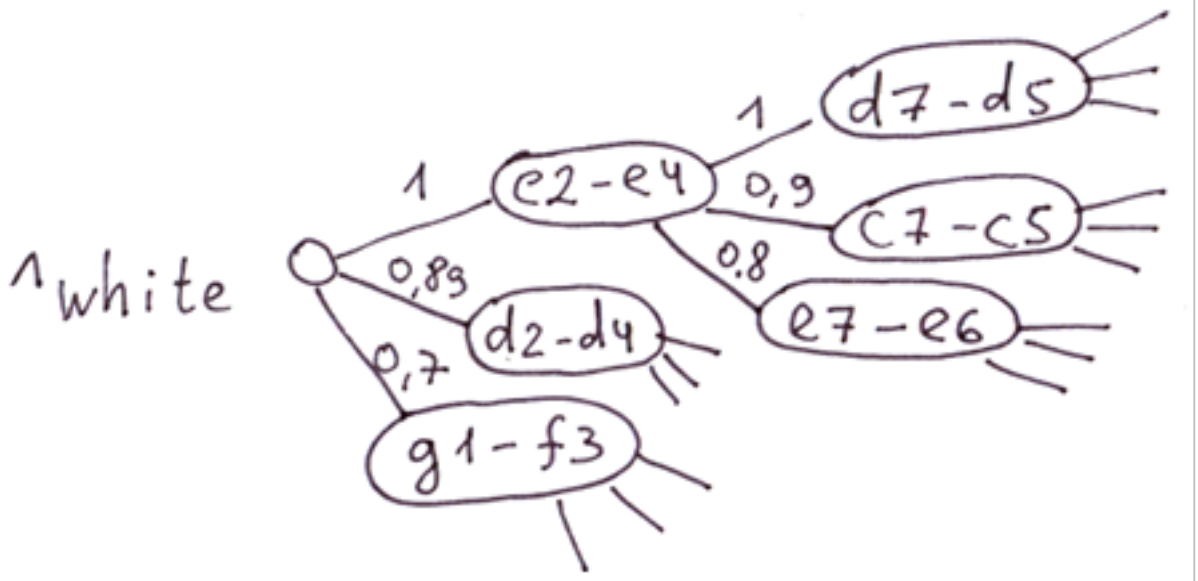
```
var document = {
  "name": "Vince Medvedev",
  "city": "Moscow",
  "threatments": {
    "surgeries": ["apedicectomy", "biopsy"],
    "radiation": ["gamma", "x-rays"],
    "physiotherapy": ["knee", "shoulder"]
  },
};
```

3.3.4 階層関係に制限された同一の構造

例：営業所の構造、ネットワークビジネス構造における人々のポジション。

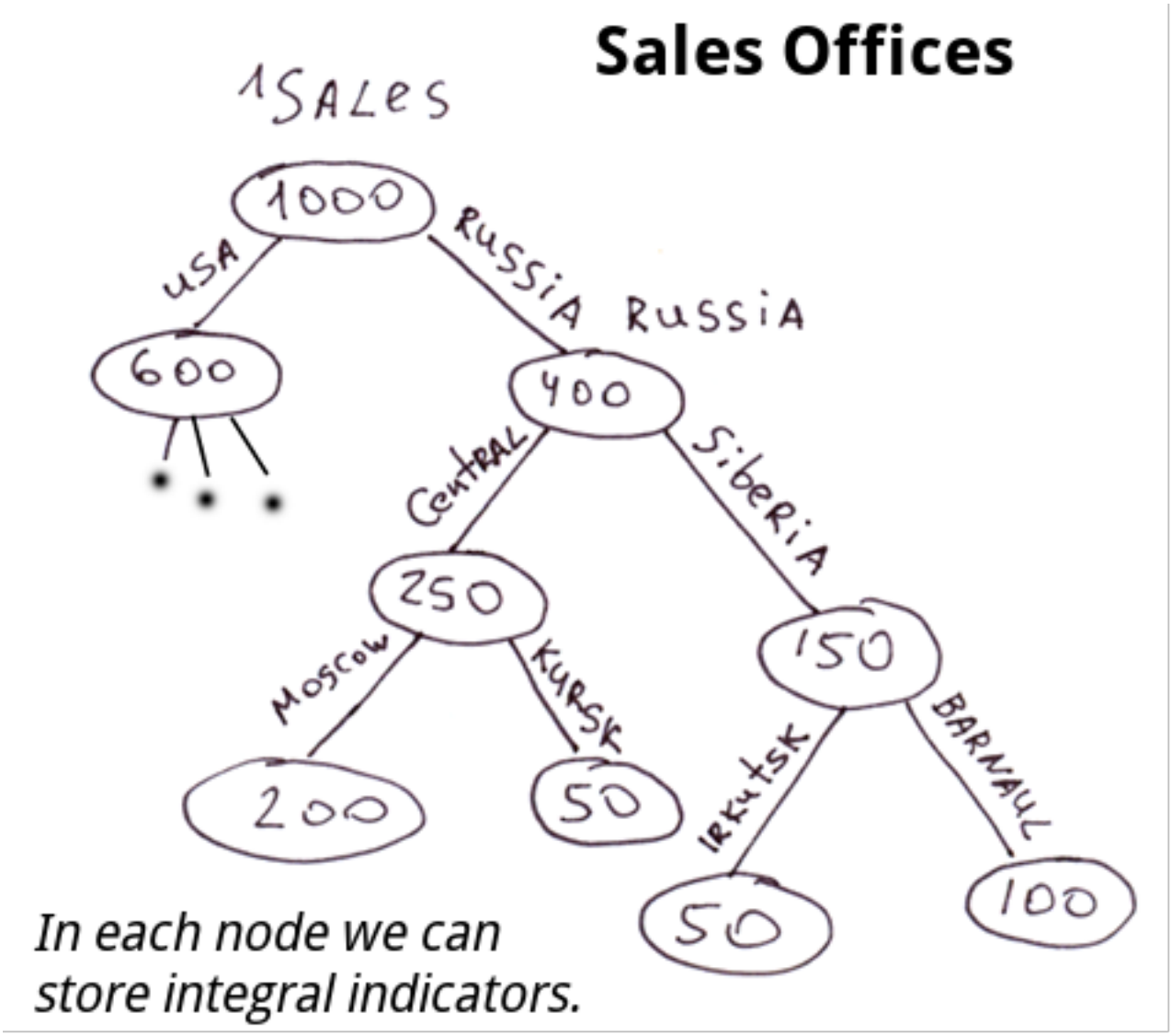
デビューのデータベース。グローバルのノードインデックスの値として、手の強さを評価に使用できます。
この場合、最高の手を決定するには、重みが最も高い枝を選択する必要があります。

グローバルでは、すべての階層のすべての枝が手の強さでソートされます。



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

営業所、ネットワークビジネス企業の人々の構造。ノードには、サブツリー全体の特性を反映するいくつかのキャッシュ値を保存できます。 例えば、この特定のサブツリーの売上高です。
いつでもすべての支店の業績について正確な情報を得ることができます。



4. グローバルの使用が役立つ場面

最初の列にはグローバルを使用した場合にパフォーマンス面でかなりのメリットが得られるケースを、2番目の列には開発またはデータモデルを単純化できる状況を一覧で掲載しています。

スピード	データ処理/表現の利便性
1. 挿入（各階層での自動ソートを伴うもの）、（主キーごとにインデックスを作成するもの） 2. サブツリーの削除 3. 個別にアクセスが必要な入れ子になったプロパティを多数持つオブジェクト 4. 存在しないものも含め、任意の枝から子の枝を探索する可能性のある階層構造 5. ツリーの深さ方向の探索	1. 不要なプロパティ/インスタンス（入れ子になったものを含む）を大量に含むオブジェクト/インスタンス 2. スキーマのないデータ。新しいプロパティが頻繁に追加され、古いプロパティが削除される可能性がある場合。 3. 標準的ではないDBを作成する必要がある場合。 4. パスデータベースとソリューションツリー。パスをツリーとして適切に表現できる場合。 5. 再帰を使用しない階層構造の削除

続きは「グローバルはデータを保存するための魔法の剣です パート3 - 疎な配列」を読み進めてください。

> 免責事項：この記事と記事に対する筆者(英語原文はSergey Kamenev氏によるものです)のコメントは、筆者の意見を反映しているにすぎず、InterSystems Corporationの公式見解とは関係ありません。

次のパート「グローバルはデータを保存するための魔法の剣です パート3 - 疎な配列 」を読み進めてください。

また、前のパート「グローバルはデータを管理するための魔法の剣です パート1」も確認してください。

#Node.js #Globals #Modelo de datos #Rendimiento #Tablas relacionales #Principiante #Caché #InterSystems IRIS
