Artículo
Minoru Horita · 30 abr, 2020 Lectura de 9 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1

データを格納するための魔法の剣であるグローバルは、かなり前から存在しています。しかしながら、これを効率的に使いこなせる人や、この素晴らしい道具の全貌を知る人はそう多くありません。 グローバルを本当に効果を発揮できるタスクに使用すると、パフォーマンスの向上やソリューション全体の劇的な単純化といった素晴らしい結果を得ることができます（1、2）。

グローバルは、SQLテーブルとはまったく異なる特別なデータの格納・処理方法を提供します。
グローバルは1966年にM（UMPS）
プログラミング言語で初めて導入され、医療データベースで使用されていました。
また、現在も同じように使用されています
が、金融取引など信頼性と高いパフォーマンスが最優先事項である他のいくつかの業界でも採用されています。

M（UMPS）は後にCaché ObjectScript
（COS）に進化しました。 COSはInterSystemsによってMの上位互換として開発されました。
元の言語は現在も開発者コミュニティに受け入れられており、いくつかの実装で生き残っています。
ウェブ上では、MUMPS Googleグループ、Mumpsユーザーグループ、ISO規格
といった複数の活動が見られます。

最新のグローバルベースのDBMSは、トランザクション、ジャーナリング、レプリケーション、パーティショニングをサポートしています。 つまり、現代的で、信頼性が高く、高速な分散システムの構築に使用できます。

グローバルは、リレーショナルモデルの限界に制限されません。
特定のタスクに最適化されたデータ構造を自由に作成できます。 多くのアプリケーションにとって、グローバルの合理的な使用は、従来のリレーショナルアプリケーション開発者の理想でしかなかった速度を実現する真の特効薬になるかもしれません。

グローバルは、高レベルおよび低レベル両方の多くの最新のプログラミング言語でデータ保存方法として使用できます。 そのため、この記事ではグローバルの由来となった言語ではなく、グローバルに限定して焦点を当てることにします。

2. グローバルの仕組み

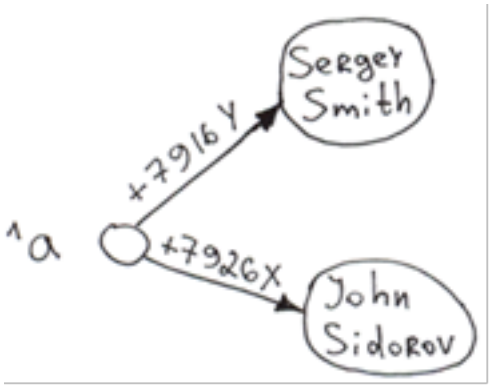
まずはグローバルの仕組みと、そのメリットを理解しましょう。
グローバルはさまざまな視点から見ることができます。
このパートでは、グローバルをツリーまたは階層型データストレージと見なします。

簡単に言えば、グローバルは永続的配列です。 つまり、自動的にディスクに保存される配列ということです。

これ以上簡単にデータを保存する方法を想像するのは困難です。 プログラムコード（COS/M言語で記述）では、その名前の前に^記号があるという点だけが通常の連想配列とは異なっています。

必要なコマンドはすべて非常に簡単で、1時間で習得が可能で、データをグローバルに保存するのにSQLの知識は必要ありません

最も単純な例である、2つの分岐を持つ単一階層のツリーから始めましょう。
以下の例はCOSで記述されています。




```
Set ^a("+7926X") = "John Sidorov"
Set ^a("+7916Y") = "Sergey Smith"
```

データがグローバルに挿入されると（Setコマンド）、次の3つの処理が自動的に行われます。

1. ディスクへのデータ**保存**。
2. インデックスの構築。 括弧内にあるのは添え字、等号の右側にあるのはノードの値です。
3. **ソート**処理。 データはキーでソートされます。 データの探索を行うと「Sergey Smith」が最初に返され、その後に「John Sidorov」が返されます。
   グローバルからユーザーのリストを取得する場合、データベースはソートに時間を費やしません。 実在しないキーも含め、任意のキーから始まるソート済みのリストをリクエストできます（出力は実在しないキーの後に続く最初の実在するキーから始まります）。

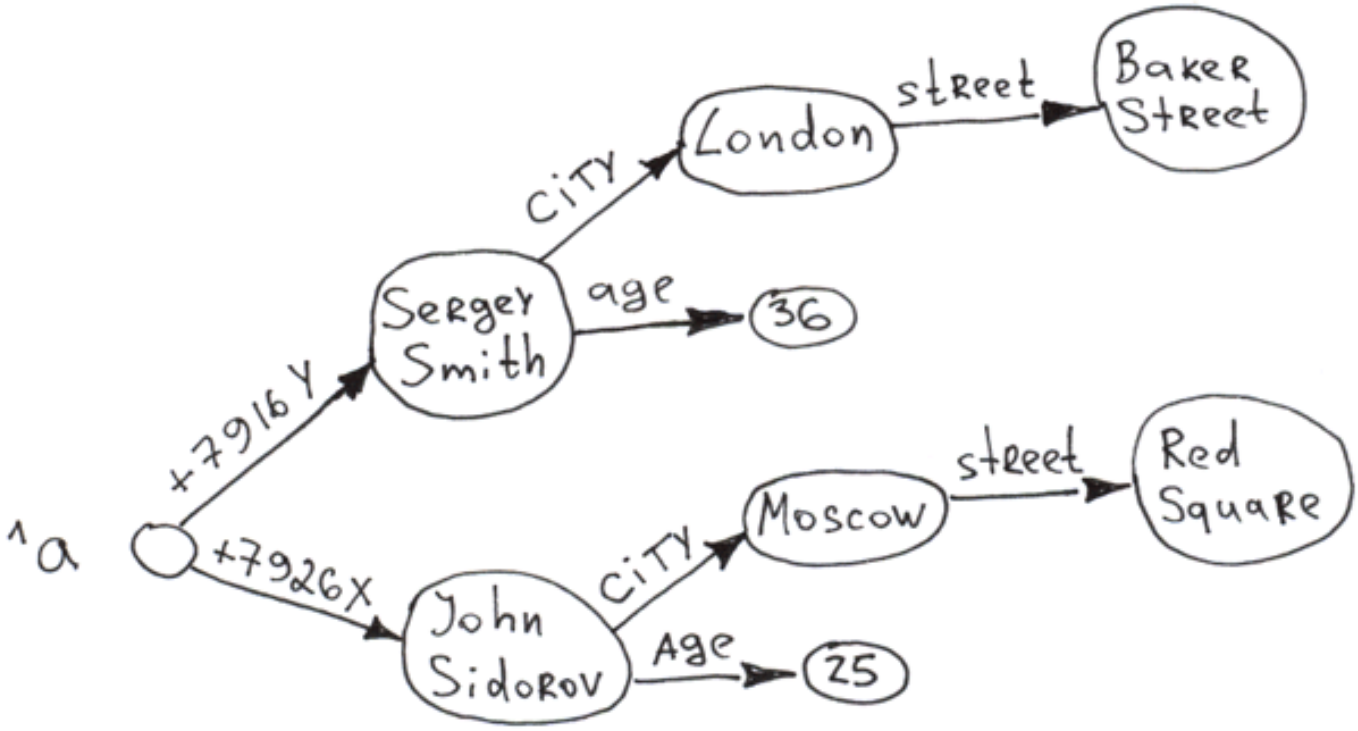
これらの処理はすべて驚異的な速度で実行されます。 筆者宅のシステム（i5-3340、16GB、HDD WD 1TB Blue）では、1回のプロセスで毎秒1,050,000件のレコードが挿入されました。
マルチコアシステムでは、毎秒数億件のレコードを挿入できる可能性があります。

もちろん、レコードの挿入速度自体にはあまり意味がありません。
噂ですが、例えばVisaではデータをテキストファイルに書き込んでいるそうです。 ただし、グローバルを使用すれば、高速で使いやすく、構造化され、インデックス化されたストレージを得ることができます。

- **グローバルの最大の強みは、新しいノードをグローバルに挿入する速度です。**
- **データは常にグローバル内でインデックス化されています。**
  単一階層や階層を下るツリー探索は常に大変高速です。

グローバルの第2階層と第3階層にいくつか枝を追加してみましょう。

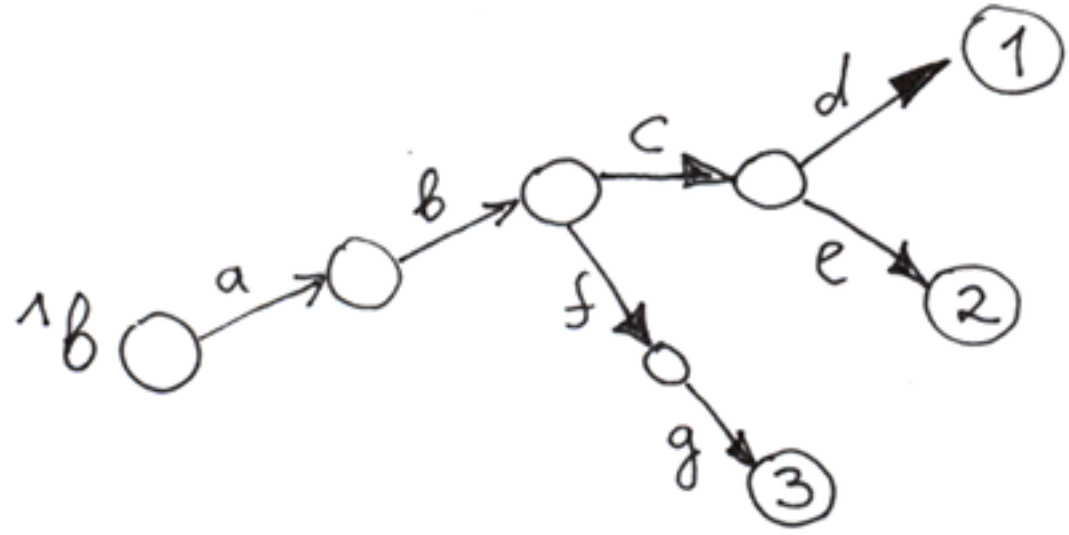
```
Set ^a("+7926X", "city") = "Moscow"
Set ^a("+7926X", "city", "street") = "Req Square"
Set ^a("+7926X", "age") = 25
Set ^a("+7916Y", "city") = "London"
Set ^a("+7916Y", "city", "street") = "Baker Street"
Set ^a("+7916Y", "age") = 36
```



、グローバルを使用して複数階層のツリーを構築することができます。
挿入が発生するたびに自動インデックス付けが行われるため、どのノードにもほぼ瞬時にアクセスできます。
ツリー内のどの階層の枝も、キーでソートされます。

ご覧のとおり、キーと値との両方にデータを保存できます。
Caché ではキーを組み合わせた長さ
（すべてのインデックスの長さの合計）は511バイトまで保存でき、値は最大3.6 MB まで保存できます。
ツリーの階層数（次元数）の上限は31です。

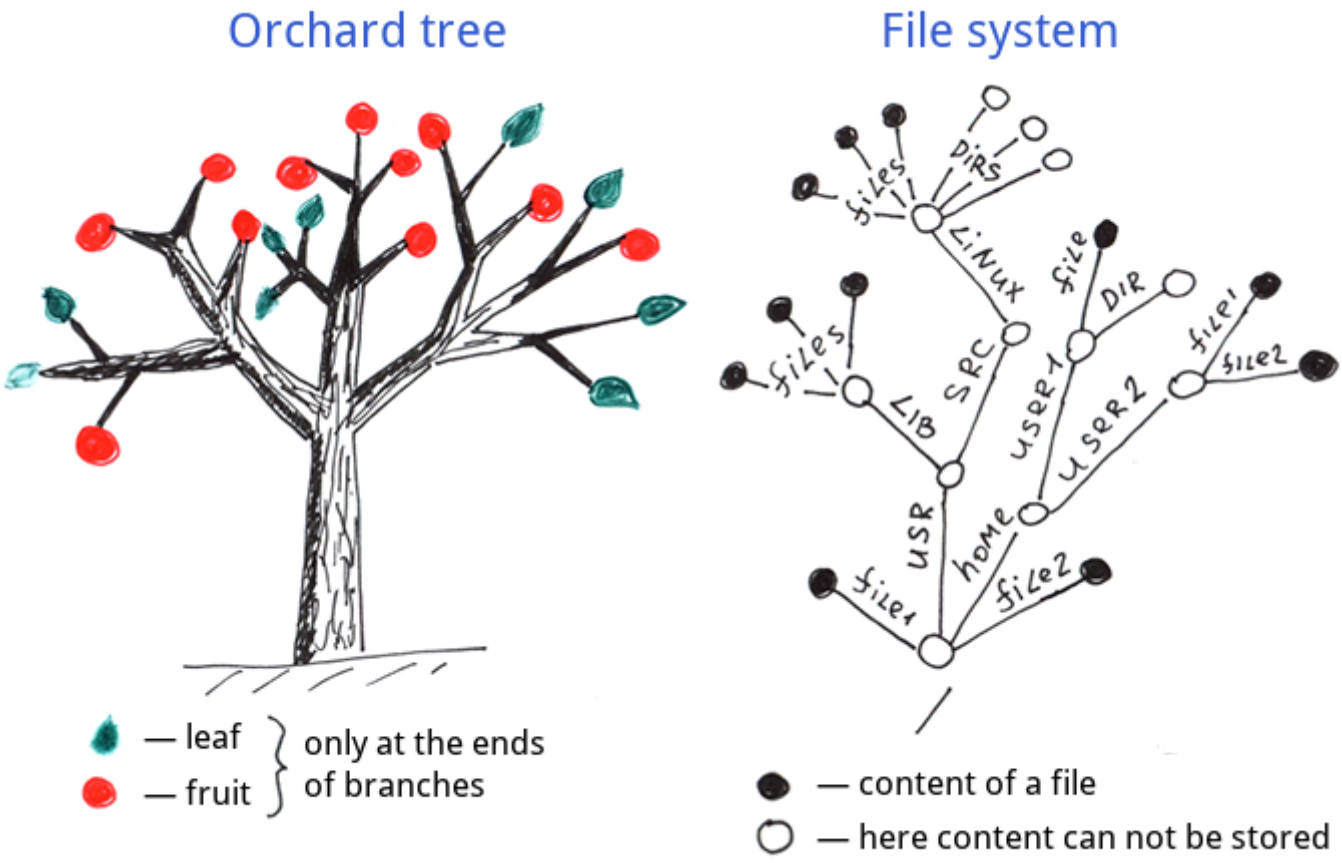
もう1つのすばらしい点は、上位階層のノードの値を定義せずにツリーを構築できることです。



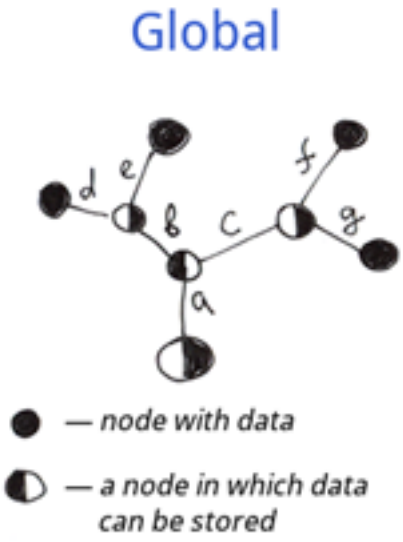
```
Set ^b("a", "b", "c", "d") = 1
Set ^b("a", "b", "c", "e") = 2
Set ^b("a", "b", "f", "g") = 3
```

空の円は値のないノードを表しています。
グローバルをより深く理解するため、グローバルを庭木やファイルシステム名ツリーと比較してみましょう。

グローバルを庭や畑で育つ普通の木やファイルシステムといった、よく見慣れた階層構造と比較してみます。



ご覧のとおり、普通の木では葉と果実は枝の先端でのみ育ちます。
ファイルシステムの場合も、情報は完全ファイル名とも呼ばれる枝の先端に保存されます。



そして、こちらがグローバルのデータ構造です。

違い：

1. **内部ノード**：グローバルの情報は枝の先端だけでなく、すべてのノードに保存できます。

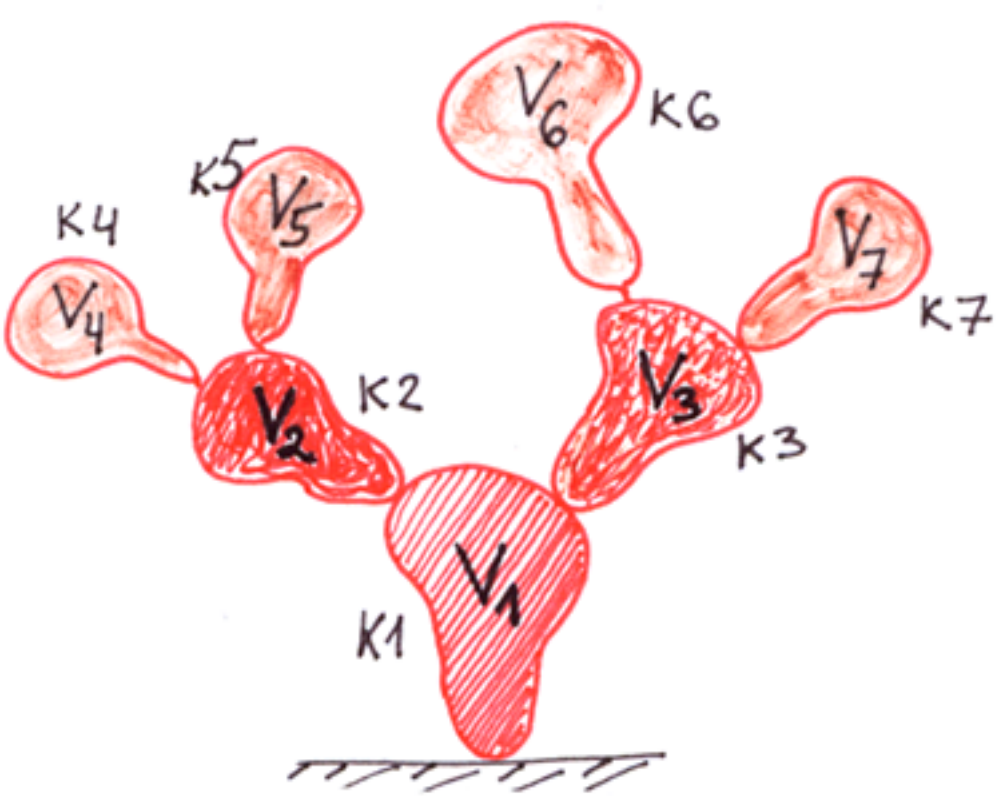
2. **外部ノード**
   ： グローバルにはファイルシステムと庭木には必須ではありませんが、定義された枝の端（値のある先端）が必要です。

内部ノードに関しては、グローバルの構造をファイルシステム名ツリーと庭木構造の上位セットとして扱うことができます。 したがって、グローバルの構造はより柔軟になっています。

大まかに言うと、グローバルは**各ノードでのデータ保存に対応した構造化ツリーです**。

グローバルの仕組みをよりよく理解するため、ファイルシステムの作成者が情報の格納にグローバルと同じ手法を採用した場合にどうなるかを想像してみましょう。

1. あるフォルダー内の最後のファイルが削除された場合、そのフォルダー自体だけでなく、その削除対象のフォルダーのみを含むすべての上位階層のフォルダーも削除されることになります。
2. この場合、フォルダーはまったく必要ありません。
   サブファイルを含むファイルとサブファイルのないファイルがあるとします。
   普通の木と比較して見ると、一つ一つの枝が実になることが分かります。



If the orchard tree was a global ...

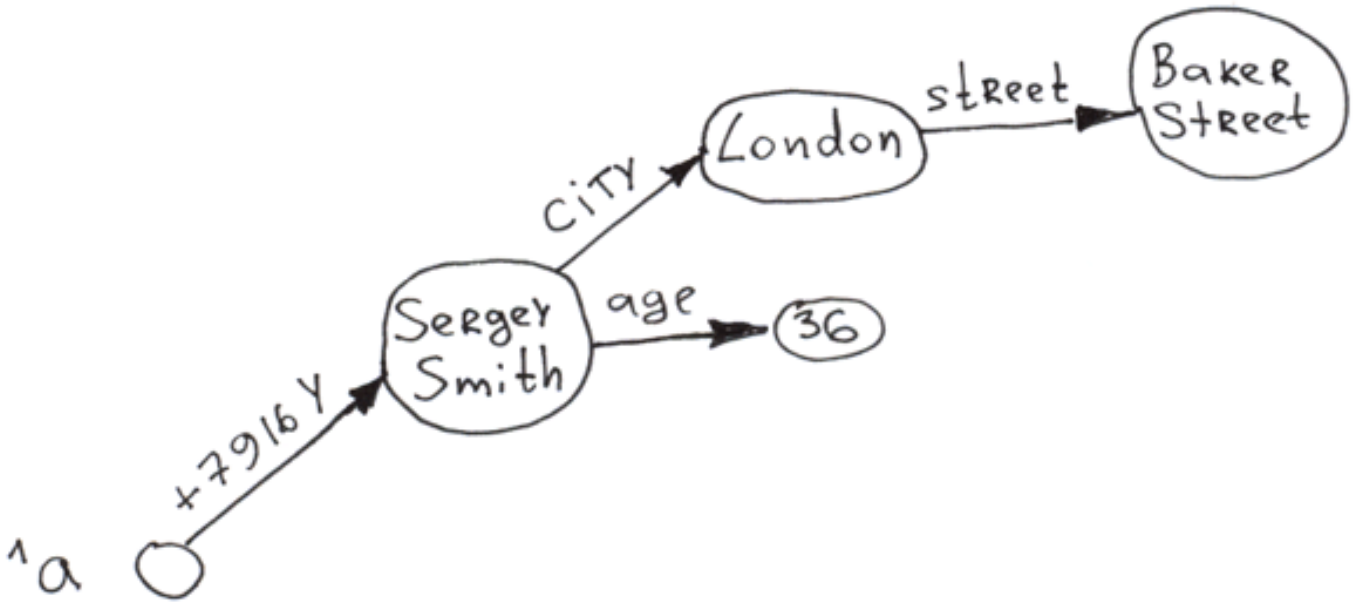
3. README.txtのようなものは不要になりそうです。 ただし、フォルダーの内容に関して言及したいすべての情報はフォルダーファイル自身に書き込まれる可能性があります。 通常、ファイル名はフォルダー名と区別されません（例えば、 /etc/readme はファイルにもフォルダーにもなり得ます）。つまり、ファイルのみを操作することで十分ということになります。

4. サブフォルダーとファイルを含むフォルダーは、はるかに高速に削除できるでしょう。 数百万件の小さなフ
ァイ
ルの削除
がいかに時間がかかり、困難であるかを伝える記事がネット上にいくつか存在します（1、2、3）。 ただし、グローバルに基づいて疑似ファイルシステムを作成した場合は数秒か数分の1秒しかかかりません。 筆者の自宅PCでサブツリーの削除をテストしたときは、HDD（SDDではない）上の2階層のツリーから9,600万ノード～3億4,100万ノードを削除できました。 また、重要な事ですが、ここで話題にしているのはグローバルを含むファイル全体の削除ではなく、グローバルツリーの一部を削除することです。

**グローバルのもう1
つの長所は、再帰的処理をすることなくサブツリーを削除できることです**。信じられないほど高速です。

上記のツリーでは、次のような**Kill**コマンド1つでサブツリーを削除できます。

```
Kill ^a("+7926X")
```



グローバルで実行できるアクションをよりよく理解できるよう、以下の小さな表にまとめています。

COSのグローバルに関連する主なコマンドと機能	
Set	ノードまでの枝（未定義の場合）とノード値を設定（初期化）します。
Merge	サブツリーをコピーします。
Kill	サブツリーを削除します。
ZKill	特定ノードの値を削除します。 そのノードから生じたサブツリーは影響を受けません。
$Query	ツリー全体を深さ優先探索します。
$Order	同じ階層の次の添え字を返します。
$Data	ノードが定義されているかどうかを確認します。
$Increment	ACIDの読み取りと書き込みを回避するため、ノード値のアトミックなインクリメント操作を実行します。 最近は、$Sequence を代わりに使用することを推奨しています。

最後までお読みいただき、ありがとうございました。喜んで皆様からのご質問にお答えします。

> 免責事項：この記事は筆者(英語原文はSergey Kamenev氏によるものです)の私見を反映したものであり、InterSystemsの公式見解とは関係ありません。

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2 - ツリー」を読み進めてください。
グローバルに表示できるデータのタイプと、グローバルが最適に機能する場所について学習します。

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