УДК 004

Азарян Магдалина Андраниковна,
студент магистратуры
1 курс, институт Цифрового развития
Северо-Кавказского федерального университета
Россия, г. Ставрополь
Насырова Бахтыгуль Халитовна,
студент магистратуры
2 курс, институт Цифрового развития
Северо-Кавказского федерального университета
Россия, г. Ставрополь
Казарян Джемма Манвеловна,
старший преподаватель
Северо-Восточного государственного университета
Россия, г. Магадан
Таволжанова Олеся Андреевна,
студент
3 курс, институт Цифрового развития
Северо-Кавказского федерального университета
Россия, г. Ставрополь

МОДЕЛИРОВАНИЕ OLAP КУБА: РЕАЛИЗАЦИЯ ГИПЕРКУБА

Аннотация: *В статье рассматривается второй этап моделирования OLAP куба, а именно его реализация. Используются структуры данных, которые обеспечивают максимальное быстродействие и минимальные расходы оперативной памяти.*

Ключевые слова: *OLAP куб, хеш-дерево, гиперкуб, сводная таблица, уникальные элементы, иерархии дерева заголовков, структуры данных.*

Annotation: *The article deals with the second stage of OLAP cube modeling, namely its implementation. Data structures are used that provide maximum performance and minimum memory consumption.*

Key words: *OLAP cube, hash tree, hypercube, pivot table, unique elements, header tree hierarchies, data structures.*

Для реализации гиперкуба нам необходимо использовать структуры данных, которые обеспечат максимальное быстродействие и минимальные расходы оперативной памяти. Очевидно, что основными у нас будут структуры для хранения словарей и таблицы фактов [4]. Рассмотрим задачи, которые должен выполнять словарь с максимальной скоростью:

- проверка наличия элемента в БД;
- добавление элемента в БД;
- поиск номеров записей, имеющих конкретное значение координаты;
- поиск координаты по значению измерения;
- поиск значения измерения по его координате.

Для реализации этих требований можно использовать различные типы и структуры данных. Например, можно использовать массивы структур. В реальном случае к этим массивам необходимы дополнительные механизмы индексации, которые позволят повысить скорость загрузки данных и получения информации. Для оптимизации работы гиперкуба необходимо определить то, какие задачи необходимо решать в первоочередном порядке, и по каким критериям нам надо добиваться повышения качества работы. Главным для нас является повышение скорости работы программы, при этом желательно, чтобы требовался не очень большой объем оперативной памяти. Повышение быстродействия возможно за счет введения дополнительных механизмов доступа к данным, например, введение индексирования. К сожалению, это повышает накладные расходы оперативной памяти. Поэтому

определим, какие операции нам необходимо выполнять с наибольшей скоростью. Для этого рассмотрим отдельные компоненты, реализующие гиперкуб. Эти компоненты имеют два основных типа – измерение и таблица фактов. Для измерения типовыми задачами будет добавление нового значения, определение координаты по значению измерения и определение значения по координате.

При добавлении нового значения элемента нам необходимо проверить, есть ли у нас уже такое значение, и если есть, то не добавлять новое, а использовать имеющуюся координату, в противном случае необходимо добавить новый элемент и определить его координату. Для этого необходим способ быстрого поиска наличия нужного элемента. Для этого оптимальным будет использование хеширование. При этом оптимальной структурой будет использование хеш-деревьев, в которых будем хранить ссылки на элементы. При этом элементами будут строки словаря измерения. Тогда структуру значения измерения можно представить следующим образом:

```
PFactLink = ^TFactLink;
TFactLink = record
   FactNo: integer; // индекс факта в таблице
   Next: PFactLink; // ссылка на следующий элемент
End;
TDimensionRecord = record
   Value: string; // значение измерения
   Index: integer; // значение координаты
   FactLink: PFactLink; // указатель на начало списка элементов таб-лицы фактов
End;
```

И в хеш-дереве будем хранить ссылки на уникальные элементы. Кроме того, нам необходимо решить задачу обратного преобразования – по координате определить значение измерения. Для обеспечения максимальной производительности надо использовать прямую адресацию. Поэтому можно использовать еще один массив, индекс в котором является координатой измерения, а значение – ссылка на соответствующую запись в словаре. Однако

можно поступить проще (и сэкономить при этом на памяти), если соответствующим образом упорядочить массив элементов так, чтобы индекс элемента и был его координатой [5].

При этом в качестве значения измерения логично хранить ссылку на соответствующий элемент таблицы измерений многомерного куба. Это позволит сократить затраты памяти для хранения среза и ускорить работу. В качестве родительских и дочерних узлов также используются ссылки. Тогда описанную структуру можно реализовать следующим пседокодом:

```
PHeaderTreeNode = ^THeaderTreeNode;
THeaderTreeNode = record
  Parent: PHeaderTreeNode; // родительский узел
  Value: PDimensionRecord; // значение измерения
  ChildCount: integer; // кол-во детей узла
  Childs: array of PHeaderTreeNode; // массив детей узла
  ...
End;
```

В этом коде пока не указан один элемент – столбец (строка) со значением. О том, чем является это поле, мы поговорим попозже. А теперь перейдем к добавлению элементов в дерево заголовков. Каждый уровень дерева будет соответствовать одному измерению в том порядке, в котором они используются для построения среза. При этом самый верхний узел дерева будет соответствовать полному итогу в сводной таблице.

Для добавления элемента в дерево необходимо иметь информацию о его местоположении в гиперкубе. В качестве такой информации надо использовать его координату, которая хранится в базе значений измерения. Рассмотрим схему добавления элемента в дерево заголовков сводной таблицы. При этом в качестве исходной информации используем значения координат измерений. Порядок, в котором эти измерения перечислены, определяется требуемым способом агрегирования и совпадает с уровнями иерархии дерева заголовков [2]. В результате работы необходимо получить список столбцов

или строк сводной таблицы, в которые необходимо осуществить добавление элемента.

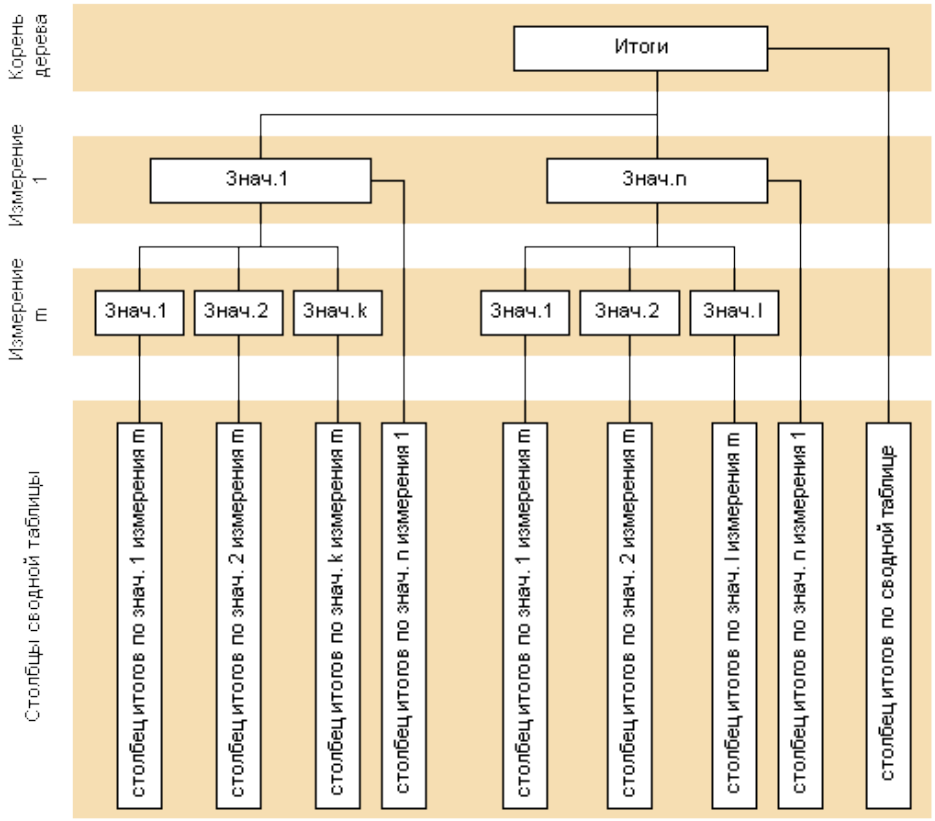

Рисунок 1. Схема хранения столбцов сводной таблицы

В качестве исходных данных для определения этой структуры используем координаты измерений. Кроме того, для определенности, будем считать, что мы определяем интересующий нас столбец в матрице. В качестве координат возьмем целые числа – номера значений измерений. Тогда это можно реализовать процедурой, имеющей следующий вид:

```
Procedure GetMatrixCol(InpCoord: array of integer; var OutCols: array of PMatrixCol);
// InpCoord – набор значений координат измерений
// OutCols – набор столбцов, в формировании значений которых ис-пользуется данный
// факт
Var
  xNode: PheaderTreeNode; // текущий узел дерева
  I: integer; // Номер текущего уровня дерева
Begin
```

```
  xNode := RootNode; // присваиваем значение текущего узла в корне-вой узел
  for I := 0 to DimensionsCols.Count – 1 do
  begin
    OutCols[I] := xNode.MatrixCol; // это ссылка на столбец матрицы
    // проверяем, есть ли у нас соответствующий дочерний узел
    if xNode.Childs[InpCoord [I]] = nil then
    begin
      inc(xNode).ChildCount; // увеличиваем счетчик потомков узла
      // теперь создаем новый узел – потомок
      xNode.Childs[InpCoord [I]] := new(TColHeaderTreeNode);
      SetLength(xNode.Childs[InpCoord [I]].Childs, DimensionsCols[I].Count);
      xNode.Childs[InpCoord [I]].ChildCount := 0;
      xNode.Childs[InpCoord [I]].Parent := xNode;
      xNode.Childs[InpCoord [I]].Value := DimensionCols[I].Value[InpCoord [I]];
    end;
    // переходим к следующему узлу
    xNode := xNode.Childs[InpCoord [I]];
  end;
End;
```

Итак, после выполнения этой процедуры получим массив ссылок на столбцы разреженной матрицы. Теперь необходимо выполнить все необходимые действия со строками. Для этого внутри каждого столбца необходимо найти нужный элемент и добавить туда соответствующее значение. Из того, что раньше не рассматривалось в процедуре встречается DimensionCols – это коллекция измерений, которые отложены в столбцах сводной таблицы. Для каждого из измерений в коллекции необходимо знать количество уникальных значений и собственно набор этих значений.

Теперь рассмотрим, в каком виде необходимо представить значения внутри столбцов – то есть как определить требуемую строку. Для этого можно использовать несколько подходов. Самым простым было бы представить каждый столбец в виде вектора, но так как он будет сильно разреженным, то память будет расходоваться крайне неэффективно. Чтобы избежать этого,

применим структуры данных, которые обеспечат большую эффективность представления разреженных одномерных массивов [1]. Самой простой из них будет обычный список, одно или двусвязный, однако он неэкономичен с точки зрения доступа к элементам. Поэтому будем использовать дерево, которое обеспечит более быстрый доступ к элементам. Например, можно использовать точно такое же дерево, как и для столбцов, но тогда пришлось бы для каждого столбца заводить свое собственное дерево, что приведет к значительным накладным расходам памяти и времени обработки. Заведем одно дерево для хранения всех используемых в строках комбинаций измерений, которое будет идентично вышеописанному, но его элементами будут не указатели на строки, а их индексы, причем сами значения индексов нас не интересуют и используются только как уникальные ключи. Затем эти ключи будем использовать для поиска нужного элемента внутри столбца. Сами же столбцы проще всего представить в виде обычного двоичного дерева. Графически полученную структуру можно представить следующим образом:

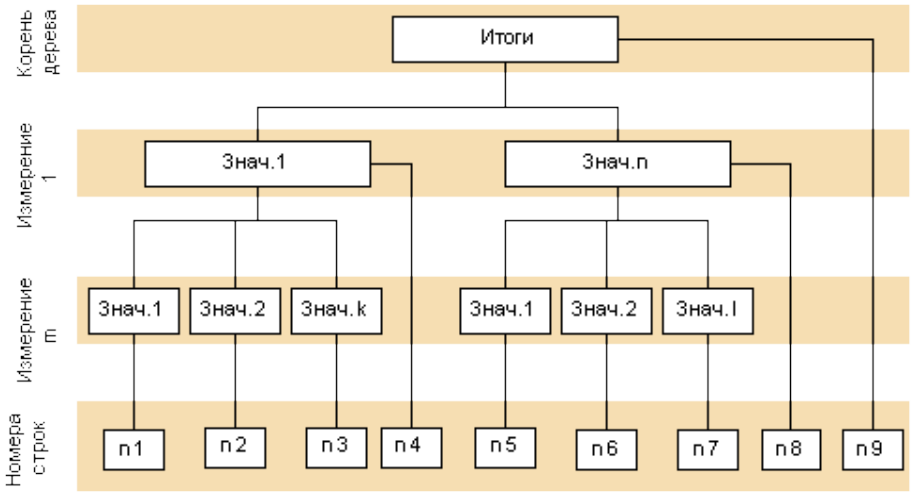

Рисунок 2. Дерево для определения строк сводной таблицы

Для определения соответствующих номеров строк можно использовать такую же процедуру, что и описанная выше процедура определения столбцов сводной таблицы. При этом номера строк являются уникальными в пределах одной сводной таблицы и идентифицируют элементы в векторах, являющихся

столбцами сводной таблицы. Наиболее простым вариантом генерации этих номеров будет ведение счетчика и инкремент его на единицу при добавлении нового элемента в дерево заголовков строк [3]. Сами эти вектора столбцов проще всего хранить в виде двоичных деревьев, где в качестве ключа используется значение номера строки. Кроме того, возможно также и использование хеш-таблиц. Так как процедуры работы с этими деревьями детально рассмотрены в других источниках, то останавливаться на этом не будем и рассмотрим общую схему добавления элемента в столбец.

В обобщенном виде последовательность действий для добавления элемента в матрицу можно описать следующим образом:

1. Определить номера строк, в которые добавляются элементы
2. Определить набор столбцов, в которые добавляются элементы
3. Для всех столбцов найти элементы с нужными номерами строк и добавить к ним текущий элемент.

После выполнения этого алгоритма получим матрицу, представляющую собой сводную таблицу, которую нам было необходимо построить.

Также добавим несколько слов про фильтрацию при построении среза. Проще всего ее осуществить как раз на этапе построения матрицы, так как на этом этапе имеется доступ ко всем требуемым полям, и, кроме того, осуществляется агрегация значений. При этом, во время получения записи из кэша, проверяется ее соответствие условиям фильтрации, и в случае его несоблюдения запись отбрасывается.

Использованные источники:

1. Илюшечкин В.М., Основы использования и проектирования баз данных / В.М. Илюшечкин. – Москва.: Высшее образование, 2011. – 213 с.
2. Цыганов А.А., Управление данными / А.А. Цыганов, А.В. Кузовкин, Б.А. Щукин. – Москва.: Academia (Академпресс), 2010. – 256 с.

3. Коннолли Т., Базы данных: проектирование, реализация, сопровождение. Теория и практика / Т. Коннолли, К. Бегг. – Москва.: Изд. дом «Вильямс», 2003. – 1440 с.
4. Многомерная модель [Электронный ресурс]. – Режим доступа: https://spravochnick.ru/bazy_dannyh/dorelyacionnye_modeli_dannyh/mnogomernaya_model/ (дата обращения 23.01.2022).