Группирование и агрегирование в концептно-ориентированной модели данных

А.А. Савинов

Институт математики и информатики АН Молдовы, Молдова
Институт автономных интеллектуальных систем, Германия

Аннотация:

В статье рассматривается организация механизма группирования и агрегирования в концептно-ориентированной модели данных. На синтаксическом уровне данные представляются с помощью множества концептов, каждый из которых является комбинацией других концептов. На семантическом уровне концепты состоят из элементов данных, каждый из которых равен комбинации других элементов. Все записи образуют решетку, где каждый элемент имеет несколько родительских и несколько дочерних элементов. Родительские элементы понимаются как группы или множества, состоящие из соответствующих дочерних элементов. В статье показывается, как такая организация данных может использоваться для простой и эффективной организации механизма группирования и агрегирования.

Введение

В основе концептно-ориентированного (КО) подхода лежит предположение, что описание системы делится на две части: синтаксическую и семантическую. *Синтаксис* системы описывает структуру пространства, тогда как *семантика* представляет объекты, которые в это пространство помещены. Предполагается, что структура пространства является более стабильной и меняется относительно редко, тогда как положение объектов в пространстве это как раз то, что обычно необходимо отслеживать или изучать. Для концептно-ориентированного подхода такое разделение на синтаксис и семантику играет первостепенную роль. Другим важным свойством является то, что с точки зрения функционирования системы пространство и его структура, описанные синтаксическими средствами, имеют не меньшее значение, чем объекты этого пространства. Существующие традиционные подходы обычно не уделяют роли пространства должного внимания, тогда как для КО парадигмы это центральный момент. Другими словами, мы просто предполагаем, что объекты не могут существовать в вакууме, а требуют определенной среды, которая оказывает на них постоянное значительное влияние и может в значительной степени определять функционирование всей системы. Особенно это касается больших систем, число которых в настоящее время стремительно увеличивается из-за происходящих процессов глобализации.

Отметим существенное отличие этого принципа от объектно-ориентированного (ОО) подхода к описанию систем, который постулирует первостепенную роль объекта, тогда как понятие пространства вообще явно не используется. Можно сказать, что в отличие от ОО подхода, где система полностью описываются функциями объектов, КО подход утверждает, что для описания системы необходимо прежде всего описать структуру и функции ее пространства, и только потом можно описывать методы объектов. При этом структура пространства, где живут объекты и его функции часто более важны, чем функции объектов. В этом смысле КО подход следует принципу «система это структура пространства, в котором живут объекты». Пространство это не что иное как среда существования, без которой объекты не только не могут функционировать, но

даже не могут быть определены и существовать как отдельные сущности. Это связано с тем, что в функции среды входят идентификация объектов, поддержание их жизненного цикла, передача взаимодействий, защита и многое другое. Примером пространства в реальной жизни может служить государство, физическая среда или биосфера без которых, очевидно, существование человека невозможно.

Основные функции, которым в КО парадигме уделяется первостепенное значение это *представление* и *доступ*. Это означает, что функции всей системы зависят от того, как в ней представляются объекты и как осуществляется доступ к ним, тогда как процессы, происходящие внутри самих объектов, в отличие от ОО парадигмы, оказывают существенно меньшее влияние на систему. Вопрос о способах представления и доступа является одним из самых сложных и полного ответа на него нет. Он фактически сводится к вопросу, что такое объекты, как они существуют, как они узнают друг о друге и как между ними осуществляется взаимодействие. Необходимо понимать, мы всегда имеем только какие-то замещающие сущности в качестве *представителей* объектов и явлений, но не можем воспринимать объекты сами по себе, а значит не знаем что это такое. В программировании это обычно ссылки, с помощью которых мы можем получать определенное содержимое, но программист на самом деле никогда не знает, что стоит за ссылкой. В частности, он не знает, хранится ли содержимое в памяти или значение генерирует какая-то процедура. На более низком уровне неизвестно, хранится ли объект в оперативной памяти или на диске и т.д. Более того, каждый объект имеет множество различных представителей и может восприниматься по разному, т.е. изменять свою форму. Например, люди могут идентифицироваться паспортами или водительскими правами, а объекты в программе локальными ссылками, удаленными ссылками или именами. Один из наиболее интересных моментов состоит в том, что сами представители могут быть объектами и иметь свои собственные идентификаторы.

В КО подходе собственно утверждается, что от того, как сущности представляются и как к ним осуществляется доступ зависят все функции системы. Функции объектов могут быть весьма просты, но система будет проявлять сложное поведение за счет механизма представления и доступа. Например, поместите объекты в одну среду, и они будут проявлять себя одним способом, тогда как в другой среде они могут вести себя и выглядеть совершенно иначе. Функции среды или пространства задействуются как раз в процессе доступа к объектам. На самом деле, это является следствием известного физического принципа, что никакие взаимодействия не распространяются мгновенно, т.е. воздействие всегда распространяется в какой-то среде с помощью определенных механизмов. Например, вызов метода в традиционном ОО программировании представляется как мгновенная процедура, т.е. сразу за вызовом метода по заданной ссылке начинается выполнение его внутренних команд. В КО ситуация меняется и все процедуры доступа являются полноценной частью программы, существенно влияя на поведение системы. Таким образом, сложной систему делает именно присущие ей способы распространения взаимодействий, т.е. процедуры доступа, и в гораздо меньшей степени методы самих объектов.

Концептно-ориентированный подход развивается в нескольких направлениях, но всех их объединяет то, что в основе лежит понятие *концепта*. В частности, в КО дизайне рассматриваются общие принципы описания систем, что включает в частности языки моделирования (аналог UML) и разметки (аналог XML). Концепт в этом случае представляет среду или контейнер, где существуют другие объекты, что может быть просто границей, через которую проходят взаимодействия, либо более сложным образованием. В КО программировании рассматривается, как функции представления и доступа можно описать в обычной программе. В

этом случае концепт является обобщением понятием класса, и его роль состоит в том, что контролировать доступ к внутренним пространствам и классам. В концептно-ориентированной модели данных концепт является контейнером, где содержатся элементы данных (записи).

Общим для всех этих подходов является не только понятие концепта как элементарного пространства, но и структура концептов. Предполагается, что концепты могут быть вложены в другие концепты в помощью отношения включения. В результате пространство имеет иерархическую многомерную структуру, формат и функции которой определяют способ представления и доступа к объектам, что в соответствии с главным предположением влияет на поведение всей системы. Каждый промежуточный концепт определяет представление и доступ к внутренним элементам, а все вместе они образуют общую среду существования объектов.

В данной статье рассматривается только концептно-ориентированная модель данных (КОМД), описанная детально в [12], а также в несколько более общей форме в [13,14]. Теоретически эта модель данных основана на теории решеток и упорядоченных множеств, а с прикладной точки зрения близка к пониманию данных, свойственному таким подходам как онтологии [5,8], многомерные базы данных и OLAP [2], формальный концептный анализ [6]. Кроме того, концептно-ориентированный подход косвенно пересекается с такими областями как грубые множества [9] и теория матроидов [11]. Суть этой модели состоит в том, что концепт (аналог таблицы или множества) определяется как комбинация других концептов так, что каждый концепт в модели имеет множество родительских и дочерних концептов. Дополнив эту структуру пространства верхним и нижним концептом, мы получаем решетку концептов, которая формально представляет пространство, в котором живут данные. Концепт состоит из множества элементов данных или записей. Каждая запись определяется как комбинация записей из родительских концептов. Структура концептов определяет синтаксис данных, а записи в них определяют семантику.

Такая модель обладает довольно интересными свойствами, и на ее основе можно построить самые разнообразные механизмы манипулирования данными. В частности, она очень удобна для моделирования связей как это делается в модели сущность-связь [10], для описания реляционных [7,3] и многомерных [2] данных, а также для представления информации в семантической паутине [1] и доступа к сервисам [4]. В данной статье рассматривается только вопрос группирования и агрегирования данных. Для этого важно понимать, что КО модель данных имеет как иерархические свойства так и многомерные, причем из ее описания мы не можем отделить их друг от друга. В частности, в модели существуют общий уровень представления, где семантика лишена всяких деталей и нижний наиболее детальный уровень представления, где объекты характеризуются многими свойствами. Соответственно, для каждой модели встает вопрос, как информация может распространяться в ней между уровнями и вдоль размерностей.

Синтаксис и семантика данных

На синтаксическом уровне *концепт* определяется как комбинация других концептов, которые называются *надконцептами* (или суперконцептами): $C = \langle C_1, C_2, \ldots, C_n \rangle$. Сам концепт по отношению к своим составляющим называется *подконцептом*. Концепт может комбинировать одинаковые концепты в своем определении, а общее количество надконцептов называется размерностью. Синтаксическое описание не допускает циклов, поскольку пространство не может включать и быть определено через самого себя. Кроме того, в синтаксическое описание включается два формальных элемента: *верхний* концепт и *нижний* концепт. Верхний концепт

является прямым или непрямым надконцептом для любого другого концепта. Нижний концепт является прямым или непрямым подконцептом для любого другого концепта. Важную роль играют *примитивные* концепты, для которых верхний концепт является непосредственным родителем (надконцептом). Предполагается, что непосредственный подконцепт (надконцепт) имеют *ранг* 1 по отношению к данному концепту, и с каждым уровнем вложения ранг увеличивается. Синтаксис концептно-ориентированной базы данных (КОБД) это набор всех концептов вместе с отношением включения между ними, а также верхним и нижним концептом. Ранг базы данных определяется максимальным рангом включения нижнего концепта.

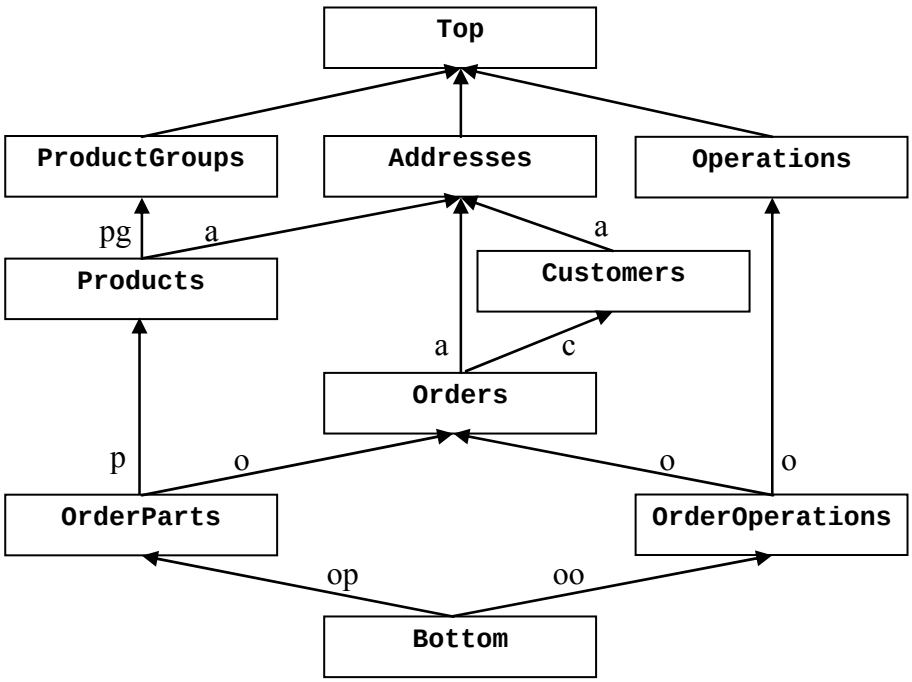

Рисунок 1. Синтаксис базы данных может быть представлен в виде ациклического графа.

Синтаксис может быть изображен в виде ациклического графа, где вершины представляют концепты, а ребра отношение подконцепт-надконцепт. Например, на рис. 1 изображена модель данных предприятия, которое получает заказы от заказчиков и затем обрабатывает их. Эта модель состоит из 10 концептов (включая вспомогательные верхний и нижний). Концепт `Orders` содержит заказы и является комбинацией концепта адресов (`Addresses`) и концепта заказчиков (`Customers`). С другой стороны, этот же концепт заказов включается в два подконцепта `OrderParts` и `OrderOperations`.

Другой, несколько менее формальны и более привычный, способ описания синтаксиса или схемы базы данных состоит в использовании понятия переменной. *Переменная* или размерность это имя, сопоставленное каждому подконцепту в определении данного концепта, причем все имена уникальны в рамках одного концепта. В результате синтаксис концепта может быть записан как комбинация переменных (а не подконцептов): $C = \langle x_1, x_2, \ldots, x_n \rangle$. Надконцепт $C_i = \mathrm{Dom}(x_i)$, соответствующий переменной из определения данного концепта, называется *доменом* или областью значений этой переменной. Например, концепт заказов на рис. 1 мог бы быть представлен как комбинация переменных *a* и *c* с доменами в концептах `Addresses` и `Customers`, соответственно.

Собственные переменные концепта имеют ранг 1 и обозначаются одним именем. Переменные более высокого ранга обозначаются с помощью последовательности имен, где каждая следующая относится к домену предыдущей переменной: $x.y.\cdots.z$, где y это переменная из $\mathrm{Dom}(x)$ и т.д. Часто в качестве первого элемента последовательности записывается имя концепта, которому она соответствует (другие концепты восстанавливаются рекурсивно). Число имен переменных в цепочке равно рангу. Например, переменная, обозначаемая цепочкой `OrderParts.o.c.a`, принадлежит концепту `OrderParts`, принимает значения из концепта `Addresses` и имеет ранг 3.

Примитивная переменная имеет примитивный концепт в качества домена последнего элемента в цепочке имен. Другими словами, примитивная переменная всегда заканчивается примитивным концептом в качестве области значений. Синтаксис может быть представлен с помощью набора концептов и их переменных. Синтаксис определенного ранга включает все переменные данного концепта, имеющие заданный ранг. Примитивный синтаксис включает все примитивные переменные и определяет полную размерность концепта. Все синтаксические параметры базы данных определяются ее нижним концептом. Например, размерность модели на рис. 1 равна 7, поскольку нижний концепт характеризуется 7 примитивными переменными, каждая из которых представляет путь из нижнего концепта в верхний.

Двойственным к переменной является понятие *подпеременной* или подразмерности. Каждая подпеременная соответствует обычной переменной, заданной цепочкой имен или концептов, но интерпретируемых в обратном направлении. Другими словами, если переменные в концептном графе это всегда путь вверх, то подпеременные соответствуют пути вниз. Значит все подпеременные можно легко получить обращением стрелок в концептном графе. Для обозначения подпеременных будем использовать последовательность имен обычных переменных, заключенную в фигурные скобки: $\{x.y.\cdots.z\}$. Это позволяет не давать подпеременным собственных имен, а получать и идентифицировать их с помощью обычных переменных. Например, концепт `Customers` характеризуется двумя подпеременными ранга 2: `{OrderParts.o.c}` и `{OrderOperations.o.c}`. Первая из них принимает значения из концепта `OrderParts`, тогда как вторая из концепта `OrderOperations`. Заметим, что имена переменных в данном случае совпадают, но соответствующие пути в графе восстанавливаются путем указания начальных концептов.

Важно понимать, что переменные всегда принимают одно значение из соответствующего надконцепта (либо не одного, если это разрешено, т.е. null), тогда как подпеременные многозначны и принимают одновременно несколько значений из соответствующего надконцепта. Например, заказ всегда характеризуется только одним заказчиком (переменная *c* концепта `Orders`), но несколькими частями из `OrderParts` и несколькими операциями из `OrderOperations`.

Семантически концепт описывается с помощью набора *элементов* данных или записей, которые он включают как множество: $C = \{r_1, r_2, \ldots, r_m\}$. Каждый элемент данных это комбинация других элементов данных, называемых надэлементами или надзаписями, взятых по одному из каждого подконцепта: $r = \langle r_1, r_2, \ldots, r_n \rangle$, где $r \in C = \langle C_1, C_2, \ldots, C_n \rangle$ и $r_i \in C_i$, $i = 1,2,\ldots,n$. Альтернативно, каждый элемент данных можно представить как набор значений всех переменных, которые они принимают из соответствующих доменов: $r = \langle x_1 = r_1, x_2 = r_2, \ldots, x_n = r_n \rangle$. Каждый элемент данных может рассматриваться как отдельный объект, состоящий из своих надэлементов, а также как надэлемент, который сам используется

для определения подэлементов из подконцептов. Таким образом, каждая запись это одновременной характеризуемый объект и характеристика для описания других объектов. В этом смысле элемент может быть описан семантически с помощью своих подэлементов, т.е. объектов, где он используется в качестве значения переменной. Для этого можно использовать подпеременные, которые однако принимают не одно, а множество значений, поскольку каждая запись может включаться в более чем одну подзапись.

При таком определении семантика может быть представлена ациклическим графом (рис. 2), где каждый элемент связан с множеством родительских надэлементов и дочерних подэлементов. Синтаксис в этом случае налагает ограничения на возможные определения новых элементов, разрешая использовать только по одному элементу из каждого надконцепта.

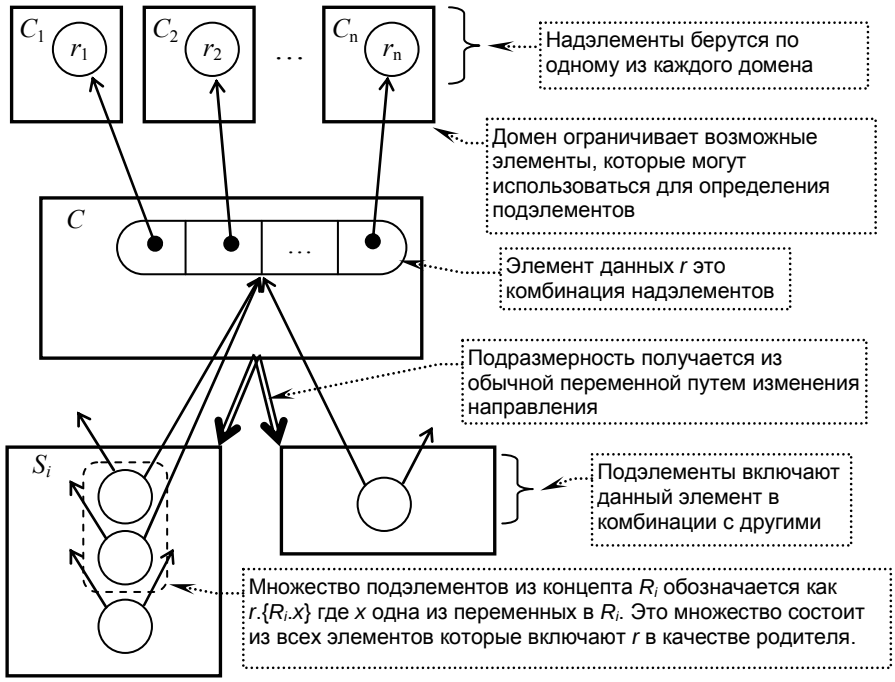

Рисунок 2. Семантика определяется тем, как элементы данных связаны друг с другом.

Один способ интерпретации такой семантики состоит в том, что каждый новый элемент характеризуется с помощью уже существующих элементов. Такая характеризация осуществляется путем присваивания значений атрибутам объектов. Если атрибуты меняются, то соответственно меняется и семантика записи. Отметим, что характеризуемые объекты одновременно являются характеристиками (значениями атрибутов) для других объектов. Например, каждый заказ на предприятии (рис. 1) характеризуется одним конкретным заказчиков и одним конкретным адресом (предполагается, что этот адрес отличается от адреса, которым характеризуется сам заказчик). С другой стороны, этот же объект-заказ может характеризовать объекты из подконцептов, в данном примере, отдельные части заказов (`OrderParts`) и отдельные операции, составляющие заказы (`OrderOperations`).

Такую характеризацию можно понимать также как позиционирование объектов в пространстве. В этом случае каждый объект имеет определенное положение относительно других объектов, которое задается его надэлементами. Эти составляющие объекта понимаются как координаты в пространстве, задаваемом соответствующим концептом. Отметим, что такое позиционирование в пространстве является иерархическим и многомерным. Многомерность

связана с одновременным позиционированием по нескольким координатам. Иерархичность связана с тем, что координата сама может быть объектом со своими собственными координатами.

Группирование данных

Кроме характеризации элементов с помощью друг друга, существует также интерпретация концептно-ориентированной семантики с помощью отношения включения. Она основана на предположении, что надэлемент играет организующую роль множества или группы по отношению к своим подэлементам. Например, каждая запись заказа организует все части этого заказа в одно множество, так же как и один адрес собирает в одну группу всех заказчиков, с ним связанных. Интересно, что с точки зрения отношения характеризации это означает, что объект принадлежит одновременно всем значениям своих атрибутов, играющих роль множеств в которые объект включен. Например, точка в координатной системе как объект принадлежит одновременно всем значениям своих координат (рис. 3). Скажем, если точка имеет координаты 5 и 10, то значит она принадлежит соответственно объектам 5 и 10, расположенных на осях. Плоскость в этом случае является подконцептом по отношению к своим осям. А точки на плоскости являются подобъектами по отношению к своим координатам на осях.

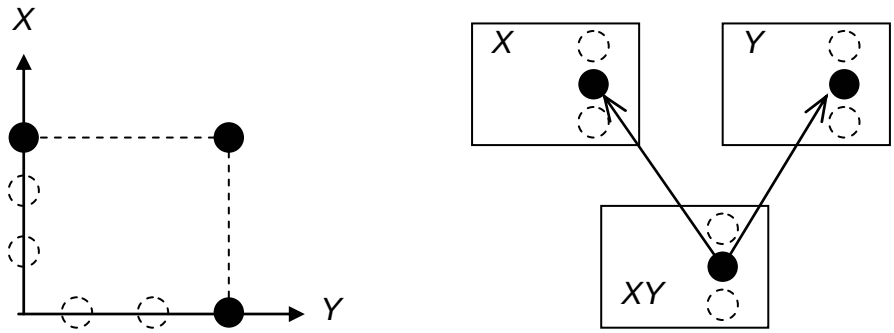

Рисунок 3. Интерпретация точек на плоскости как элементов данных.

Такое одновременное включение в несколько родительских объектов эквивалентно многомерности пространства. Другим важным свойством концептно-ориентированного подхода является то, что сами значения координат могут быть объектами со своими собственными координатами, что позволяет описать не только многомерность, но и иерархичность. Например, значения на оси координат могут быть сами многомерными объектами с координатами в других пространствах, а точки на плоскости могли бы использоваться как координаты для позиционирования подобъектов Отметим, однако, что в общем случае иерархичность и многомерность могут переплетаться весьма сложным образом, когда их нельзя разделить.

С точки зрения группирования и агрегирования данных важно, что каждый элемент играет роль группы, категории или множества для своих подэлементов, находящихся в подконцептах. Соответственно, сам элемент является членом групп, представленных своими надэлементами. В частности, все записи в базе данных прямо или косвенно принадлежат группе, соответствующей верхнему элементу. Используя такую структуру данных, можно рассматривать каждый элемент данных как множество, состоящее из других элементов, которые в свою очередь имеют свой состав. Таким образом, состав любого элемента данных можно раскрыть, двигаясь вниз в иерархии. Отметим, что состав имеет семантику включения в множество, т.е. логического ИЛИ.

Например, предприятие состоит из множества адресов, каждый из которых состоит из множества заказчиков, которые в свою очередь состоят из заказов и т.д. В отличие от этого, двигаясь вверх в иерархии, можно раскрыть двойственный по смыслу состав элементов, имеющий семантику логического И (т.е. раскрываются комбинации элементов). Например, каждая часть заказа состоит из одного заказа, которому эта часть принадлежит, и одного продукта, который ей соответствует, а заказ и продукт в свою очередь являются комбинациями своих надэлементов и т.д. вверх по решетке.

Каждый элемент это не просто однородное множество всех его подэлементов. Его состав разбивается на подмножества, состоящие из подэлементов одного типа, т.е. взятых из одного подконцепта. Эти подмножества называются альтернативными представлениями состава элемента, т.е. элемент данных может быть представлен как состоящий из множества подэлементов одного типа, либо, альтернативно, из множества элементов другого типа:

$r = \{R_1, R_2, \ldots, R_n\}$, где $R_i = \{s \in S_i \mid s.x_i = r\}$

Здесь из каждого подконцепта S_i выбирается подмножество элементов R_i, у которых в качестве непосредственного родителя по переменной x_i находится рассматриваемый элемент $r \in C$. То же самое можно записать как значение соответствующей подпеременной: $R_i = r.\{S_i.x_i\}$. Таким образом, один и тот же элемент r можно рассматривать как состоящий из элементов одного из множеств R_i. Отметим, что каждый из элементов $s \in R_i$, составляющих элемент r, в свою очередь состоит из альтернативных множеств. Этот процесс можно рекурсивно продолжить вниз по иерархии, раскрывая каждый раз состав элементов на альтернативные множества элементов из подконцептов.

Процедура альтернативного раскрытия состава каждого элемента используется для представления данных в виде многомерной иерархии, которое также называется аналитическим представлением. В основе лежит идея, что данные могут рассматриваться на разных уровнях абстракции, начиная с самого верхнего уровня, где вообще нет никаких деталей, и заканчивая нижними уровнями, на которых данные содержат существенно больше информации. Корень иерархии соответствует верхнему элементу в концептно-ориентированной модели. В рассматриваемом примере верхний элемент соответствует всему предприятию как единому целому без рассмотрения его состава. Далее вопрос состоит в том, что необходимо раскрыть содержимое данного элемента. Для этого необходимо указать состав какого типа необходимо раскрыть, что делается путем выбора какого-то подконцепта (либо путем указания подпеременной). В результате верхний элемент представляется как множество подэлементов из указанного концепта. Например, предприятие может рассматриваться как набор категорий товаров, либо как набор адресов. Далее для выбранного элемента из нового представления можно проделать ту же процедуру, т.е. раскрыть его состав, указав какой-то подконцепт. Этот процесс продолжается до тех пор, пока не будет достигнут нужная детальность представления или масштаб.

Состав каждого элемента представляется в виде списка его подэлементов определенного типа. Однако, элементы из этого списка могут быть представлены по разному поскольку характеризуются набором свойств. Таким образом, на каждом уровне представления можно настроить детальность путем выбора того, какие параметры элементов показываются. Этот процесс соответствует раскрытию двойственного состава, поскольку мы движемся вверх в пространстве концептов. Дело в том, что указание свойства объекта означает выбор какого-то надконцепта (а не подконцепта), а само свойство показывается как элемент этого надконцепта. Соответственно, для этого надэлемента также можно выбрать показываемые свойства путем

указания следующего надконцепта и т.д. вверх в решетке концептов. Например, если мы раскрыли состав заказов (`Orders`) в виде его частей (`OrderParts`), то записи из этого концепта могут быть представлены либо с помощью своих идентификаторов, либо путем указания идентификатора продукта, либо еще выше в иерархии путем указания категории этого продукта.

В результате такая процедура позволяет настроить представление с двух точек зрения. Во-первых, можно выбрать детальность и тип состава двигаясь из каждого узла вниз по иерархии. Во-вторых, можно для каждого узла настроить глубину раскрытие его свойств, двигаясь вверх по иерархии. Здесь важно понимать двойственность этих двух процессов. Кроме этого, способ аналитического представления данных совмещает иерархический взгляд на данные в виде дерева и многомерное их представление в виде таблицы. Действительно, каждый узел является локальной таблицей, состоящей из выбранных элементов одного концепта, но с другой стороны, каждый элемент раскрывается в множество других элементов и представляется в виде узла дерева.

Другой важный механизм состоит в возможности вводить ограничения на данные в рамках одного концепта, которые будут автоматически распространяться на все другие концепты. Например, можно выбрать только одну или несколько категорий продуктов, и это ограничение будет автоматически распространено по всей модели данных. В частности, останутся и будут доступны только заказы, продукты, операции, адреса, которые связаны с выбранными категориями. Распространение ограничений в решетке концептов происходит следующим образом. Сперва все ограничения, наложенные на различные концепты, распространяются вниз до нижнего концепта. После этого, если это необходимо, они агрегируются в обратном направлении.

Агрегирование данных

В реляционных базах данных не существует встроенного механизма группирования, который бы действовал постоянно в течение всего времени существования данных. Другими словами, реляционная база данных не знает о том, как управляемые ею данные будут группироваться. Группирование в этом случае осуществляется вручную с помощью средств языка запросов каждый раз, когда это необходимо. Для этого в языке запросов SQL существует специальное ключевое слово `GROUP`. Идея состоит в том, чтобы существующие в таблице записи распределяются по группам таким образом, что каждая группа характеризуется одним и тем же уникальным значением какого-то свойства объекта. Свойство, которое используется для группирования, указывается после ключевого слова `GROUP`, а результатом такого запроса является набор групп. Например, запрос `SELECT * FROM Persons GROUP BY sex` вернет в качестве результата две записи, которые соответствуют группе мужчин и группе женщин. На самом деле такой запрос не закончен и не будет работать, поскольку для полноты необходимо еще указать свойства всей группы, которые необходимо вернуть. Свойства группы вычисляются путем агрегирования каких-то характеристик объектов этой группы. Для этого необходимо указать функцию агрегирования, которая возвращает одно значение свойства группы, вычисленное по всем ее элементам. Например, запрос `SELECT avg(age) FROM Persons GROUP BY sex` вернет средний возраст мужчин и женщин в предположении, что каждый человек в таблице `Persons` характеризуется численным значением возраста, сохраненным в поле `age`. Критерий группирования может быть более сложным и включать вычисляемые свойства. Функция агрегирования также может быть сложным выражением. Например, мы

могли бы разбить всех людей на группы в зависимости от разницы между их датой рождения и датой рождения ребенка.

Такой подход к группированию и агрегированию довольно гибок, поскольку позволяет строить весьма разнообразные схемы, которые удовлетворяют текущим потребностям. Проблема состоит в том, что база данных знает о схеме группирования и агрегирования только на время запроса, тогда как в остальное время данные это просто записи в таблицах. В концептно-ориентированной модели данных предполагается, что записи естественным образом связаны в иерархическую многомерную модель, где у каждой из них есть несколько родительских и дочерних записей, т.е. каждая запись является одновременно и группой для других записей и членом других групп. Важно, что такая организация поддерживается самой базой данной, поскольку эта структура отражает семантику предметной области. Другими словами, задачей концептно-ориентированной базы данных как раз и является поддержание этой структуры связей между записями, а не самих записей как в реляционной модели.

В результате того, что концептно-ориентированная семантика собственно выражается через группирование элементов, задача агрегирования существенно упрощается. Теперь нет необходимости описывать конкретный состав групп путем указания свойства, значения которого должны быть равны для всех записей одной группы. Записи уже заранее принадлежат различным группам и остается только указать какие свойства групп требуется вернуть в качестве результата. В терминах реляционных баз данных это означает, что каждой записи заранее приписаны значения в виде надзаписи, которые могут совпадать и в этом случае считается, что они составляют одну группу.

Группирование в концептно-ориентированной модели данных тесно связано с рассмотрением многозначных свойств, определяемых с помощью подпеременных. Дело в том, что именно то, что такие свойства принимают целое множество значений, мы не можем однозначно охарактеризовать объект. Например, каждое государство как единый объект или запись может характеризоваться множеством регионов, которые его составляют. Но проблема как раз в том, что часто необходимо проводить анализ именно на уровне государства без учета специфики регионов, тогда как исходная информация представлена именно на их уровне. Например, запись о государстве не хранит информацию о его площади, но такая информация имеется для каждого отдельного региона. В этом случае решением как раз и является применение подпеременных, которые возвращают информацию из подконцептов, но ассоциируют ее с надконцептом. Но поскольку значения подпеременных неоднозначны, а представляются множеством, то часто применяются функции агрегирования. Так, для государства можно вычислить его площадь как сумму площадей всех регионов. Для этого надо применить сумму к свойству, возвращающему состав государства.

При таком подходе каждый концепт (и все его элементы) характеризуется двумя типами свойств: однозначные, соответствующие обычным переменным, направленным вверх по направлению к надконцептам, и многозначные, соответствующие подпеременным, направленным вниз по направлению к подконцептам. Но последние для того, чтобы интерпретироваться как действительно свойства объекта, должны быть преобразованы в однозначную форму путем агрегирования. После применения функции агрегирования к такому свойству получаем только одно значение, которое характеризует данных объект. Такая операция формально соответствует проекции. Ее результатом является уменьшение размерности всего пространства и соответственно упрощение представления. Например, таким способом можно из двумерного пространства получить одномерное. В результате все координаты вдоль одной размерности будут представлены только одним числом, а сама размерность удалена.

Для агрегирования данных необходимо, прежде всего, выделить набор объектов, которые необходимо охарактеризовать, используя информацию из объектов более низкого уровня. Собственно эти объекты представляют группы. После этого информация с нижних уровней «подтягивается» до уровня данных объектов и преобразуется в нужную форму. Идея концептно-ориентированного подхода к этой проблеме состоит в том, что группирование встроено в модель и следует из базовой семантики данных. Таким образом, для каждого характеризуемого объекта на нижнем уровне уже определен его состав в виде набора подобъектов. Остается только найти их, получить их свойства и далее вычислить агрегированное значение.

Можно взглянуть на этот процесс с другой стороны, а именно, с точки зрения объектов нижнего уровня, свойства которых поднимаются вверх и одновременно агрегируются. Каждый концепт в модели данных содержит набор объектов со своими свойствами. Если же такой уровень рассмотрения является слишком низким и содержит лишние детали, тогда механизм агрегирования позволяет его повысить. Для этого необходимо выбрать один из надконцептов и сгруппировать записи из данного концепта вокруг соответствующих надзаписей с последующим агрегированием нужных свойств. Этот процесс можно продолжить, т.е. выбрать далее следующий надконцепт и еще больше уменьшить детальность представления. В конечном итоге можно вообще свести представление к одной верхней записи с набором агрегированных свойств. Например, можно представить предприятие как набор его усредненных свойств, которые вычисляются по всем остальным записям в базе данных.

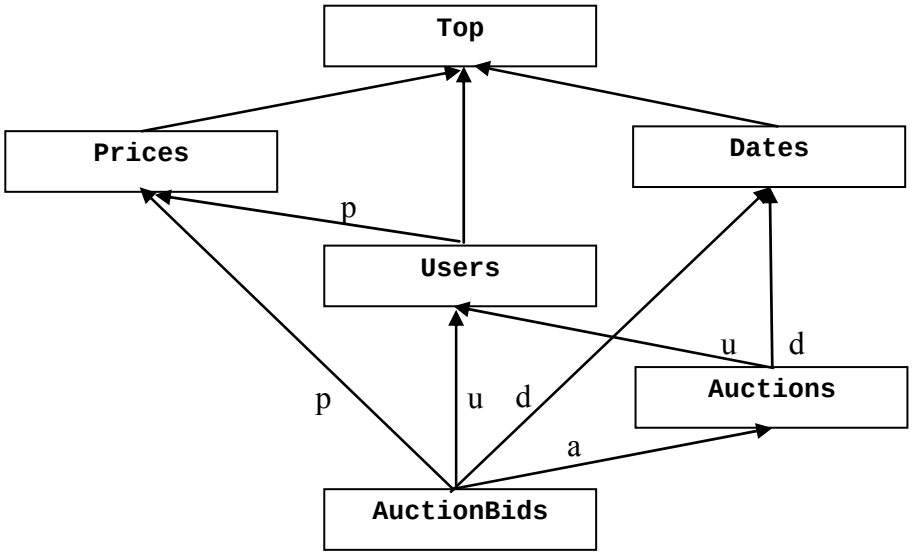

Рисунок 4. Агрегирование осуществляется вдоль размерностей в схеме базы данных.

Рассмотрим как осуществляется группирование и агрегирование на примере, показанном на рис. 4. Предметная область состоит из концепта пользователей или участников (`Users`), аукционов (`Auctions`), всевозможных дат (`Dates`) и цен (`Prices`). Подконцепт `AuctionBids` содержит заявки на покупку, которые выставляются участниками на аукционе с указанием цены. На верхнем уровне предметная область может быть представлена как набор дат, цен, аукционов или участников. На нижнем уровне она представляется как набор заявок, которые характеризуются вышеуказанными параметрами. Каждый аукцион характеризуется набором заявок и набором участников.

Чтобы вычислить среднюю цену заявки, необходимо просто для всех элементов из концепта `AuctionBids` найти среднее значение свойства цена (размерность `p`) следующим образом:

```
{auctionBids : AuctionBids}<avg(auctionBids.p)>
```

Здесь в фигурных скобках описывается набор элементов, а в угловых указывается функция агрегирования с параметров цены заявки (язык запросов описан в работе [12]). Этот запрос вернет всего один элемент с одним полем, которое будет содержать среднюю цену всех заявок в базе денных.

Обычно, однако, требуется вернуть агрегированные значения, вычисленные для набора групп. Например, было бы интересно найти среднее значение цены для каждой существующей даты, что можно сделать следующим образом:

```
{date : Dates}<avg(date.{AuctionBids.d}<p>)>
```

Здесь внутренний запрос `date.{AuctionBids.d}<p>` возвращает набор заявок для каждой отдельной даты, рассматриваемой как группа. В угловых скобках указывается, что необходимо вернуть не всю заявку, а только ее цену. Далее эта группа заявок, характеризующаяся одной датой, передается в качестве параметра функции агрегирования. Среднее значение далее возвращается в качестве параметра данной даты. В результате запрос возвращает набор средних цен аукциона, вычисленных для каждой даты. Тот же самый шаблон можно использовать для вычисления средних цен по всем участниками или по всем аукционам.

Часто агрегирование выполняется по более крупным группам, например, это могут быть группы пользователей, дни недели или диапазон цен. Таким группы могут заранее существовать в базе данных, как например, группы пользователей, либо определяться непосредственно в самом запросе. Если группы определены в базе данных, то применяется метод, описанный выше, т.е. просто перечисляются элементы группы (возможно, с ограничениями), и для каждой из них находится набор подэлементов, после чего выполняется агрегирование свойств.

Для создания групп внутри запроса необходимо понимать, что группа это просто набор элементов, который создается и существует как обычный концепт. Единственное отличие состоит в том, что такой концепт существует в контексте запроса, а не в глобальном контексте базы данных. Например, можно разбить всех участников на «любителей» и «профессионалов» в зависимости от количества проведенных аукционов (мало или много) и далее для каждой группы вычислить среднюю цену заявки.

Заключение

В работе был описан механизм группирования и агрегирования в концептно-ориентированной модели данных. В основе этого механизма лежит многомерная иерархическая организация пространства данных, свойственная данной модели. В частности, каждый элемент данных понимается как группа или категория для элементов данных нижнего уровня. В этом случае предметная область может естественным образом рассматриваться с разным уровнем детализации. Агрегирование информации тогда понимается как ее распространение с нижних детальных уровней вверх в направлении более абстрактных концептов в модели данных. В основе этого процесса лежит понятие подразмерности, которое двойственно понятию размерности. В частности, если размерность направлена всегда вверх и принимает только одно значение, то подразмерности направлены вниз (к более детальным уровням) и принимают множество значений. Агрегирование выполняется путем указания подразмерности с помощью пути в графе концептов. После этого к возвращаемому множеству значений применяется

функция агрегирования, а ее результат рассматривается как свойство объекта. Данный подход является весьма общим и может быть применен в самых различных областях, где информация представляется с помощью концептно-ориентированной модели. В частности, описанный механизм лежит в основе логического вывода, который выполняется путем распространения исходных ограничений вниз в пространстве концептов с последующим агрегированием вверх в направлении целевых атрибутов. Этот подход можно также использовать в сложных зигзагообразных запросах к данным, который требуют движения в различных направлениях в структуре концептов (попеременно вверх и вниз). Главным преимуществом является простота и общность представления синтаксиса и семантики, которые позволяют развить самые разные подходы к представлению и обработке информации в базе данных.

Библиография:

[1] Tim Berners-Lee, James Hendler and Ora Lassila, The Semantic Web, Scientific American, May 2001.
[2] A. Berson and S.J. Smith, Data warehousing, data mining, and OLAP, New York, McGraw-Hill, 1997.
[3] Codd, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM 13 (6), 377-387.
[4] Michael C. Daconta, Leo J. Obrst, Kevin T. Smith: The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management, John Wiley & Sons
[5] Dieter Fensel, Ontologies: a silver bullet for knowledge management and electronic commerce.
[6] Bernhard Ganter, Rudolf Wille, Formal Concept Analysis: Mathematical Foundations, Springer, 1999.
[7] Hector Garcia-Molina, Jeff Ullman, Jennifer Widom, Database Systems: the Complete Book, Prentice Hall, 2003.
[8] Gruber, T.R. (1993). A Translation Approach to Portable Ontology Specification. Knowledge Acquisition 5: 199-220.
[9] Pawlak, Z. (1982). Rough sets. Internat. J. Comput. Inform. Sci., 11, 341-356.
[10] Peter Pi-Shan Chen: The Entity-Relationship Model. Toward a Unified View of Data. In: ACM Transactions on Database Systems 1/1/1976 ACM-Press ISSN 0362-5915, S. 9-36
[11] Rabe von Randow, Introduction to the Theory of Matroids, Lecture notes in economics and mathematical systems 109, Springer, 1975.
[12] A. Savinov, Principles of the Concept-Oriented Data Model, Research Report, Institute of Mathematics and Informatics, 2004, 54pp. http://conceptoriented.com/savinov/publicat/imi-report'04.pdf
[13] A. Savinov, Analytical space for data representation and interactive analysis, Computer Science Journal of Moldova, Vol. 10, No. 1, 59-80, 2002.
[14] А. Савинов. Применение иерархических многомерных пространств для описания структуры предметной области, Proc. Conf. On Applied Mathematics & Informatics — AM&I'98, Chisinau, Moldova, August 25-27, 1998

Об авторе:

Александр Савинов, д.т.н., старший научный сотрудник

Лаборатория интеллектуальных систем
Институт математики и информатики
Академия наук Молдовы
ул. Академией 5, MD-2028 Кишинев

Knowledge Discovery Team
Fraunhofer Institute for Autonomous Intelligent Systems
Schloss Birlinghoven, Sankt-Augustin, D-53754 Germany

E-mail: savinov@conceptoriented.com

Home page: http://conceptoriented.com/savinov