Язык запросов при организации данных в виде графа. Концепция и реализация

К.Б. Потапов

Аннотация. В статье рассматривается язык запросов, предназначенный для доступа к данным, организованным в виде графа, и способный к интеграции в язык SQL. Приводятся общие требования к такому языку, основные положения его реализации, а также краткое описание разрабатываемой в ИПИ РАН реализации, получившей название GSQL.

Ключевые слова: язык запросов, сетевая база данных, объектно-ориентированная база данных.

Введение

Не секрет, что в современном мире наибольшую популярность завоевали реляционные базы данных, имеющие в основе реляционную алгебру. Теория и практика их создания и применения наиболее развита, и такие базы можно встретить во всех отраслях и применительно к самому широкому спектру задач. Это и ERP-системы [1], без которых немыслимо эффективное функционирование крупных компаний, и различные специализированные продукты, и даже средства, логическая структура данных в которых с трудом приводится к реляционной модели.

В то же время авторы многих проектов отказываются от реляционных баз данных, ссылаясь на ряд их недостатков. Предполагаются базы данных с иной структурой (NoSQL), или, по крайней мере, реляционные базы данных с отказом от языка SQL, ставшего стандартом для классических реализаций. [2]

Имеют место и утверждения, предполагающие обреченность реляционных баз данных [3], однако, по крайней мере, признается, что и реляционная модель, и язык SQL являются важными и полезными инструментами, но ни в коем случае не универсальными. Это касается как проблем, присущих реализациям реляционных баз данных (например, масштабируемость при стремительном росте числа серверов), так и банального неудобства доступа к данным, логическая структура которых многое теряет при приведении к реляционной модели.

Почему же реляционные базы данных [4], тем не менее, остаются популярными и используются даже там, где, казалось бы, их использовать просто неудобно? В первую очередь благодаря наличию стандартизованного инструмента доступа и манипулирования данными в виде языка SQL [5] огромному множеству сопутствующих средств разного назначения и степени универсальности, но ориентированных именно на работу с реляционными базами (в качестве примера можно привести конструкторы отчетов).

Реляционные базы просто предлагают наилучший компромисс между простотой, гибкостью, производительностью и т.д. Обычно проще настроить под задачу качественную реляционную базу данных и быть уверенным в наличии сопутствующих решений, нежели рисковать использовать решения, не настолько проверенные временем и масштабами использования. Однако сейчас ситуация начинает меняться, и первые тому признаки – направление NoSQL.

Но нужно ли отказываться от опыта, накопленного за годы разработки и использования как самих реляционных СУБД, так и всех со-

путствующих продуктов? Не разумнее ли попытаться объединить новые структуры, более подходящие для конкретных задач, с реляционным подходом и языком SQL, чтобы и облегчить переход на новые модели организации данных.

Ответ очевиден. Более того, некоторые системы *предполагают* совмещение различных подходов не просто на переходном этапе, а как наилучший вариант организации данных – просто потому, что должны оперировать разнотипными данными.

Примером могут служить ERP-системы и среды бизнес-моделирования [6]. Первые традиционно соответствуют реляционной модели, и для них она весьма эффективна. Вторые же, хотя в ряде случаев и используют реляционную базу данных, логически соответствуют иной структуре – как раз той, что будет рассматриваться в данной статье. При этом и ERP-система, и среда бизнес-моделирования предназначаются для управления организацией: ERP-система имеет все данные об эффективности текущих бизнес-процессов, а среда бизнес-моделирования позволяет улучшить их [7], снизить риски [8] и, при наличии связи, соответственно настраивать модули ERP-системы в непрерывном режиме, а не один раз при внедрении, как это обычно делается сейчас. Потребность в такой связи при динамике современной жизни очевидна.

Данная статья рассматривает задачу взаимной интеграции реляционной базы данных и базы с иной структурой, данные в которой организуются в виде графа. Основным требованием при этом является преемственность решения относительно существующего стандарта: языка SQL как средства доступа и манипулирования данными.

1. Обобщенная модель данных для граф-подобной структуры

Организация данных в виде ориентированного графа, о которой пойдет речь далее, является естественной для множества систем. Именно граф как средство отображения связей между элементами хорошо подходит в виде дополнения к реляционной модели, которая такие отображения вынуждена собирать вместе и хранить всем множеством в таблице, что не наглядно и требует масштабных операций соединения при извлечении данных.

Модель, подразумевающая организацию данных в виде графа, традиционно именуется сетевой. Рассматриваемая в рамках статьи модель организации данных достаточно универсальна, на ее основе может быть наращена как полноценная объектная модель, так и многие другие. Сама же по себе такая модель естественна как для сред бизнес-моделирования [9] (где именно таким образом обычно реализуется программный интерфейс доступа к данным), так и для геоинформационных систем [10].

Как и ориентированный граф, рассматриваемая граф-подобная обобщенная модель (далее ГПОМ) содержит два основных типа элементов: узлы (вершины) и дуги (направленные ребра). Далее используется терминология, приближенная к объектной модели: в узлах графа находятся *объекты*, обладающие набором *атрибутов* (иначе *свойств*). Каждый атрибут имеет название, поясняющее его смысл, и тип значения. Набор значений атрибутов полностью описывает все характеристики объекта. Дуги же графа далее именуются *связями* между объектами. Связи также могут иметь набор атрибутов, полностью их характеризующий.

В рамках ГПОМ предполагается, что список возможных атрибутов един для всех объектов и – отдельно – для всех связей. Данное предположение является ключевым в согласовании ГПОМ с реляционной моделью. При этом от конкретного типа объекта (как и связи) зависят допустимые значения того или иного аргумента, т.е. предположение никак не ограничивает предметную область? просто некоторые атрибуты для некоторых типов объектов (связей) могут иметь исключительно пустое (NULL) значение.

Как объекты, так и связи разбиваются на типы (соответственно, тип является одним из атрибутов, присутствующих как в списке атрибутов связей, так и в списке атрибутов объектов), причем обычно удобно выделять хотя бы два уровня типов. От типа, как уже было сказано, зависит набор значений, допустимых для каждого атрибута. Кроме того, типами двух объек-

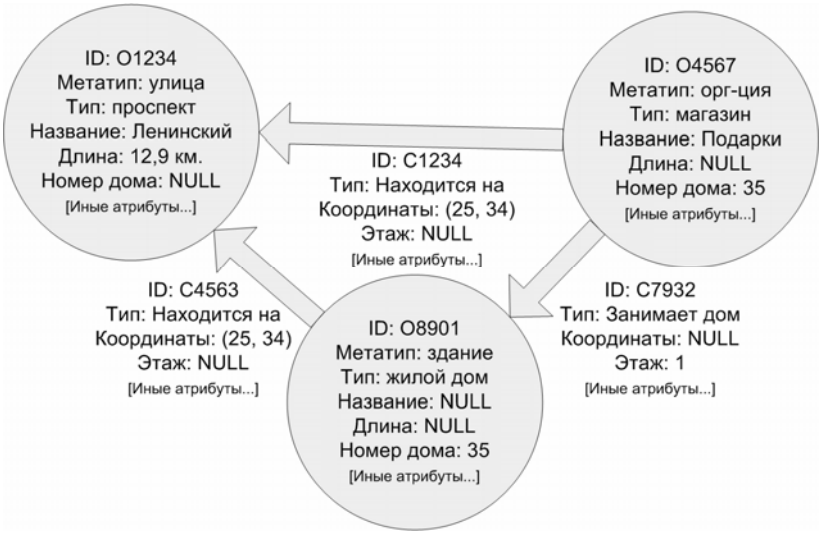


Рис. 1. Пример данных, организованных в соответствии с ГПОМ

тов определяются и типы связей, допустимых между этими объектами. Кроме типа, очевидна необходимость наличия атрибута - идентификатора, позволяющего гарантированую различать объекты и связи даже в случае идентичности других атрибутов. На Рис. 1 представлен пример геоинформационных данных, организованных согласно описанным выше положениям.

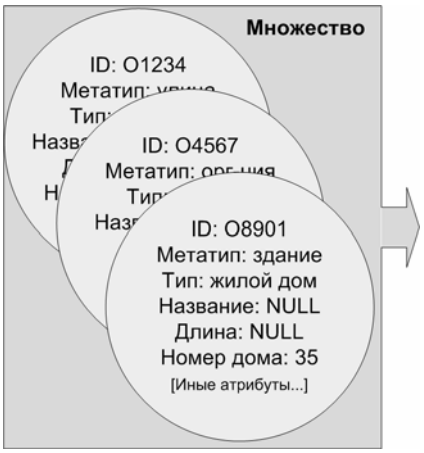
При всем удобстве ГПОМ для представления данных, естественно отметить сложность формулирования запроса к такой базе, присуще. сетевой модели. Отчасти по этой причине в настоящее время доступ и манипулирование такими данными обычно реализуются при помощи программного интерфейса. Конечно, использование языка программирования предоставляет широчайший спектр возможностей, но и накладывает серьезные ограничения на квалификацию оператора, на трудозатраты для организации связи с реляционными базами данных и т.д. Стандартизованные решения здесь не работают. Однако потребность в стандартизации интерфейсов безусловно существует, например, для поддержки сотрудничества организаций в реализации общих бизнес-процессов и управления ими, а в конечном итоге для перехода к сервис-ориентированной архитектуре [11].

Поэтому рассматриваемая задача не имеет универсального решения, а все имеющиеся реализации предназначаются для осуществления вполне конкретных операций между двумя определенными продуктами и не претендуют на нечто большее. Таким образом, альтернативного SQL языка для ГПОМ не существует, не говоря уже о возможности объединения с реляционной моделью в рамках одного способа доступа и одной системы.

2. Приведение данных из обобщенной модели к табличной форме

В постановке задачи четко обозначена преемственность решения относительно SQL, подразумевающая легкую его интеграцию в существующие реляционные базы данных и взаимодействие с сопутствующими программными продуктами. Потому очевидно оформление решения в виде языка запросов, построенного на основе SQL. Более того, такой язык должен интегрироваться в SQL-запрос, а не просто быть сходным по синтаксису. Поэтому, прежде чем начать его описание, следует упомянуть основные свойства языка SQL. Следует также заметить, что далее используется терминология, присущая SQL, а не реляционной модели, поскольку речь идет об интеграции с реальными системами, а не теоретическими базами данных (как известно, SQL не в полной мере соответствует реляционной модели).

В основе языка SQL, по крайней мере, на логическом уровне, лежат таблицы [12]. Данные



ID	Название	Длина	Номер дома	[Иные атрибуты]
O1234	Ленинский	12,9 км.	NULL	…
O4567	Подарки	NULL	35	…
O8901	NULL	NULL	35	…

Рис. 2. Преобразование множества в таблицу

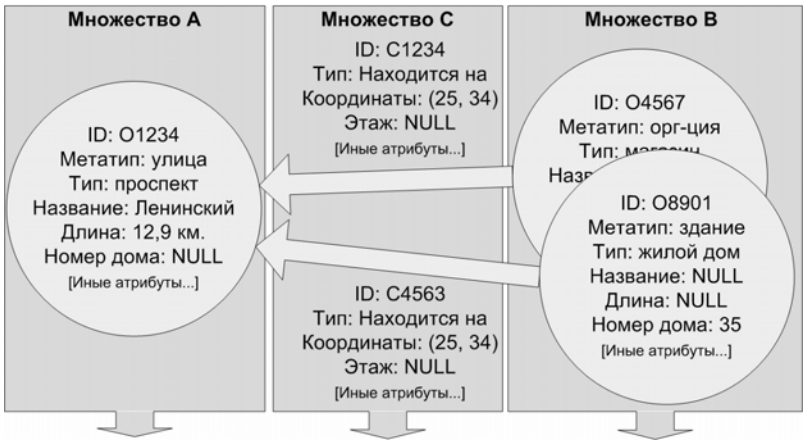
таблицы могут быть физическими, а могут быть полученны в результате подзапросов, соединений и иных операций, могут быть даже виртуальными, но логика, лежащая в основе SQL все равно оперирует исключительно таблицами. Посредством соединений и фильтрации таблиц формируется выборка [13]. Путем расшифровки ячеек результирующих таблиц, так или иначе ссылающихся на ячейки исходных, осуществляется модификация данных.

Все операции, связанные с соединением таблиц, располагаются в частях WITH и FROM SQL-запроса – собственно, и отвечающих за формирование источника данных. Остальные же части работают с одной большой таблицей (по крайней мере, логической), сформированной по итогам выполнения данных частей. Причем любая SQL-таблица обладает наперед известным количеством и типами столбцов, а строки могут «прирастать» или удаляться в ходе операций.

Поскольку таблицы являются основой SQL, целесообразно свести задачу именно к формированию таблиц путем обработки данных, организованных в соответствии с ГПОМ, набором особых операторов. Результат же в табличной форме может быть подставлен как вместо любой таблицы для последующего применения к нему операций соединения и прочих операций WITH- и FROM-частей, так и в качестве источника данных для всех остальных частей запроса. В последнем случае, при отказе от основных реляционных операций, будет получен SQL-подобный язык для доступа и манипулирования данными, организованными в соответствии с ГПОМ, который можно использовать во всех средствах работы с реляционными СУБД, имеющих дело только с результатом выполнения запроса.

Самый простой способ формирования таблицы на основе объектов (и связей) графа следующий. Заметим, что атрибуты во многом эквивалентны столбцам таблицы. Далее, основываясь на предположении о единстве списка атрибутов для всех объектов (связей), автоматически получим, что любое множество, состоящее из объектов (или связей), может быть представлено в виде таблицы, где столбцами являются атрибуты, каждая строка соответствует одному объекту (связи) множества, а ячейка – значению соответствующего атрибута для соответствующего объекта (Рис. 2). Предположив единство списка атрибутов в рамках всех элементов, т.е. и для объектов, и для связей одновременно, можно объединить и их в рамках одного множества, однако это уже идет в разрез с логикой ГПОМ, по которой объекты и связи – совершенно разнородные элементы.

В языке SQL, как видно, например, из операции соединения, связи выражаются путем помещения связанных данных на одну строчку таблицы. Применим это правило и к разрабатываемому языку: пусть объекты, соединенные заданными связями, попадут на одну строку таблицы. Объекты, как и ранее, будем группировать по множествам – тогда связи будут существовать между объектами в различных множествах. Чтобы различать атрибуты, будем



A.ID	A.Длина	A.Номер дома	C.ID	C.Коорд.	B.ID	B.Длина	B.Номер дома
O1234	12,9	NULL	C1234	(25, 34)	O4567	NULL	35
O1234	12,9	NULL	C4563	(25, 34)	O8901	NULL	35

Рис. 3. Преобразование множеств связанных объектов в таблицу

именовать множества так, как именуются таблицы. Результат применения такого подхода показан на Рис. 3.

Как видно из рисунка, описанный принцип позволяет представить множества связанных объектов в виде таблицы с сохранением всех особенностей SQL. Сохраняются и особенности именования (*множество.атрибут* вместо *таблица.столбец*), и дублирование связанных данных при наличии более чем одной связи. Кроме того, атрибуты, характеризующие сами связи, также могут быть помещены в таблицу между атрибутами соответствующих объектов. При отсутствии связей с каким-то объектом, по аналогии с SQL, в соответствующих множествах атрибуты принимают NULL-значения (как при внешних соединениях в SQL). Отличить в таком случае «пустой» объект можно по пустому значению одного из обязательных атрибутов, например, идентификатора.

Имея набор попарно связанных множеств и пользуясь описанным принципом, можно преобразовать извлекаемые данные в табличную форму. Естественно, такая таблица скорее будет логической, нежели реально существовать в памяти: допустимых атрибутов в реальных системах сотни, если не тысячи. Однако оптимизация выполнения запросов находится за рамками данной статьи. Важным же фактом является автоматическое связывание множеств, полученных друг из друга в результате иных операций – например, объединения и т.д. Такие множества удобно считать связанными «связью эквивалентности», когда на одной строчке оказываются одинаковые объекты. Такая связь, естественно, не имеет атрибутов и существует на логическом уровне для удобства преобразования в табличный вид любых множеств, участвующих в запросе.

3. Основные операнды и операции языка запросов для обобщенной модели

Из сказанного очевидно, что основным операндом разрабатываемого языка должны быть множества объектов. Связи при этом не рассматриваются отдельно, а всегда привязаны к объектам, что соответствует логике ГПОМ. В таком случае необходимо выделение нескольких *базовых множеств* объектов, на основе которых и будут производиться дальнейшие операции. Очевидно, что в совокупности базовые множества должны покрывать множество всех объектов базы данных, при этом они вовсе не обязаны быть попарно непересекающимися.

Конкретные наборы, очевидно, зависят и от предметной области, и от программной реализации базы данных.

Точно так же, как и в случае с таблицами, базовые множества (однако, помимо ряда наперед заданных) допустимо позволить формировать оператору, просто объединяя в них объекты на основании запросов (точнее, не полноценных запросов, а набора операций над иными базовыми множествами), как представления (view) в реляционных СУБД и т.п. Как и в SQL, следует предусмотреть операцию присвоения множеству псевдонима для дальнейшей ссылки на него. Названия базовых множеств удобно рассматривать как преопределенные псевдонимы.

Кроме стандартных операций над множествами, необходимость которых очевидна, столь же понятно и естественно наличие операции фильтрации, позволяющей исключить из множества объекты по результату вычисления для их атрибутов определенного выражения. Подобная операция реализуется частью WHERE SQL-запроса, но следует помнить, что роль разрабатываемого языка состоит в формировании таблицы – дальше уже таблицей оперирует язык SQL, т.е. «внутри» разрабатываемого языка конструкции SQL недоступны иначе как в подзапросах, возвращающих таблицы.

Далее, если в SQL циклическая конструкция появилась не сразу и не во всех диалектах, то для выборки данных из графа наличие такой конструкции необходимо, поскольку между двумя нужными узлами может быть наперед неизвестное количество переходных. Такая конструкция должна позволять задавать правила перехода и при этом, желательно, по самой своей стурктуре избегать возможности зацикливания.

Потому реализовывать ее предлагается по следующему принципу. В основе такой операции, как и прочих, лежит множество. Однако подразумевается, что работа ведется не с самим этим множеством, а целым блоком операций, приводящих к его формированию. В основе этого блока лежат ссылки на множества по их псевдонимам (не обязательно на базовые множества – это могут быть и множества, псевдонимы для которых объявлены вне блока, в ранее исполняемой части запроса).

Один из таких псевдонимов указывается в качестве *множества цикла*. Важно заметить, что множество цикла определяется именно как псевдоним, а не как ссылка на множество, и потому одно и то же множество, если ему присвоено два псевдонима, может в блоке операций быть как множеством цикла, так и обычным множеством, что добавляет операции гибкости.

При первой итерации цикла блок вычисляется как обычно, давая в результате некое множество R_1. Затем, на второй итерации, R_1 подставляется в блоке на все те места, где указан псевдоним, объявленный как множество цикла, и блок операций выполняется снова. Можно заметить, что, в принципе, множеством цикла можно объявлять и несколько псевдонимов – тогда один и тот же результат итерации будет подставляться на место каждого.

Пусть на второй итерации вычисление блока дает в результате множество Q_2. Но теперь, как и на последующих итерациях, полученное в результате вычислений множество Q_i, $i=2,3,\ldots$ уже не считается результатом итерации, предварительно из него исключаются все объекты, входившие в результаты предыдущих итераций: $R_i = Q_i \setminus U_i$, $U_i = (\bigcup_{k=1}^{i-1} R_k)$, $i=2,3,\ldots$. Цикл прекращается, когда на очередной итерации будет получено пустое множество, т.е. $R_i = \varnothing$. Благодаря исключению на каждом шаге всех объектов, полученных на предыдущих итерациях, зацикливания произойти не может – ведь в базе данных конечное число объектов.

Для удобства описанная конструкция дополняется условием выхода из цикла, проверяющимся для объектов множества, полученного в результате каждой итерации, и специальным атрибутом уровня, добавляемым к объектам и соответствующим номеру итерации, на которой они были добавлены в множество. Кроме того, в качестве результата операции цикла возвращается множество, состоящее из объединения как результатов одной или нескольких итераций (последней, например, или всех), так и начального значения множества, на которое ссылался псевдоним, указанный как множество цикла. В языке возможные вариан-

ты регулируются опциями и константами. Дополнительно операцию цикла поясняет Рис. 4.

Следующей необходимой операцией над множествами является связывание, как раз и отбирающее из имеющихся между объектами те связи, которые будут представлены в итоговой таблице. Кроме того, данная операция может и создавать связи – например, обозначить декартово произведение множеств. Такие связи, однако, по понятным причинам не будут содержать атрибутов, как и связь эквивалентности.

Однако, поскольку связывание должно органично вписываться в блок операций над множествами, необходимо определить выходное множество объектов для нее. И лучшим решением видится совмещение данной операции с операцией перехода между объектами – последней, требуемой в рамках языка.

В результате операция связывания определяется следующим образом: во-первых, она содержит набор *условий соединения*, по которому отбираются или создаются связи между объектами двух множеств (Рис. 5, этап 1). Во-вторых, на основе отобранных связей формируется множество-результат операции: в него попадают те и только те объекты, одного (указанного) из множеств, связи между которыми и объектами «другого» множества были отобраны (Рис. 5, этап 2). Наконец, для результирующей таблицы запоминаются связи между множеством-результатом операции и «другим» множеством (Рис. 5, этап 3), что более корректно при представлении операции связывания именно как операции над множествами.

Из описания операции связывания очевидно, что для нее необходимо добавление ряда опций, таких как, например, предписание возвращать исходное множество целиком с запоминанием связей; или наоборот, не запоминание их вообще. Кроме того, именно в рамках этой операции можно присвоить псевдоним множеству связей для именования их атрибутов в результирующей таблице.

Таков набор необходимых операций для языка запросов, оперирующего множествами объектов и преобразующего результат выборки в таблицу. Как видно, в результирующей таблице сохраняется прямая связь и с объектами (связями), «породившими» ту или иную ячейку,

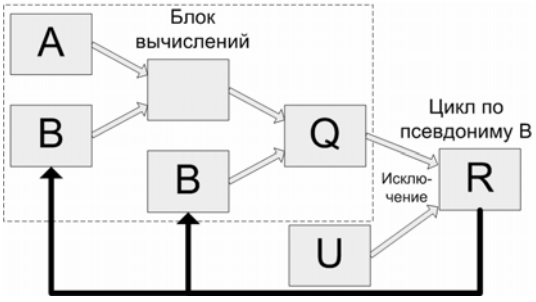


Рис. 4. Графическая иллюстрация операции цикла

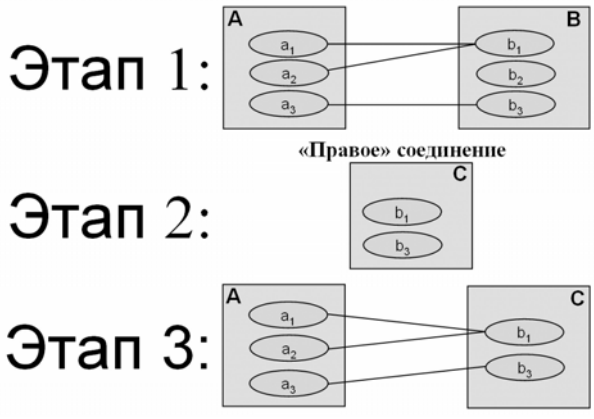
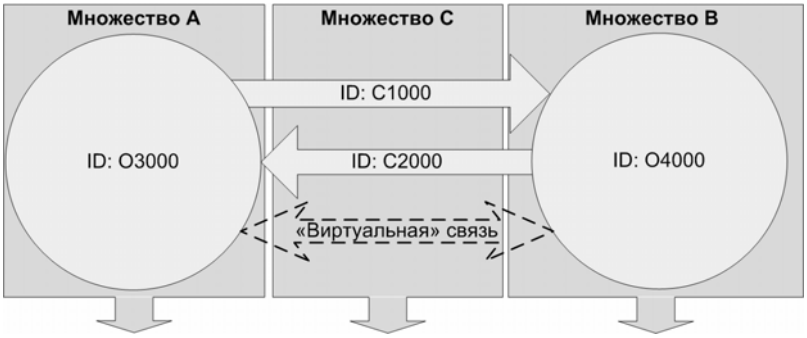


Рис. 5. Порядок действий при операции связывания

и с множествами, а значит, на основе этих операций можно реализовать и модификацию данных, и редактирование базовых множеств. Расширение набора операций возможно, однако на данном этапе развития языка представляется избыточным.

4. Условия соединения и базовые принципы отбора связей

Как и в случае с объектами, связи также удобно объединять во множества, тем более в таком виде они попадают в результирующую таблицу. Однако следует заметить, что когда пользователь определяет условия соединения объектов, то из самой формулировки названия конструкции видно, что речь идет скорее о выражении, об условии, нежели о формировании множества. По этой причине более удобно работать с множествами связей, но оформить их выборку как набор условий (каждое из которых формирует «базовое» множество удовлетворяющих ему связей), между собой соединенных



Вар. 1

A.ID	C.ID	B.ID
O3000	C1000	O4000

Вар. 2

A.ID	C.ID	B.ID
O3000	C1000	O4000
O3000	C2000	O4000

Рис. 6. Два варианта преобразования множеств связанных объектов в таблицу при наличии нескольких отобранных связей

логическими операторами, равносильными стандартным операциям над множествами. Так, логическое «И» равносильно пересечению множеств, логическое «ИЛИ» - объединению, а исключение естественно так и именовать: «за исключением».

В принципе, между двумя объектами в графе существует ограниченное количество связей, потому возможно определение и оператора отрицания. Однако не следует забывать, что логика операции связывания допускает и создание «виртуальных» связей, как было замечено выше – связей без атрибутов, не существующих в графе непосредственно. Поскольку такие связи не существуют в рамках графа, но также могут учитываться пользователем, применение оператора отрицания нежелательно.

Как видно из принципов формирования результирующей таблицы, в ней связи между объектами множеств не являются направленными просто потому, что логика SQL не предусматривает очевидного выражения подобного свойства. По сути, выходное множество связей просто может быть представлено в виде пар связанных объектов, где первый объект принадлежит, для определенности, «левому» множеству операции, а второй – «правому». Каждой паре, если связь в ней не виртуальная, сопоставляется также набор значений атрибутов, а направление может быть учтено в условиях соединения.

При этом возможны два пути вывода множества связей, определяемые опциями операции связывания. Если пользователю необходимо просто знать, что между двумя объектами существует хотя бы одна связь указанного типа – в множестве следует оставить только одну связь для каждой пары объектов (для определенности – с минимальным идентификатором; возможны и иные критерии), отдавая предпочтение «реальным» связям, которым сопоставлен набор атрибутов (Рис. 6, вар. 1).

В противном же случае, когда пользователю важно вывести данные о всех связях, существующих в множестве остаются все пары, хотя «виртуальные» ввиду невозможности различить их из-за отсутствия значений атрибутов всё равно «сжимаются» до одной и, при условии наличия «реальных» связей, исключаются из множества вовсе. Тогда для каждой связи соответствующая строчка таблицы дублируется (Рис. 6, вар. 2).

В основу отбора связей можно положить два принципа формирования множеств: на основе отношения между объектами и на основе свойств самой связи. К первому случаю относятся такие характеристики, как направление («связь направлена из объекта левого множест-

ва к объекту правого» – этот случай следует выделить особым коротким оператором, имеющим наглядную форму стрелки); возможно, ряд особых вариантов связей графа; а также набор операторов для создания «виртуальных» связей (оператор построения декартова произведения, например). Такие *операторы связи* не требуют никаких дополнительных данных.

Второй случай, который удобнее именовать *свойством связи*, предназначен для «реальных» связей с атрибутами и осуществляет отбор связи по принципу сравнения определенного атрибута (а может, некой вычисляемой характеристики) с вычисленным или заданным значением (как вариант, равенства этому значению). Значение это, или вычисляющее его выражение, и будет дополнительными данными, необходимыми для свойства связи, что и отличает его конструкцию от конструкции оператора связи.

Таким образом, наряду с функциями (о которых, как и о выражениях языка вообще, нет необходимости говорить отдельно ввиду аналогий с SQL), операторы связи и свойства связи возможно в рамках языка доопределять, настраивая его под логику и потребности конкретной системы. Также можно заметить, что перечисленные пять операций (три стандартные плюс конструкции операторов и свойств связи) полностью покрывают потребности в плане отбора связей, ведь фильтрация множеств является одним из вариантов применения свойства связи, а циклическое выполнение и тем более связывание для связей применять не имеет смысла.

5. Язык GSQL и его реализация

Такова базовая концепция языка, предназначенного дополнить SQL в плане доступа и манипулирования данными, организованными в соответствии с ГПОМ. Воплощением этой концепции является язык GSQL (от «Graph SQL» – «SQL для графа»), разрабатываемый в настоящий момент в ИПИ РАН [14]. Как подтверждает опыт его проектирования и создания экспериментальной реализации для среды бизнес-моделирования ARIS, практическое воплощение озвученной выше концепции имеет ряд подводных камней и тонких моментов. Однако в условиях фактического отсутствия альтернатив решений невозможно произвести сравнение с иными подходами.

Хотя в настоящей момент испытания реализации только начинаются, первые результаты показали значительное упрощение создания выходных документов на основе шаблонов (задача, давшая импульс началу разработки языка) [15]. Несмотря на то, что семейство ARIS лидирует на рынке средств бизнес-моделирования, изначально заполнение шаблонного документа данными требовало серьезных навыков программирования. Применение же GSQL позволило отделить описание выборки данных от вставки строк в определенные места шаблонного документа образом, привычным для пользователя по работе с реляционными СУБД. В масштабной системе, где логически данные представлены и в форме графа, и в реляционной форме, обеспечить единообразие формирования отчетов, очевидно, крайне важно.

Как и было заявлено в начале статьи, реализация GSQL возможна и в виде дополнения к существующему диалекту (более того – процедурному расширению также) языка SQL, и в виде отдельного языка. При этом вне зависимости от варианта реализации, через интерфейсы (например, ODBC) возможно обеспечить функционирование приложений, изначально предназначавшихся для реляционных СУБД (особенно это касается конструкторов отчетов).

Поскольку именование атрибутов множеств как внутри запроса, так и в результирующей таблице осуществляется по аналогии с именованием столбцов таблиц в SQL, оригинальный язык придется дополнить лишь конструкцией, вводящей в части WITH и FROM запроса GSQL-блок. Возвращать такая конструкция (на логическом уровне) будет таблицу, аналогичную полученной рядом соединений (т.е. каждое множество в результирующей таблице с точки зрения SQL будет реальной таблицей, подвергшейся соединению).

Что касается модификации данных, то аналогии с SQL приводят к запросам на удаление объектов и изменение атрибутов объектов. Запросы INSERT, изначально осуществляющие вставку данных в таблицу, в применении к GSQL означали бы создание объекта в базовом множестве, что, вообще говоря, не всегда и не

совсем верно (объект создается в базе данных вообще; в базовое множество помещаются уже созданные объекты). По этой причине применение запросов INSERT для GSQL-блоков следует запретить. Редактирование же графа (в т.ч. и в плане создания новых объектов и связей) при необходимости таких инструкций возможно осуществить через полностью новые конструкции, аналогичные тем, что в SQL предназначены для вставки таблицы. Что, однако, вполне естественно и не оказывает существенного влияния на основное достоинство предлагаемого языка – возможность объединения в одной СУБД как реляционной модели, так и ГПОМ, с универсальным и преемственным механизмом доступа и модификации данных.

В качестве самостоятельного языка GSQL реализуется путем добавления конструкций SQL, не относящихся к частям WITH и FROM. При этом источником данных служит результирующая таблица, полученная в результате одних лишь GSQL-операций, с которыми остальные части запроса оперируют обычным образом.

Следует, однако, заметить, что реализация в рамках существующего диалекта SQL *объединяет* [16] два подхода к извлечению данных – соответствующий реляционной алгебре и соответствующий ГПОМ, т.е. теории графов [17]. Это увеличивает гибкость языка даже в случае, когда предполагается выборка одних лишь данных, организованных в виде графа.

В перспективе GSQL можно и должно дорабатывать согласно практическими потребностями. Очевидная (по аналогии с SQL) необходимость в процедурной надстройке, как упоминалось выше, удовлетворяется встраиванием GSQL в процедурные расширения SQL тем же образом, что и в один из диалектов языка.

Таким образом, использование GSQL, во-первых, позволяет развить существующие системы управления, связав единообразным, стандартизированным способом все необходимые их компоненты и максимально использовав при этом накопленный опыт. В случае же геоинформационных и иных типов систем с организацией данных в виде графа, применение GSQL на первом этапе позволяет обеспечить доступ к данным через привычный по реляционным СУБД и достаточно удобный интерфейс, а в перспективе – построить более качественные системы, совмещающие обе модели (реляционную и ГПОМ).

Литература

1. Дэниел О'Лири. ERP системы. Современное планирование и управление ресурсами предприятия. Выбор, внедрение, эксплуатация /Пер. с англ. Ю.И.Водяновой. – М.: ООО «Вершина», 2004 – 272 с.
2. NoSQL Relational Database Management System: Home Page //Strozzi.it. URL: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page (дата обращения: 08.05.2011).
3. Tony Bain. Is the Relational Database Doomed? //ReadWriteEnterprise. URL: http://www.readwriteweb.com/enterprise/2009/02/is-the-relational-database-doomed.php (дата обращения: 08.05.2011).
4. Джеффри Ульман, Дженнифер Уидом. Основы реляционных баз данных. – М.: Издательство «Лори», 2006. – 384 с.
5. Грофф Дж., Вайнберг П. Энциклопедия SQL. – СПб.: Питер, 2003. – 896 с.: ил.
6. Киселев А.Г. Основы организационного моделирования (бизнес-моделирования) //Сайт «Оргструктура.ру». URL: http://orgstructura.ru/?q=organizational-modelling-methodology (дата обращения: 08.05.2011).
7. Андерсен Бьерн. Бизнес-процессы. Инструменты совершенствования /Пер. с англ. С.В. Ариничева /Науч. ред. Ю.П. Адлер. – М.: РИА «Стандарты и качество», 2003. – 272 с.
8. Костогрызов А.И., Степанов П.В. «Инновационное управление качеством и рисками в жизненном цикле систем» – М.: Изд-во ВПК, 2008 – 404с.
9. Ковалев С., Ковалев В., Риб С.И., Кремлева И. В., Дворников А., Смирнова Н. //Сайт компании БИТЕК. URL: http://www.betec.ru/index.php?id=06&sid=01 (дата обращения: 08.05.2011).
10. Скворцов А.В., Поспелов П.И., Крысин С.П. Геоинформатика в дорожной отрасли (на примере IndorGIS). – М.: Изд-во МАДИ, 2005. – 11 с.
11. Сорокин А. В. Предприятие и Интернет следующего поколения / А. В. Сорокин // Системы и средства информатики, вып. 18 (дополнительный выпуск). М.: Наука. – 2008. – С. 86-117.
12. ISO/IEC 9075:1992, Database Language SQL – July 30, 1992. URL: http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt (дата обращения: 08.05.2011).
13. SQL Grammar. URL: http://www.h2database.com/html/grammar.html (дата обращения: 08.05.2011).
14. Потапов К.Б. Язык запросов для системы управления регламентами // Первая школа молодых ученых ИПИ РАН. Сборник докладов. – М.: ИПИ РАН, 2010. – с. 4-14

15. Потапов К.Б. Управление регламентами как необходимый модуль систем управления в производственной и экономической сферах. Язык запросов // Труды 12-ой международной научно-технической конференции «Кибернетика и высокие технологии XXI века» (C&T-2011), Воронеж 11-13 мая 2011 – т. 1 – с. 334-341
16. Потапов К.Б. Организация данных как графа. Специальный язык запросов GSQL // Системы компьютерной математики и их приложения Выпуск 12. Издательство Смоленского государственного университета, 2011 – с. 116-118
17. Зыков А. А. Основы теории графов. – М.: «Вузовская книга», 2004. — 664 с.

Потапов Кирилл Борисович. Программист 2 категории ИПИ РАН. Окончил Московский авиационный институт в 2007 году. Автор трех публикаций. Область научных интересов: общие вопросы проектирования и реализации системы управления регламентами. E-mail: pokibor@yandex.ru