Оперативная интеграция данных на основе XML: системная архитектура BizQuery*

К.В. Антипин, А.В. Фомичев, М.Н. Гринев, С.Д. Кузнецов, Л.Г. Новак, П.О. Плешачков, М.П. Рекуц, Д.Р. Ширяев

Аннотация. При возрастающем интересе к решению проблемы интеграции распределенных и разнородных источников данных виртуальный подход представляется перспективным и многообещающим. В своей общей постановке эта проблема исключительно сложна, и до сих пор ее решению уделялось недостаточное внимание. Однако быстрое развитие XML – многофункционального формата представления данных – и языков запросов к XML-данным, таких как XQuery, позволяет по-новому взглянуть на старую проблему.

В статье содержится описание общей архитектуры BizQuery – системы виртуальной интеграции, основанной на модели данных XML. В системе локальные источники единообразно отображаются на глобальную схему как ее представления в терминах XML и UML на основе использования декларативных языков XQuery и UQL. Обсуждаются вопросы отображения схем, оптимизации, декомпозиции и выполнения запросов в системе виртуальной интеграции.

Ключевые слова: виртуальная интеграция данных, XML, XQuery, UML, оптимизация запросов, декомпозиция запросов, обработка запросов, трансформация схем

1. Введение

Интеграция разнородных гетерогенных данных является одной из старейших задач в области разработки баз данных и информационных систем. Кратко, проблема может быть сформулирована следующим образом. Предположим, имеются несколько гетерогенных источников данных, которые каким-то образом связаны на логическом уровне. Имеется задача предоставить программное обеспечение, которое обеспечивало бы возможность унифицированного доступа к этим данным, как будто бы они имели единое логическое и

* Проект выполнялся исследовательской группой MODIS Института системного программирования РАН (http://www.ispras.ru/groups/modis/modis.html) в сотрудничестве с международной компанией ATS (www.atssoft.com) при поддержке РФФИ (гранты 02-01-01088-а и 02-07-90300-в).

физическое представление. Мы не будем обосновывать очевидную важность этой проблемы.

Существуют два фундаментальных подхода к решению этой проблемы. Первый подход связан с построением хранилищ данных, когда интегрируемые данные из разных источников трансформируются в соответствии с целевой моделью данных и помещаются в одну локальную базу данных. По поводу этого подхода имеется обширная литература, современное состояние дел описывается, например, в [1].

Второй подход связан с понятием виртуальной интеграции гетерогенных источников данных, когда данные не материализуются в локальной базе данных, а используется промежуточное программное обеспечение, которое транслирует пользовательские запросы в подзапросы к источникам и формирует окончательный результат. Краткий обзор эволюции систем, использующий виртуальный подход, включая мультибазы данных [2] и федеративные базы данных [3], может быть найден в [4]. Подход этих систем был связан, прежде всего, с интеграцией данных с четкой структурой (хотя структура могла быть разная). Следующим этапом было возникновение систем интеграции на базе медиаторов [5], которые создавались на основе полуструктурированных данных [6]. Возникновение XML [7] и сопутствующих технологий (XSLT [8], XQuery [9]) вызвало всплеск новых разработок по тематике виртуальной интеграции [10], [11] и т.д.

Система виртуальной интеграции BizQuery на основе технологий XML [7] и UML [12], обсуждаемая в этой статье, является результатом работы исследовательской группы, которая на протяжении последних четырех лет занимается вопросами исследования и разработки методов управления XML-данными. Основные возможности BizQuery заключаются в следующем:

- интегрированный доступ к нескольким источникам данных, которые могут быть реляционными или содержать XML-данные;
- использование XML как для внутреннего представления данных, так и для представления результата;
- представление глобальной схемы интегрированных данных как в терминах UML, так и в терминах XML;
- возможность формулировки запросов к интегрированным данным с использованием декларативных языков запросов UQL (разработка группы [4]) и XQuery в терминах UML и XML соответственно;
- развитая обработка запросов, включая оптимизацию запросов; декомпозицию запросов на частичные запросы, адресуемые к индивидуальным источникам данных; формирование окончательного результата с потенциальным выполнением соединений и трансформаций данных.

Основная цель проекта BizQuery состояла в проведении исследований по проблематике подхода виртуальной интеграции и реализации “приближенной к жизни” системы виртуальной интеграции на основе XML-технологий.

Участники проекта пытались показать, что создание практической системы на основе XML технологий с учитом прошлого опыта разработки систем виртуальной интеграции вполне возможно. В статье рассматривается общая архитектура системы и концепции, которые были заложены в ее основу, объясняется, почему были выбраны именно такие подходы к решению задачи.

Основная часть статьи организована следующим образом. В разделе 2 описывается общая архитектура системы BizQuery. В разделах 3 и 4 обсуждаются детали двух основных компонентов системы — BizQuery Mapper и BizQuery Integration Server. В разделе 5 приводятся некоторые результаты, демонстрирующие производительность системы при ее испытаниях на тестовых наборах данных и запросов. Раздел 6 является заключением статьи.

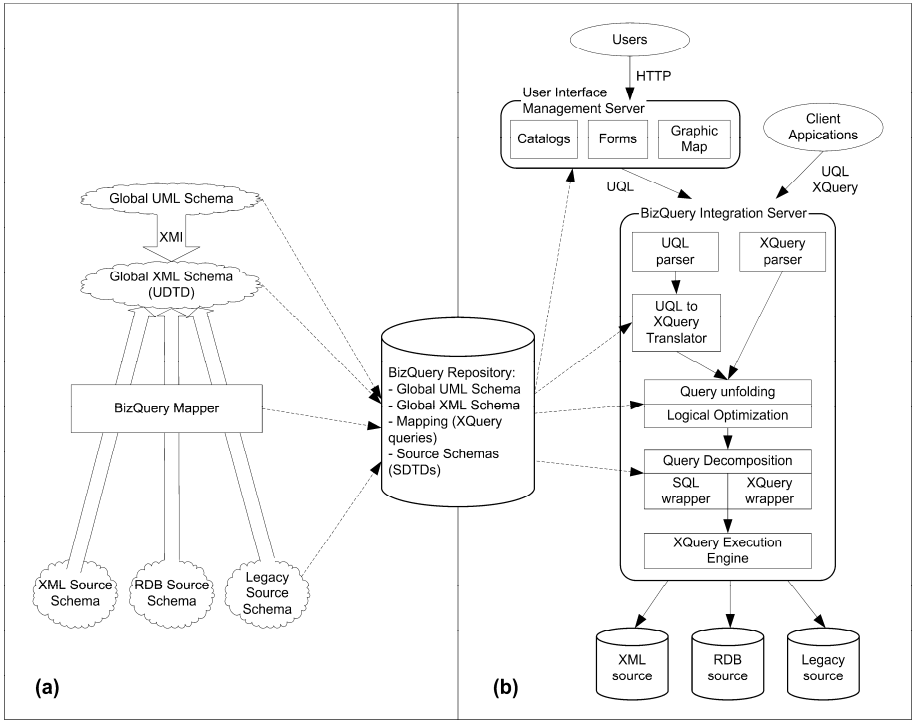

Рис 1. Структура системы: (a) – подготовительная фаза; (b) – рабочая фаза

2. Архитектура и структура системы

В этом разделе рассматривается архитектура системы BizQuery и поясняется назначение основных компонентов. Прежде всего, выделяются две фазы использования системы: подготовительная фаза (Рис. 1(a)) и рабочая фаза, на которой система обрабатывает запросы к интегрируемым данным (Рис. 1(b)).

Прежде чем адресовать запросы интеграционной системе, необходимо выполнить ряд подготовительных действий, таких как создание глобальной схемы инте-грируемых данных в терминах UML и XML, сбор информации о схемах инте-грируемых источников и построение отображения схем источников на глобальную схему. Эти подготовительные действия происходят при выполнении подготовительной фазы. Метаданные, созданные на этом этапе, помещается в BizQuery Repository, и затем они используется во время выполнения запросов. Необходимо заметить, что во время развертывания системы происходит работа только с метаданными, то есть со схемами источников и глобальной схемой, а не с данными.

2.1. Фаза развертывания системы

BizQuery предоставляет два интерфейса доступа к данным: в терминах глобальной UML-схемы и в терминах глобальной XML-схемы. Процесс внедрения начинается с конструирования глобальной UML-схемы, которая служит для моделирования предметной области и представляет собой диаграмму классов UML. При конструировании этой схемы во внимание должны приниматься одновременно два фактора. Во-первых, схема должна соответствовать требованиям будущих пользователей. Во-вторых, схема должна быть адекватной доступным или предполагаемым источникам данных, которые требуется интегрировать. За отслеживание этих факторов отвечает специалист, выполняющий подготовительную фазу. Затем эта схема автоматически преобразуется в XML-документ в формате XMI [13], который представляет собой глобальную XML-схему. Необходимо отметить, что глобальная XML-схема является ключом для функционирования системы, не важно, каким образом эта схема была произведена. Иначе говоря, для интеграции данных в терминах XML совершенно не обязательно создавать UML-схему, если есть возможность сразу предоставить XML-схему.

От интегрируемых источников на этом этапе требуется только схема хранимых данных. Поскольку для системы интеграции нужна, прежде всего, структурная информация об источниках, для хранения схемы использовался формат DTD. В настоящее время происходит миграция на Relax NG[14]. Для XML-источников схема данных должна быть предоставлена вручную, для реляционных источников схема получается автоматически средствами системы.

Завершающий и наименее тривиальный этап фазы развертывания состоит в построении отображения схем локальных источников на глобальную XML-схему. Выполнение этого процесса поддерживается компонентом системы BizQuery Mapper, который обсуждается в разд. 3.

2.2. Архитектура времени выполнения

Подсистема BizQuery времени выполнения содержит два основных компонента: BizQuery Integration Server (BQIS) and User Interface Management Server (UIMS). BQIS отвечает за обработку запросов, сформулированных на

языках UQL или XQuery. Однако все UQL-запросы транслируются в XQuery-запросы, и реальное выполнение запроса производится в терминах XML. Эта трансляция возможна благодаря наличию метаданных, поскольку, с одной стороны, эти метаданные описывают исходную модель, а с другой стороны, сами представляются в формате XML.

В прямо сформулированный или полученный из UQL-запроса XQuery-запрос подставляются представления локальных источников, хранимые в BizQuery Repository (заметим, что в результате этой подстановки запрос переформулируется в терминах схем локальных источников, и структура запроса существенно усложняется). После этого запрос оптимизируется логическим оптимизатором путем применения правил перезаписи, что приводит к значительному упрощению и "улучшению" структуры переформулированного запроса. Это один из наиболее важных шагов обработки запроса. Обсуждение применяемых методов содержится в разд. 4.

Далее, оптимизированный запрос декомпозируется в набор частичных запросов (по прежнему на XQuery), каждый из которых формулируется в терминах локальной схемы соответствующего локального источника данных (по одному частичному запросу на один локальный источник). Каждый частичный запрос транслируется соответствующей "оберткой" (wrapper) на язык запросов, понимаемый локальным источником (в настоящее время поддерживаются обертки для SQL и XQuery). Трансляция на XQuery тривиальна, но пере-формулировка произвольного XQuery-запроса к реляционным данным в SQL-запрос не очень проста. Этот вопрос обсуждается в статье позже.

После декомпозиции запрос разбивается на набор частичных запросов к локальным источникам и так называемую "межисточниковую часть" (т.е. часть запроса, для выполнения которой нужны данные одновременно от нескольких локальных источников), которая должна выполняться самой системой интеграции, в случае BizQuery за это отвечает компонент, называемый XQuery Execution Engine.

Выше мы привели лишь краткую характеристику основных компонентов BQIS и шагов обработки запросов, адресованных к системе интеграции. Детали приводятся в разд. 4.

BQIS реализует API для взаимодействия клиентских приложений с интеграционным сервером посредством адресации XQuery и UQL запросов, то есть реализует низкоуровневый интерфейс доступа к данным, который, вообще говоря, не пригоден для конечных пользователей. Компонент UIMS призван сгладить разрыв между пользователем и BQIS. Он предоставляет три графических интерфейса пользователя доступа к данным в терминах UML. *Catalogs* обеспечивает навигационный интерфейс доступа к данным, предоставляя пользователю возможность просматривать существующие экземпляры классов UML-модели и переходить по ссылкам от одного экземпляра к другому. Два других интерфейса — *Forms* и *Graphic Map* — декларативные. Они позволяют перемещаться по UML-модели и накладывать

условия на атрибуты экземпляров класса. В обоих случаях в результате действия пользователя генерируется UQL-запрос, который адресуется BQIS, а полученный результат отображается на экране. Поскольку пользователь оперирует только понятиями UML-модели, выразительных средств UQL достаточно для формулирования запроса.

UIMS реализован в виде Web-приложения и обладает развитыми возможностями настройки, благодаря использованию XSLT. Графический интерфейс пользователя генерируется автоматически по UML-модели, хранящейся в BizQuery Repository.

2.3. Декларативные языки запросов системы BizQuery

В соответствии с двумя уровнями интеграции данных (XML и UML), поддерживаются два языка запросов интегрированных данных, а именно XQuery и UQL. Язык XQuery, развиваемый консорциумом W3C, был выбран как будущий стандарт языков запросов XML-данных.

В системе BizQuery пользователь с помощью XQuery может оперировать двумя типами сущностей: виртуальными документами, которые стоят за глобальной XML схемой (посредством указания ключевого слова virtual в функции document, например `document("virtual:foo.xml")`), и реальными документами, которые представляют собой действительно существующие документы XML-источников или таблицы реляционной базы данных (посредством указания ключевого слова real, например `document("real:sql/foo")`). В первом случае пользователь оперирует виртуальным документом или представлением, за которым стоят запросы к реальным документам. Во втором случае пользователь работает с сущностями интегрируемых источников (поскольку вся обработка данных производится в терминах XML, реляционные таблицы путем тривиального отображения представляются в виде XML документов).

В ходе работы над средствами манипулирования данными была осознана потребность в создании языка запросов, который бы оперировал в терминах UML-модели. Этим языком стал UQL. Он служит для выборки экземпляров класса (т.е. объектов, которые соответствуют структуре, определенной на UML) построенной диаграммы классов (глобальной UML-схемы в случае BizQuery).

В основу UQL был положен язык OCL [12], который является частью спецификации UML, с несколько измененным и синтаксисом и семантикой. Основное назначение OCL состоит в определении ограничений данных, соответствующих модели (диаграмме классов), в терминах этой UML-модели.

Коротко говоря, UQL позволяет запрашивать экземпляры классов посредством накладывания условий на атрибуты, перемещения по связям между экземплярами классов и использования кванторов всеобщности и существования. Более подробное описание UQL см. в [4].

Вот пример UQL-запроса: “Найти все открытые аукционы (экземпляры класса open_auction), в которых цена больше 40, и покупатель имеет годовой доход не меньше 50000”.

```
context model-id("1803"):
extent(closed_auction)=>
select(c|c.price>"40" and
c!buyer@person=>exist(p|p.income>="50000"))
```

3. BizQuery Mapper

Как отмечалось выше, основная задача фазы развертывания состоит в построении отображения локальных источников данных на глобальную XML схему. Понятно, что глобальная схема данных может быть, вообще говоря, произвольной, и отображение локальных источников на нее является нетривиальным: документы могут подвергаться сложным трансформациям, на их основе могут строиться новые документы, которые затем вновь подвергаются трансформациям и т.д.

Прежде, чем рассказывать про BizQuery Mapper — компонент системы, который занимается решением поставленной задачи, обсудим возможные способы построения отображения между схемами. В общем смысле эта задача может быть переформулирована следующим образом. Пусть имеются некоторые данные, соответствующие схеме A. Как преобразовать эти данные, чтобы они соответствовали схеме B? Имеется несколько методов решения этой проблемы:

1. **Написание программы.** Можно написать программу на языке общего назначения, например, таком как C или Java, которая преобразует данные, соответствующие схеме A, в данные, соответствующие схеме B.
2. **Преобразование вручную.** Пользователь может написать запросы на языке запросов, которые применяются к данным, представленным в схеме A, и возвращают данные в представлении, соответствующем схеме B. Этот метод, как и все последующие, обладает тем преимуществом перед первым методом, что запросы могут подвергаться оптимизации.
3. **Преобразования с использованием высокоуровневых операций.** В этом случае от пользователя требуется описать преобразования схемы A в терминах операций над деревьями, а не узлами, как принято в XML. После этого выражения высокоуровневой алгебры деревьев транслируются в запросы на языке запросов, которые применяются к данным. Результаты запросов должны соответствовать схеме B.
4. **Сопоставление схем.** В этом подходе предполагается, что сама система отыскивает узлы, соответствующие один другому, и пользователь получает возможность более точного выполнения отображения. В результате формируется запрос на языке запросов.

5. **Автоматическое преобразование с использованием высокоуровневых операций.** Этот метод представляет собой вариант метода 3, но преобразование производится автоматически на основе семантики и статистики данных.
6. **Автоматическое сопоставление схем.** Этот метод является вариантом метода 4, но он полностью автоматизируется.

В соответствии с приведенной классификацией BizQuery Mapper базируется преимущественно на методе 3 и, частично, на методе 4. Основным соображением при выборе подхода было то, что у пользователя не должно быть принципиальных ограничений при построении отображения одной схемы на другой. При этом он должен иметь относительно удобный интерфейс и манипулировать высокоуровневыми терминами. Алгоритмы сопоставления схем являются приятным, но не обязательным дополнением. От пользователя не должны требоваться такие дополнительные сведения, как описание семантики данных или статистика.

Итак, схема B получается из схемы A итеративным путем, посредством последовательного применения функций трансформации. Операции трансформации замкнуты относительно множества схем, то есть представляют собой алгебру. Единственным ограничением является то, что результирующая схема должна быть принципиально выводимой из исходной. На практике это означает, что в трансформациях не должно быть зависимости по данным; например, ситуация, где содержимое элемента становится именем нового элемента, недопустима. Ниже приведен список функций трансформации (в основу была положена не минимальность набора, а удобство использования):

1. Простые преобразования
2. Преобразование соединения
3. Преобразование корня
4. Вертикальная проекция
5. Горизонтальная проекция
6. Базовые конструкторы
7. Реструктуризация
8. Преобразования объединения
9. Сложные конструкторы.

С помощью приведенных операций, которые манипулируют поддеревьями, а не узлами или последовательностями узлов XML-документа, многие запросы, такие, как получить исходный документ, в котором атрибут по определенному пути удваивается, выражаются проще, чем на XQuery. Тем не менее, отображение, построенное в высокоуровневых терминах, переводится в XQuery — целевой язык системы интеграции, микрооперации которого проще оптимизировать и выполнять.

BizQuery Mapper, обеспечивающий описанную выше функциональность, реализован в виде отдельного компонента, который предоставляет удобный графический интерфейс в стиле drag-and-drop для проведения трансформаций.

4. BizQuery Integration Server

В этом разделе обсуждается компонент BizQuery, отвечающий за выполнение XQuery- и UQL-запросов. Он состоит из следующих частей: синтаксический анализатор языков UQL и XQuery, транслятор UQL-запросов в запросы на языке XQuery, оптимизатор, подсистема выделения подзапросов, подсистема выполнения XQuery-запросов и подсистема оберток для связи с источниками данных.

Не секрет, что при построении систем виртуальной интеграции приходится сталкиваться с существенными ограничениями производительности таких систем, и как следствие, время ожидания ответа пользователем может быть неудовлетворительным. В качестве основных причин можно указать отсутствие актуальной статистики распределения данных и отсутствие структур данных, позволяющих оптимизировать доступ к данным (то есть индексов). Определенные проблемы связаны также с задержкой передачи данных по сети и их последующим преобразованием во внутренний формат для обработки. Эти проблемы возникают из-за того, что виртуальная интеграционная система не материализует интегрируемые данные.

По всей видимости, системы виртуальной интеграции никогда не смогут достигнуть производительности систем, основанных на хранилищах данных, однако они вполне могут быть пригодны в тех случаях, когда действительно необходимы актуальные данные.

В ходе работы над ядром BizQuery Integration Server были выделены три составляющие ядра, которые, вероятно, являются ключевыми для эффективной обработки запросов (перечислены в порядке убывания важности).

1. Логическая оптимизация на основе перезаписи запросов (преобразование и упрощение);
2. Декомпозиция запросов (выделение максимального частичного подзапроса, адресованного к локальному источнику);
3. Потоковая обработка на стороне сервера и, по возможности, на уровне источников данных.

Далее эти составляющие рассматриваются подробнее, и поясняется, почему они являются ключевыми.

4.1. Логическая оптимизация

Значимость оптимизатора запросов для СУБД трудно переоценить: разница во времени выполнения исходного запроса и оптимизированного запроса может различаться на порядки. В настоящее время во многих промышленных СУБД

используется так называемая оценочная оптимизация (cost-based optimization), то есть выбор оптимального плана выполнения запроса из множества возможных планов производится на основе оценки стоимости выполнения операции. Оптимальным считается тот план, совокупная стоимость операций которого является наименьшей. Другой метод оптимизации основан на правилах эквивалентного преобразования запросов на основе эвристик с целью получения нового запроса, который является эффективнее первого. Такой вид оптимизации носит название оптимизации на основе правил (rule-based optimization), или перезаписи запросов (query rewriting). Именно этот подход применяется для оптимизации в BizQuery. Это связано со следующими основными причинами:

- в виртуальной интеграционной системе отсутствует статистика, на основе которой можно было бы оценить стоимость операций (данные в источниках могут изменяться без ведома системы интеграции);
- обычно пользовательский запрос адресуется к виртуальному документу (т.е. к представлению), после выполнения подстановки тела представления содержит много лишней информации и, таким образом, поддается существенному упрощению.

Особенно важен последний пункт, так как в зависимости от сложности виртуального документа размер запроса может отличаться на порядки.

Перечислим список целей, достигаемых посредством перезаписи запросов в системе BizQuery:

- сокращение объема запроса после подстановки тела представления;
- “выталкивание” (push down) предикатов (как можно более раннее применение предикатов) и устранение избыточных вычислений (например, конструирования элементов до обработки путевых выражений);
- повышение уровня декларативности запроса (например, путем нахождения конструкций, эквивалентных соединениям);
- частичная перезапись запроса, содержащего вызовы рекурсивных функций, в запрос без рекурсии (с использованием схемы данных) с последующей оптимизацией;
- преобразование запроса в более “целенаправленную” форму на основе информации из схемы с возможным устранением избыточных просмотров данных (например, замена метасимвола в путевом выражении именем некоторого XML-элемента).

Необходимо заметить, что поскольку в XQuery отсутствует явное понятие соединения, потребовалось ввести такую логическую (и физическую) операцию, что привело к созданию расширенной модели данных XQuery. Обычно соединение выражается через FLWR-выражение (что навязывает конкретный алгоритм выполнения — соединение путем использования вложенных циклов), и поэтому выделение соединения в виде отдельной операции повышает декларативность запроса, так как становится возможным

использовать другие, зачастую более эффективные алгоритмы. Тем не менее, наличие упорядоченности в XQuery понижает уровень значимости этой декларативности (например, из-за того, что нельзя произвольным образом изменять порядок выполнения соединений).

Этап логической оптимизации на основе перепизаписи крайне важен еще и по следующей причине. Результатом переписывания запроса является запрос вполне определенного (“нормализованного”) вида. Условно, дерево запроса можно поделить на три части: в листьях дерева расположены операции выборки данных с накладываемыми на данные условиями; в центре дерева находятся операции соединения; верхняя часть запроса состоит преимущественно из трансформаций данных (рис. 2a). Подобная нормализация запроса играет важнейшую роль на этапе декомпозиции запросов, который будет рассматриваться ниже.

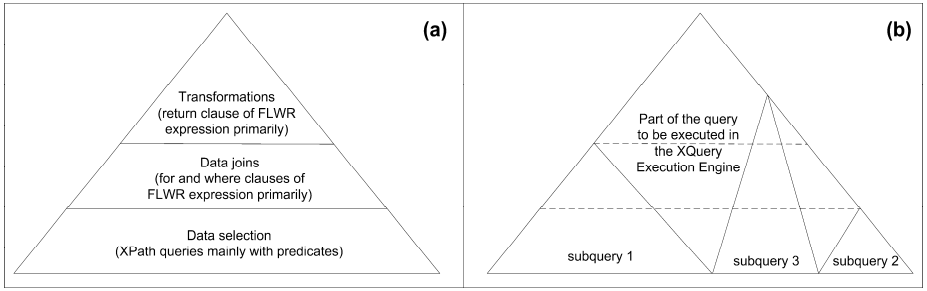

Рис. 2. Схема XQuery-запроса: (a) — после нормализации; (b) — после декомпозиции запроса

Подробнее правила перезаписи XQuery-запросов обсуждаются в [15].

4.2. Декомпозиция запросов

Задача декомпозиции запроса состоит в том, что требуется разбить запрос на части таким образом, что определенные части будут выполняться на стороне источников данных, а оставшаяся часть – на стороне интеграционной системы. В зависимости от типов интегрируемых источников существуют две диаметрально противоположные позиции по этому вопросу. Одна позиция связана с рассмотрением источников, которые предоставляют крайне ограниченные выразительные средства для формулирования запросов, например, имеют в качестве интерфейса HTML-формы [16]. Другая позиция подразумевает, что в источниках поддерживаются развитые, декларативные языки запросов. Система BizQuery ориентирована на второй случай. Более точно, как отмечалось ранее, BizQuery ориентируется на два типа источников — реляционные системы с интерфейсом SQL и системы, поддерживающие язык XQuery. Тогда задача декомпозиции запроса сводится к выделению “максимального” частичного запроса к источнику. Другими словами,

источникам адресуются максимально возможные компоненты исходного запроса, которые они в состоянии выполнить. Например, если в исходном запросе имеются такие потребляющие много ресурсов операции, как соединение или сортировка, относящиеся к одному источнику, то они отдаются в источник при условии, конечно, что источник в состоянии их выполнить. Преимущества выделения максимального подзапроса состоят в следующем:

- Параллельное выполнение максимальных частей исходного запроса (части исходного запроса, адресуемые различным источникам, могут выполняться параллельно).
- Развитые источники данных зачастую способны выполнять запросы быстрее, чем интеграционная система, поскольку в них имеется более полная информация о хранимых данных (например, развитая структура индексов).
- В типичных случаях размер передаваемого по сети результата частичного запроса намного меньше общего размера документа, хранимого в соответствующем источнике данных (в частности, по причине наличия в частичном запросе условий выборки).

Итак, в случае BizQuery задача разбивается на нахождение в дереве запроса максимальных поддеревьев, относящихся к одному источнику, и перевод их на язык запросов, поддерживаемый источником. Оставшаяся часть дерева запроса содержит операции, для выполнения которых требуются данные из разных источников (для краткости будем называть такие операции “кроссдоменными”), и эти операции должны быть выполнены на стороне интеграционной системы (рис. 2b).

Необходимо заметить, что для полноценного выделения частичных запросов крайне важно, чтобы запрос находился в нормальной форме, о которой говорилось выше (рис. 2a). Поскольку листья дерева запроса представляют собой выражения XPath, можно гарантировать возможность их выполнения в источниках данных, поддерживающих соответствующий язык запросов. Выше по дереву находятся операции соединения и полусоединения, которые тоже могут быть переданы для выполнения источникам данных, если их операнды черпаются только из одного источника данных. Таким образом, на стороне интеграционной системы приходится выполнять только те операции соединения и полусоединения, которые являются кроссдоменными (в том числе, если они прямо или косвенно опираются на результаты других кроссдоменных операций). Если в дереве запроса существует хотя бы одна кроссдоменная операция, которая должна быть выполнена до уровня трансформации, то трансформация тоже выполняется на стороне интеграционной системы.

Приведем пример декомпозиции запроса. Предположим, что имеется запрос, содержащий обращения к двум документам одного реляционного источника (схемы документов представлены в табл. 1 в виде DTD); запрос выдает данные об отделах, в которых имеются сотрудники моложе 20 лет.

```
for $d in document("real:sql1/deps")/table/tuple
where some $e in document("real:sql1/emps")/table/tuple[age < 20]
      satisfies $d/id = $e/dep_id
return element dep {$d/name, element additional {$d/address}}
```

document("real:sql1/deps")	document("real:sql1/emps")
<!ELEMENT table (tuple)*> <!ELEMENT tuple (id, name, address)> <!ELEMENT id (#PCDATA)> <!ELEMENT name (#PCDATA)> <!ELEMENT address (#PCDATA)>	<!ELEMENT table (tuple)*> <!ELEMENT tuple (name, age, dep_id)> <!ELEMENT name (#PCDATA)> <!ELEMENT age (#PCDATA)> <!ELEMENT dep_id (#PCDATA)>

Табл. 1. Схемы документов источников данных в виде DTD

Запрос содержит полусоединение (определяется разделами for и where) двух документов одного источника, которое может быть выполнена на стороне источника, а также трансформацию, не поддерживаемую реляционным источником. Ниже показан подзапрос к источнику sql1.

```
select * from deps
where (exists (select * from emps
         where deps.id = emps.dep_id and emps.age <
20))
```

Заметим, что представленную ранее нормальную форму XQuery-запроса можно подвергнуть определенной критике. Очевидно, что существует ряд простых трансформаций, которые могут быть выполнены ранее операций соединения (например, вертикальная проекция на стороне реляционного источника). Однако для выполнения таких трансформаций не требуется слишком много ресурсов, и их выполнение на стороне интеграционной системы не сильно сказывается на производительности. Значительно важнее выделить именно "тяжелые" операции, такие как соединения и сортировки.

4.3. XQuery-процессор

После стадии декомпозиции исходное дерево запроса преобразуется в физический план выполнения запроса, листья которого представляют собой подзапросы в терминах источников. При этом физический план, вообще говоря, может быть достаточно сложным для выполнения. Это связано преимущественно со следующими причинами:

- исходный запрос содержит кроссдоменные операции (например, соединение);

- запрос содержит трансформации, не поддерживаемые источником (например, реляционные источники способны выполнять крайне ограниченный набор трансформаций).

По этим причинам в состав BizQuery входит полнофункциональный процессор подмножества языка XQuery. Введенная разработчиками явная операция соединения реализована в виде физической операции, что позволило повысить производительность системы.

Была адаптирована итераторная модель (называемая также моделью "top-down") выполнения запросов [17], которая широко применяется в реляционных СУБД. Это позволяет не материализовывать промежуточные результаты после вычисления каждой операции. На самом деле, такой подход к организации обработки запросов имеет далеко идущие последствия.

Для многих пользовательских запросах в результате получается не единичный документ, а последовательность из нескольких (часто однотипных) XML элементов. Особенно это характерно для UQL-запросов, в результате которых получается последовательность элементов — экземпляров класса. Часто, особенно при использовании графического интерфейса, пользователю достаточно получить в качестве ответа на запрос первые n XML-элементов и иметь возможность в дальнейшем просмотреть оставшиеся элементы. Это подход соответствует понятию курсора, широко применяемому в реляционных СУБД, и позволяет существенно сократить время отклика системы на запрос пользователя.

Строго говоря, в XQuery-процессоре был реализован так называемый потоковый или конвейерный подход к организации вычислений. В контексте того, что XQuery является функциональным языком (по крайней мере, его подмножество, поддерживаемое в BizQuery), это выразилось в реализации "ленивой" семантики (lazy semantics) XQuery. Вообще говоря, это является расхождением со спецификацией, так как по стандарту XQuery является языком со строгой семантикой (strict semantics). Однако применение ленивой семантики ни в коей мере не сокращает класс вычислимых запросов: если запрос вычислим в системе, использующей строгую семантику, то он вычислим и в системе с ленивой семантикой. Обратное утверждение неверно: некоторые запросы, не вычислимые в соответствии со спецификацией, могут быть вычислены системой BizQuery.

5. Анализ производительности

В этом разделе представлены результаты тестирования производительности системы BizQuery. Насколько известно авторам, в открытом доступе сегодня не существует средств тестирования производительности систем, подобных BizQuery. Поэтому пришлось воспользоваться средством для тестирования XML СУБД — пакетом XMark [18] и адаптировать его для BizQuery.

Документ, соответствующий DTD XMark auctions, был разбит на части, которые были распределены по трем источникам — одному XML-источнику и двум реляционным. В качестве XML-источника выступала программа QuiP [19] компании Software AG, представляющая собой XQuery-процессор над файловой системой. В качестве реляционного источника использовалась СУБД Oracle 8i. Следует заметить, что QuiP не является полноценной СУБД, а также является прототипом. Поэтому производительность этого программного средства была невысокой, что повлияло на распределение данных. Кроме того, чтобы представить определенные части исходного документа в виде реляционных таблиц, пришлось упростить эти части (например, путем избавления от вложенности элементов).

В результате первый реляционный источник содержал 5 таблиц, суммарное количество картежей которых превышало 1,8 миллиона. Второй реляционный источник содержал 3 таблицы с более чем 4,2 миллионами кортежей. Суммарный объем XML-файлов, по которым строились реляционные таблицы, превышает 700Mb. XML-источник содержал 4 файла, общим объемом 5,8Mb.

В табл. 2 показаны запросы, для которых приводятся результаты измерения.

Q1	`for $x in document("real:sql1/item")/table/tuple`
	`where $x/QUANTITY="5" and $x/location="Germany"`
	`return $x`

Q2	`for $y in document("real:sql2/interest")/table/tuple`
	`for $z in document("real:sql2/people")/table/tuple[business= "yes" and city="Moscow"]`
	`for $x in document("real:sql1/categories")/table/tuple [name="all"]`
	`where $y/ref_category=$x/id_category and $y/ref_person= $z/id_person`
	`return ($x, $z)`

Q3	`document("virtual:closed_auction.xml")`

Q4	`for $v in document("virtual:item.xml")/item[location= "United States"]`
	`for $z in document("real:sql1/mailbox")/table/tuple[mail_date="12/11/99"]`
	`where $v/id=$z/ref_item`
	`return element {name($v)} {$v/*[not empty(./text())]}`

Табл. 2. XQuery-запросы, использованные при тестировании

Для измерения производительности использовалась следующая конфигурация. BizQuery Integration Server работал на Pentium-IV 1500Mhz с 512Mb RAM. Для XML-источника данных (QuiP) использовалась такая же машина. Оба сервера реляционных СУБД (Oracle) работали на машинах одинаковой конфигурации — Pentium-III 733Mhz с 256Mb RAM. Все машины работали под управлением Windows 2000.

Запрос Q1 адресуется реальному документу item (таблице реляционного источника 1) и просто налагает условие на данные. Запрос Q2 содержит два соединения, одно из которых выполняется между документами одного источника, а второе соединение — кроссдоменное — между разными реляционными источниками. Запрос Q3 показывает возможность построения виртуального документа closed_auctions.xml целиком. И, наконец, запрос Q4 выполняет соединение между виртуальным документом item.xml и реальным документом mailbox из второго реляционного источника. Результаты выполнения представлены в табл. 3.

Номер запроса	Размер результата (в Kb)	Общее время работы источников	Время работы BizQuery	Общее время выполнения запроса
Q1	23	5,984	0,078	6,062
Q2	12	82,485	1,796	84,281
Q3	3673	67,907	3,093	71,000
Q4	27	52,766	0,938	53,704

Табл. 3. Результаты тестирования производительности BizQuery

Во всех четырех запросах, вне зависимости от общего времени выполнения, чистое время работы BizQuery (включающее оптимизацию и, если необходимо, выполнение кроссдоменных операций) относительно невелико. Это обеспе-чивается посредством перезаписи запросов (особенно в случае запроса Q4, где производится соединение виртуального документа с реальным) и декомпозиции запросов, которая приводит к выделению наиболее дорогостоящей части запроса и передачи ее для выполнения источнику. Заметим, что общее время выполнения запроса можно было бы снизить, введя дополнительные индексы в источниках, однако рассмотрение данного вопроса выходит за рамки статьи. Ограниченный объем статьи не позволяет также привести примеры работы логического оптимизатора по существенному упрощению запросов, содержащих сложные трансформации (особенно те запросы, которые генерируются автоматически). Эти примеры чрезвычайно объемны.

6. Заключение

В статье представлена архитектура системы виртуальной интеграции данных BizQuery, основанной на модели данных XML/XQuery, которая позволяет обращаться к данным как в терминах XML, так и в терминах UML. Обсуждены разработанные средства построения отображения глобальной схемы на локальные схемы источников. Рассмотрена роль декларативных языков запросов UQL и XQuery в системе BizQuery. Представлены средства автоматического построения пользовательского интерфейса по описанию модели интегрируемых данных в виде диаграммы классов UML. Обсуждена проблема

производительности систем виртуальной интег-рации, и были выделены три ключевых задачи, от решения которых зависит пригодность системы для пользователя с точки зрения времени ожидания ответа на запрос. Эти задачи были детально проработаны, и предложены решения. Правильность подхода подтверждена экспериментальными результатами.

Проблема виртуальной интеграции данных была и остается исключительно сложной проблемой в области управления данными. Развитие технологии XML обеспечивает простую возможность унифицированного представления всех видов данных, но в то же время порождает ряд новых проблем, включающих более сложные методы оптимизации и выполнения запросов. При работе над проектом BizQuery авторы стремились обозначить эти проблемы и предложить их решения. Это позволило создать эффективную систему виртуальной интеграции данных и доказать практическую применимость подхода.

Литература

1. A Selection of Papers on Datawarehousing, Computer, Vol. 14, No. 12 (2001)
2. Batini, C., Lenzerini, M., and Navathe, S.: A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computer Surveys 18(4) (1986) 323-364
3. Sheth, A., Larson, J.: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases, ACM Computing Surveys 22(3) (1990) 183-236
4. Grinev, M., Kuznetsov, S.: UQL: A Query Language on Integrated Data in Terms of UML, Programming and Computer Software, Vol. 28, No. 4 (2002) 189-196
5. Wiederhold, G.: Mediators in the Architecture of Future Information Systems, IEEE Computer 25(3) (1992) 38-49
6. Chawathe, S., Garcia-Molina, H., Hammer J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: The TSIMMIS Project: Integration of Heterogeneous Information Sources, IPSJ (1994) 7-18
7. Extensible Markup Language (XML) 1.0, W3C Recommendation, 2nd edition (2000) http://www.w3.org/TR/2000/REC-xml-20001006
8. XSL Transformations (XSLT) 2.0, W3C Working Draft 15 November 2002, http://www.w3.org/TR/2002/WD-xslt20-20021115/
9. XQuery 1.0: An XML Query Language, W3C Working Draft 15 November 2002, http://www.w3.org/TR/2002/WD-xquery-20021115/
10. The Tukwila Data Integration System, University of Washington, http://data.cs.washington.edu/integration/tukwila/
11. Xperanto Project, IBM Almaden Research Center, http://www.almaden.ibm.com/software/dm/Xperanto/index.shtml
12. Unified Modeling Language (UML), Specification Version 1.4, http://www.omg.org/technology/documents/formal/uml.htm
13. XML Metadata Interchange (XMI), Version 1.2 http://www.omg.org/technology/documents/formal/xmi.htm
14. RELAX NG Specification, Committee Specification 3 December 2001, http://www.oasis-open.org/committees/relax-ng/spec-20011203.html

15. Grinev, M., Kuznetsov S.: Towards an Exhaustive Set of Rewriting Rules for XQuery Optimization: BizQuery Experience, 6th East-European Conference on Advances in Databases and Information Systems (ADBIS), LNCS 2435 (2002) 340-345
16. Levy, A., Rajaraman, A., Ullman J. D.: Answering Queries Using Limited External Query Processors, PODS (1996) 227-237
17. Graefe, G.: Query Evaluation Techniques for Large Databases, ACM Computing Surveys 25(2) (1993) 73-170
18. XMark — An XML Benchmark Project, http://www.xml-benchmark.org
19. QuiP. Software AG's prototype of XQuery, http://developer.softwareag.com/tamino/quip/