ГЛАВА 10

XML И БАЗЫ ДАННЫХ

Объем XML, используемый в бизнесе, стремительно растет с увеличением количества пересылаемых предприятиями сообщений в виде XML. Многие веб-сайты используют XML в качестве хранилищ данных, информация из которых затем преобразуется в HTML или XHTML для вывода пользователям. Растет и многообразие исходных данных. Например, новое поколение продуктов и технологий, таких как Microsoft InfoPath и W3C XForms, также начинают работать с XML-данными, перенося их из различных форм непосредственно в такие хранилища данных, как Microsoft Access или SQL Server.

Для отслеживания бизнес-активности необходимо иметь возможность обмениваться большими объемами данных в виде XML, хранить их, а также ясно представлять преимущества гибкости XML для отражения структуры бизнес-данных и для их обработки и обмена ими в дальнейшем. Кроме того, XML все чаще используется для хранения критичных данных, которые могут быть особо конфиденциальны и должны быть надежно скрыты от посторонних глаз. Поэтому при использовании XML для хранения данных возникает большое количество вопросов, которые следует рассмотреть и разрешить. Недостаточно просто хранить данные в виде XML; также следует обязательно позаботиться о масштабируемости и безопасности.

В главе 9, “XQuery”, вы познакомились с XQuery, языком запросов XML, разрабатываемым W3C. В этой главе рассматривается более широкий круг вопросов, связанных с использованием XML с базами данных. Эти вопросы проиллюстрированы примерами использования XML с “родными” XML-базами данных и с двумя SQL-базами данных, способными к работе с XML.

В этой главе рассматриваются следующие вопросы.

- Использование баз данных, способных работать с XML
- Выполнение фундаментальных задач с помощью eXist — XML-базы данных с открытыми исходными текстами
- Использование XML-функциональности в Microsoft SQL Server и MySQL — основных реляционных базах данных с XML-функциональностью

Потребность в эффективных хранилищах данных XML

При рассмотрении вопросов хранения данных в виде XML очень важным критерием является эффективность. Если XML хранится как текстовый документ, каким образом можно обеспечить его эффективную обработку? Когда объемы данных измеряются гигабайтами, создание полностью размещаемой в памяти объектной модели документа (DOM) становится в большинстве случаев непрактичным, и следует обязательно рассмотреть иные подходы.

С ростом объемов XML-данных становится все более важной эффективность поиска. Какой бы метод не использовался для хранения XML-данных, индексирование для ускорения поиска является насущной необходимостью. Во многих практических случаях важна эффективность выборки данных. Например, когда XML-данные используются для веб-служб или для взаимодействия с пользователем через веб, производительность должна быть достаточно высокой, чтобы обеспечить приемлемое для пользователя время отклика.

Если данные хранятся некоторым иным способом, не как XML-данные, встает вопрос о скорости преобразования хранящихся данных в XML-формат. В состоянии ли база данных предоставлять данные в XML-виде с достаточной скоростью, без излишних задержек?

Не менее важны и вопросы надежности. Можно разработать хорошо работающую XML-базу данных, по крайней мере хорошо работающую тогда, когда она работает, но если она будет не в состоянии работать постоянно, 100% времени, то такая база данных может просто оказаться ненужной, не обеспечивающей требуемой надежности.

Увеличение количества XML

Я начал работать с XML в 1999 году и впервые написал о его использовании примерно год спустя. Сначала XML представлялся мне узкоспециализированной, абстрактной темой, которую, как мне представлялось, многие просто не сочтут заслуживающей внимания. Я, конечно, понимал, что XML может стать важной технологией, но совершенно не представлял, насколько важной (и насколько быстро).

Одним из факторов, обеспечивающих возрастающее применение XML, является то, что XML имеет аномальную гибкость в представлении данных. Он в состоянии представить структуры данных, которые в виде реляционных данных представляются сложно или неэффективно. В некоторых ситуациях XML-базы данных для работы с XML добиваются большего успеха, чем реляционные базы данных. Однако очень сложно сказать, что же такое “XML-база данных”. Вероятно, наиболее практичным определением “родной” XML-базы данных (native XML

database) будет база данных, разработанная, в первую очередь, для работы с XML-данными (или только с ними).

Те, кто пришли к XML-базам данных от реляционных баз данных, обычно считают реляционные данные *структурированными*. Однако, если оглянуться вокруг, можно встретить массу иных типов данных, которые также являются структурированными, но структурированными иным, более сложным или более изменчивым способом. В этой книге термин *структурированные данные* (structured data) будет относиться, прежде всего, к реляционным данным.

Термины *полуструктурированные данные* (semi-structured data) и *свободно структурированные данные* (loosely structured data) четких границ не имеют. *Полуструктурированные данные* представляют собой термин, часто применяемый работающими с реляционными базами данных для указания нереляционных данных, зачастую — XML-данных. *Свободно структурированные данные* обычно означают документоцентричный XML. XML-документы, такие как XHTML веб-страницы или документы DocBook, могут очень сильно отличаться по своей структуре. Они, конечно, остаются структурированными и, в предположении их корректности, соответствуют схеме. Серьезное различие между реляционными данными и документами XHTML или DocBook заключается в большей гибкости и вариантности XHTML-документов по сравнению с реляционными базами данных.

Рассматриваете ли вы реляционные данные как негибкие и просто структурированные или нет — вопрос скорее философский. Точно так же, рассматриваете ли вы XML-документы как свободно структурированные (точка зрения работающих с реляционными базами данных) или как сильно структурированные, это никак не влияет на гибкость XML.

Сравнение XML-данных и реляционных данных

Перед тем как приступить к рассмотрению использования XML в современных базах данных, кратко сравним структуру реляционных данных и XML. Если вы не разберетесь в этом вопросе сейчас, многое из последующего материала может оказаться для вас непонятным.

В реляционной базе данных информация хранится в таблицах, состоящих из строк и столбцов. В столбце хранятся данные определенного вида для всех записей таблицы. Каждая запись таблицы представлена строкой. Порядок строк в таблице не связан ни с каким упорядочением данных, в отличие от XML, где внутренне присутствующий порядок документа влияет, например, на данные, возвращаемые такой функцией XPath, как `position()`.

Только простейшие реляционные данные могут храниться в одной таблице; типичная реляционная база данных имеет множество таблиц со сложными логическими отношениями между ними. Данные в разных таблицах связываются

с помощью *ключей*. Например, в таблице `Customers` может иметься поле (или столбец) `CustomerID`. Идентификация заказов для конкретного пользователя облегчается наличием соответствующего значения в столбце `CustomerID` таблицы `Orders`.

Отношения между данными могут быть "один-к-одному" (например, у одного сына может быть только один отец), "один-ко-многим" (у одного сына два родителя, у одного пользователя — несколько заказов) или "многие-ко-многим" (один товар может быть во многих заказах, в каждом заказе могут быть разные товары). Каждое из этих отношений может быть представлено путем хранения данных в двух или более связанных таблицах.

Реляционные базы данных обычно не имеют иерархии как таковой в отличие от XML-документов, которым присуща внутренняя иерархичность, как иллюстрирует модель данных XPath, DOM или XML Infoset.

Данные XML внутренне упорядочены, как в этом простом примере.

```
<Orders>
  <Order Customer="Acme Industries" Date="2003-12-11"
         Value="1234.56" Currency="US Dollars" />
  <Order Customer="Fiction Fabricators" Date="2004-02-11"
         Value="4300.12" Currency="US Dollars" />
  <Order Customer="Aspiring Assemblers" Date="2005-07-11"
         Value="10000.00" Currency="US Dollars" />
</Orders>
```

Внутренняя иерархия XML является условием, определяемым критерием правильного построения XML-документа. Сохранение даже этих простых данных в реляционной таблице приведет к потере их упорядоченности.

Подходы к хранению XML-данных

Необходимость хранения XML-данных возникла не на пустом месте. До того как был изобретен XML, было накоплено огромное количество сохраненных данных, причем большинство — в обычных СУРБД.

Хранение XML в файловых системах

Хотя данная глава посвящена XML и базам данных, не следует забывать, что большинство XML-документов хранится в файловых системах. Сама идея XML-"документа" предполагает хранение на диске, так же как вы храните любые другие "документы" на своем рабочем столе. Многие приложения никогда не идут дальше этого первого шага и всегда хранят XML-документы в файловой системе.

Хранение XML-документов в файловой системе простое и естественное не только потому, что сам термин "XML-документ" подразумевает его, но и в силу

того, что иерархическая организация файловой системы очень похожа на иерархическую организацию документа. Имеется четкая параллель между синтаксисом URL или путей файлов и простейшими выражениями XPath, так что вполне естественным выглядит обращение к узлу "/bat/baz" в документе "/foo/bar.xml". Перед тем как перейти к "настоящим" XML-базам данных, стоит рассмотреть, каковы ограничения хранения XML-документов в файловых системах.

Размер документа

Имеет смысл хранить XML-документы на диске в том случае, когда вам нужно работать со статическими документами небольших размеров в WWW. Файловые системы в настоящее время могут эффективно поддерживать гигабайтные файлы; так что, зная путь к любому XML-документу, вы можете эффективно получить доступ к хранящейся в нем информации. Важным фактором является "зернистость" информации, к которой требуется доступ. Если вам всегда нужен полный документ, эта система вполне хорошо срабатывает. Однако если вам необходимо получать только небольшие части больших документов с помощью DOM или XPath, то у вас возникнут огромные накладные расходы из-за необходимости чтения всего документа перед тем, как вы сможете выделить из него интересующую вас часть.

Кроме того, не забывайте, что вам придется анализировать эти документы всякий раз, когда вы обращаетесь к ним посредством DOM или XPath. Конечно, это соображение применимо только к данному виду работы с документами. Если все, что вам нужно, — это работать с документами без их модификации или преобразования в WWW, то лучше подготовить их к работе в виде XML.

Обновления

Еще одним важным вопросом, возникающим при хранении XML-документов на диске, являются обновления. Если вы вручную управляете небольшим набором XML-документов на рабочем столе или веб-сервере, обновления не вызывают затруднений. Но как только вам потребуется обеспечить возможность внесения обновлений многими пользователями или, что еще хуже, если вы разрабатываете транзакционное приложение, вам потребуется предпринять ряд дополнительных мер для выполнения обновлений. Одним из способов решения этой задачи является хранение документов в репозитории WebDAV, который решает вопросы блокировки и параллельного обращения вместо вас. Если вас заинтересовал этот подход, можете попробовать воспользоваться системой управления версиями, такой как Subversion (`http://subversion.tigris.org/`). Subversion может работать как репозиторий WebDAV и предоставляет все возможности системы управления версиями, включая фиксацию истории любых модификаций ваших документов. Для ряда приложений это очень важная возможность, причем эта

возможность — одна из тех, которые непосредственно не поддерживаются базами данных, рассматриваемыми в данной главе.

Что бы вы не предприняли для обновления документов, вам все равно придется рассматривать вопросы детализации информации. Если большинство обновлений работают с мелкими частями больших документов, у вас будут большие накладные расходы из-за необходимости замещать полный документ обновленной версией. В транзакционных приложениях это также означает, что при обновлении одного из узлов должен блокироваться весь документ, а это может вызвать серьезные последствия для производительности приложений.

Индексы

Последний вопрос, который может возникнуть при хранении документов на диске, — это запросы. Например, если в большой коллекции документов нужно найти все документы, написанные определенным автором, вам может потребоваться реализовать индексацию того или иного вида. Если для индексирования имеется несколько предопределенных полей, в качестве индекса можно использовать структуру каталогов: для индексирования авторов для каждого автора имеется свой каталог. Если при этом требуется индексирование по дате, можно добавить символьные связи для создания виртуальных каталогов для дат. Чтобы увеличить количество полей или обеспечить поддержку полнотекстового поиска, придется использовать какой-то иной метод индексирования.

При использовании системы управления версиями, описанной в предыдущем разделе, можно воспользоваться преимуществами, которые предоставляют возможности такой системы. Например, при использовании Subversion можно легко получить список документов для определенной версии с изменениями, внесенными определенным пользователем, измененными между двумя датами и т.д. Кроме того, Subversion позволяет добавить собственные пользовательские свойства, которые можно сохранять и запрашивать. Это удобно, например, если нужно хранить имя автора документа, а оно отлично от имени пользователя, внесшего изменения в документ, или когда дата создания текста отличается от даты обновления. Изюминкой Subversion является то, что команды этой системы управления версиями имеют опции для форматирования результатов в виде XML; это означает, в свою очередь, что данные результаты могут быть отформатированы с помощью XSLT для представления на веб-странице.

Если вас, в первую очередь, интересует полнотекстовый поиск, можете воспользоваться поисковым механизмом наподобие Lucene (`http://lucene.apache.org/`). Lucene поставляется с API, который позволяет определять, какие элементы должны индексироваться и как они должны рассматриваться. Он эффективно поддерживает большие коллекции документов и предоставляет возможности, знакомые большинству пользователей (поскольку эти возможности те же, что и у большинства поисковых систем в WWW).

Даже если после изучения главы 9, “XQuery”, вы убеждены, что XQuery — это именно то, что вам нужно, все равно реализации XQuery работают с коллекциями XML-документов, сохраненных на диске, как, например, в случае XQEngine (`http://xqengine.sourceforge.net/`).

Построение собственного решения

Хотя большинство вопросов можно разрешить, хранение XML-документов на диске с доступом для записи и индексами требует большого объема работы, в особенности работы, связанной с интеграцией. XML же базы данных оказываются существенно более интегрированным решением. Само собой, эта интегрированность представляет собой определенный компромисс, так что вы можете найти XML-базы данных худшим решением по сравнению с тем, которое вы можете разработать самостоятельно для своих конкретных нужд. В качестве примера можно взять систему управления версиями: в XML-базах данных обычно нет тех возможностей, которые есть в системах управления версиями, и нет полнотекстового поиска. Но, с другой стороны, использование надежной XML-базы данных вместо разработки программного обеспечения для той же функциональности в случае хранения документов в файловой системе может помочь вам сэкономить массу времени.

Использование XML с обычными базами данных

Реляционные базы данных — один из наиболее популярных способов хранения данных. Они давно и тщательно проработаны, хорошо подходят для хранения структурированных данных, хранят огромные количества существующей информации и хорошо знакомы большому количеству разработчиков. Эти причины делают СУРБД хорошими кандидатами на совместное использование с XML, и у вас есть масса возможностей в этом убедиться.

Получение XML из реляционных баз данных

Большое количество HTML и XHTML веб-сайтов используют, непосредственно или опосредованно, реляционные данные. В этой области широко применяются комбинации PHP с базой данных MySQL или ASP или ASP.NET с SQL Server или Microsoft Access. Данные обычно хранятся как реляционные таблицы, и программист пишет код для создания HTML или XHTML, иногда с применением XML в качестве промежуточной стадии. XHTML представляет собой язык приложения XML. Создание XHTML веб-страниц из реляционных данных демонстрирует один из способов, которыми эти данные могут использоваться для получения XML-кода для представления данных. Распространенность этой технологии является наилучшим доказательством возможности отображения реляционных данных на иерархические структуры XML и передачи этих структур пользователю.

Например, при использовании PHP для запроса данных из нескольких таблиц в MySQL для представления их пользователю в виде веб-страницы вряд ли разработчик пожелает представить данные в виде таблиц, аналогичных структуре базы данных. Скорее всего, он преобразует неупорядоченную, неиерархичную структуру реляционных данных в нечто как минимум упорядоченное и иерархичное, поскольку именно такие данные успешно представимы в виде веб-страниц HTML и XHTML, которые сами по себе иерархичны.

Если отдельные программисты находят способы преобразования реляционных данных в XML, то нет ничего удивительного в том, что разработчики различных баз данных также обнаружили возможность вступить на растущий рынок XML и предложить на нем возможность экспорта хранящихся реляционных данных в виде XML. Фактически многие реляционные базы данных позволяют пользователям получать XML из данных, хранящихся в реляционных таблицах.

Перемещение XML в реляционные базы данных

Аналогично многие СУРБД обладают способностью получения XML-данных от пользователей, преобразования их в реляционный вид с последующим сохранением в реляционных таблицах. В зависимости от того, сохраняются ли метаданные, связанные с упорядочением, в процессе преобразования XML в информацию, которую способна обработать реляционная база данных, может оказаться возможным в последующем восстановить исходный XML-документ. Во многих ситуациях, впрочем, такое точное восстановление не является необходимым.

Возможность получения и возврата данных для имитации XML-функциональности прекрасна, в первую очередь, потому, что это автоматизация очень громоздкой и утомительной для ручной работы задачи.

Связывание данных

Схемы связывания данных, учитывая тот факт, что в приложениях должны сосуществовать несколько представлений одних и тех же данных, автоматизируют отображения между этими представлениями. Представлениями, наиболее часто поддерживаемыми схемами связывания данных, являются XML, базы данных SQL и объекты.

Схемы связывания данных, которые могут непосредственно отображать друг на друга XML и базы данных SQL, включают ADO.NET (`http://msdn.microsoft.com/data/ref/adonet/`) в мире Microsoft и Castor (`http://www.castor.org/`) в сообществе программного обеспечения Java с открытым кодом. Они работают в качестве “клея” между XML и реляционными базами данных, которые слишком громоздки для разработки вручную. Они очень гибки, могут настраиваться разными способами и являются хорошим решением задачи представления содержимого существующих баз данных в виде XML.

В дополнение к такому непосредственному связыванию XML и реляционных баз данных можно встретить и более сложные сценарии, как, например, сценарии, когда реляционная база данных используется в качестве основы для объектов, которые, в свою очередь, генерируют XML- или XHTML-документы. В таких сценариях основа — всего лишь схема связывания данных, использующаяся для отображения реляционных баз данных на объекты.

В зависимости от ситуации XML или XHTML генерируется вручную, с помощью шаблонов или другой библиотеки связывания данных.

Если же вас привлекает более интегрированный подход, познакомьтесь с истинными XML-базами данных.

Истинные XML-базы данных

Что собой представляют истинные XML-базы данных (native XML database)? Вряд ли вы найдете единственное устраивающее всех определение. Простое и вполне разумное определение может заключаться в том, что истинная XML-база данных разработана для хранения XML-данных. Если она при этом хранит и другие структуры данных, отличные от XML, то разве это заставляет ее перестать быть истинной XML-базой данных?

Истинная XML-база данных может реализовывать XML с использованием модели наподобие XML Infoset, XML DOM, XPath или Simple API for XML (SAX). Она, вероятно, хранит и различные аспекты XML-документа, такие как порядок документа.

Технология реляционных баз данных в настоящее время хорошо разработана, имеет солидную теоретическую основу и несколько десятилетий практического применения в широко распространенных программных продуктах. Истинные же XML-базы данных — недавняя разработка; у них пока нет такого теоретического базиса, как у реляционных баз данных, они находятся в стадии развития и, вероятно, будут находиться в ней еще как минимум несколько лет.

Каким бы ни был лежащий в основе этих баз данных механизм хранения информации, истинная XML-база данных отображает XML-документ на модель хранения. Отображение может существенно отличаться в разных базах данных, например зависеть от деталей преобразования XML-документа в реляционную базу данных.

Истинные XML-базы данных часто хранят XML-документы в коллекциях, и запросы могут обращаться к целым коллекциям. В зависимости от конкретного продукта коллекция может определяться схемой или содержать документы с разными структурами. Последний вариант, вероятно, вызовет ужас у тех, кто привык к предсказуемости реляционной модели.

На момент написания этой книги многие истинные XML-базы данных использовали в качестве языка запросов XQuery, хотя он пока еще и не получил статус

рекомендации W3C. Однако лишь немногие из них поддерживают возможности W3C XML Schema в XQuery.

Обновления в истинных XML-базах данных в настоящее время испытывают недостаток в стандартизации. Отсутствие функциональности вставки, удаления и обновления в XQuery 1.0 означает присутствие нестандартных механизмов обновления в истинных XML-базах данных по крайней мере еще в течение некоторого времени.

На практике граница между истинными XML-базами данных и реляционными базами данных с поддержкой XML становится все более размытой. Например, Microsoft SQL Server, Oracle, Sybase Adaptive Server Enterprise и IBM DB2 9 обладают возможностью хранения нового типа данных `xml` без утраты своей традиционной мощи в качестве СУРБД.

Для многих практических целей не имеет значения, используется ли истинная XML-база данных или реляционная база данных с поддержкой XML. В качестве пользователя или разработчика вы передаете обоим видам баз данных XML, и получаете XML от них; так зачем волноваться о том, что происходит за сценой? Обычно в этом нет никакой необходимости. Выбор между истинной XML-базой данных и реляционной базой данных с поддержкой XML аналогичен любому другому выбору программного обеспечения. Просто определите свои нужды и найдите продукт, наиболее полно им соответствующий, с учетом цены, поддержки операционной системы (или операционных систем) и других критериев.

В оставшейся части этой главы в качестве примеров истинных XML-баз данных и реляционных баз данных с поддержкой XML рассматриваются три очень разные базы данных.

- eXist — одна из наиболее сложившихся XML-баз данных с открытым кодом, написанная на Java
- SQL Server — СУРБД уровня предприятия от Microsoft, в которой реализована определенная XML-функциональность
- MySQL — база данных с открытым исходным кодом, широко используемая множеством веб-сайтов; ее XML-возможности значительно уступают возможностям коммерческих конкурентов. (В настоящей главе мы рассмотрим возможности бета-версии 5.1.2 этой базы данных.)

Использование истинных XML-баз данных

Как упоминалось ранее, истинные XML-базы данных отличаются своими подходами. Отдельные базы данных в категории истинных XML-баз данных могут существенно отличаться возможностями. В этой главе для изучения работы истинной XML-базы данных используется eXist.

Получение и инсталляция eXist

Перед тем как приступить к работе с базой данных eXist, посетите ее сайт по адресу `http://exist-db.org/`. Здесь вы найдете ссылки, позволяющие загрузить последнюю версию базы данных. Ее можно загрузить в разных вариантах — как автономный инсталлятор и как инсталлятор на основе IzPack (на момент написания настоящей книги эта версия называлась `eXist-1.1.1-newcore.jar`). eXist написана на Java, так что перед инсталляцией загруженного jar-файла убедитесь, что у вас установлена последняя версия Java. В настоящее время требуется JDK версии не ниже 1.4.2.

> *Если вы не знаете, какая версия Java установлена на вашем компьютере, введите команду `java -version` в окне командной строки или терминале Unix.*

Если вы загрузили инсталлятор и у вас установлена правильная версия Java, можете установить eXist, щелкнув на загруженном jar-файле на правильно настроенной машине. Если вам не удалось это сделать, откройте окно командной строки и введите в нем следующую команду.

```
java -jar eXist-<version>.jar
```

При этом должен запуститься графический инсталлятор, который поможет вам в и без того простом процессе установки eXist.

После завершения установки у вас имеется “готовая к употреблению” истинная XML-база данных, которая может использоваться в трех разных режимах.

- Можно использовать eXist в качестве библиотеки Java для встраивания сервера базы данных в собственное Java-приложение.
- Можно работать с приложением как с автономным сервером базы данных, как если бы вы работали, например, с сервером SQL-базы данных.
- Можно запустить ее как веб-сервер и получить как возможности автономного сервера базы данных, так и веб-интерфейс для доступа к базе данных.

После инсталляции eXist можно использовать в двух последних режимах с помощью различных сценариев, которые находятся в подкаталоге `bin`.

- `server` (`.sh` или `.bat` в зависимости от вашей платформы) запускает eXist как автономный сервер базы данных.
- `startup` (`.sh` или `.bat`) запускает eXist как встроенный веб-сервер, а `shutdown` (`.sh` или `.bat`) останавливает его. Для примеров в данной главе будет использоваться именно этот режим, поскольку в нем доступно больше всего возможностей.

Чтобы убедиться в корректности инсталляции, запустите `startup.sh` или `startup.bat` в окне командной строки. Вы получите ряд предупреждений и информацию, которая завершится строками наподобие следующих.

```
20 Nov 2006 16:47:55,485 [main] INFO (SocketListener.java
➥  [start]:204) - Started SocketListener on 0.0.0.0:8080
20 Nov 2006 16:47:55,485 [main] INFO (HttpServer.java
➥  [start]:690) - Started org.mortbay.jetty.Server@858bf1
```

Эти строки означают, что jetty (веб-сервер Java в составе eXist) готов принимать соединения по порту 8080.

> *По умолчанию веб-сервер прослушивает порт 8080. Это означает, что он не сможет запуститься, если в системе имеется другая служба, связанная с этим портом. В таком случае необходимо либо остановить эту службу перед запуском eXist, либо изменить настройки eXist так, чтобы он прослушивал другой порт. Соответствующие инструкции вы найдете на сайте eXist.*

Последняя проверка заключается в запуске вашего браузера. В его адресной строке введите `http://localhost:8080/exist/` и убедитесь, что начальная страница eXist, показанная на рис. 10.1, успешно открылась.

Рис. 10.1

Взаимодействие с eXist

Поздравляем, вы успешно установили и запустили истинную XML-базу данных! Теперь рассмотрим, каким образом можно с ней взаимодействовать.

Использование веб-интерфейса

Первый из вариантов — использование веб-интерфейса по адресу `http://localhost:8080/exist/`. Прокрутите страницу вниз до раздела Administration в левой части экрана. Щелкните на кнопке Admin для перехода по ссылке `http://localhost:8080/exist/admin/admin.xql`, где вы должны войти в систему как пользователь `admin` с пустым паролем (который рекомендуем немедленно заменить чем-то более безопасным).

Войдя в систему, вы получите доступ к командам из левой части меню. Чувствуйте себя как дома и немного освойтесь в системе, воспользовавшись примерами, которые eXist предоставляет в процессе инсталляции.

Когда вы будете готовы к продолжению этой беглой экскурсии по eXist, щелкните на ссылке Browsing Collection (рис. 10.2). XML-документы организованы в *коллекции*, которые представляют собой эквивалент каталогов в файловой системе. Это действительно одна и та же концепция. Вы можете рассматривать базу данных eXist как черный ящик, в котором хранятся новые возможности по работе с XML-документами — возможности, которых у вас не было при простом хранении файлов на диске. При этом основная парадигма иерархической структуры коллекций, или каталогов, остается неизменной.

Вновь установленная база данных eXist имеет несколько существующих коллекций, но мы создадим новую коллекцию `blog` с помощью кнопки Create Collection. После создания коллекции следуйте по ссылке для ее просмотра. Пока что новая коллекция пуста. Воспользуйтесь кнопкой Upload, чтобы загрузить в нее файлы `blogitem1.xml` и `blogitem2.xml`, которые можно загрузить вместе с прочими примерами кода к данной главе с сайта Wrox. Эти документы представляют собой примеры записей в блогах. Например, вот как выглядит файл `blogitem1.xml`.

```
<?xml version="1.0"?>
<item id="1">
  <title>Working on Beginning XML</title>
  <description>
    <p>
      <a href=
       "http://www.wrox.com/WileyCDA/WroxTitle/
         ➥   productCd-0764570773.html">
        <img src="http://media.wiley.com/product_data/
         ➥   coverImage/73/07645707/0764570773.jpg"
         align="left"/>
      </a> I am currently working on the next edition
      of <a href="http://www.wrox.com/WileyCDA/
```

Рис. 10.2

```
        ➥  WroxTitle/productCd-0764570773.html">
      WROX's excellent "Beginning XML".</a>
    </p>
    <p>It's the first time I am working on editing a
       book that I haven't written and I must say that
       I like it even better than I had expected when I
       accepted WROX's offer.</p>
    <p>I knew that the book was solid and that I would be
       working with a team of very professional authors,
       but what I hadn't anticipated is how fun it can be
       to create a new version out of existing material.
       You have a lot of room to reorganize what you are
       editing and when the material is good, it's like
       creating your own stuff, except that 80% of the
       hard work is already done!</p>
  </description>
  <category>English</category>
  <category>XML</category>
  <category>Books/Livres</category>
  <pubDate>2006-11-13T17:32:01+01:00</pubDate>
  <comment-count>0</comment-count>
</item>
```

Загруженные в коллекцию документы можно будет вывести, щелкнув на соответствующих ссылках. Теперь, когда у вас есть документы в коллекции `/db/blog`, можете выполнять к ним запросы с помощью того же веб-интерфейса. Для этого щелкните на ссылке Home, чтобы вернуться к начальной странице, а затем перейдите по соответствующей ссылке к XQuery (`http://localhost:8080/exist/sandbox/sandbox.xql`).

Теперь самое время вспомнить все, чему вы научились в главах 7, "XPath", и 9, "XQuery": текстовое поле ожидает, что вы введете в него запрос, написанный на XPath или XQuery! Начав с чего-нибудь простого наподобие `/item[@id='1']` и щелкнув на кнопке Send, вы получите все документы из коллекции, у которых имеется корневой элемент `item` с атрибутом `id`, равным 1. Если вы строго следовали приведенным здесь инструкциям, то полу́чите только первую запись блога.

Конечно, можно написать и более сложный запрос. Например, если вы хотите определить названия, идентификаторы и ссылки записей блога со ссылками на сайт Wrox, можете воспользоваться запросом `xquery1.xq`, показанным на рис. 10.3.

```
for $item in /item
  where .//a[contains(@href, 'wrox.com')]
    return <match>
      <id>{string($item/@id)}</id>
      {$item/title}
      {$item//a[contains(@href, 'wrox.com')]}
    </match>
```

Немного "поиграйте" с запросами, а когда вы достаточно освоитесь, мы перейдем к рассмотрению клиента eXist.

Использование клиента eXist

Клиент eXist представляет собой автономный графический инструмент, который может выполнить те же виды операций, что и веб-интерфейс. Чтобы начать с ним работать, просто щелкните на сценарии `client.sh` или `client.bat` (в зависимости от используемого вами окружения). Вы должны получить экран входа в систему. Если вы уже установили пароль администратора (пользователя `admin`), введите его. Перед тем как щелкнуть на кнопке OK, обратите внимание на поле ввода URL. По умолчанию в нем содержится значение `xmldb:exist://localhost:8080/exist/xmlrpc`. Мы не будем рассматривать здесь все компоненты этого URL; взглянем только на часть `localhost:8080`. Она означает, что данный клиент использует для соединения с базой данных eXist протокол HTTP и то, что вы можете администрировать базы данных eXist удаленно на других машинах.

Рис. 10.3

Следующий экран позволяет просмотреть коллекции или вашу базу данных. Если вы щелкните на Blog, то опять увидите две записи; если щелкнуть на одной из них, появится окно, в котором ее можно редактировать. В основном окне кнопка с биноклем открывает диалоговое окно Query, в котором вы можете снова поработать с XPath и XQuery. Обратите внимание на вкладку Trace в окне Results в нижней части экрана: здесь вы найдете путь выполнения ваших запросов, который может содержать полезную информацию для их отладки или оптимизации. На рис. 10.4 показан приведенный выше запрос, запущенный в клиенте eXist.

С помощью клиента можно получить гораздо больше информации; он может также сохранять и восстанавливать коллекции и целые базы данных. Теперь же мы узнаем, как использовать eXist в качестве сервера WebDAV.

Использование WebDAV

WebDAV означает Web-based Distributed Authoring and Versioning (протокол распределенной работы над документами в WWW). Он представляет собой множество документов IETF RFC, определяющих, как использовать HTTP не только для чтения ресурсов, но и для записи. WebDAV реализован во множестве распространенных операционных систем и инструментов, и возможность представлять

Рис. 10.4

коллекции eXist в виде репозиториев WebDAV может существенно облегчить импорт и экспорт документов.

> *IETF (Internet Engineering Task Force — инженерная группа по развитию Интернета) представляет собой организацию, занимающуюся стандартизацией и публикующую большинство спецификаций протоколов Интернета, включая HTTP. Спецификации IETF называются RFC (Requests For Comments, дословно — "запрос на комментарий"); невзирая на свои названия они являются стандартами де факто.*

Чтобы познакомиться с WebDAV, направьте ваш браузер по адресу `http://localhost:8080/exist/webdav/db/`. Здесь придется снова выполнить процедуру входа в систему, после чего вы увидите страницу, на которой сможете просмотреть коллекции и содержимое вашей базы данных. Браузер без расширений обеспечивает доступ только для чтения; чтобы получить возможность записи и просмотра базы данных eXist в виде репозитория, установите клиент WebDAV.

Документация eXist, имеющаяся в вашей локальной базе данных по адресу `http://localhost:8080/exist/webdav.xml`, включает детальные инструкции по настройке Microsoft Windows, KDE Konqueror, oXygen и XML Spy

для использования WebDAV. Поддержка WebDAV встроена и в Mac OS X. Windows XP эта возможность известна как *веб-папки* (web folders) и очень легко настраиваема. Поскольку эти настройки описаны в документации eXist, мы не будем их рассматривать; однако здесь будут рассмотрены очень похожие настройки клиента WebDAV в GNOME.

В GNOME следует начать с пункта меню Places⇨Connect to Server. Выберите Select WebDAV (HTTP) в качестве Service type и введите информацию, соответствующую вашей конфигурации: `Server=localhost`, `Port=8080`, `Folder=/exist/webdav/db/` и `User Name=admin`. Выберите имя, которое будет использоваться в качестве метки данного соединения, например “eXist (local)”, и щелкните на кнопке Connect. На рис. 10.5 показано диалоговое окно Connect to Server GNOME WebDAV.

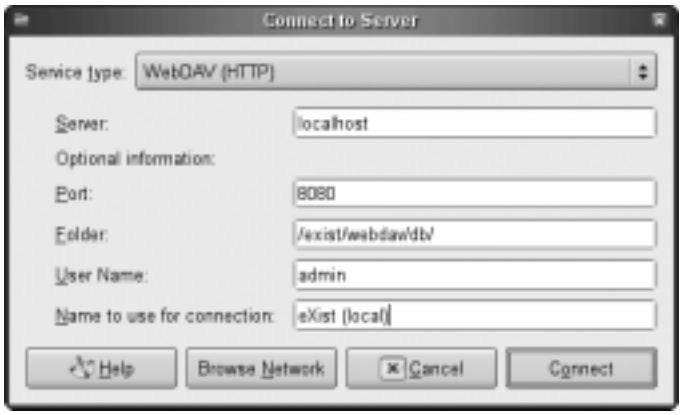

Рис. 10.5

Ваш клиент WebDAV настроен. Для его использования выберите в меню Places только что настроенное соединение. Вам будет предложено ввести пароль пользователя admin, и Nautilus (файловый менеджер GNOME по умолчанию) откроет перед вами окно для просмотра содержимого вашей базы данных eXist точно так, как если бы это была обычная файловая система.

Теперь, в каком бы окружении вы ни находились, вы должны иметь возможность обращаться к ресурсам в базе данных eXist с использованием вашего любимого файлового менеджера. Это означает, что вы можете не только открывать найденные документы, но и редактировать их, перемещать ресурсы между eXist и вашей локальной файловой системой, создавать новые коллекции, удалять существующие и т.д. На рис. 10.6 показан файловый менеджер GNOME по умолчанию, Nautilus, который просматривает базу данных eXist, представленную как репозиторий WebDav.

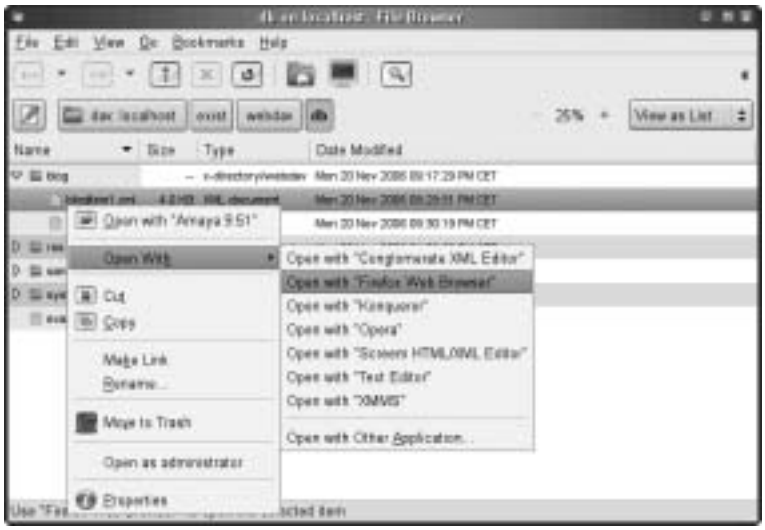

Рис. 10.6

Если ваш клиент eXist или веб-браузер остается открытым с интерфейсом администратора, то вы можете убедиться, что вносимые посредством WebDAV изменения немедленно становятся видимыми в базе данных, и наоборот — изменения, внесенные непосредственно в базу данных, становятся видимыми, как только вы обновляете содержимое вашего файлового менеджера.

Единственная возможность, отсутствующая при использовании интерфейса WebDAV, — это возможность выполнения запросов, но вы вскоре увидите, что вы можете вновь ее обрести при применении XML IDE.

Использование XML IDE

Ваша любимая среда XML (XML IDE), вероятно, в состоянии предоставить доступ к базе данных eXist посредством WebDAV. Если он является интерфейсом к eXist, вы также можете выполнять запросы из самой среды. Примером может являться среда oXygen 8.0, доступная для бесплатного 30-дневного пробного периода на сайте `http://www.oxygenxml.com/`.

Для настройки подключения к вашей базе данных eXist выберите базу данных с помощью пиктограммы в полосе инструментов или в меню Perspective. Затем щелкните на кнопке Configure Database Sources, расположенной в верхнем правом углу окна Database Explorer. При этом перед вами откроется окно настроек. Создайте новый источник данных с типом eXist и добавьте библиотеки `exist.jar`, `xmldb.jar` и `xmlrpc-1.2-patched.jar` (находящиеся в вашей установленной базе данных eXist). Простейший способ найти эти зависящие от конкретной версии библиотеки — поискать их в подкаталогах каталога с установленной базой данных eXist. Сохраните этот источник данных и создайте ис-

пользующее его соединение с параметрами соединения с eXist. Сохраните данное соединение и настройки, и можете считать работу сделанной.

Database Explorer должен показать вам вновь созданное соединение, и теперь вы можете просматривать и обновлять базу данных eXist так же, как если бы вы просматривали и обновляли документы в вашей локальной файловой системе.

До сих пор все, что бы вы ни делали, может быть сделано с помощью WebDAV. Для выполнения запроса создайте новый документ, используя пиктограмму File New или меню. Выберите для документа тип XQuery и введите свой запрос. Сделав все это, щелкните на кнопке Apply Transformation Scenario на панели инструментов или выберите это действие в меню Document⇨XML Document⇨Apply Transformation Scenario. Поскольку с документом пока что не связан ни один сценарий, будет открыто диалоговое окно Configure Transformation Scenario. Сценарий по умолчанию использует механизм Saxon XQuery. Для использования механизма eXist XQuery нужно создать новый сценарий и выбрать в качестве Transformer соединение с вашей базой данных eXist. Сохраните этот сценарий и щелкните на кнопке Transform Now для выполнения запроса. На рис. 10.7 показано выполнение того же запроса в oXygen.

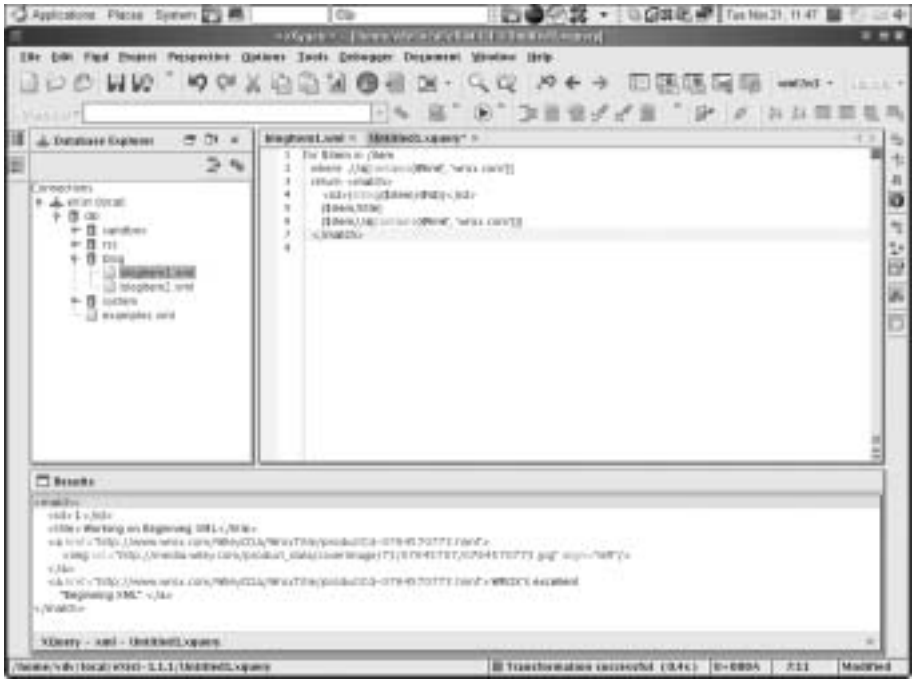

Рис. 10.7

Теперь, когда этот сценарий присоединен к документу запроса, можете обновить запрос и щелкнуть на кнопке Apply Transformation Scenario для его запуска без повторной настройки.

Все описанные методы обеспечивают удобные пользовательские интерфейсы, но вам следует еще узнать, как с вашей базой данных могут работать веб-приложения.

Использование интерфейса REST

Какой способ взаимодействия базы данных с веб-приложением может быть лучше, чем использование HTTP? Именно в этом состоит назначение интерфейса REST. Чтобы познакомиться с ним, направьте свой браузер по адресу `http://localhost:8080/exist/rest/`. Это действие покажет вам содержимое корня вашей базы данных в виде XML-документа. Этот XML-формат несколько менее дружествен по сравнению с просмотром содержимого коллекций через веб-интерфейс или с просмотром репозитория WebDAV, но зато существенно более прост в обработке приложением!

Благодаря этому интерфейсу доступно полное содержимое базы данных. Например, адрес `http://localhost:8080/exist/rest/db/blog/` соответствует содержимому коллекции `blog`, а `http://localhost:8080/exist/rest/db/blog/blogItem1.xml` дает первый элемент блога. Все становится более интересным при работе со строками запросов. Интерфейс REST принимает ряд параметров, включая параметр `_query`, который можно использовать для немедленной пересылки простых запросов XPath или XQuery.

Например, если вы хотите получить все ссылки из всех документов коллекции `/db/blog`, запросите URL `http://localhost:8080/exist/rest/db/blog/?_query=//a`. В текущей версии (`eXist-1.1.1-newcore`) тип медиа для таких запросов неверно сконфигурирован как `text/html`, так что браузер будет пытаться вывести его так, как если бы это был HTML-документ, как показано на рис. 10.8.

Этот результат может показаться странным, но, взглянув на исходный текст документа, вы увидите, что это не что иное, как XML-документ.

```
<exist:result
    xmlns:exist="http://exist.sourceforge.net/NS/exist"
    exist:hits="4" exist:start="1" exist:count="4">
    <a href="http://www.wrox.com/WileyCDA/WroxTitle/
    ➥ productCd-0764570773.html">
        <img src="http://media.wiley.com/product_data/
        ➥ coverImage/73/07645707/0764570773.jpg"
        align="left"/>
    </a>
    <a href="http://www.wrox.com/WileyCDA/WroxTitle/
    ➥ productCd-0764570773.html">
```

Рис. 10.8

```
    WROX's excellent"Beginning XML".</a>
    <a href="http://exist-db.org/">eXist</a>
    <a href="http://prdownloads.sourceforge.net/exist/
    ➥  eXist-1.1.1-newcore-build4311.jar">
    eXist 1.1.1-newcore</a>
</exist:result>
```

К этому XML можно применить преобразование XSLT, чтобы получить настоящий HTML; если вы помните материал главы 8, “XSLT”, то сообразите, что простое преобразование наподобие приведенного ниже приведет к существенно лучшему результату, чем показанный выше.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:exist="http://exist.sourceforge.net/NS/exist"
     version="1.0">
  <xsl:template match="/exist:result">
    <html>
      <head>
        <title>Query results</title>
      </head>
```

```
      <body>
        <h1>eXist query results</h1>
        <p>Showing results
          <xsl:value-of select="@exist:start"/> to
          <xsl:value-of select="@exist:end"/> out of
          <xsl:value-of select="@exist:hits"/>:
        </p>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="*">
    <p>
      <xsl:copy-of select="."/>
    </p>
  </xsl:template>
</xsl:stylesheet>
```

Хорошая новость состоит в том, что интерфейс eXist REST может при необходимости выполнить это преобразование вместо вас, но сначала следует сохранить преобразование в базе данных. Для этого можно воспользоваться любым из методов загрузки документов в базу данных (веб-интерфейс, клиент eXist, WebDAV или ваша любимая среда XML). Поскольку в этом разделе рассказывается об интерфейсе REST, мы воспользуемся именно им.

Сохранение документа с применением интерфейса REST основано на HTTP-запросе `PUT`; к сожалению, сделать то же самое с помощью веб-браузера нельзя. Для отправки HTTP-запроса `PUT` необходимо либо немного попрограммировать (все языки программирования имеют библиотеки, поддерживающие это действие), либо воспользоваться соответствующей утилитой типа curl (`http://curl.haxx.se/`).

У этой программы большое количество разных опций командной строки. Если curl установлен на вашей машине, для сохранения документа `rest-query-results.xsl` по адресу `http://localhost:8080/exist/rest/db/xslt/` просто введите следующую команду в окне командной строки.

```
curl -T rest-query-results.xsl
➥  http://localhost:8080/exist/rest/db/xslt/
```

Эта команда просто отправляет документ, используя метод `PUT` протокола HTTP. Интерфейс eXist REST поддерживает и запрос HTTP `DELETE`, так что вы можете также удалить этот документ. Для этого можно использовать опцию `-X`, которая определяет, какой именно метод HTTP должен использоваться.

```
curl -X DELETE localhost:8080/exist/rest/
➥  db/xslt/rest-query-results.xsl
```

Само собой разумеется, выполнив эту команду, вы должны заново загрузить преобразование перед тем, как его использовать! Теперь, когда ваша таблица стилей хранится в базе данных, для его применения просто добавьте параметр `_xsl`, определяющий его местоположение. Теперь в вашем браузере необходимо ввести в адресной строке URL

```
http://localhost:8080/exist/rest/db/blog/
➥  ?_query=//a&_xsl=/db/xslt/rest-query-results.xsl
```

Результат выполнения запроса показан на рис. 10.9.

Рис. 10.9

Теперь вы знаете, как использовать методы HTTP `GET`, `PUT` и `DELETE`. Если вы знакомы с HTTP, вас не должно удивлять то, что интерфейс REST поддерживает метод HTTP `POST`. Этот метод используется для отправки запросов, которые слишком велики, чтобы быть размещенными в строке запроса метода HTTP `GET`. Такие запросы “заворачиваются” в XML-документ, структура которого определяется в документации eXist. Например, рассматривавшийся ранее запрос принимает следующий вид.

```
<?xml version="1.0" encoding="UTF-8"?>
<query xmlns="http://exist.sourceforge.net/NS/exist">
  <text>
    <![CDATA[
```

```
      for $item in /item
        where .//a[contains(@href, 'wrox.com')]
        return <match>
          <id>{string($item/@id)}</id>
          {$item/title}
          {$item//a[contains(@href, 'wrox.com')]}
          </match>
    ]]>
  </text>
</query>
```

Обратите внимание на то, как предусмотрительно запрос встроен в раздел `CDATA`, так что он квалифицируется как правильно построенный XML. Для его отправки с помощью интерфейса REST можно использовать curl и опцию `-d`. Например, если на рабочей станции Linux запрос хранится в файле `linksToWrox.xml`, можно ввести следующую команду.

```
curl -d @linksToWrox.xml http://localhost:8080/exist/rest/db/
```

Другие интерфейсы

Вы познакомились с четырьмя способами работы с eXist, но существуют еще и другие. В этом разделе кратко рассматриваются только некоторые из них (поскольку имеется много другого интересного материала, который хотелось бы рассмотреть в этой главе).

Первый из этих методов — XML:DB API. XML:DB API представляет собой распространенный API, определенный рядом редакторов XML-баз данных. Его изначальное назначение — определить независимый от производителя программного обеспечения API, который играл бы ту же роль для XML-баз данных, что и JDBC для SQL-баз данных. К сожалению, проект не привлек коммерческих производителей, и, похоже, момент для него был упущен. XML:DB остался API для доступа к вашей базе данных eXist при программировании на Java.

Вторым является интерфейс XML-RPC, который охватывает все, что возможно при применении интерфейса REST, плюс некоторые дополнительные возможности, например можно обновить фрагмент без загрузки всего документа и полностью администрировать базу данных, пользуясь лишь этим интерфейсом.

Интерфейс SOAP обладает теми же возможностями, что и XML-RPC интерфейс, так что это выбор для тех, кто предпочитает SOAP поверх XML-RPC.

Интерфейс Atom Publishing Protocol (APP) был разработан совсем недавно, но для работы с базой данных вы можете воспользоваться и им.

Выбор интерфейса

Как при наличии такого богатого выбора остановиться на чем-то одном? Спросите сами себя, что именно играет для вас самую важную роль? Вы можете представить базу данных eXist как черный ящик, инкапсулирующий ваши

XML-документы. Они собраны в коллекции подобно файлам в каталогах. Черный ящик работает как файловая система с дополнительными возможностями XQuery и предоставляет ряд различных интерфейсов для доступа к одному и тому же набору данных различными способами. Какой бы интерфейс вы ни использовали, результат будет один и тот же. От случая к случаю можете выбирать разные интерфейсы, наиболее удобные для решаемых вами конкретных задач.

Если вам требуется доступ к документам, напоминающий работу с файловой системой, неплохим выбором станет WebDAV. Если все, что у вас есть, — это браузер, то веб-интерфейс — это то, что вам нужно. Если ваша среда XML поддерживает eXist, это облегчит вашу жизнь. Если вы используете инструментарий, который поддерживает разные HTTP-запросы, то можете предпочесть REST-интерфейс. Если вы — разработчик на Java, обратите внимание на XML:DB API. Если вы хотите интегрировать свою базу данных с инструментарием Atom, то имейте в виду, что для этого случая разработан APP-интерфейс, а если вы — фанат веб-служб, то выбирайте между интерфейсами XML-RPC или SOAP.

Богатство этого множества интерфейсов означает, что ваши документы никогда не окажутся запертыми в базе данных и будут доступны в любом окружении.

XML в коммерческих СУРБД

Реальная ситуация такова, что огромные объемы данных в настоящее время хранятся в системах управления реляционными базами данных (СУРБД). Перемещение этих данных в XML-хранилища, даже если бы это было желательно и возможно, было бы огромной технической задачей. По причинам, схожим с причинами использования многими предприятиями языка программирования COBOL, эта задача никогда не будет решена. Реляционные данные хорошо подходят для многих практических целей, а многие бизнес-процессы критически зависят от как минимум некоторых из этих реляционных данных, так что было бы неоправданным риском вмешиваться в хорошо работающую систему только для того, чтобы перенести реляционные данные в XML.

Однако имеется и обратный эффект: желание использовать данные из традиционных реляционных хранилищ в современных бизнес-процессах, основанных на XML, внутри предприятия или для обмена информацией между ними. Поэтому вопрос ставится таким образом: *каким образом добавить XML-функциональность к существующим реляционным базам данных*?

Производители большинства СУРБД масштаба предприятия, таких как IBM DB2, Oracle и Microsoft SQL Server, использовали различные подходы. В этом разделе мы вкратце рассмотрим XML-функциональность в SQL Server 2000, который можно считать первым поколением СУРБД с поддержкой XML. Мы также более подробно рассмотрим SQL Server 2005, в который добавлено существенно

больше возможностей, включая, в частности, поддержку XQuery. Наконец мы бегло ознакомимся с некоторыми из возможностей, заявленных в следующей версии, в настоящее время известной под названием Katmai.

SQL Server доступен бесплатно в редакции, известной как SQL Express. Загрузить базу данных и инструментарий можно по адресу `http://msdn.microsoft.com/vstudio/express/sql/`. Советуем вам загрузить версию с расширенными службами, а также с системой Books Online и примерами баз данных для тестирования кода из этой главы. После инсталляции можете открыть менеджер базы данных SSMS, выбрав пункт системного меню Start⇨All programs⇨Microsoft SQL Server 2005⇨SQL Server Management Studio Express (Пуск⇨Программы⇨Microsoft SQL Server 2005⇨SQL Server Management Studio Express).

XML-функциональность в SQL Server 2000

SQL Server 2000 был первой версией SQL Server с поддержкой XML-функциональности. Однако, познакомившись с XML-функциональностью SQL Server 2005, вы увидите, что SQL Server 2000 обладает всего лишь частичной поддержкой XML-функциональности, которая включает следующее.

- Поддержка схем XDR (позже обновлена до схем XSD)
- HTTP-доступ к SQL Server 2000
- Функциональность SQLXML (добавлена в SQLXML 1.0)
- Компонент SOAP (добавлен в SQLXML 3.0)
- Выборка XML с использованием инструкции `SELECT` и конструкции `FOR XML`
- Запись XML с использованием `OPENXML`-провайдера
- Выборка XML-данных с помощью XPath 1.0

Ключевые слова SQL не чувствительны к регистру. Многие работающие с SQL программисты используют прописные буквы для ключевых слов SQL, но это не более чем соглашение.

Большинство этих возможностей доступно и в SQL Server 2005, хотя некоторые из способов обращения к XML-данным в SQL Server 2000, в частности посредством HTTP, в настоящее время рассматриваются как рискованные с точки зрения безопасности или слишком зависящие от других приложений, таких как IIS, так что в новой версии им предлагается более безопасная замена. SQL Server 2005 включает также собственный веб-сервер.

XML-функциональность в версии SQL Server 2005, последовавшей за SQL Server 2000, содержит множество новых возможностей. Она, кроме того, обладает способностью использовать код .NET для создания хранимых процедур, а также улучшенным T-SQL и встроенной поддержкой веб-служб.

Фундаментальные возможности XML, такие как возврат записей в виде XML вместо табличных данных, обратно совместимы с SQL Server 2000, хотя в настоящее время имеется больше возможностей по предоставлению в точности требуемого формата данных. Все рассматриваемые далее примеры выполнялись с применением SQL Server 2005, но при этом указано, какие из примеров будут работать и с предыдущей версией SQL Server.

Получение данных в виде XML с помощью FOR XML

`FOR XML` позволяет большинству стандартных SQL-запросов возвращать данные в виде XML, а не множества записей.

Имеется ряд опций, обеспечивающих управление форматом XML, именами элементов, тем, должны ли данные выводиться как элементы или атрибуты и как должны быть вложены дочерние записи. Рассмотрим приведенный далее запрос, который использует каталог `AdventureWorks`, включенный в качестве примера в состав сервера, который мы ранее рекомендовали вам загрузить с сайта Microsoft.

```
SELECT [PurchaseOrderID]
      ,[Status]
      ,[EmployeeID]
      ,[VendorID]
      ,[ShipMethodID]
      ,[OrderDate]
      ,[ShipDate]
      ,[SubTotal]
      ,[TaxAmt]
      ,[Freight]
      ,[TotalDue]
  FROM [AdventureWorks].[Purchasing].[PurchaseOrderHeader]
  WHERE [TotalDue] > 300000
```

Этот запрос выбирает перечисленные столбцы из таблицы `PurchaseOrderHeader` и ограничивается только теми, общая сумма которых превышает 300 000.

Вывод результатов запроса показан на рис. 10.10.

Чтобы преобразовать запрос для вывода XML, к нему следует добавить фразу `FOR XML`, за которой следует тип требуемого XML-запроса. Имеется четыре варианта — `RAW`, `AUTO`, `EXPLICIT` и `PATH` (последний — новинка, появившаяся в SQL Server 2005).

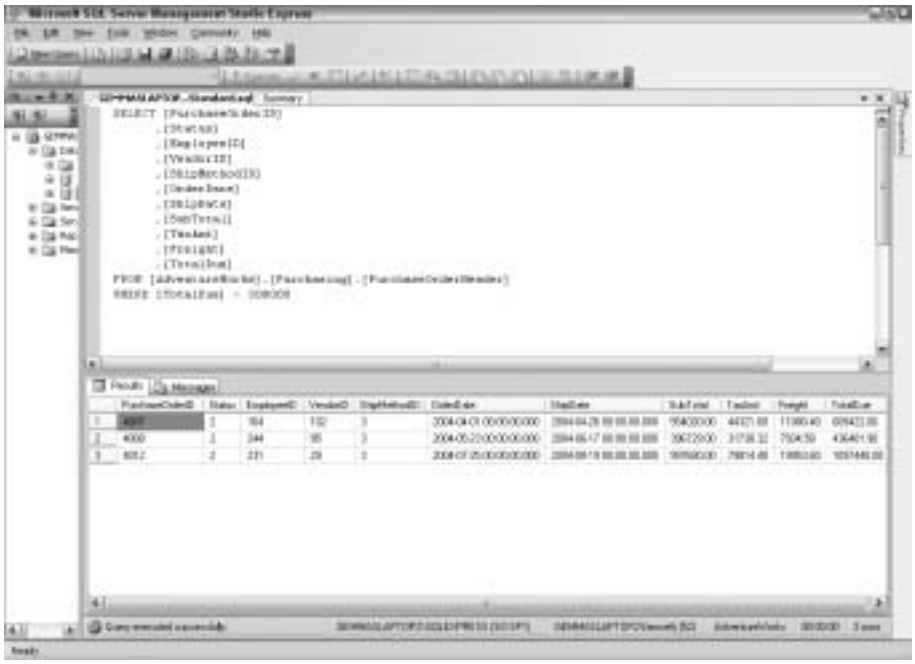

Рис. 10.10

Применение FOR XML RAW

Для выполнения запроса с помощью варианта `RAW` используется следующий синтаксис.

```
SELECT [PurchaseOrderID]
      ,[Status]
      ,[EmployeeID]
      ,[VendorID]
      ,[ShipMethodID]
      ,[OrderDate]
      ,[ShipDate]
      ,[SubTotal]
      ,[TaxAmt]
      ,[Freight]
      ,[TotalDue]
  FROM [AdventureWorks].[Purchasing].[PurchaseOrderHeader]
  WHERE [TotalDue] > 300000
FOR XML RAW
```

`FOR XML RAW` возвращает данные в виде атрибутов, “завернутых” в обобщенный элемент `<row>`. Возвращаемый XML представляет собой фрагмент, так как по умолчанию элемент документа не возвращается. Вот как выглядит одна из строк результата.

```
<row PurchaseOrderID="4007" Status="2" EmployeeID="164"
     VendorID="102" ShipMethodID="3"
     OrderDate="2004-04-01T00:00:00"
     ShipDate="2004-04-26T00:00:00" SubTotal="554020.0000"
     TaxAmt="44321.6000" Freight="11080.4000"
     TotalDue="609422.0000" />
```

Если вам нужны элементы, добавьте к запросу `,ELEMENTS`.

```
SELECT -- Тот же запрос, что и ранее
FOR XML RAW, ELEMENTS
```

Теперь результат приобретет следующий вид (показан только первый элемент `<row>`).

```
<row>
    <PurchaseOrderID>4007</PurchaseOrderID>
    <Status>2</Status>
    <EmployeeID>164</EmployeeID>
    <VendorID>102</VendorID>
    <ShipMethodID>3</ShipMethodID>
    <OrderDate>2004-04-01T00:00:00</OrderDate>
    <ShipDate>2004-04-26T00:00:00</ShipDate>
    <SubTotal>554020.0000</SubTotal>
    <TaxAmt>44321.6000</TaxAmt>
    <Freight>11080.4000</Freight>
    <TotalDue>609422.0000</TotalDue>
</row>
```

Обратите внимание на две вещи, которые вы, возможно, решите изменить при использовании `FOR XML RAW`: вы можете изменить имя элемента по умолчанию и превратить фрагмент в документ, добавив в него всеохватывающий элемент документа.

Чтобы изменить имя элемента по умолчанию, укажите нужное вам имя после ключевого слова `RAW`: `FOR XML RAW('Order')`. Чтобы указать элемент документа, добавьте ключевое слово `ROOT`, за которым следует имя в скобках.

```
SELECT -- Тот же запрос, что и ранее
FOR XML RAW('Order'), ROOT('Orders') , ELEMENTS
```

Вывод результата этого запроса показан на рис. 10.11.

Распространенной проблемой при взаимодействии с данными, хранящимися в реляционной базе данных, из объектно-ориентированных языков программирования, таких как C# или Java, является обработка пустых данных. Она же возникает и при возврате данных в виде XML. Традиционный подход в XML состоит в опускании в выходных данных элемента или атрибута с пустым значением. Например, если заказ имеет вид

```
<Order>
    <PurchaseOrderID>4007</PurchaseOrderID>
```

Рис. 10.11

```
    <Status>2</Status>
    <EmployeeID>164</EmployeeID>
    <VendorID>102</VendorID>
    <ShipMethodID>3</ShipMethodID>
    <OrderDate>2004-04-01T00:00:00</OrderDate>
    <ShipDate>2004-04-26T00:00:00</ShipDate>
    <SubTotal>554020.0000</SubTotal>
    <TaxAmt>44321.6000</TaxAmt>
    <Freight>11080.4000</Freight>
    <TotalDue>609422.0000</TotalDue>
    <OrderQty>5000</OrderQty>
    <ProductID>849</ProductID>
    <UnitPrice>24.7500</UnitPrice>
</Order>
```

то, если дата поставки — NULL, получим следующее.

```
<Order>
    <PurchaseOrderID>4007</PurchaseOrderID>
    <Status>2</Status>
    <EmployeeID>164</EmployeeID>
    <VendorID>102</VendorID>
    <ShipMethodID>3</ShipMethodID>
    <OrderDate>2004-04-01T00:00:00</OrderDate>
```

```
    <SubTotal>554020.0000</SubTotal>
    <TaxAmt>44321.6000</TaxAmt>
    <Freight>11080.4000</Freight>
    <TotalDue>609422.0000</TotalDue>
</Order>
```

Однако это не всегда удобно для обработки. Иногда для представления пустого значения проще иметь пустой элемент.

```
<ShipDate/>
```

Чтобы ваш запрос имел такой формат, добавьте параметр `XSINIL` после ключевого слова `ELEMENTS`.

```
SELECT -- Тот же запрос, что и ранее
FOR XML RAW('Order'), ROOT('Orders'), ELEMENTS XSINIL
```

При этом для заказа с отсутствующей датой вы получите следующий результат.

```
<Orders
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Order>
    <PurchaseOrderID>4007</PurchaseOrderID>
    <Status>2</Status>
    <EmployeeID>164</EmployeeID>
    <VendorID>102</VendorID>
    <ShipMethodID>3</ShipMethodID>
    <OrderDate>2004-04-01T00:00:00</OrderDate>
    <ShipDate xsi:nil="true" />
    <SubTotal>554020.0000</SubTotal>
    <TaxAmt>44321.6000</TaxAmt>
    <Freight>11080.4000</Freight>
    <TotalDue>609422.0000</TotalDue>
  </Order>
</Orders>
```

> *Если вы хотите выполнить приведенный запрос, но записи с пустой датой в базе данных нет, вы всегда можете сначала модифицировать таблицу.*

Обратите внимание, как атрибут `xsi:nil` используется для того, чтобы указать, что содержимое элемента не определено, а не является просто пустой строкой. Префикс `xsi` связан со стандартным пространством имен в элементе `Orders`.

Еще одна опция может оказаться полезной при передаче результатов запросов третьей стороне: к данным может быть привязана XML-схема, описывающая формат XML. Для этого нужно просто добавить запятую, за которой следует слово `XMLSCHEMA`, к существующему запросу.

```
FOR XML RAW('Order'), ROOT('Orders'),
    ELEMENTS XSINIL, XMLSCHEMA
```

Результаты будут идентичны предыдущим, но при этом отобразится схема наподобие приведенной далее.

```
<Orders xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xsd:schema
    targetNamespace="urn:schemas-microsoft-com:sql:SqlRowSet1"
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:sqltypes=http://schemas.microsoft.com/
    ➥ sqlserver/2004/sqltypes
    elementFormDefault="qualified">
    <xsd:import
      namespace=http://schemas.microsoft.com/
      ➥ sqlserver/2004/sqltypes
      schemaLocation="http://schemas.microsoft.com/
      ➥ sqlserver/2004/sqltypes/sqltypes.xsd"/>
    <xsd:element name="Order">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="PurchaseOrderID"
               type="sqltypes:int" nillable="1" />
          <xsd:element name="Status"
               type="sqltypes:tinyint" nillable="1" />
          <xsd:element name="EmployeeID"
               type="sqltypes:int" nillable="1" />
          <xsd:element name="VendorID"
               type="sqltypes:int" nillable="1" />
          <xsd:element name="ShipMethodID"
               type="sqltypes:int" nillable="1" />
          <xsd:element name="OrderDate"
               type="sqltypes:datetime" nillable="1" />
          <xsd:element name="ShipDate"
               type="sqltypes:datetime" nillable="1" />
          <xsd:element name="SubTotal"
               type="sqltypes:money" nillable="1" />
          <xsd:element name="TaxAmt"
               type="sqltypes:money" nillable="1" />
          <xsd:element name="Freight"
               type="sqltypes:money" nillable="1" />
          <xsd:element name="TotalDue"
               type="sqltypes:money" nillable="1" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <!-- Остальная часть заказа, такая же, как и ранее -->
</Orders>
```

Применение FOR XML AUTO

Хотя применение FOR XML RAW дает неплохую гибкость, проблемы начинаются, когда ваш запрос приводит к вложенным данным, таким как все заказы с их отдельными элементами. Вместо иерархического XML-документа вы получите один элемент, что усложняет обработку информации и приводит к дополнительным расходам ресурсов и потере одной из самых сильных сторон XML — его возможности представления вложенных данных. Для преодоления этих проблем можно воспользоваться конструкцией FOR XML AUTO, являющейся темой приведенного далее практического примера.

Попробуй Использование FOR XML AUTO

1. Чтобы ознакомиться с работой FOR XML AUTO, просто замените ключевое слово RAW словом AUTO в запросе из предыдущего раздела.

```
SELECT [PurchaseOrderID]
      ,[Status]
      ,[EmployeeID]
      ,[VendorID]
      ,[ShipMethodID]
      ,[OrderDate]
      ,[ShipDate]
      ,[SubTotal]
      ,[TaxAmt]
      ,[Freight]
      ,[TotalDue]
FROM [AdventureWorks].[Purchasing].[PurchaseOrderHeader]
WHERE [TotalDue] > 300000
FOR XML AUTO
```

Вы не увидите больших отличий результатов этого запроса по сравнению с RAW-версией, если не считать тот факт, что имя элемента, хранящего данные, порождается из имени таблицы, а не представляет собой обобщенный элемент row.

```
<AdventureWorks.Purchasing.PurchaseOrderHeader
    PurchaseOrderID="4007" Status="2" EmployeeID="164"
    VendorID="102" ShipMethodID="3"
    OrderDate="2004-04-01T00:00:00"
    ShipDate="2004-04-26T00:00:00"
    SubTotal="554020.0000" TaxAmt="44321.6000"
    Freight="11080.4000" TotalDue="609422.0000" />
```

Результат представляет собой, как и ранее, фрагмент без охватывающего элемента документа.

2. Реальное отличие становится понятным, когда выполняется запрос на получение данных из двух связанных таблиц. Приведенный далее SQL выводит все заказы вместе с их отдельными элементами.

```
SELECT [PurchaseOrderHeader].[PurchaseOrderID]
      ,[PurchaseOrderHeader].[Status]
      ,[PurchaseOrderHeader].[EmployeeID]
      ,[PurchaseOrderHeader].[VendorID]
      ,[PurchaseOrderHeader].[ShipMethodID]
      ,[PurchaseOrderHeader].[OrderDate]
      ,[PurchaseOrderHeader].[ShipDate]
      ,[PurchaseOrderHeader].[SubTotal]
      ,[PurchaseOrderHeader].[TaxAmt]
      ,[PurchaseOrderHeader].[Freight]
      ,[PurchaseOrderHeader].[TotalDue]
      ,[PurchaseOrderDetail].[OrderQty]
      ,[PurchaseOrderDetail].[ProductID]
      ,[PurchaseOrderDetail].[UnitPrice]
FROM [Purchasing].[PurchaseOrderHeader] PurchaseOrderHeader
INNER JOIN
    Purchasing.PurchaseOrderDetail PurchaseOrderDetail
    ON PurchaseOrderHeader.[PurchaseOrderID] =
       PurchaseOrderDetail.[PurchaseOrderID]
WHERE [PurchaseOrderHeader].[TotalDue] > 300000
```

Здесь таблицы получают псевдонимы и объединяются по полю `PurchaseOrderId`. Результат запроса показан на рис. 10.12.

Как это работает

Три последних столбца в результирующем множестве принадлежат отдельным пунктам заказа, и для каждого пункта одного и того же заказа предыдущие столбцы с информацией о заказе одинаковы, так что в таком табличном представлении результатов вложенность данных оказывается потерянной.

Если этот запрос модифицировать с помощью конструкции `FOR XML AUTO`, то иерархичность немедленно становится совершенно очевидной.

```
SELECT [PurchaseOrderHeader].[PurchaseOrderID]
      ,[PurchaseOrderHeader].[Status]
      ,[PurchaseOrderHeader].[EmployeeID]
      ,[PurchaseOrderHeader].[VendorID]
      ,[PurchaseOrderHeader].[ShipMethodID]
      ,[PurchaseOrderHeader].[OrderDate]
      ,[PurchaseOrderHeader].[ShipDate]
      ,[PurchaseOrderHeader].[SubTotal]
      ,[PurchaseOrderHeader].[TaxAmt]
      ,[PurchaseOrderHeader].[Freight]
      ,[PurchaseOrderHeader].[TotalDue]
```

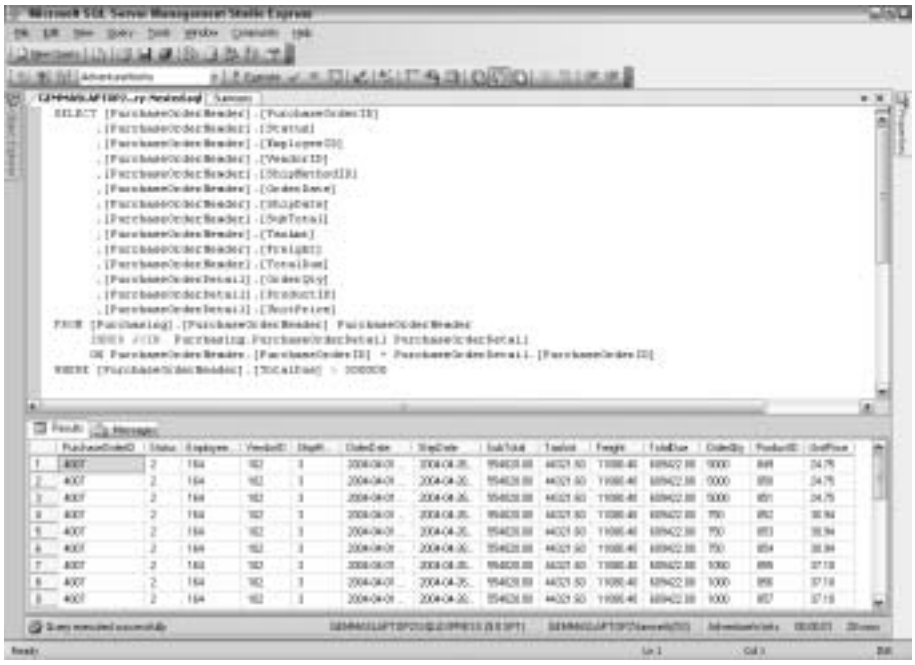

Рис. 10.12

```
      ,[PurchaseOrderDetail].[OrderQty]
      ,[PurchaseOrderDetail].[ProductID]
      ,[PurchaseOrderDetail].[UnitPrice]
FROM [Purchasing].[PurchaseOrderHeader] PurchaseOrderHeader
INNER JOIN Purchasing.PurchaseOrderDetail PurchaseOrderDetail
        ON PurchaseOrderHeader.[PurchaseOrderID] =
           PurchaseOrderDetail.[PurchaseOrderID]
WHERE [PurchaseOrderHeader].[TotalDue] > 300000
FOR XML AUTO, ROOT('Orders')
```

Обратите внимание на определение корневого элемента, такого же, как и в случае RAW. Результат выполнения запроса имеет следующий вид.

```
<Orders>
  <PurchaseOrderHeader PurchaseOrderID="4007" Status="2"
      EmployeeID="164" VendorID="102" ShipMethodID="3"
      OrderDate="2004-04-01T00:00:00"
      ShipDate="2004-04-26T00:00:00" SubTotal="554020.0000"
      TaxAmt="44321.6000" Freight="11080.4000"
      TotalDue="609422.0000">
    <PurchaseOrderDetail OrderQty="5000" ProductID="849"
      UnitPrice="24.7500" />
    <PurchaseOrderDetail OrderQty="5000" ProductID="850"
```

```
      UnitPrice="24.7500" />
    <!-- Другие элементы PurchaseOrderDetail -->
  </PurchaseOrderHeader>
  <PurchaseOrderHeader PurchaseOrderID="4008" Status="2"
      EmployeeID="244" VendorID="95" ShipMethodID="3"
      OrderDate="2004-05-23T00:00:00"
      ShipDate="2004-06-17T00:00:00" SubTotal="396729.0000"
      TaxAmt="31738.3200" Freight="7934.5800"
      TotalDue="436401.9000">
    <PurchaseOrderDetail OrderQty="700" ProductID="858"
      UnitPrice="9.1500" />
    <PurchaseOrderDetail OrderQty="700" ProductID="859"
      UnitPrice="9.1500" />
    <!-- Другие элементы PurchaseOrderDetail -->
  </PurchaseOrderHeader>
  <PurchaseOrderHeader PurchaseOrderID="4012" Status="2"
      EmployeeID="231" VendorID="29" ShipMethodID="3"
      OrderDate="2004-07-25T00:00:00"
      ShipDate="2004-08-19T00:00:00" SubTotal="997680.0000"
      TaxAmt="79814.4000" Freight="19953.6000"
      TotalDue="1097448.0000">
    <PurchaseOrderDetail OrderQty="6000" ProductID="881"
      UnitPrice="41.5700" />
    <PurchaseOrderDetail OrderQty="6000" ProductID="882"
      UnitPrice="41.5700" />
    <!-- Другие элементы PurchaseOrderDetail -->
  </PurchaseOrderHeader>
</Orders>
```

Другие опции, доступные в конструкции `FOR XML RAW`, такие как `ELEMENTS`, `XSINIL` и `XMLSCHEMA`, доступны и в конструкции `FOR XML AUTO`.

> *Доступны и некоторые реже используемые возможности, такие как возврат бинарных данных и использование конструкции `GROUP BY` в XML-запросах. Подробнее они рассматриваются в справке SQL SERVER 2005 Books Online (BOL), доступной для загрузки по адресу `www.microsoft.com/technet/prodtechnol/sql/2005/downloads/books.mspx`.*

Несмотря на различные опции, доступные как в версии `RAW`, так и в версии `AUTO` конструкции `FOR XML`, вы, вероятно, столкнетесь с ситуациями, когда ни одна из них не даст требуемого результата. Наиболее распространенный сценарий — когда вам нужна комбинация и элементов, и атрибутов, а не только те или другие. Для таких случаев предназначены конструкции `FOR XML EXPLICIT` и `FOR XML PATH`; последняя представляет собой новую возможность, доступную только в SQL Server 2005.

Применение FOR XML EXPLICIT

Опция EXPLICIT предоставляет практически неограниченный контроль над получаемым XML-форматом, но за это приходится платить определенную цену. Используемый синтаксис труден для понимания, а поскольку механизм, используемый для построения результирующего XML основан на однонаправленной записи XML, результаты должны быть сгруппированы и упорядочены весьма специфичным способом. Если вы не строго ограничены использованием SQL Server 2000, Microsoft и другие эксперты советуют вместо этой опции применять опцию PATH. Если же вы вынуждены использовать EXPLICIT, полную информацию о том, как это сделать, вы найдете в SQL Server BOL.

Применение FOR XML PATH

Опция PATH делает построение вложенного XML с комбинацией элементов и атрибутов относительно простым делом. Вернемся к первому примеру запроса, в котором выбирается информация о заказах общей суммой свыше 300 000 и из нее с помощью опции AUTO формируется XML с атрибутами.

```
<Orders>
  <PurchaseOrderHeader PurchaseOrderID="4007" Status="2"
    EmployeeID="164" VendorID="102" ShipMethodID="3"
    OrderDate="2004-04-01T00:00:00"
    ShipDate="2004-04-26T00:00:00" SubTotal="554020.0000"
    TaxAmt="44321.6000" Freight="11080.4000"
    TotalDue="609422.0000" />
  <PurchaseOrderHeader PurchaseOrderID="4008" Status="2"
    EmployeeID="244" VendorID="95" ShipMethodID="3"
    OrderDate="2004-05-23T00:00:00"
    ShipDate="2004-06-17T00:00:00" SubTotal="396729.0000"
    TaxAmt="31738.3200" Freight="7934.5800"
    TotalDue="436401.9000" />
  <PurchaseOrderHeader PurchaseOrderID="4012" Status="2"
    EmployeeID="231" VendorID="29" ShipMethodID="3"
    OrderDate="2004-07-25T00:00:00"
    ShipDate="2004-08-19T00:00:00" SubTotal="997680.0000"
    TaxAmt="79814.4000" Freight="19953.6000"
    TotalDue="1097448.0000" />
</Orders>
```

Но что делать, если требуется иная схема, например если PurchaseOrderID, EmployeedID и status должны быть атрибутами, а остальные данные должны быть представлены элементами? Опция PATH использует псевдонимы столбцов для указания, как именно следует структурировать XML. Используемый синтаксис подобен синтаксису XPath, рассмотренному в главе 7, “XPath”, откуда и происходит ключевое слово PATH.

Запрос данных о заказах как с атрибутами, так и с элементами использует PATH следующим образом.

```
SELECT [PurchaseOrderID] [@PurchaseOrderID]
      ,[Status] [@Status]
      ,[EmployeeID] [@EmployeeID]
      ,[VendorID]
      ,[ShipMethodID]
      ,[OrderDate]
      ,[ShipDate]
      ,[SubTotal]
      ,[TaxAmt]
      ,[Freight]
      ,[TotalDue]
FROM [AdventureWorks].[Purchasing].[PurchaseOrderHeader]
     PurchaseOrderHeader
WHERE [TotalDue] > 300000
FOR XML PATH('Order'), ROOT('Orders')
```

Обратите внимание на то, как данные, которые должны возвращаться в виде атрибутов, используют псевдонимы имен столбцов, начинающиеся с @. Столбцы без псевдонимов возвращаются в виде элементов. Результаты этого запроса дают XML следующего вида.

```
<Orders>
  <Order PurchaseOrderID="4007" Status="2" EmployeeID="164">
    <Vendor>102</Vendor>
    <ShipMethodID>3</ShipMethodID>
    <OrderDate>2004-04-01T00:00:00</OrderDate>
    <ShipDate>2004-04-26T00:00:00</ShipDate>
    <SubTotal>554020.0000</SubTotal>
    <TaxAmt>44321.6000</TaxAmt>
    <Freight>11080.4000</Freight>
    <TotalDue>609422.0000</TotalDue>
  </Order>
  <!-- Прочие элементы Order -->
</Orders>
```

Опция PATH предоставляет также возможности управления вложением. Распространенным способом решения этой задачи, кроме применения JOIN, как в предыдущем примере, является использование подзапросов. Приведенный далее фрагмент показывает заголовок запроса с использованием атрибутов, а детали запроса — как вложенные элементы.

```
SELECT [PurchaseOrderHeader].[PurchaseOrderID]
                            [@PurchaseOrderID]
      ,[PurchaseOrderHeader].[Status] [@Status]
      ,[PurchaseOrderHeader].[EmployeeID] [@EmployeeID]
```

```
      ,[PurchaseOrderHeader].[VendorID] [@VendorID]
      ,[PurchaseOrderHeader].[ShipMethodID] [@ShipMethodID]
      ,[PurchaseOrderHeader].[OrderDate] [@OrderDate]
      ,[PurchaseOrderHeader].[ShipDate] [@ShipDate]
      ,[PurchaseOrderHeader].[SubTotal] [@SubTotal]
      ,[PurchaseOrderHeader].[TaxAmt] [@TaxAmt]
      ,[PurchaseOrderHeader].[Freight] [@Freight]
      ,[PurchaseOrderHeader].[TotalDue] [@TotalDue]
      ,(
        SELECT [PurchaseOrderDetail].[OrderQty]
              ,[PurchaseOrderDetail].[ProductID]
              ,[PurchaseOrderDetail].[UnitPrice]
        FROM [Purchasing].[PurchaseOrderDetail]
                          PurchaseOrderDetail
        WHERE PurchaseOrderHeader.[PurchaseOrderID] =
              PurchaseOrderDetail.[PurchaseOrderID]
        ORDER BY PurchaseOrderDetail.[PurchaseOrderID]
        FOR XML PATH('OrderDetail'), TYPE
       )
FROM [Purchasing].[PurchaseOrderHeader] PurchaseOrderHeader
WHERE [PurchaseOrderHeader].[TotalDue] > 300000
FOR XML PATH('Order'), ROOT('Orders')
```

Основная часть запроса, без вложенного `SELECT`, практически та же, что и ранее, но все выводимые столбцы определены как атрибуты, как указывают имена псевдонимов, начинающиеся с символа `@`.

```
SELECT [PurchaseOrderHeader].[PurchaseOrderID]
                            [@PurchaseOrderID]
      ,[PurchaseOrderHeader].[Status] [@Status]
      ,[PurchaseOrderHeader].[EmployeeID] [@EmployeeID]
      ,[PurchaseOrderHeader].[VendorID] [@VendorID]
      ,[PurchaseOrderHeader].[ShipMethodID] [@ShipMethodID]
      ,[PurchaseOrderHeader].[OrderDate] [@OrderDate]
      ,[PurchaseOrderHeader].[ShipDate] [@ShipDate]
      ,[PurchaseOrderHeader].[SubTotal] [@SubTotal]
      ,[PurchaseOrderHeader].[TaxAmt] [@TaxAmt]
      ,[PurchaseOrderHeader].[Freight] [@Freight]
      ,[PurchaseOrderHeader].[TotalDue] [@TotalDue]
      ,(
        -- Внутренний запрос
       )
FROM [Purchasing].[PurchaseOrderHeader] PurchaseOrderHeader
WHERE [PurchaseOrderHeader].[TotalDue] > 300000
FOR XML PATH('Order'), ROOT('Orders')
```

Внутренний запрос возвращает детальную информацию о заказе, определяемом внутренним запросом. Это достигается путем приравнивания поля `PurchaseOrderDetail.PurchaseOrderId` из внешнего запроса полю `PurchaseOrderDetail.PurchaseOrderID` во вложенном запросе. (В терминах SQL это называется *связанным подзапросом* (correlated subquery).)

```
SELECT [PurchaseOrderDetail].[OrderQty]
      ,[PurchaseOrderDetail].[ProductID]
      ,[PurchaseOrderDetail].[UnitPrice]
FROM [Purchasing].[PurchaseOrderDetail] PurchaseOrderDetail
WHERE PurchaseOrderHeader.[PurchaseOrderID] =
      PurchaseOrderDetail.[PurchaseOrderID]
ORDER BY PurchaseOrderDetail.[PurchaseOrderID]
FOR XML PATH('OrderDetail'), TYPE
```

Обратите внимание на опцию `TYPE` в конце подзапроса. Она появилась в SQL Server 2005 и указывает, что результирующие данные должны быть приведены к типам данных XML (более детально этот вопрос рассматривается ниже). Эта опция гарантирует, что данные вставляются, как XML, а не как строки. Вывод данного запроса имеет следующий вид.

```
<Orders>
  <Order PurchaseOrderID="4007" Status="2"
    EmployeeID="164" VendorID="102"
    ShipMethodID="3" OrderDate="2004-04-01T00:00:00"
    ShipDate="2004-04-26T00:00:00" SubTotal="554020.0000"
    TaxAmt="44321.6000" Freight="11080.4000"
    TotalDue="609422.0000">
    <OrderDetail>
      <OrderQty>5000</OrderQty>
      <ProductID>849</ProductID>
      <UnitPrice>24.7500</UnitPrice>
    </OrderDetail>
    <OrderDetail>
      <OrderQty>5000</OrderQty>
      <ProductID>850</ProductID>
      <UnitPrice>24.7500</UnitPrice>
    </OrderDetail>
    <OrderDetail>
      <OrderQty>5000</OrderQty>
      <ProductID>851</ProductID>
      <UnitPrice>24.7500</UnitPrice>
    </OrderDetail>
    <!-- Прочие элементы OrderDetails -->
  </Order>
  <Order PurchaseOrderID="4008" Status="2"
    EmployeeID="244" VendorID="95"
```

```
    ShipMethodID="3" OrderDate="2004-05-23T00:00:00"
    ShipDate="2004-06-17T00:00:00" SubTotal="396729.0000"
    TaxAmt="31738.3200" Freight="7934.5800"
    TotalDue="436401.9000">
    <OrderDetail>
      <OrderQty>700</OrderQty>
      <ProductID>858</ProductID>
      <UnitPrice>9.1500</UnitPrice>
    </OrderDetail>
    <!-- Прочие элементы OrderDetails -->
  </Order>
  <Order PurchaseOrderID="4012" Status="2"
    EmployeeID="231" VendorID="29" ShipMethodID="3"
    OrderDate="2004-07-25T00:00:00"
    ShipDate="2004-08-19T00:00:00" SubTotal="997680.0000"
    TaxAmt="79814.4000" Freight="19953.6000"
    TotalDue="1097448.0000">
    <OrderDetail>
      <OrderQty>6000</OrderQty>
      <ProductID>881</ProductID>
      <UnitPrice>41.5700</UnitPrice>
    </OrderDetail>
    <!-- Прочие элементы OrderDetails -->
  </Order>
</Orders>
```

Поскольку во внутреннем запросе не используются псевдонимы, его столбцы представлены элементами XML.

> *Если вы удалите из внутреннего запроса подстроку* `,TYPE`*, детальная информация о запросе будет вставлена как управляющие последовательности XML, поскольку будет рассматриваться, как текстовые данные, а не разметка.*

Имеется масса других опций. Еще один пример показывает, как группировать данные внутри элементов. Две даты, связанные с заказом, группируются в элемент `Dates`, а элемент `OrderDetails` используется для хранения отдельных пунктов заказа.

```
SELECT [PurchaseOrderHeader].[PurchaseOrderID]
                            [@PurchaseOrderID]
      ,[PurchaseOrderHeader].[Status] [@Status]
      ,[PurchaseOrderHeader].[EmployeeID] [@EmployeeID]
      ,[PurchaseOrderHeader].[VendorID] [@VendorID]
      ,[PurchaseOrderHeader].[ShipMethodID] [@ShipMethodID]
      ,[PurchaseOrderHeader].[SubTotal] [@SubTotal]
      ,[PurchaseOrderHeader].[TaxAmt] [@TaxAmt]
```

```
      ,[PurchaseOrderHeader].[Freight] [@Freight]
      ,[PurchaseOrderHeader].[TotalDue] [@TotalDue]
      ,[PurchaseOrderHeader].[OrderDate] [Dates/Order]
      ,[PurchaseOrderHeader].[ShipDate] [Dates/Ship]
      ,(
        SELECT [PurchaseOrderDetail].[OrderQty]
              ,[PurchaseOrderDetail].[ProductID]
              ,[PurchaseOrderDetail].[UnitPrice]
        FROM [Purchasing].[PurchaseOrderDetail]
                          PurchaseOrderDetail
        WHERE PurchaseOrderHeader.[PurchaseOrderID] =
              PurchaseOrderDetail.[PurchaseOrderID]
        ORDER BY PurchaseOrderDetail.[PurchaseOrderID]
        FOR XML PATH('OrderDetail'), TYPE
       ) [OrderDetails]
FROM [Purchasing].[PurchaseOrderHeader] PurchaseOrderHeader
WHERE [PurchaseOrderHeader].[TotalDue] > 300000
FOR XML PATH('Order'), ROOT('Orders')
```

В приведенном коде ключевым моментом является изменение `OrderDate` и `ShipDate` во внешнем `SELECT`. Столбцы получают псевдонимы `Date/Order` и `Dates/Ship`, так что SQL Server создает новый элемент, `Dates`, для хранения этих двух значений. Имеется также псевдоним для всего подзапроса, `OrderDetails`, который заставляет все результаты быть сгруппированными в одном элементе. В результате получается следующий XML.

```
<Orders>
  <Order PurchaseOrderID="4007" Status="2" EmployeeID="164"
    VendorID="102" ShipMethodID="3" SubTotal="554020.0000"
    TaxAmt="44321.6000" Freight="11080.4000"
    TotalDue="609422.0000">
    <Dates>
      <Order>2004-04-01T00:00:00</Order>
      <Ship>2004-04-26T00:00:00</Ship>
    </Dates>
    <OrderDetails>
      <OrderDetail>
        <OrderQty>5000</OrderQty>
        <ProductID>849</ProductID>
        <UnitPrice>24.7500</UnitPrice>
      </OrderDetail>
      <OrderDetail>
        <OrderQty>5000</OrderQty>
        <ProductID>850</ProductID>
        <UnitPrice>24.7500</UnitPrice>
      </OrderDetail>
      <OrderDetail>
```

```
        <OrderQty>5000</OrderQty>
        <ProductID>851</ProductID>
        <UnitPrice>24.7500</UnitPrice>
      </OrderDetail>
      <!-- Другие элементы OrderDetail -->
    </OrderDetails>
  </Order>
  <!-- Другие элементы Order -->
</Orders>
```

У запросов PATH имеется масса других опций, включая управляющие созданием комментариев и текстового содержимого и добавлением объявлений пространств имен. Полное описание всех опций можно найти в Books Online.

В этом разделе мы рассмотрели получение XML для заданного реляционного ресурса. Следующая тема, OPENXML, связана с противоположной задачей — анализом XML и вставкой данных в реляционные таблицы.

Применение OPENXML

Существует три основные стадии обработки XML и вставки его содержимого в стандартные таблицы базы данных, процесса, именуемого также *измельчением* (shredding).

Первая стадия, аналогичная загрузке документа DOM, использует специальную сохраненную процедуру sp_xml_prepareDocument, которая предоставляет указатель, используемый другими методами для обращения к XML. Вот используемый для этого синтаксис.

```
DECLARE @XmlData NVARCHAR(MAX)
DECLARE @XmlPointer INT
SET @XmlData =
  '<root><element>One</element><element>Two</element></root>'
EXEC sp_xml_preparedocument @XmlPointer OUTPUT, @XmlData
```

NVARCHAR(MAX) — новый тип переменной в SQL Server 2005, созданный для замены других типов для больших текстовых данных, таких как NTEXT. В рассматриваемых примерах при использовании SQL Server 2000 вместо NVARCHAR(MAX) можно воспользоваться NVARCHAR(4000).

На заднем плане SQL Server анализирует текст XML и сохраняет детальную информацию во внутренней таблице. Переменная @XmlPointer используется для дальнейшего получения этих данных.

Следующая стадия состоит в действительной обработке XML. Ключевое слово OPENXML используется для представления XML в виде таблицы. За этим словом следует схема, указывающая, какие атрибуты и элементы необходимо вернуть.

```
SELECT *
FROM OPENXML (@XmlPointer, '/root/element', 2)
WITH (newName NVARCHAR(100) '.')
```

Первый аргумент `OPENXML` представляет собой указатель на проанализированный XML-документ. Второй аргумент — выражение XPath для узлов, которые будут использоваться в каждой строке вывода; в нашем случае для формирования выходной строки используется каждый элемент `element`. Третий аргумент определяет отображение данных: 1 — для атрибутов, 2 — для элементов.

Конструкция `WITH` отображает данные из каждого элемента на выходные данные. Первый элемент — имя столбца, второй — тип данных SQL и третий — выражение XPath, определяющее данные. В результате будет получена таблица из одной строки, как показано на рис. 10.13.

Рис. 10.13

В качестве альтернативы определения отображения с использованием конструкции `WITH` можно определить так называемую краевую таблицу (edge table); полное описание этого метода имеется в Books Online.

Последняя стадия использования `OPENXML` состоит в удалении XML-документа из внутренней таблицы SQL Server. Если это не сделать, то в конечном итоге память сервера исчерпается, и вы не сможете создавать новые XML-документы.

Удаление осуществляется с помощью сохраненной процедуры sp_xml_removedocument.

```
EXEC sp_xml_removedocument @XmlPointer
```

Одним большим преимуществом OPENXML является то, что вы можете создавать сохраненные процедуры, которые принимают аргументы в виде массивов. Это практически невозможно при использовании стандартного SQL. В качестве реального примера представьте себе приложение электронной коммерции, использующее концепцию традиционной корзины покупателя. После того как пользователь завершит выбор, вам нужно передать детальную информацию о корзине сохраненной процедуре для подсчета общей суммы или иных действий. Поскольку корзина может хранить переменное количество товаров, разработка сохраняемой процедуры оказывается непростым делом. При использовании строки XML ситуация существенно упрощается.

Сначала вы должны решить вопрос о формате корзины. Предположим, что каждый товар представлен элементом item с двумя атрибутами — идентификатором товара и его количеством.

```
<basket customerId="12345">
    <item productId="a123" quantity="1"/>
    <item productId="b456" quantity="3"/>
    <item productId="c789" quantity="2"/>
</basket>
```

Приведенный далее сценарий SQL преобразует эти данные в таблицу, готовую для дальнейшей обработки или хранения.

```
DECLARE @BasketXml NVARCHAR(MAX)
DECLARE @BasketPointer INT
SET @BasketXml = '<basket customerId="12345">
                    <item productId="a123" quantity="1"/>
                    <item productId="b456" quantity="3"/>
                    <item productId="c789" quantity="2"/>
                  </basket>'

EXEC sp_xml_preparedocument
    @BasketPointer OUTPUT, @BasketXml

DECLARE @BasicBasket TABLE
(
    ProductId NVARCHAR(20),
    Quantity INT
)

INSERT @BasicBasket
    SELECT productId ProductId, quantity Quantity
```

```
    FROM OPENXML (@BasketPointer, '/basket/item', 1)
    WITH (productId NVARCHAR(20), quantity INT)

EXEC sp_xml_removedocument @BasketPointer

SELECT * FROM @BasicBasket
```

Первая часть этого кода аналогична ранее рассмотренному блоку. Здесь имеются две переменные: `@BasketXml` — для XML-строки и `@BasketPointer` — для хранения указателя на проанализированные XML-данные. В реальной ситуации сохраненная процедура получит `@BasketXml` в качестве одного из своих параметров.

XML анализируется и преобразуется во внутренний формат, как и ранее.

```
EXEC sp_xml_prepareDocument @BasketPointer OUTPUT, @BasketXml
```

Затем для хранения извлеченных XML-данных объявляется простая табличная переменная.

```
DECLARE @BasicBasket TABLE
(
    ProductId NVARCHAR(20),
    Quantity INT
)
```

И наконец, далее следует ключевая инструкция `OPENXML`.

```
SELECT productId ProductId, quantity Quantity
FROM OPENXML (@BasketPointer, '/basket/item', 1)
WITH (productId NVARCHAR(20), quantity INT)
```

В этом примере третий параметр `OPENXML` равен `1`, что указывает на отображение данных в атрибуты. Конструкция `WITH` гласит, что два результирующих столбца таблицы — `NVARCHAR(20)` и `INT`.

Наконец, освобождается указатель XML и выводятся результаты, как показано на рис. 10.14.

В следующем разделе будет рассмотрено наиболее важное новшество в SQL Server 2005 — тип данных XML.

Тип данных xml

SQL Server 2005 добавляет новый тип данных `xml`, который означает, что XML-документы могут храниться в базе данных SQL Server 2005 без разделения на части (это единственный способ хранения в SQL Server 2000) и сохранения в разных реляционных таблицах, которые удовлетворяют реляционной модели данных, или в виде простой последовательности символов с потерей логического содержимого XML-документа. XML-данные, хранящиеся как тип данных `xml`, по сути, могут рассматриваться так, как если бы они оставались XML-документом. Фактически тип данных `xml` хранится в соответствующем бинарном формате, но,

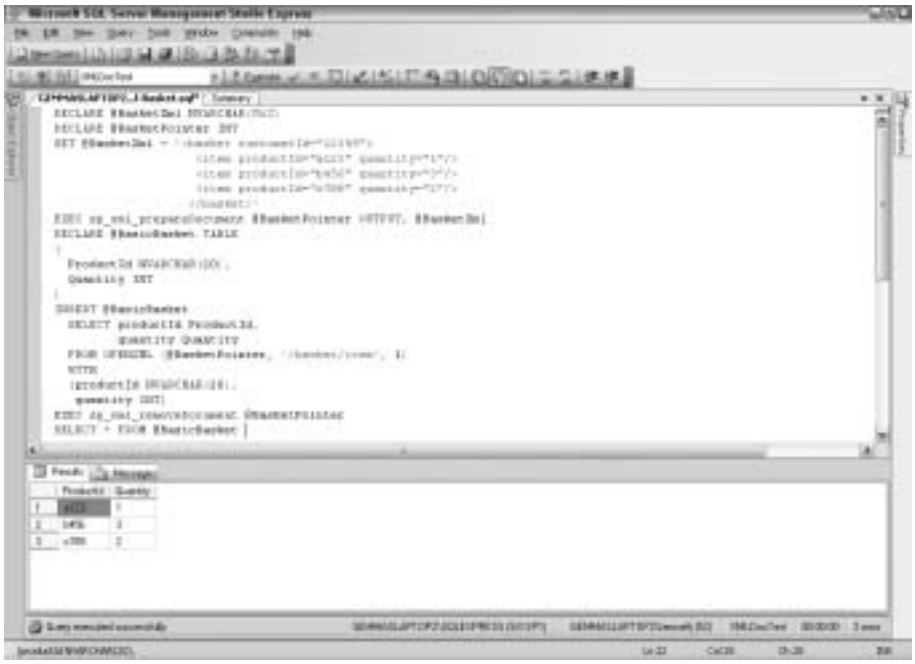

Рис. 10.14

с точки зрения разработчика, данные остаются доступными как XML, с нетронутой логической структурой.

> *Имеется одно или два отличия между данными, сохраненными SQL Server, и исходным документом, так что невозможно выполнить прямое и обратное преобразования и получить идентичную исходному документу копию, хотя XML Infoset и сохраняется (дополнительную информацию ищите в главе 7, "XPath").*

Это важное усовершенствование по сравнению с SQL Server 2000, где XML-документы либо преобразовывались с помощью `OpenXML`, как описывалось выше, либо сохранялись в текстовом формате, так что выборка представляла собой обычную последовательность символов. Запросы к таким документам не могли полагаться на правильность построения выбираемого XML, а вся обработка XML должна была иметь место вне базы данных, зачастую на среднем уровне трехуровневого приложения.

Наличие типа данных `xml` означает, что XML-документы, сохраненные, например, в столбце SQL Server 2005, могут рассматриваться, как если бы они были коллекциями XML-документов на вашем жестком диске. Конечно, детали интерфейса к этому XML специфичны для SQL Server 2005, так же как при обращении

к XML, сохраненным в базе данных eXist, имелись аспекты, специфичные для eXist.

Среди общих преимуществ хранения в SQL Server 2005 нужно выделить то, что при таком хранении XML обеспечиваются безопасность, масштабируемость и другие аспекты СУРБД уровня предприятия. Можно также связать со столбцом схемы XML, и при запросе документа будет возвращаться соответствующий тип. Это существенное усовершенствование по сравнению с предыдущей версией, требовавшей большого количества `CAST` или `CONVERT`.

XML-документы, хранящиеся в версии 2005, могут рассматриваться как XML и во многих других отношениях. Один из практических результатов этого — возможность использовать XQuery (см. главу 9, "XQuery") для запросов этих столбцов XML. Вероятно, неожиданным окажется то, что в этой версии нельзя сравнивать два экземпляра XML-документов, частью из-за гибкости синтаксиса XML. Рассмотрим, например, тонкости попыток сравнить два длинных XML-документа, которые могут использовать одинарные или двойные кавычки для значений атрибутов, иметь разные порядки атрибутов, использовать разные префиксы пространств имен (при том, что URI пространства имен может быть одним и тем же) и иметь пустые элементы, записанные с помощью как начальных и конечных дескрипторов, так и дескрипторов пустых элементов.

Документы, сохраненные как тип данных `XML`, могут (необязательно) быть проверены на соответствие некоторому документу W3C XML Schema. XML-данные, не связанные с документом схемы, называются *нетипизированными* (untyped), а XML, связанные со схемой, — *типизированными* (typed).

Рассмотрим создание простой таблицы для хранения XML-документов в SQL Server 2005. Графический интерфейс SQL Server 2005 существенно изменился по сравнению с графическим интерфейсом SQL Server 2000. Основным графическим инструментом для управления объектами базы данных и написания SQL кода является SQL Server Management Studio. SQL Management Studio основан на Microsoft Visual Studio, и в SQL Server 2005 разработчики могут создавать решения и проекты способами, которые будут знакомы им по работе в Visual Studio.

Попробуй Создание XML-документов в SQL Server

1. После выполнения всех инструкций по установке на странице `http://msdn.microsoft.com/vstudio/express/sql/` откройте Management Studio и подключитесь к интересующему вас экземпляру SQL Server.
2. В Object Explorer раскройте узлы так, чтобы увидеть список User Databases. Щелкните правой кнопкой мыши, выберите пункт New Database, и откроется диалоговое окно, в котором вы вводите имя базы данных (в данном

примере — XMLDocTest). Перед тем как щелкнуть на кнопке OK, убедитесь в том, что установлен флаг Full Text Indexing.

3. Создайте таблицу Docs с помощью следующего SQL.

```
CREATE TABLE dbo.Docs (
    DocID INTEGER IDENTITY PRIMARY KEY,
    XMLDoc XML
)
```

Столбец XMLDoc имеет тип xml (поскольку мы имеем дело с инструкцией SQL, тип данных не чувствителен к регистру). Теперь у вас есть пустая таблица.

4. Для нашего примера достаточно добавить простой документ со следующей структурой.

```
<Person>
    <FirstName></FirstName>
    <LastName></LastName>
</Person>
```

5. Вставьте XML-документ с помощью инструкции SQL INSERT, приведенной далее (которая вставляет в таблицу XML-документ).

```
INSERT Docs
    VALUES ('<Person><FirstName>Joe</FirstName>
             <LastName>Fawcett</LastName></Person>'
)
```

6. После изменения значений элементов FirstName и LastName и добавления нескольких новых документов в столбец XMLDoc убедитесь с помощью приведенной далее инструкции SQL, что выборка работает корректно.

```
SELECT XMLDoc FROM Docs
```

Результат работы данного запроса показан на рис. 10.15.

Значения, содержащиеся в столбце XMLDoc, выводятся на нижней панели. Ниже мы создадим несколько простых запросов XQuery.

Как это работает

Первый шаг состоял в создании таблицы Docs, в которой имеется один столбец XmlDoc, определенный как имеющий новый тип данных xml. На следующем этапе для добавления некоторого текста в этот столбец использован традиционный запрос INSERT. Поскольку столбец определен как имеющий тип xml, данные конвертируются из текста в XML-документ. Этот документ может быть получен с помощью стандартного запроса SELECT.

Альтернативой получения всего XML-документа является получение только его части (которое будет продемонстрировано ниже в данной главе).

* * *

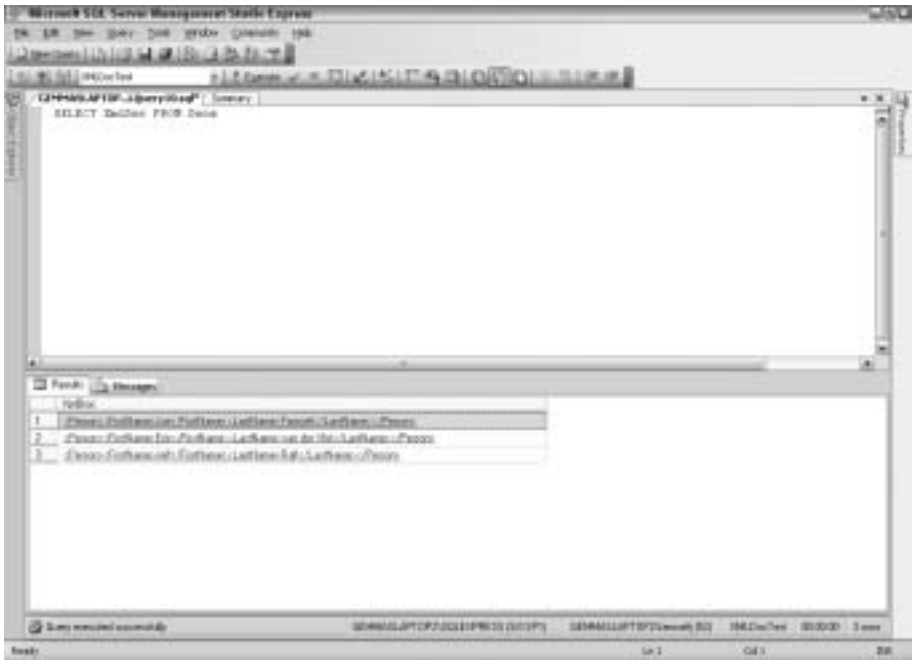

Рис. 10.15

XML-документы в SQL Server 2005 могут быть индексированы для более эффективной выборки; при этом может быть создан (необязательный) полнотекстовый индекс. Для создания полнотекстового индекса документа воспользуйтесь командой наподобие следующей.

```
-- Если каталог пока что не существует
CREATE FULLTEXT CATALOG ft ON DEFAULT
CREATE FULLTEXT INDEX ON dbo.Docs(XmlDoc)
    KEY INDEX <primary key name>Doc)
```

Тип данных `xml` позволяет использовать следующие методы: `query()`, `value()`, `exist()`, `modify()` и `nodes()`.

XQuery в SQL Server 2005

Тип данных `xml` может быть запрошен с использованием языка XQuery, с которым вы познакомились в главе 9, “XQuery”. В SQL Server 2005 выражения XQuery встроены в Transact-SQL. Transact-SQL — разновидность языка SQL, используемая в SQL Server.

Стандарт XQuery детально рассмотрен в главе 9, “XQuery”, так что в последующих разделах мы сосредоточимся только на дополнениях, предоставляемых SQL Server.

Компания Microsoft кое-чему научилась на своих ошибках и отделила нестандартную функциональность от стандартного XQuery. Таким образом, можно будет добавить возможности вставки, удаления и замены, когда они станут частью XQuery, не затрагивая при этом существующий код. Нестандартные расширения при этом станут не рекомендуемыми к использованию, а со временем и вовсе сойдут на нет, и от них можно будет спокойно отказаться.

Расширения XQuery в SQL Server 2005

Спецификация W3C XQuery ограничена в том плане, что запрашивать можно только источники XML-данных (или с поддержкой XML). В XQuery 1.0 нет возможности выполнить удаления, вставку новых данных или их изменение (которое, по сути, является комбинацией первых двух действий). В SQL Server 2005 язык модификации XML-данных (XML Data Modification Language — DML) добавляет три ключевых слова к функциональности, имеющейся в XQuery 1.0.

- `delete`
- `insert`
- `replace value of`

Обратите внимание, что, хотя SQL сам по себе не чувствителен к регистру, перечисленные команды должны использоваться только в нижнем регистре. Если вы напишете `DELETE` вместо `delete`, то получите загадочное сообщение об ошибке.

Попробуй Удаление с помощью XML DML

Рассмотрим, как используется ключевое слово `delete`. Далее приведен пример такого использования.

```
DECLARE @myDoc XML
    SET @myDoc = '<Person><FirstName>Joe</FirstName>
                  <LastName>Fawcett</LastName></Person>
                 '
SELECT @myDoc
SET @myDoc.modify(' delete /Person/*[2]
                  ')
SELECT @myDoc
```

Если вы работаете с SQL Server 2005, выполните следующие действия.

1. Откройте SQL Server Studio.
2. Подключитесь к экземпляру по умолчанию.
3. Выберите в меню New SQL Server Query.

4. Введите приведенный выше код.
5. Нажмите <F5> для запуска кода SQL. Если вы ввели его корректно, будет выведен исходный документ, а ниже — модифицированный. В модифицированном документе удален элемент `LastName`.
6. Измените ширину столбцов, чтобы увидеть весь XML.

Как это работает

Первая строка кода объявляет переменную `myDoc` и указывает, что ее тип данных — `xml`. Инструкция `SET`

```
SET @myDoc = '<Person><FirstName>Joe</FirstName>
              <LastName>Fawcett</LastName></Person>
             ')
```

указывает значение переменной `myDoc`. Это знакомый элемент `Person` с дочерними элементами `FirstName` и `LastName` и соответствующим текстовым содержимым.

Инструкция `SELECT`, следующая за инструкцией `SET`, приводит к выводу значения `myDoc`. Затем для изменения значения с типом `xml` использована функция `modify`.

```
SET @myDoc.modify('
    delete /Person/*[2]
')
```

Инструкция DML в функции `modify` чувствительна к регистру, как и XQuery. Ключевое слово `delete` используется для того, чтобы указать, какая часть XML-документа должна быть удалена. В данном случае выражение XPath `/Person/*[2]` указывает, что следует удалить второй дочерний элемент элемента `Person`, т.е. элемент `LastName`.

Последняя инструкция `SELECT` выводит значение `myDoc` после удаления. На рис. 10.16 приведены результаты выполнения обеих инструкций `SELECT`.

Попробуй Вставка с помощью XML DML

В этом примере используется ключевое слово `insert`. Код Transact-SQL приведен далее.

```
DECLARE @myDoc XML
SET @myDoc = '<Person><LastName>Fawcett</LastName></Person>'
SELECT @myDoc
SET @myDoc.modify(' insert <FirstName>Joe</FirstName> as
                    first into /Person[1] ')
SELECT @myDoc
```

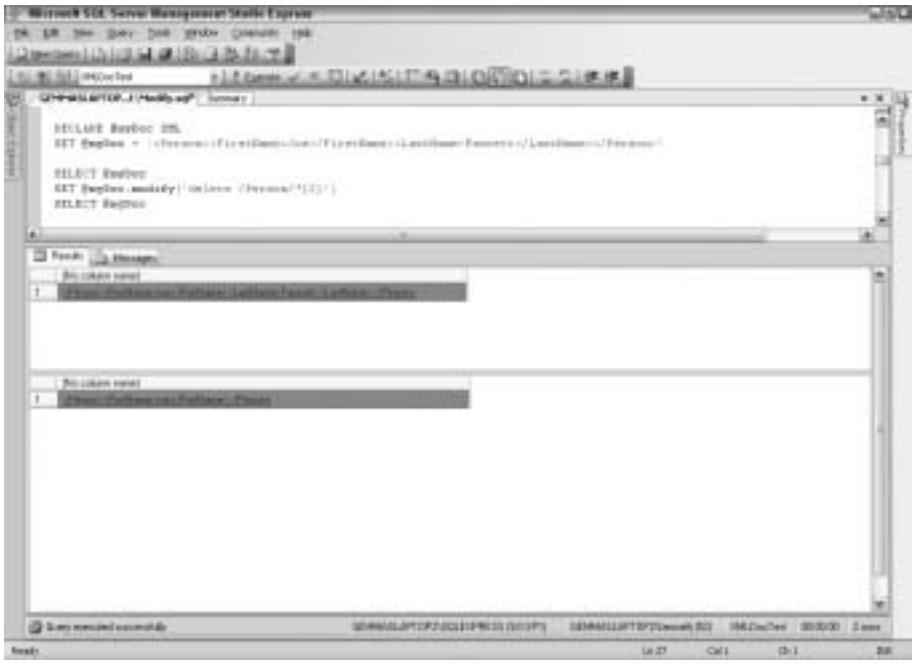

Рис. 10.16

1. Откройте SQL Server Studio.
2. Подключитесь к экземпляру по умолчанию.
3. Выберите в меню New SQL Server Query.
4. Введите приведенный выше код.
5. Нажмите <F5> для запуска кода SQL. Если вы ввели его корректно, будет выведен исходный документ, а ниже — модифицированный. В модифицированном документе имеется новый элемент `FirstName`.
6. Измените ширину столбцов, чтобы увидеть весь XML.

Как это работает

Первая строка кода объявляет переменную `myDoc` и указывает, что ее тип данных — `xml`. В коде

```
SET @myDoc = '<Person><LastName>Fawcett</LastName></Person>'
```

вы устанавливаете значение переменной `myDoc` и определяете его как элемент `Person`, содержащий единственный элемент `LastName`, который содержит текст `Fawcett`.

Функция `modify` предназначена для использования расширений XQuery. Ключевое слово `insert` указывает, что модификация представляет собой операцию

вставки. Вставляемый XML следует за ключевым словом `insert`. Обратите внимание на то, что он не заключен ни в одинарные, ни в двойные кавычки. Конструкция `as first` указывает, что вставляемый XML будет вставлен первым, а ключевое слово `into` использует выражение XPath `/Person`, чтобы указать, что элемент `FirstName` и его содержимое добавляются в качестве дочернего элемента к элементу `Person`. Конструкция `as first` говорит нам о том, что элемент `FirstName` является первым дочерним элементом элемента `Person`.

Альтернативами ключевому слову `into` являются слова `after` и `before`. В то время как `into` добавляет дочерний узел к родительскому, `after` или `before` добавляют братьев. Приведенный выше запрос можно переписать следующим образом.

```
DECLARE @myDoc XML
SET @myDoc = '<Person><LastName>Fawcett</LastName></Person>'
SELECT @myDoc
SET @myDoc.modify(' insert <FirstName>Joe</FirstName>
    before (/Person/LastName)[1]')
SELECT @myDoc
```

При запуске Transact-SQL первая инструкция `SELECT` выводит исходный XML, а вторая — XML после завершения операции `insert`. Результат приведен на рис. 10.17.

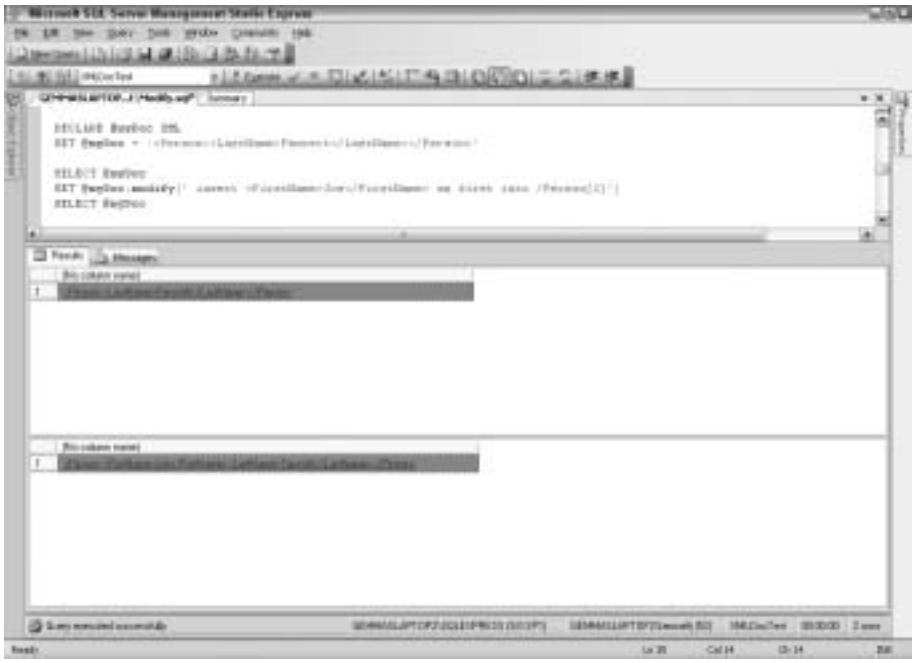

Рис. 10.17

Попробуй Обновление с помощью XML DML

Последний пример, использующий DML, обновляет содержимое переменной XML, так что значение элемента `FirstName` изменяется с `Joe` на `Gillian`. Код запроса приведен далее.

```
DECLARE @myDoc XML
SET @myDoc = '<Person><FirstName>Joe</FirstName>
              <LastName>Fawcett</LastName></Person>'
SELECT @myDoc
SET @myDoc.modify(' replace value of
                   (/Person/FirstName/text())[1]
                   with "Gillian" ')
SELECT @myDoc
```

1. Откройте SQL Server Studio.
2. Подключитесь к экземпляру по умолчанию.
3. Выберите в меню New SQL Server Query.
4. Введите приведенный выше код.
5. Нажмите <F5> для запуска кода SQL. Если вы ввели его корректно, будет выведен исходный документ, а ниже — модифицированный. Теперь в документе вместо `Joe` в качестве содержимого элемента `FirstName` содержится `Gillian`.
6. Измените ширину столбцов, чтобы увидеть весь XML.

Как это работает

Взгляните на функцию `modify`.

```
SET @myDoc.modify(' replace value of
                    (/Person/FirstName/text())[1]
                    with "Gillian" ')
```

Выражение `replace value of` указывает, что будет выполняться обновление, а выражение XPath указывает, какая именно часть XML будет обновлена. В нашем случае это текстовый узел, являющийся дочерним по отношению к элементу `FirstName`, другими словами — значение элемента `FirstName`, указанное выражением XPath `/Person/FirstName/text()`.

Результат выполнения двух инструкций `SELECT` показан на рис. 10.18.

Одна из основных проблем при использовании метода `modify()` заключается в том, что в качестве его аргумента ожидается жестко кодированная строка. Таким образом, сложно создавать динамические запросы, которые чаще всего требуются в реальном мире, например запросы, в которых новый XML порождается из другой таблицы. Имеется два способа обхода этого ограничения. Вы можете

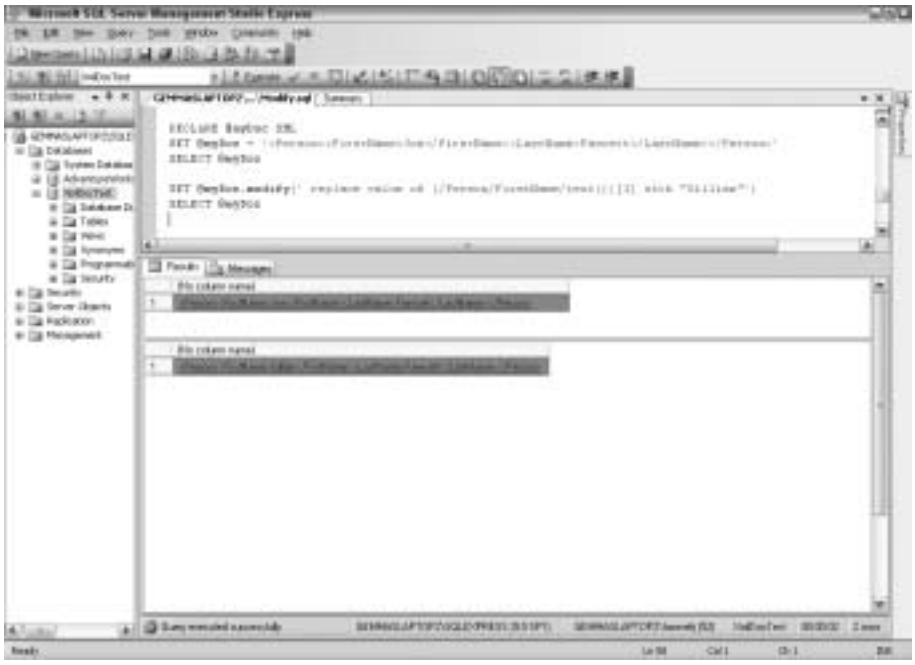

Рис. 10.18

построить запрос в виде строки и выполнить его динамически с использованием конструкции `EXEC`. В качестве альтернативы можно использовать встроенные функции `sql:column` и `sql:function`. Далее приведены примеры каждого из описанных методов.

В этих примерах можно использовать созданную ранее таблицу `Docs`. Сначала напомним, как выглядит статическое обновление.

```
UPDATE Docs
SET XmlDoc.modify (' replace value of
    (/Person/LastName/text())[1] with "Salt"')
WHERE DocId = 1
```

Теперь предположим, что нужно заменить жестко закодированное значение `Salt` переменной. Можно было бы попробовать поступить так.

```
DECLARE @NewName NVARCHAR(100)
SET @NewName = N'Salt'
UPDATE Docs
    SET XmlDoc.modify(' replace value of
        (/Person/LastName/text())[1] with "' +
        @NewName + '"')
WHERE DocId = 1
```

К сожалению, этот способ не работает. Метод `modify()` сообщит о том, что ему нужен строковый литерал. Один из способов выхода из этой ситуации — построить динамически всю SQL-инструкцию.

```
DECLARE @NewName NVARCHAR(100)
SET @NewName = N'Salt'
DECLARE @SQL NVARCHAR(MAX)
SET @SQL = 'UPDATE Docs SET XmlDoc.modify(
    '' replace value of (/Person/LastName/text())[1] with "'
    + @NewName + '"'') WHERE
DocId = 1'
PRINT(@SQL)
EXEC(@SQL)
```

Перед выполнением SQL-инструкция выводится для просмотра с помощью команды `PRINT` и имеет следующий вид.

```
UPDATE Docs SET XmlDoc.modify (' replace value of
➥  (/Person/LastName/text())[1] with "Salt"') WHERE DocId = 1
```

Как видите, мы получили точно ту же инструкцию, что и в оригинальном запросе.

Рекомендуемый путь, однако, состоит в использовании встроенных функций `sql:column` и `sql:variable`. Функция `sql:column` применяется тогда, когда из таблицы выбираются новые данные, так что в нашей ситуации нужна функция `sql:variable`.

```
DECLARE @NewName NVARCHAR(100)
SET @NewName = N'Salt'
UPDATE Docs
SET XmlDoc.modify
    (' replace value of (/Person/LastName/text())[1] with
     sql:variable("@NewName")')
WHERE DocId = 1
```

Синтаксис представляет собой имя переменной в двойных кавычках, переданное в качестве аргумента функции `sql:variable()`.

Метод query()

Метод `query()` позволяет построить инструкцию XQuery в SQL Server 2005. При этом используется тот же синтаксис, что и синтаксис XQuery, рассматриваемый в главе 9, “XQuery”. Все запросы в данной главе могут применяться к подходящему столбцу с XML-данными.

В приведенном далее запросе метод `query()` используется для вывода имен каждого человека во вновь построенный элемент `Name`, значение которого образуется из значения элемента `LastName`, за которым следуют запятая и значение элемента `FirstName`. Ниже приведен код запроса.

```
SELECT XMLDoc.query
  ('for $p in /Person return
```

```
  <Name>{$p/LastName/text()}, {$p/FirstName/text()}</Name>')
FROM Docs
```

Первая строка указывает, что выбор делается с использованием метода `query()`, и применяется к столбцу `XMLDoc` (который, конечно же, имеет тип данных `xml`).

Конструкция `for` указывает, что переменная `$p` связана с узлом элемента `Person`.

Конструкция `return` определяет, что элемент `Name` строится с применением конструктора элемента. Первая часть содержимого каждого элемента `Name` создается путем вычисления выражения XQuery `$p/LastName/text()`, которое представляет собой текстовое содержимое элемента `LastName`. Затем выводится запятая и вычисляется выражение XQuery `$p/FirstName/text()`.

На рис. 10.19 показан результат выполнения инструкции `SELECT`, содержащей запрос XQuery.

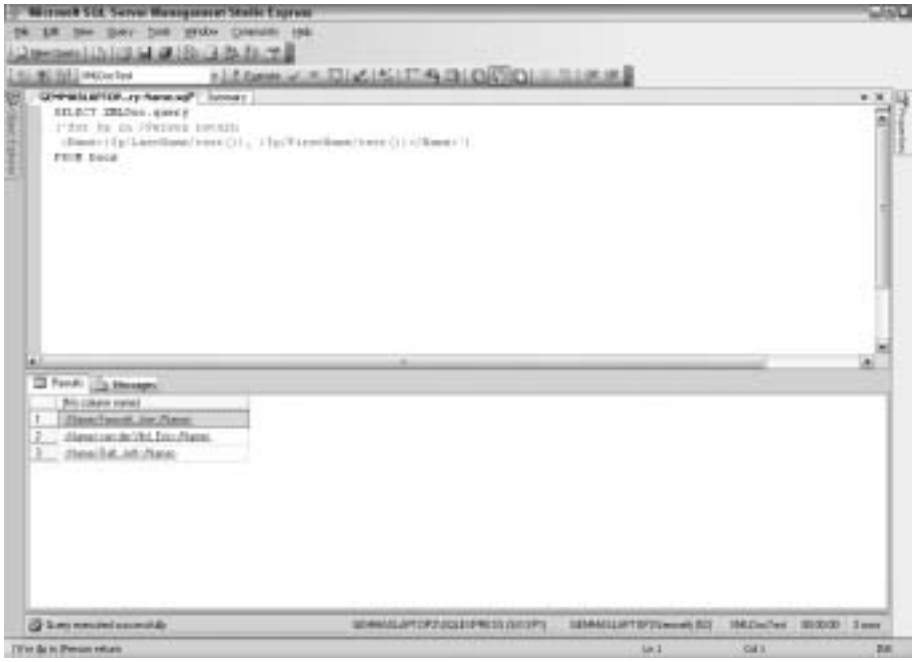

Рис. 10.19

W3C XML Schema в SQL Server 2005

Ранее упоминалось, что новый тип данных `xml` теперь имеет все права типа данных в SQL Server 2005. Он может использоваться для хранения нетипизированных и типизированных XML-данных, так что вас не должно удивлять, что так же, как реляционные данные определяются схемой, так и новый тип данных

xml может быть связан с документом W3C XML Schema, который определяет его структуру. Язык схем XDR, использовавшийся в SQL Server 2000, в SQL Server 2005 заменен W3C XML Schema.

Давайте посмотрим, как можно указать схему для данных типа xml. Первая задача — создание коллекции схемы вместе с XML Schema. Коллекции необходимо присвоить имя (в данном примере — EmployeesSchemaCollection), а сам документ W3C XML Schema должен быть выделен с помощью одинарных кавычек. Например, создать очень простую схему для документа, который может содержать элемент Person и дочерние элементы FirstName и LastName, можно с помощью следующего синтаксиса.

```
CREATE XML SCHEMA COLLECTION EmployeesSchemaCollection AS
  '<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     targetNamespace="http://wiley.com/namespaces/Person"
     xmlns="http://wiley.com/namespaces/Person">
     <xsd:element name="Person">
       <xsd:complexType>
         <xsd:sequence>
           <xsd:element name="FirstName" />
           <xsd:element name="LastName" />
         </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
   </xsd:schema>'
```

Чтобы удалить коллекцию XML Schema, нужно выполнить инструкцию DROP XMLSCHEMA.

```
DROP XML SCHEMA COLLECTION EmployeesSchemaCollection
```

Если схема уже имеется в наличии, новую схему можно добавить с помощью следующего синтаксиса.

```
ALTER XML SCHEMA COLLECTION EmployeesSchemaCollection ADD
    '<xsd:schema>
       <!-- Здесь находится новая схема -->
     </xsd:schema>'
```

Нетипизированные и типизированные данные xml могут использоваться в столбце, переменной или параметре SQL Server. Создать таблицу Docs и связать с ней документ W3C XML Schema можно с помощью кода следующего вида.

```
CREATE TABLE [dbo].[Docs](
    [DocID] [int] IDENTITY(1,1) PRIMARY KEY,
    [XMLDoc] [xml] (EmployeesSchemaCollection))
```

Преимущество от применения схемы двойное. Во-первых, она действует как средство проверки корректности; XML, не соответствующий схеме, будет отвергнут так же, как столбец, объявленный как INT, не примет произвольные текстовые

данные. Во-вторых, запросы к XML будут возвращать типизированные данные, соответствующие схеме, а не просто обобщенный текст.

Для оптимизации XML Schemas анализируются и сохраняются в базе данных во внутреннем формате. Большинство схем могут быть реконструированы из этого формата в XML-документ с помощью встроенной функции `xml_schema_namespace`. Следовательно, если вы импортировали ранее показанную схему, можете получить ее с помощью следующего кода.

```
SELECT xml_schema_namespace(N'dbo',
        N'EmployeesSchemaCollection')
```

Запомните также, что может иметься несколько путей написания функциональности, эквивалентной документу W3C XML Schema, например с использованием ссылок и именованных или анонимных типов. SQL Server не отличает такие различия при реконструкции документа схемы.

Кроме того, части схемы, являющиеся документацией, например аннотации и комментарии, не сохраняются во внутреннем формате SQL Server. Следовательно, для точного восстановления исходный документ W3C XML Schema необходимо хранить отдельно в сериализованном виде. Одна из возможностей — его хранение в столбце типа `xml` или `varchar(max)` в отдельной таблице.

Поддержка веб-службы

SQL Server 2000 предоставляет различные средства доступа посредством HTTP к XML-данным в реляционных таблицах каталога. Большинство из них доступны в некоторой степени и в SQL Server 2005, но ряд из них заменен встроенной поддержкой веб-служб. К сожалению, эта мощь имеет свою цену; новую функциональность сложнее настраивать, кроме того, для получения даже базовой службы она использует ряд совместно работающих разных компонентов.

> *Подробно эти вопросы рассматриваются в SQL Server Books Online, в разделе "Using Native XML Web Services".*

Основные преимущества использования встроенной функциональности перечислены ниже.

- **Бóльшая доступность.** Любой клиент, имеющий возможность использовать HTTP-запросы и анализировать XML, в состоянии воспользоваться новыми службами.
- **Соответствие стандартам.** Стандарт SOAP хорошо проработан и повсеместно используется в WWW. Клиент не только не использует специальные способы обращения к данным, но и может даже не знать, что он пользуется возможностями SQL Server.
- **Встроенный веб-сервер.** Нет необходимости в работе отдельного экземпляра IIS.

- **Бóльшая безопасность.** SQL Server имеет относительно простую в использовании модель безопасности, которая обеспечивает соответствующую аутентификацию и авторизацию для всех предлагаемых служб.
- **Повторное использование существующего инструментария мониторинга.** Большое количество имеющегося инструментария может использоваться для мониторинга и тонкой настройки производительности служб при их работе на сервере.

XML в СУРБД с открытым кодом

Основными факторами, способствовавшими добавлению поддержки XML в СУРБД с открытым кодом, были те же, что и для их коммерческих двойников, но СУРБД с открытым кодом в этой области оказались несколько отстающими. Причины этого различны. Проекты с открытым кодом менее коммерческих склонны к гигантомании и включению в себя всего, до чего можно дотянуться. Они также в большей степени совместно используются с другими проектами. Кроме того, на них меньше влияют модные лозунги и заклинания типа "XML — в каждый дом!" Ну и, конечно, на их разработку обычно имеется существенно меньше денег, чем у коммерческих конкурентов.

Из трех основных реляционных баз данных с открытым кодом две (PostgreSQL и Ingres) не имеют поддержки XML; третья, MySQL, получила первые XML-возможности в версии 5.1, которая на момент написания этой книги находилась в статусе бета-версии.

Установка MySQL

MySQL можно загрузить с сайта `http://dev.mysql.com/downloads/`. Следуйте по ссылкам на Community Server и выберите версию 5.1 (или более позднюю, если таковая доступна). Устойчивые версии MySQL доступны в различных дистрибутивах Linux, но в настоящее время они нам не подходят, поскольку XML-функциональность доступна только в версии 5.1, которая пока что имеет статус бета-версии.

Страница загрузки содержит исходные тексты и ряд бинарных файлов для наиболее распространенных платформ, включая Windows, различные дистрибутивы Linux и Mac OS X. Выберите вариант, соответствующий вашей платформе, и следуйте инструкциям сайта.

Материал этой главы требует установки как сервера, так и клиентской программы. Если вы — пользователь Debian или Ubuntu, то выберите пункт "Linux x86 generic RPM (dynamically linked) downloads", преобразуйте `.rpm`-пакеты в `.deb`-пакеты Debian с помощью команды `alien` и установите их так же, как любые другие пакеты Debian с применением команды `dpkg -i`.

Если вы предпочитаете красивый пользовательский интерфейс, можете загрузить и инсталлировать инструментарий MySQL, использующий графический пользовательский интерфейс. Этот инструментарий доступен и для версии MySQL 5.0. Он не полностью совместим с версией сервера 5.1, но, тем не менее, большинство его возможностей доступны и с этой версией; функции, которые не работают с версией 5.1, просто не работают, не нанося никакого ущерба базе данных. (Поскольку собственного графического инструментария у версии 5.1 пока нет, вы можете воспользоваться инструментарием, например, из вашего дистрибутива Linux.)

Если вы устанавливаете базу данных MySQL не только для работы с примерами из данной книги, защитите ее, корректно установив учетную запись и пароль. Для выполнения примеров из главы для простоты можете оставить конфигурацию, заданную по умолчанию.

Добавление информации в MySQL

Для взаимодействия с MySQL можно воспользоваться графическим инструментарием, но, если вы действительно хотите понять, что же происходит за сценой, вашим лучшим другом должна стать утилита командной строки `mysql`. Откройте терминал Unix или окно командной строки и введите `mysql -u root`. Если все введено верно и работает, появится приглашение `mysql`.

```
vdv@grosbill:~/beginning-xml/0764570773/ch 10 $ mysql -u root
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 5.1.12-beta-log MySQL Community Server (GPL)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

Если вы создали пароль для корневого пользователя при установке MySQL, в командную строку следует добавить опцию `-p`.

В следующем практическом примере создается новая база данных и в нее вносится информация.

Попробуй Создание и заполнение базы данных MySQL

Перед тем как добавлять информацию в MySQL, необходимо создать базу данных. База данных работает как контейнер, в котором вы группируете информацию, связанную с проектом. Обратите внимание на то, что, в отличие от коллекций eXist, в MySQL имеется только один уровень баз данных.

1. Для создания базы данных `myBlog` с UTF-8 в качестве набора символов введите следующее.

```
mysql> create database myBlog DEFAULT CHARACTER SET 'utf8';
Query OK, 1 row affected (0.00 sec)
mysql>
```

2. Перейдите во вновь созданную базу данных, введя следующую команду.

```
mysql> use myBlog;
Database changed
mysql>
```

Существенным отличием между истинной XML-базой данных, такой как eXist, и реляционной базой данных является то, что реляционная база данных сильно структурирована. Во всех примерах работы с eXist мы хранили документы без определения какой-либо схемы или DTD. Эта база данных изучала структуру ваших документов при их загрузке, не требуя предварительного определения. В случае реляционной базы данных такое невозможно. В реляционной базе данных информация хранится в таблицах, в строках и столбцах. Эти таблицы похожи на обычные электронные таблицы, за исключением того, что имена и типы столбцов должны быть определены до их использования.

3. Создайте одну из таблиц для хранения информации, необходимой для последующих примеров данной главы. Имя таблицы — `entries`; для простоты в нее входят два столбца.

   - Столбец `id`, используемый в качестве первичного ключа при выборке определенной записи
   - Столбец `content` для хранения записи блога в формате XML

   Конечно, это минимальная схема. Вы можете добавить в таблицу и другие столбцы для упрощения и оптимизации запросов.

   Для создания таблицы введите следующее.

```
mysql> create table entries (
    -> id int PRIMARY KEY,
    -> content LONGTEXT
    -> );
Query OK, 0 rows affected (0.27 sec)
mysql>
```

   Обратите внимание на то, что вы не вводите символы `->` в начале второй и последующих строк инструкции SQL `create table`; это приглашения утилиты командной строки `mysql`.

4. База данных готова для ввода записей блога. В реальной ситуации записи вносились бы некоторым веб-приложением, но в данной главе мы введем данные с помощью утилиты командной строки `mysql`. Добавление информации в SQL выполняется с помощью команды `insert`. Введите несколько собственных записей по следующему образцу.

```
mysql> insert into entries values (
-> 1,
-> '<?contentxml version="1.0"?>
-> <item id="1">
-> <title>Working on Beginning XML</title>
-> <description>
-> <p>
-> <a href="http://www.wrox.com/WileyCDA/0764570773.html">
-> <img
-> src="http://media.wiley.com/coverImage/73/0764570773.jpg"
-> align="left"/>
-> </a> I am currently working on the next edition of <a
-> href="http://www.wrox.com/WileyCDA/0764570773.html">
-> WROX\'s excellent "Beginning XML".</a>
-> </p>
-> <p>It\'s the first time I am working on editing a book
-> that I haven\'t written and I must say that I like it even
-> better than I had expected when I accepted WROX\'s offer.
-> </p><p>I knew that the book was solid and that I would be
-> working with a team of very professional authors, but
-> what I hadn\'t anticipated is how fun it can be to create
-> a new version out of existing material. You have a lot of
-> room to reorganize what you are editing and when the
-> material is good, it\'s like creating your own stuff,
-> except that 80% of the hard work is already done!</p>
-> </description>
-> <category>English</category>
-> <category>XML</category>
-> <category>Books/Livres</category>
-> <pubDate>2006-11-13T17:32:01+01:00</pubDate>
-> <comment-count>0</comment-count>
-> </item>
-> ');
Query OK, 1 row affected (0.34 sec)
mysql>
```

Обратите внимание на то, что XML-документ заключен в строку SQL, ограниченную одинарными кавычками. Все одинарные кавычки в вашем документе должны быть заменены управляющими символами \'.

Как это работает

В данном практическом примере созданы база данных, которая служит контейнером для хранения информации, и таблица, определяющая структуру ваших данных. Затем в эту структуру вводятся данные.

Теперь, когда ваша база данных готова к дальнейшему использованию, можно поработать с `mysql-admin`, графическим интерфейсом пользователя, поставляемым с MySQL для ее администрирования. После его запуска подключитесь к серверу базы данных. Если вы не изменяли настроек, то параметр `hostname` должен иметь значение `localhost`, имя пользователя — `root`, а пароль — оставаться пустым. Помимо всего прочего, `mysql-admin` позволяет выводить и даже обновлять определения таблиц, как показано на рис. 10.20.

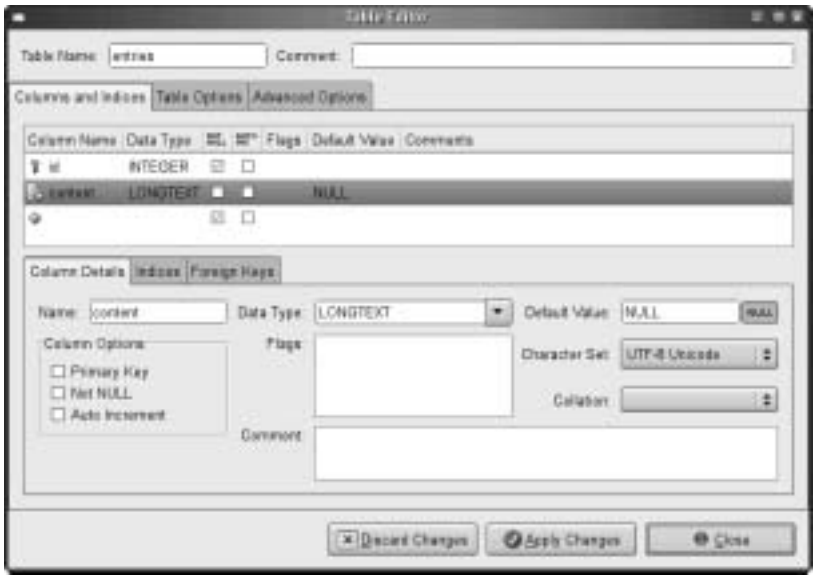

Рис. 10.20

Запросы к MySQL

Теперь, когда у вас есть две записи, что можно с ними делать? Конечно, MySQL представляет собой SQL-базу данных, так что можно воспользоваться SQL для запроса содержимого вашей базы данных. Чтобы вывести все записи, введите следующее.

```
select * from entries;
```

Результат имеет слишком большой размер, чтобы полностью привести его здесь, так что мы покажем лишь несколько первых символов каждой записи.

```
mysql> select id, substring(content, 1, 80) from entries;
+----+----------------------------------------...---------+
| id | substring(content, 1, 80)                           |
+----+----------------------------------------...---------+
| 1  | <?xml version="1.0"?>
<item id="1">
```

```
<title>Working on Beginning XML</title>
|
| 2  | <?xml version="1.0"?>
<item id="2">
<title>eXist: getting better with each rel
|
+----+-----------------------------------...---------+
2 rows in set (0.00 sec)
mysql>
```

Или, если вас интересует только количество записей:

```
mysql> select count(*) from entries;
+----------+
| count(*) |
+----------+
| 2        |
+----------+
1 row in set (0.00 sec)
mysql>
```

Все в порядке, но это — чистый SQL и все это можно получить в любой другой SQL-базе данных без поддержки XML. А вот как вывести, например, содержимое элемента `title`?

Поддержка XML в MySQL 5.1 состоит из двух XML-функций XML, документированных по адресу `http://dev.mysql.com/doc/refman/5.1/en/xml-functions.html`. Это функции `ExtractValue` и `UpdateXML`. В приведенном далее практическом примере показано, как использовать функцию `ExtractValues` для запроса данных.

Попробуй Использование функции `ExtractValue`

1. Первая функция, `ExtractValue`, вычисляет выражение XPath для XML-фрагмента, переданного в виде строки. Заметим, что пока реализовано очень ограниченное подмножество XPath, которое сильно ограничивает ваши возможности запросов к XML-фрагментам, но которого достаточно, чтобы получить значения элементов `title` из содержимого столбцов.

```
mysql> select id, ExtractValue(content, '/item/title')
    ->    as title FROM entries;
+----+---------------------------------------+
| id | title                                 |
+----+---------------------------------------+
| 1  | Working on Beginning XML              |
| 2  | eXist: getting better with each release |
+----+---------------------------------------+
```

```
2 rows in set (0.03 sec)
mysql>
```

Эта функция может использоваться в любом месте инструкции SQL.

2. Для получения идентификатора `id` записи с определенным элементом `title` воспользуйтесь следующим запросом.

```
mysql> select id from entries where
    ->   ExtractValue(content, '/item/title') =
    ->   'eXist: getting better with each release';
+----+
| id |
+----+
| 2  |
+----+
1 row in set (0.00 sec)
mysql>
```

Как это работает

Поведение функции `ExtractValue` зачастую противоречит привычным представлениям, если вы знакомы с XPath. Например, если вы попытаетесь применить тот же метод для получения элемента `description` ваших записей, вы получите следующее.

```
mysql> select id, ExtractValue(content, '/item/description')
    ->    as title FROM entries;
+----+-----------------------+
| id | title                 |
+----+-----------------------+
| 1  |                       |
| 2  |                       |
+----+-----------------------+
2 rows in set (0.00 sec)
mysql>
```

Если вы пользовались функциональностью XPath, которая транслирует элементы в строки путем конкатенации текстовых узлов из всех их потомков, то вы можете счесть, что `ExtractValue` делает то же самое. Однако это не так: `ExtractValue` конкатенирует только текстовые узлы, непосредственно встроенные в элементы. В нашем случае единственными текстовыми узлами, являющимися непосредственными потомками элементов `description`, являются пробельные символы, что и поясняет поведение базы данных при последнем запросе.

Для получения поведения XPath по умолчанию необходимо явно указать, что требуются текстовые узлы на любом уровне.

```
mysql> select id, ExtractValue(content,
    ->   '/item/description//text()') as description
    ->   FROM entries;
```

```
+----+--------------------------...------+
| id | description                        |
+----+--------------------------...------+
| 1  |
I am currently working on the next edition of WROX's
excellent "Beginning XML".
.
.
.
2 rows in set (0.00 sec)
mysql>
```

(Приведенная таблица отредактирована для краткости и удобочитаемости.)

* * *

Как выбрать записи, которые содержат изображения? В XPath можно использовать `//img` непосредственно в проверке, и такое выражение будет истинно тогда и только тогда, когда где-то в документе будет присутствовать по крайней мере один элемент `img`. Если вы знакомы с XPath, можете попытаться написать запрос наподобие следующего.

```
mysql> select id, ExtractValue(content, '/item/title')
    ->   as title from entries
    ->   where ExtractValue(content, '//img') != '';
Empty set (0.00 sec)
mysql>
```

Такой подход не сработает, поскольку элементы `img` пустые: у них нет дочерних текстовых узлов и `ExtractValue` конвертирует их в пустые строки. Чтобы запрос начал работать, необходимо выбрать узел, который будет иметь значение (такое, как `//img/@src`) или подсчитать количество элементов `img` и проверить, больше ли нуля полученный результат.

```
mysql> select id, ExtractValue(content, '/item/title'
    ->   as title from entries
    ->   where ExtractValue(content, '//img/@src') != '';
+----+--------------------------+
| id | title                    |
+----+--------------------------+
| 1  | Working on Beginning XML |
+----+--------------------------+
1 row in set (0.00 sec)
mysql> select id, ExtractValue(content, '/item/title')
    ->   as title from entries
    ->   where ExtractValue(content, 'count(//img)') > 0;
+----+--------------------------+
| id | title                    |
```

```
+----+--------------------------+
| 1  | Working on Beginning XML |
+----+--------------------------+
1 row in set (0.00 sec)
mysql>
```

При использовании этой функции очень скоро вы столкнетесь с еще одним ограничением: большинство строковых функций XPath не реализовано. Например, чтобы найти записи со ссылками на сайт Wrox, можете попытаться написать запрос примерно такого вида.

```
select id, ExtractValue(content, '/item/title') as title
from entries
where
  ExtractValue(content,
    'count(//a[starts-with(@href,"http://www.wrox.com")])')
    >0;
```

Увы, функция `starts-with` не реализована; чтобы добиться того, что не удается с помощью XPath, нужно воспользоваться SQL.

```
mysql> select id, ExtractValue(content, '/item/title')
    ->   as title from entries
    ->   where ExtractValue(content, '//a/@href')
    ->   like '%http://www.wrox.com%';
+----+--------------------------+
| id | title                    |
+----+--------------------------+
| 1  | Working on Beginning XML |
+----+--------------------------+
1 row in set (0.01 sec)
mysql>
```

Если вы не поклонник командной строки, можете выполнять все команды с помощью еще одного графического инструмента — `mysql-query-browser`, который можно загрузить с сайта MySQL и который показан на рис. 10.21.

Этот инструмент не только позволяет выполнить все запросы, с которыми вы сталкивались ранее, но и предоставляет возможность редактирования табличных значений аналогично редактированию электронных таблиц и даже отдельных полей, что особенно удобно при редактировании хранящихся в базе данных XML-документов. На рис. 10.22 показано редактирование в действии.

Обновление XML в MySQL

Вторая XML-функция, введенная в MySQL 5.1, называется `UpdateXML`. Подобно любой другой SQL-функции, `UpdateXML` не выполняет обновления базы данных, но очень удобна при использовании в инструкциях обновления.

Рис. 10.21

`UpdateXML` получает три аргумента. Первый — строка, содержащая XML-документ, второй — выражение XPath, которое указывает на элемент, и третий — фрагмент XML. `UpdateXML` получает XML-документ, заменяет элемент, на который указывает выражение XPath, фрагментом XML, переданным в качестве третьего аргумента, и возвращает результат этой операции в виде строки.

Например, для замены названия второй записи можно использовать следующий запрос.

```
mysql> update entries
    ->   set content = UpdateXml(content,
    ->   '/item/title',
    ->   '<title>eXist DB is getting much
↪          better with each release</title>')
    ->   where id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
mysql> select id, ExtractValue(content, '/item/title')
    ->   as title FROM entries;
```

Рис. 10.22

```
+----+-------------------------------------------------+
| id | title                                           |
+----+-------------------------------------------------+
| 1  | Working on Beginning XML                        |
| 2  | eXist DB is getting much better with each release |
+----+-------------------------------------------------+
2 rows in set (0.01 sec)
mysql>
```

Удобство этой функции в данной ситуации очевидно, но обратите внимание, что выражение XPath должно указывать на элемент. Это означает, что обновления возможны на уровне элемента, так что, если вам нужно обновить значение атрибута, вам не повезло...

Применение XML в MySQL

После приведенного введения в XML-возможности MySQL 5.1 вы, вероятно, задумаетесь, насколько применимы эти возможности в реальных приложениях? Чтобы ответить на этот вопрос, заметим сначала, что поддержка XML в MySQL 5.1 ограничена приведенными двумя функциями для работы со строками. Другими

словами, здесь нет такой вещи, как XML-тип столбца. Ваши документы хранятся как текст и должны анализироваться всякий раз при использовании одной из этих функций.

Рассмотрим еще раз один запрос, с которым мы уже сталкивались.

```
select id from entries
    where ExtractValue(content, '/item/title') =
          'eXist: getting better with each release'
```

Для его обработки механизм базы данных должен читать полное содержимое всех записей, анализировать его и применять выражение XPath для получения значения элемента `title`. Это неплохо работает для нашей пары записей, но вряд ли вы захотите использовать такой метод работы, если у вас будут миллионы записей.

Для оптимизации создаваемой базы данных вы, вероятно, выделите информацию, наиболее часто применяемую в пользовательских запросах, и перенесете ее в столбцы таблицы. В нашем примере очевидными кандидатами на такой перенос являются название, категории и дата публикации. Если эти данные будут доступны как столбцы, база данных получит к ним непосредственный доступ. Дальнейшая оптимизация может заключаться в индексировании этих столбцов.

Не следует забывать о расхождениях между текущей реализацией и использованием XPath. Вы видели пример того, как нужно явно указывать необходимость конкатенации текстовых узлов всех потомков. При использовании функций вы встретитесь и с другими примерами. Возможно, такое несоответствие будет устранено в более поздних версиях, но это может привести к появлению несовместимости.

С учетом всех этих ограничений нет никаких причин игнорировать XML-возможности в MySQL. Они не превращают MySQL в истинную XML-базу данных, но являются шагом в верном направлении.

Поддержка XML на стороне клиента

Все рассмотренные выше возможности являются серверными, реализуемыми механизмом базы данных. Вам не нужна никакая поддержка XML на стороне клиента, и использовать любой язык программирования для преобразования результатов запросов SQL в XML очень легко. Чтобы вы не остались разочарованными, рассмотрим поддержку XML на стороне клиента, в утилите командной строки `mysql`.

Чтобы увидеть ее в действии, добавьте к команде `mysql` параметр `--xml`.

```
vdv@grosbill $ mysql -u root --xml myBlog
Reading table information for completion of table
➥    and column names
You can turn off this feature to get a quicker startup with -A
```

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 31
Server version: 5.1.12-beta-log MySQL Community Server (GPL)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

Этот параметр командной строки включает XML-режим, и все результаты запросов теперь будут выводиться в виде XML.

```
mysql> select id, ExtractValue(content, '/item/title')
    ->   as title FROM entries;
<?xml version="1.0"?>
<resultset
    statement="select id, ExtractValue(content, '/item/title')
    as title FROM entries;">
    <row>
        <field name="id">1</field>
        <field name="title">Working on Beginning XML</field>
    </row>
    <row>
        <field name="id">2</field>
        <field name="title">eXist DB is getting much better
            with each release</field>
    </row>
</resultset>
2 rows in set (0.02 sec)
mysql>
```

Конечно, такой вывод не слишком удобочитаем или полезен сам по себе, но это неплохая возможность при использовании `mysql` в сценариях оболочки: вы получаете результаты в виде XML-документов, которые можно обработать с помощью XML-инструментария, например такого, как преобразования XSLT. Если вам нужен действительно простой способ превратить результат запроса в XHTML, то эта возможность окажется для вас большим подспорьем.

Выбор базы данных для хранения XML

В настоящее время в мире существует огромное количество данных, хранящихся в обычных реляционных системах управления базами данных, и имеется высококвалифицированный персонал с большим опытом работы с этими базами данных. По этой причине основное внимание к XML сосредоточивается на добавлении поддержки XML в существующих СУРБД, таких как IBM DB2, Oracle, Sybase, MySQL и Microsoft SQL Server. При добавлении XML-функциональности к таким продуктам непонятно, остаются ли они СУРБД или превращаются в некий гибрид. Впрочем, для большинства применений это сугубо академиче-

ский вопрос. Им просто нужна база данных, которая работает, безопасна, надежна, способна выдержать быстрый рост объемов данных и количества пользователей, проста в управлении. . .

С другой стороны, в некоторых ситуациях лучше подходит пользовательское приложение, применяющее истинную XML-базу данных наподобие eXist.

Заглядывая вперед

В настоящее время не существует стандарта обновления данных в истинной XML-базе данных, и не похоже, чтобы он появился раньше, чем в версии XQuery после 1.0. В идеальном случае следовало бы иметь стандартный язык с обновлениями, поскольку XQuery в значительной степени является стандартом для коммерческих баз данных. Отсутствие такого строгого стандарта приводит к появлению различных частных решений в разных реализациях, как, в частности, в SQL Server 2005. Вероятно, со временем W3C разработает стандартный язык с возможностью модификации данных — как более позднюю версию XQuery или как дополнение к нему. Пока же пользователи истинных XML-баз данных и реляционных баз данных с поддержкой XML вынуждены выбирать, какой из частных языков с возможностью модификации данных предпочесть.

Резюме

В этой главе вы познакомились с быстро развивающейся областью применения XML для хранения и представления данных. Здесь приведены характеристики баз данных с поддержкой XML и три примера таких баз данных. Первая из рассмотренных баз данных, eXist, представляет собой истинную XML-базу данных. Затем рассмотрены реляционная база данных Microsoft SQL Server 2000 и ее наследник — SQL Server 2005 — и показано, как к промышленной реляционной базе данных может быть добавлена богатая XML-функциональность. Наконец, рассмотрена последняя версия популярной реляционной базы данных с открытым кодом MySQL — первая версия, включающая некоторые XML-возможности.

Упражнения

Варианты ответов к упражнениям приведены в приложении А.

Упражнение 1

Перечислите причины, по которым добавление XML-функциональности к СУРБД может быть предпочтительнее использования истинной XML-базы данных.

Упражнение 2

Какие методы работы с данными в столбце с типом `xml` доступны в SQL Server 2005?

Упражнение 3

Напишите SQL-запрос для получения идентификатора и названия записи блога из базы данных MySQL. Следует ли ожидать масштабируемости запроса при росте блога и увеличении количества записей? Что можно сделать для повышения производительности в этой ситуации?