УДК 004.655.3

ПОДДЕРЖКА ОБНОВЛЕНИЯ XML-ПРЕДСТАВЛЕНИЙ НАД РЕЛЯЦИОННЫМИ БАЗАМИ ДАННЫХ

Н. А. Самохвалов[1]

В статье рассматриваются способы реализации задачи построения механизма обновляемых XML-представлений над реляционными базами данных с использованием техники деревьев запросов и предлагаются способы расширения множества поддерживаемых представлений. Приведенные правила построения деревьев запросов для стандартных SQL/XML конструкций делают возможным реализацию данного подхода в виде модуля реляционной системы управления базами данных (СУБД). Приводится описание реализации метода деревьев запросов для создания инструмента разработки web-интерфейсов к реляционным данным и обзор возможных областей применения данного подхода.

Ключевые слова: XML, XML-представления, SQL/XML, реляционные представления, обновление представлений, базы данных.

1. Введение. Задачи преобразования данных из реляционной модели данных в модель XML [1] и обратно сегодня чрезвычайно актуальны. В то время как XML активно используется для обмена данными в Internet, реляционные СУБД продолжают оставаться основным классом систем хранения данных. Например, типичной можно считать ситуацию, когда при наличии уже существующей системы, использующей для хранения данных реляционную СУБД, ставится задача создания (или усовершенствования) web-приложения, которое должно поставлять конечному пользователю документы в формате XHTML, WML, RSS и т.п. Таким образом, в общем случае web-приложение можно рассматривать как некоторую систему, выступающую в роли "посредника" между двумя моделями данных — реляционной и XML.

Безусловно, существующие реализации реляционных СУБД обладают чрезвычайно важными качествами, среди которых можно отметить богатые возможности поддержки целостности данных, механизмы обеспечения надежности хранения, защиты информации. С другой стороны, XML является стандартом de facto обмена информацией в Internet. Именно поэтому ведущие производители реляционных СУБД развивают средства работы с данными типа XML в своих продуктах. Проблема поддержки таких средств состоит из следующей совокупности задач.

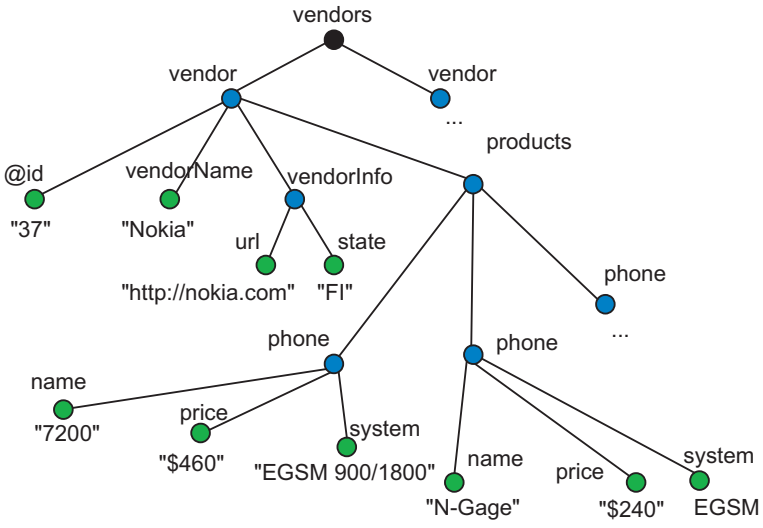

Рис. 1. Представление View_1 — производители и телефоны

1. Задача *экспорта* реляционных данных в виде XML [10, 16, 19]. Стандарт SQL:2003 (SQL/XML [2]) описывает расширение оператора SELECT языка SQL конструкциями для получения XML-документов (или их частей) из реляционных данных.

2. *Хранение* XML-данных в реляционной базе. Существует два различных пути решения этой задачи: хранение XML-данных посредством расширенного типа CLOB и специальная декомпозиция XML-данных с целью последующего хранения в "разобранном" виде (*shredding*) [12]. Существующие реализации, как правило, комбинируют эти два подхода, сохраняя данные в цельном виде и создавая расширенный индекс с применением техники shredding.

[1] Московский физико-технический институт, факультет управления и прикладной математики, Институтский переулок, 9, 141700, г. Долгопрудный, Московская область; e-mail: samokhvalov@gmail.com

3. *Манипуляционные задачи:* осуществление операции выборки XML-данных (например, с помощью некоторого расширения языка SQL, поддерживающего XPath-выражения [5]), сохраненных в реляционной базе; создание эффективных индексов для выполнения операции выборки; возможность осуществления операций обновления данных.

Заметим, что свойства модели данных XML достаточно глубоко исследованы [15, 17, 18, 20], а основные коммерческие реализации предоставляют достаточно мощные средства для решения упомянутых проблем. Тем не менее не все задачи решаются эффективным образом; кроме того, существуют и вовсе нерешенные проблемы. К ним относится универсальная поддержка трансляции обновлений XML-представлений над реляционными базами (созданных, к примеру, с помощью конструкций SQL/XML) на исходные реляционные таблицы. Эта задача возникает, например, при обновлении XML-представлений удаленными агентами, от которых должна быть скрыта полная информация о природе данных; при создании универсального средства для создания web-интерфейсов работы с данными и т.д.

В данной работе рассматривается подход [7–9], основывающийся на технике сопоставления XML-представления с группой реляционных представлений и трансляции операций обновления XML-документа в набор SQL-выражений обновления над реляционными представлениями. Данный подход позволяет переложить на плечи реляционной СУБД вопросы возможности проведения самой операции обновления и дальнейшей трансляции ее на исходные таблицы, что существенно упрощает задачу (вообще говоря, задача реализации механизма обновления реляционных представлений [11] является нетривиальной, поэтому использование механизмов поддержки операций обновления представлений, которыми обладают текущие реляционные СУБД, представляется чрезвычайно выгодным). Техника сопоставления XML-представлений с группой реляционных представлений базируется на понятии *дерева запросов*. В данной работе предложено расширение класса поддерживаемых данным методом XML-представлений. Кроме того, описывается создание деревьев запросов для стандартных SQL/XML конструкций [2], что делает возможным реализацию данного подхода в виде модуля реляционной СУБД. В заключение приводится обзор областей применения метода деревьев запросов, описание реализации метода для создания инструмента разработки web-интерфейсов к реляционным данным и анализ возможности создания обновляемых XML-представлений в объектно-реляционной СУБД PostgreSQL.

Крупнейшие производители СУБД уже несколько лет предлагают механизмы для экспорта данных в XML и импорта XML-документов в реляционную БД. К сожалению, подходы к реализации данных механизмов у разных производителей имеют принципиальные различия. Поэтому использование подобных механизмов, например, в СУБД MS SQL Server [16] (стоит заметить, что данная СУБД поддерживает обновления реляционных таблиц с помощью XML-пакетов) существенно снизило бы универсальность системы.

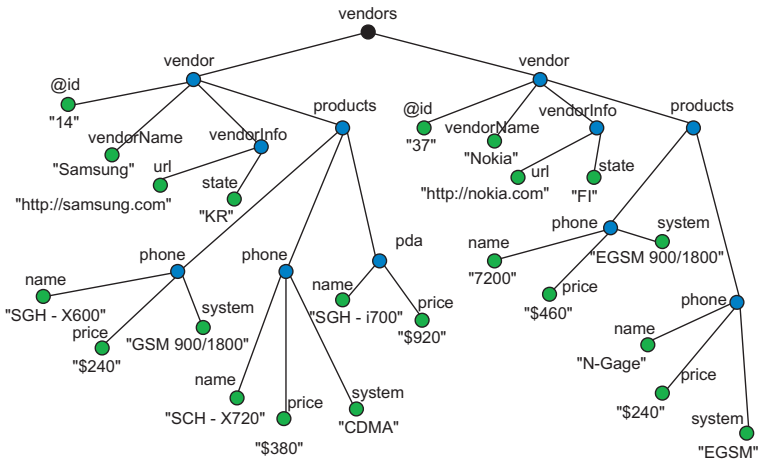

Рис. 2. Представление View_2 — производители, телефоны и КПК

Между тем, задачу построения системы поддерживающих операции обновления XML-представлений можно решить независимым от СУБД способом — построив соответствие XML-представлений наборам обычных реляционных представлений [7, 8]. При этом часть ответственности за реализацию операций обновления перекладывается на плечи реляционной СУБД, что позволяет разработчику целиком сосредоточиться на вопросах определения правильной структуры XML-документов, поставляемых конечному пользователю. Для реализации данного подхода в [8] вводится понятие *деревъя запросов*. Как мы увидим в дальнейшем, использование деревьев запросов на этапе построения XML-представлений позволяет легко перейти ко второй нормальной форме XML-документов (что, по сути дела, и делает возможным построение взаимно-однозначного соответствия XML-документов и реляционных таблиц), а использование их на этапе отображения операции обновления частично позволяет отказаться от рассмотрения проблемы возможности осуществления самой операции обновления, воспользовавшись механизмами обновления представлений, заложенными в реляционную СУБД. Подробнее о нормальных

формах XML-документов см. в [18].

Множество представлений ограничим представлениями, построенными с помощью операций выборки, ограничения и соединения. Что касается XML-документов, сфокусируем внимание на общей форме XML-представлений, разрешающих операции вложения, композиции атрибутов, гетерогенные множества и повторяющиеся элементы. Пример такого представления — представление View_2, описанное на рис. 2. В этом XML-представлении узлы phone вложены в узлы products, узел vendorInfo содержит композицию атрибутов url и state. Узлы products могут содержать кортежи из элементов двух типов: phone и pda.

В качестве примера реляционной базы данных рассмотрим БД, схема которой частично изображена в приложении А.

2. Деревья запросов. Деревья запросов можно рассматривать как некоторую переходную конструкцию от реляционной базы данных (БД) к документам XML. Деревья запросов позволяют работать с обновлением представлений, поддерживающих гетерогенные множества, в то время как другие решения проблемы обновления XML-представлений такие классы представлений не рассматривают. Например, подход с использованием алгебры вложенных отношений (NRA, Nested Relational Algebra), описанный в [13, 14], позволяет работать с представлениями, подобными View_1 (рис. 1), но не View_2. Как мы увидим далее, система, основанная на деревьях запросов, поддерживает оба типа представлений. Кроме того, использование такой абстрактной системы, а не какого-либо синтаксиса языка запросов XML удобно по следующим причинам. Во-первых, облегчается задача определения возможности обновления представления: на основе дерева запросов генерируется набор реляционных представлений, далее операция обновления XML преобразуется (опять-таки с помощью дерева запросов) в набор SQL-предложений обновления этих реляционных представлений и далее реляционная СУБД решает, являются ли такие операции корректными. Во-вторых, деревья запросов просты для понимания и в то же время являются достаточно мощными, предоставляя возможность работы с большим классом представлений.

2.1. Определение деревьев запросов. Далее будем полагать, что D обозначает реляционную БД, над которой определяются XML-представления. Пусть T — множество имен таблиц D, а A_T — множество атрибутов таблицы $T \in \text{T}$ (множество имен столбцов в T).

Определение 1 (*дерево запросов*). Деревом запросов, определенным над базой D, называется дерево с множеством вершин N и множеством ребер E, в котором верны следующие положения. Множество ребер содержит простые ребра и *-ребра. Ребро является *простым*, если вершине, в которую оно входит, соответствует ровно один элемент в соответствующем XML-представлении, в противном случае оно является **-ребром*. Вершины должны удовлетворять следующим требованиям:

— все вершины имеют имя, соответствующее тегу элемента в итоговом XML-представлении;

— все листовые вершины имеют значения;

— листовые вершины, имена которых начинаются с “@”, рассматриваются как XML-атрибуты;

— *-вершины (вершины со входящими *-ребрами) имеют один или более дескрипторов источника, ноль или более дескрипторов ограничения и ноль или более дескрипторов сортировки.

Вершины могут содержать дескрипторы нескольких типов. *Дескрипторы источника* связывают переменные с реляционными таблицами (например, [$t := table(“tree”)]), а *дескрипторы ограничения* содержат ограничения на реляционные таблицы (пример: [where $p/product_price > 300]). Введем еще один тип дескрипторов — *дескрипторы сортировки*, которые указывают, каким образом будет произведена сортировка в результирующем представлении (например, [orderby $p/product_price desc]).

Определение 2 (*дескриптор источника*). Дескриптор источника s в *-вершине n записывается в виде $x := table(T), где $x означает переменную, а $\text{T} \in \text{T}$ — реляционную таблицу. Будем говорить, что $x *связана с* T *при помощи* s.

Определение 3 (*дескриптор ограничения*). Дескриптор ограничения в *-вершине n записывается в виде $\texttt{where } \$x_1/A_1 \texttt{ op } Z_1 \texttt{ AND ...AND } \$x_k/A_k \texttt{ op } Z_k\texttt{, } k \geqslant 1$, где $A_i \in AT_i$ и x_i связана с T_i при помощи дескриптора источника в n или каком-либо предке n. Оператор op — это какой-либо оператор сравнения ($=$, $\neq$, $>$, $<$, $\leqslant$, $\geqslant$, LIKE и т.д.). Z_i — либо литеральная константа, либо выражение в форме $y/B, где $B \in A_T$ и $y связана с T при помощи дескриптора источника в n или каком-либо предке n.

Возможно использование любого условного выражения (см. спецификацию conditional_expression в [7]) в качестве дескриптора ограничения. При этом может возникнуть ситуация, когда реляционная СУБД не в состоянии выполнить операцию обновления соответствующих реляционных представлений. Здесь выбрано подмножество условных выражений, которое: а) допускает рассмотрение таких реляционных представлений, для которых большинство развитых реляционных СУБД позволяют осуществить операцию обновления; б) соответствует ситуациям, часто встречающимся при разработке приложений.

Определение 4 (*дескриптор сортировки*). Дескриптор сортировки в *-вершине n записывается в

виде `orderby $x/A [{asc | desc}], k` $\geqslant$ `1`, где `A` $\in$ $\mathtt{A}_\mathtt{T}$ и `$x` связана с `T` при помощи дескриптора источника в `n` или каком-либо предке `n`. Ключевые слова `asc` и `desc` указывают на направление сортировки, `asc` — прямой порядок (данное значение является значением по умолчанию), `desc` — обратный.

Определение 5 (*значение вершины*). Значение листовой вершины `n` записывается в виде `$x/A`, где `A` $\in$ $\mathtt{A}_\mathtt{T}$ и `$x` связана с `T` при помощи дескриптора источника в `n` или каком-либо предке `n`.

Пример дерева запросов приведен на рис. 3. В этом дереве описаны указания к получению из базы данных (БД) всех телефонов Samsung, которые продаются по цене более \$ 300. Дерево запросов очень близко по структуре к результирующему XML-документу. Корневая вершина дерева соответствует корневой вершине результата. Листовые вершины соответствуют атрибутам реляционных таблиц; ребра, отмеченные звездой, соответствуют кортежам элементов. Результат данного запроса представлен на рис. 3 справа.

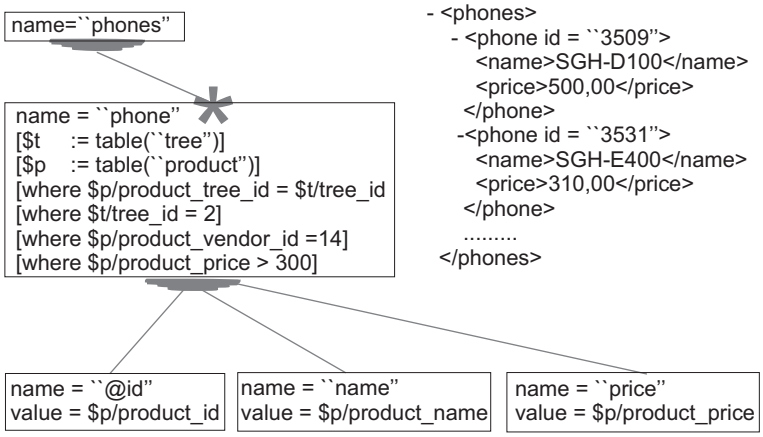

Рис. 3. Пример дерева запросов и построенного с его помощью XML-представления

Легко видеть, что из корня (phones) выходит *-ребро к дочерней вершине phone, что означает то, что в соответствующем XML-документе у корневого элемента phones может быть несколько дочерних элементов phone. Простое ребро от вершины phone к вершине name означает, что у элемента phone в XML-представлении может быть только один дочерний элемент name. Вершине @id в XML-документе соответствует не элемент, а атрибут. Далее, вершина phone содержит дескрипторы источника и ограничения. Дескрипторы источника связывают переменную \$p с реляционной таблицей Product и переменную \$t с реляционной таблицей Tree. Дескрипторы ограничения указывают на то, что производится выборка телефонов (tree_id = 2) производителя Samsung (product_vendor_id = 14), задают ограничения значений цены телефонов в представлении снизу величиной \$300 и определяют условие соединения таблиц Product и Tree. Значение вершины @id определяется в виде \$p/product_id, что указывает на то, что содержимое атрибута id в XML-представлении будет определяться значением столбца product_id таблицы Product.

Более сложный пример дерева запросов изображен на рис. 4 (смысл обозначений τ в правых нижних углах вершин будет пояснен позже). Это дерево запросов извлекает из базы производителей (`vendors`), для каждого производителя (`vendor`) его `@id, vendorName, url, state` и кортежи телефонов (`phone`) и КПК (`pda`), которые включаются в общую "оболочку" `products`. Корень `vendors` содержит множество дочерних вершин `vendor` (*-ребро). В вершине `vendor` переменная `$v` связывается с таблицей Vendor; эта вершина имеет несколько дочерних вершин (`@id, vendorName, vendorInfo`), которые соединены с ней простыми ребрами и отражают информацию о производителе. Значение атрибута `id` определяется выражением пути вида `$v/vendor_id`, аналогично для вершины `vendorName` — `$v/vendor_name`. Вершина `vendorInfo` устроена немного сложнее, она состоит из субвершин `url` и `state`.

Вершина `products` имеет две дочерних вершины, с которыми она соединена *-ребрами — `phone` и `pda`. Дескрипторы источника вершины `phone` связывают переменную `$p` с таблицей `Product` и `$t` — с `Tree`, а дескрипторы ограничения соединяют кортежи таблицы `Product` с кортежами `Vendor` и кортежи `Product` с кортежами `Tree` (условия соединения), а также ограничивают выборку продуктами-телефонами (`$t/tree_id = 2`). Вершина `pda` описывается аналогично с той лишь разницей, что выборка ограничивается продуктами-КПК (`$t/tree_id = 4`). Результат работы этого дерева запросов — представление $\mathtt{View}_2$, показанное на рис. 2.

С этого момента мы будем предполагать, что дерево запросов не пусто, т.е. корневая вершина не является единственной.

2.2. Абстрактные типы вершин. Вершины в дереве запросов можно разделить на группы, исходя из их семантики. Для этого в [8] определяется пять типов вершин: τ, τ_T, τ_N, τ_C и τ_S. Будем называть их *абстрактными* для того, чтобы отличать от типов DTD XML-документов.

Определение 6 (*абстрактные типы*). Абстрактные типы приписываются вершинам согласно следующим правилам:

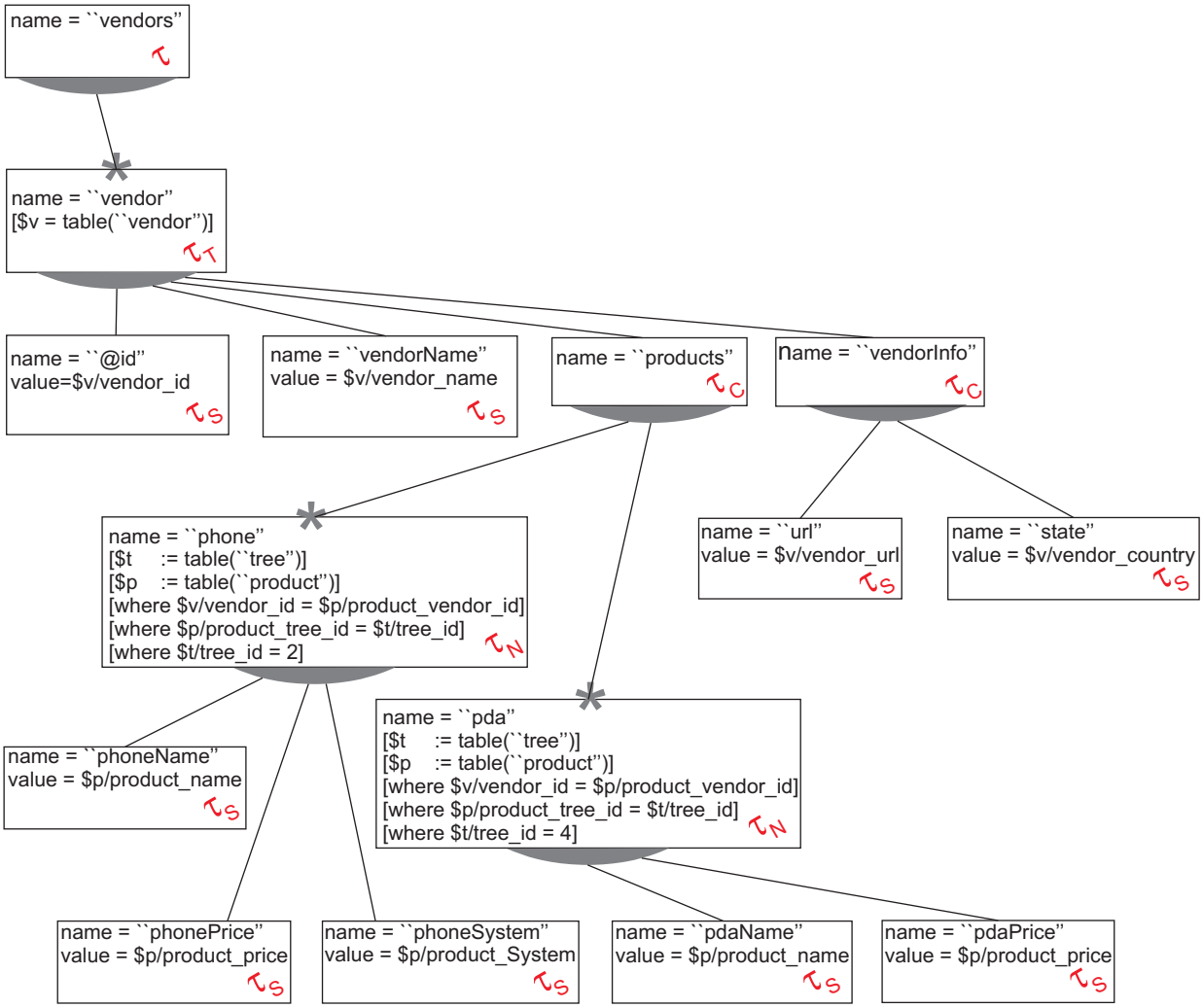

Рис. 4. Дерево запросов для View_2 — qt_1

— корень имеет абстрактный тип τ;

— каждая листовая вершина имеет абстрактный тип τ_{S} (*Simple*);

— каждая нелистовая вершина с простым входящим ребром имеет абстрактный тип τ_{C} (*Complex*);

— каждая *-вершина, которая является либо листовой, либо все ее потомки соединены только простыми ребрами, имеет абстрактный тип τ_{N} (*Nested*);

— все остальные *-вершины имеют абстрактный тип τ_{T} (*Tree*).

Заметим, что все вершины имеют ровно один тип, за исключением некоторых *-вершин, которые обладают двумя типами — τ_{S} и τ_{N}.

В качестве примера рассмотрим дерево запросов, изображенное на рис. 4, где рядом с каждой вершиной указан ее тип. Так как `phone` и `pda` являются *-вершинами, а все их потомки — простые вершины, то они имеют тип τ_{N}, а не τ_{T}.

Причины введения такого определения абстрактных вершин таковы. Для преобразования операций обновлений XML-представления в операции обновления реляционных представлений мы должны выявить соответствие атрибутов кортежей реляционной базы элементам или атрибутам XML-представления. В идеале такое соответствие должно оказаться взаимно-однозначным: каждый атрибут кортежа имеет ровно один образ в XML-представлении и, таким образом, может подвергаться операциям обновления без каких-либо сторонних эффектов в представлении. В общем случае, однако, это отображение типа 1:n. Класс представлений, порождаемый деревьями запросов, позволяет осуществлять такого рода отображения.

Итак, семантика абстрактных типов вершин деревьев запросов такова:

— вершинам типов $\tau_{\mathrm{T}}/\tau_{\mathrm{N}}$ соответствуют кортежи в реляционной базе; вершина типа τ_{T} может иметь дочерние вершины типа τ_{T}, т.е. разрешены вложения таких вершин;

— τ_S описывает реляционные атрибуты (столбцы); вершина типа τ_S должна иметь предка типа τ_T или τ_N; листовые вершины, отмеченные звездочкой, являются исключением из этого правила: не требуется, чтобы они имели предка такого типа;

— τ_C определяет сложные XML-элементы; так как они не содержат значений, вершинам такого типа нет соответствия в реляционной модели; вершины типа τ_C вводятся для создания более гибких XML-представлений, однако они не важны в процессе отображения.

Назовем XML-представления, полученные при помощи деревьев запросов и ассоциированных с ними абстрактных типов, *доброкачественными*, ввиду того что, как будет показано ниже, они могут легко быть отображены на соответствующие реляционные представления. Дерево запросов, в котором каждой вершине сопоставлен ее абстрактный тип, далее будем называть *типизированным* деревом запросов.

Прежде чем переходить к рассмотрению механизма трансляции операций обновления XML-представлений на реляционные представления, приведем без доказательства два утверждения (доказательства данных утверждений можно найти в [8]).

Утверждение 1. Существует по крайней мере одна τ_N-вершина в любом типизированном дереве запросов `qt`.

Утверждение 2. В любом типизированном дереве запросов qt вдоль любого пути от листа к корню имеется не более одной τ_N-вершины.

Под *абстрактным типом элемента* понимается абстрактный тип вершины, использующейся при получении этого элемента, за которым идет имя элемента. Например, элементу pda представления View_2 приписан абстрактный тип τ_N(pda), а его DTD-типом является <`!ELEMENT pda(pdaName, pdaPrice)`>.

2.3. Отображение XML-представлений на множество реляционных представлений. Рассмотрим, как XML-представление, полученное с помощью дерева запросов, можно преобразовать в набор реляционных представлений. Как уже упоминалось, это делается для того, чтобы воспользоваться механизмом обновления представлений, который есть в большинстве развитых реляционных СУБД, для проверки корректности операции обновления XML-представления применительно к реляционной базе и, если операция корректна, переноса этой операции на базовые реляционные таблицы.

Сначала рассмотрим построение одного реляционного представления для XML-представления, в котором есть только одна τ_N-вершина, далее перейдем к более сложным случаям и опишем алгоритм разделения (расщепления) сложных представлений (несколько τ_N-вершин) на несколько простых.

Отображение (*mapping*). Если задано дерево запросов `qt` с единственной τ_N вершиной, SQL-предложение для соответствующего реляционного представления получается следующим образом. Соединяем все таблицы, которые присутствуют в дескрипторах источника (назовем эти таблицы *таблицами-источниками*) данной вершины `n` в дереве `qt`, используя дескрипторы ограничения, которые соответствуют условиям соединений таблиц-источников `n`, а точнее, внутренних соединений (`INNER JOIN`). Если нет условий соединения, используем "тавтологическое" условие (т.е. `1=1`) в качестве условия соединения, результатом чего будет декартово произведение. Назовем такие выражения *выражениями соединенных источников*. Спускаясь по дереву от корня к листьям, соединяем выражения соединенных источников, которые встречаются на пути от корня к вершине с помощью операции левого внешнего соединения (`LEFT OUTER JOIN`). Условия для внешних соединений получаются очевидным образом: если вершина `a` является предком вершины `n` и дескриптор ограничения в `n` определяет условие соединения таблицы в `n` с таблицей в `a`, то используем это определение как условие соединения для операции внешнего соединения. Так же, как и в случае с внутренними соединениями, если нет условия для внешнего соединения, используем выражение вида `1=1`. Далее используем оставшиеся дескрипторы ограничения (те, которые не были использованы для построения условий внутренних и внешних соединений) в предложении `WHERE` языка SQL и обрабатываем значения листовых вершин. Заметим, что в отличие от [8] в условиях ограничения могут быть не только простые сравнения (операции <, >, =, <=, >=), но и другие выражения, такие как `IS NULL, LIKE exp` и т.д.

Примеры выражений соединенных источников:

```
<source table> AS <source variable> INNER JOIN
<source table> AS <source variable> INNER JOIN ...
ON <inner joincond>
```

Соответствующее SQL-выражение:

```
SELECT
<leaf value> AS <leaf name>,
...,
```

```
<leaf value> AS <leaf name>
FROM
(<source join expression>
LEFT JOIN <source join expression> ON <outer joincond>)
LEFT JOIN ...
WHERE
<remaining ‘‘where’’ annotation> AND
... AND
<remaining ‘‘where’’ annotation>
```

Например, реляционное представление, соответствующее дереву запросов на рис. 3, выглядит так:

```
SELECT
p.product_id AS id, p.product_name AS name, p.product_price AS price
FROM
(Tree AS t
INNER JOIN Product AS p
ON p.product_tree_id = t.tree_id)
WHERE t.tree_id = 2 AND p.product_vendor_id = 14 AND p.product_price > 300
```

Расщепление (*split*). Для дерева запросов с более чем одной τ_{N}-вершиной описанный процесс некорректен. Например, рассмотрим дерево запросов $\mathtt{qt}_1$ (рис. 4), которое имеет две τ_{N}-вершины (`phone` и `pda`). При попытке применить описанную выше процедуру отображения (mapping) таблица `Product` будет соединена сама с собой, в результате получится декартово произведение (точнее, почти декартово произведение: каждый кортеж-телефон будет соединен с каждым кортежем-КПК). Это очевидным образом нарушит семантику дерева запросов. Значит, перед созданием соответствующего реляционного представления необходимо расщепить дерево запросов на поддеревья, которые содержат ровно одну τ_{N}-вершину. После процесса расщепления на основе каждого полученного поддерева запросов создается реляционное представление с помощью процедуры mapping.

Процесс расщепления состоит в изолировании вершины `n` типа τ_{N} в дереве запросов `qt` и работе с полученным поддеревом так, как будто ее вершины-предки и неповторяющиеся вершины-потомки (типов τ_{C} и τ_{S}) формируют новое дерево $\mathtt{qt}_{\mathrm{i}}$. Вспомним, что, согласно утверждению 1, любое $\mathtt{qt}_{\mathrm{i}}$ должно иметь по меньшей мере одну τ_{N}-вершину.

Результат работы этого алгоритма над деревом запросов (рис. 4) показан на рис. 5 и 6. Используя данные отщепленные деревья, получаем следующие реляционные представления `ViewPhone` и `ViewPda` (по этим именам будем ссылаться на данные реляционные представления далее):

```
CREATE VIEW ViewPhone AS p
SELECT
v.vendor_id AS id, v.vendor_name AS vendorName, v.vendor_url AS url,
v.vendor_country AS state, p.product_name AS phoneName, p.product_price AS
phonePrice, p.product_system AS phoneSystem
FROM
(Vendor AS v
LEFT JOIN (Tree AS t
INNER JOIN Product AS p ON p.product_tree_id = t.tree_id)
ON v.vendor_id = p.product_vendor_id)
WHERE t.tree_id = 2;
```

```
CREATE VIEW ViewPda AS p
SELECT
v.vendor_id AS id,
v.vendor_name AS vendorName,
v.vendor_url AS url,
v.vendor_country AS state,
p.product_name AS pdaName,
p.product_price AS pdaPrice
FROM
```

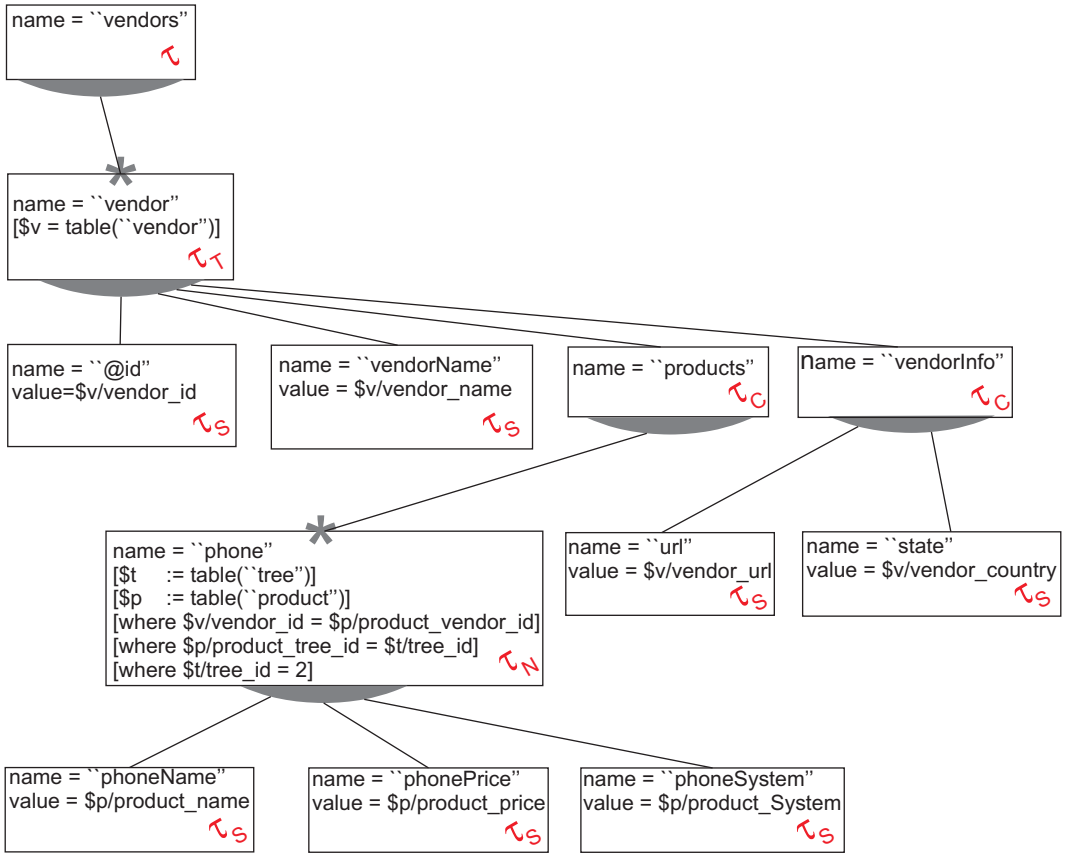

Рис. 5. Отщепленное дерево запросов для τ_{N}(phone)

```
(Vendor AS v
LEFT JOIN (Tree AS t
INNER JOIN Product AS p ON p.product_tree_id = t.tree_id)
ON v.vendor_id = p.product_vendor_id)
WHERE t.tree_id = 4;
```

3. Обновление деревьев запросов. Рассмотрим, как операцию обновления XML-представления можно преобразовывать в совокупность операций обновления соответствующего ему набора реляционных представлений.

Все операции обновления делятся на три группы: операции вставки, удаления и модификации. Сначала обсудим способы описания операций обновления, а потом рассмотрим отдельно каждый из трех классов операций и методы преобразования.

3.1. Язык обновлений. Обновления определяются с использованием выражения пути для того, чтобы указать множество вершин в XML-дереве, которые подлежат обновлению. Для вставок и изменений в операции обновления также должно быть указано значение Δ, содержащее новые значения.

Определение 7 (*операция обновления*). Операция обновления u — это тройка <t, Δ, ref>, где t есть тип операции (insert, delete или modify); Δ — XML-дерево, подлежащее вставке, или, в случае модификации, атомарное значение (один элемент); ref — простое выражение пути на языке XPath [5], которое обозначает, где именно должно произойти обновление.

Выражение пути ref применяется от корня дерева, и в результате применения мы получаем множество вершин, которые будем называть *целевыми*. В случае модификации это должны быть листовые вершины. Ограничим множество фильтров, используемых в ref, конъюнкциями сравнений атрибутов или дочерних элементов с атомарными значениями. Будем называть выражение ref, из которого удалены фильтры, *абсолютной частью* ref. К примеру, абсолютной частью выражения `/vendors/vendor[@id=‘‘01’’]` является `/vendors/vendor`.

Определение 8 (*корректность выражения пути*). Выражение пути ref является корректным по отношению к дереву запросов qt, если абсолютная часть ref полностью применима к этому дереву (т.е.

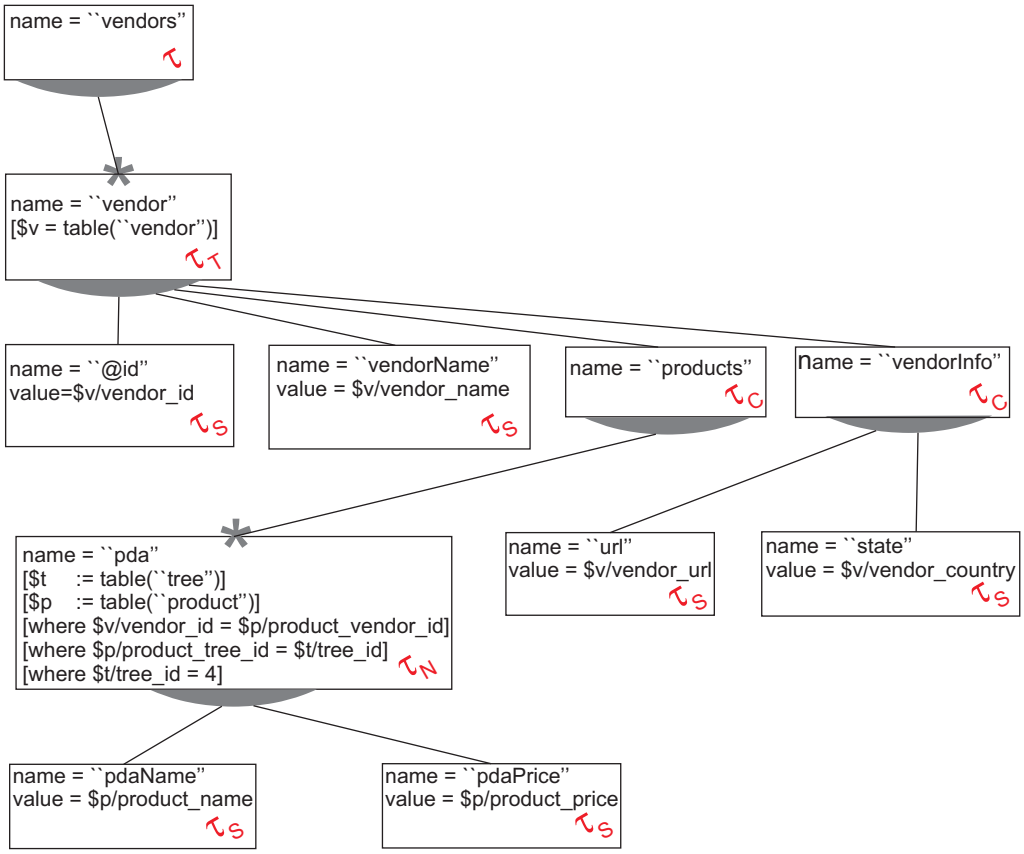

Рис. 6. Отщепленное дерево запросов для $\tau_{\mathrm{N}}(\mathrm{pda})$

результат применения всего выражения пути — непустое множество).

К примеру, `/vendors/vendor[@id=‘‘01’’]/vendorName` — корректный путь по отношению к дереву запросов $\mathtt{qt}_1$ (рис. 4), т.к. путь `/vendors/vendor/vendorName` применим к этому дереву.

Во время операции вставки Δ вставляется как дочернее поддерево к вершине, указанной выражением `ref`; для операции модификации значение листовой вершины, указанной в `ref`, заменяется на Δ; в процессе удаления поддерево, исходящее из вершины, указанной с помощью `ref`, удаляется.

Рассмотрим несколько примеров для представления View_2 (рис. 2).

Пример 1. Для вставки нового телефона, продаваемого по цене \$ 430, производитель которого имеет id=“14” (Samsung), надо указать: `t = insert, ref = /vendors/vendor[@id=‘‘14’’]/products,`

```
Δ = {<phone>
     <name>SGH-D410</name>
     <price>430</price>
<system>EGSM 900/1800/1900</system>
     </phone>}.
```

Пример 2. Для изменения vendorName производителя с id = “42” с “Trium” на “Mitsubishi” надо указывать: `t = modify, ref = /vendors/vendor[@id=‘‘42’’]/vendorName, Δ = Mitsubishi.`

Пример 3. Удаление всех телефонов с названием “1100”:

```
t = delete, ref = /vendors/vendor/products/phone[name=‘‘1100’’].
```

Не все операции вставки и удаления имеют смысл, так как получившееся после их применения XML-представление может не удовлетворять DTD, которое было получено с помощью дерева запросов. Например, удаление, заданное выражением пути `/vendors/vendor/vendorName` для представления $\mathtt{View}_2$ (рис. 2 и 4), приведет к нарушению DTD, так как vendorName является обязательным дочерним элементом элемента `vendor`. Аналогично, необходимо проверять корректность операций вставки и модификации.

Определение 9 (*корректность операции обновления*). Обновление <`t, Δ, ref`> XML-представления, построенного с помощью дерева запросов `qt`, является корректным тогда и только тогда, когда выполнены следующие условия:

— `ref` является корректным по отношению к `qt`;

— если `t = modify`, то абсолютная часть `ref`, примененная к `qt`, приводит к вершине типа τ_S;

— если `t = insert`, то абсолютная часть `ref` + корень Δ, примененные к `qt`, приводят к вершине, у которой входящее ребро есть *-ребро (т.е. вершина типа τ_T или τ_N);

— если `t = delete`, то абсолютная часть `ref`, примененная к `qt`, приводит к вершине, чье входящее ребро есть *-ребро;

— если Δ не пусто, то оно удовлетворяет DTD элемента, к которому приводит `ref`.

Рассмотрим операцию удаления в примере 3. Она корректна, так как `phone` является дочерней *-вершиной вершины `product`. Удаление, задаваемое выражением пути `/vendors/vendor/vendorName`, не является корректным, так как `vendorName` имеет абстрактный тип τ_S, так же, как и, например, удаление, задаваемое некорректным выражением пути `/vendors/vendor/pda`.

Операция вставки из примера 1 является корректной, так как вершина phone (к которой приводит выражение пути `/vendors/vendor/products` + корень Δ — phone) является дочерней *-вершиной вершины products, DTD для phone есть <`!ELEMENT phone (name,price,system)`> и Δ удовлетворяет ей.

3.2. Трансляция XML-обновлений на реляционные представления. Теперь рассмотрим, как корректные обновления XML-представлений преобразуются в SQL-операции обновления соответствующих реляционных представлений.

В дальнейших примерах будет использоваться представление $\texttt{View}_2$ (рис. 2 и 4). Реляционные представления `ViewPhone` и `ViewPda`, которые соответствуют этому XML-представлению, были описаны в п. 2.3.

Алгоритмы трансляции для операций вставок, удалений и модификаций (`doInsert, doDelete` и `doModify` соответственно) предполагают, что операция обновления u является корректной (т.е. проверка на корректность уже проведена).

3.2.1. Вставка. Для трансляции операции вставки в XML-представление на реляционные представления делается следующее. Во-первых, абсолютная часть выражения пути ref используется для нахождения той вершины в дереве запросов, где будет произведена вставка. Вместе с Δ это будет использовано для определения реляционного представления, подвергаемого обновлению. Во-вторых, формируется SQL-выражение для каждого подвергаемого обновлению реляционного представления с использованием информации в Δ, а также информации о метках и значениях в поддеревьях, исходящих из вершин на пути от каждой целевой вершины к корню XML-документа.

Заметим, что согласно утверждению 2 вдоль пути из любой вершины к корню в дереве запросов есть только одна вершина типа τ_N, кроме того, вставки никогда не производятся ниже вершины типа τ_N, так как все вершины ниже вершины типа τ_N по определению должны иметь тип τ_S или τ_C.

Рассмотрим трансляцию операции обновления, описанной в примере 1. Безусловная часть выражения пути `/vendors/vendor/products/` применяется к дереву запросов qt_1 (рис. 4), при этом выходит, что тип целевой вершины — τ_C(`products`). Продолжая от τ_N(`products`) с использованием структуры Δ находим, что единственная τ_N-вершина в Δ — это корень, τ_N(`phone`). Реляционное представление, подлежащее операции обновления, таким образом, — `ViewPhone`. Далее, для определения элементов в XML-документе, которые подлежат обновлению, используем выражение пути `ref = /vendors/vendor[@id=‘‘14’’]/products`. В нашем случае это единственный элемент. Таким образом, должно быть сгенерировано только одно SQL-выражение вставки для `ViewPhone`.

Для формирования SQL-выражения вставки надо найти значения всех атрибутов (столбцов) представления. Некоторые из этих пар “атрибут–значение” извлекаются из Δ, другие должны быть взяты из XML-документа при анализе пути от каждой целевой вершины к корню и получения пар “атрибут–значение” из листовых вершин поддеревьев, исходящих из вершин этого пути. В примере 1 Δ задает `name=‘‘SGH-D410’’, price=‘‘430’’, system=‘‘EGSM 900/1800/1900’’`. Вдоль пути от целевой вершины к корню находим: `id=‘‘14’’, vendorName=‘‘Samsung’’, url=‘‘http://samsung.com’’, state=‘‘KR’’`. Комбинируя эту информацию, мы создаем следующее SQL-выражение вставки:

```
INSERT INTO
ViewPhone(id, vendorName, url, state, name, price, system)
VALUES
(‘‘14’’, ‘‘Samsung’’, ‘‘http://samsung’’, ‘‘KR’’, ‘‘SGH-D410’’, ‘‘430’’,
‘‘ EGSM 900/1800/1900’’)
```

3.2.2. Модификация. По определению, модифицировать можно только значения листовых вершин. Для выполнения операции модификации необходимо следующее. Во-первых, выражение ref используется для определения имен реляционных представлений, над которыми будет произведена попытка выпол-

нения операции. Это достигается нахождением первого предка вершины, определяемой выражением ref, который имеет тип τ_N или τ_T с последующим поиском всех τ_T-вершин его поддерева. (По крайней мере одна τ_N-вершина должна существовать.) Если целевая листовая вершина и есть эта τ_N-вершина, то это дает гарантии того, что обновление будет транслировано только на реляционное представление, соответствующее этой вершине. Во-вторых, создается SQL-выражение. При этом атрибуты, которые задействованы в ref, комбинируются со значениями из Δ, далее с использованием фильтров в ref вычисляется предложение WHERE SQL-выражения.

3.2.3. Удаление. Операция удаления является самой простой из трех. Сначала абсолютная часть используется для нахождения вершины дерева, которую необходимо удалить (это не обязательно листовая вершина — т.е. возможно удаление поддерева). Далее с использованием дерева запросов определяется, какие реляционные представления подвергаются операции обновления, и создаются SQL-выражения — по одному для каждого из соответствующих реляционных представлений.

4. SQL/XML и деревья запросов. В 14-й части стандарта SQL:2003 описывается расширение языка SQL (т.н. SQL/XML) для поддержки XML в реляционных СУБД: приводятся правила преобразования SQL-данных в XML и обратно и описываются дополнительные конструкции SQL, позволяющие получать XML-документы (или их части) с помощью расширенного оператора SELECT.

Опишем здесь основные конструкции для получения XML-данных с помощью расширенных SQL-запросов, далее перейдем к рассмотрению правил построения деревьев запросов на основе расширенных предложений SELECT.

Здесь приведен неполный список конструкций. Исчерпывающий список, содержащий формальные определения всех функций, можно найти в тексте стандарта SQL/XML [2].

XMLElement и XMLAttributes. XMLElement создает элемент документа XML, при этом в качестве операндов выступают имя и содержимое элемента. Имя указывается в качестве первого аргумента функции, причем перед именем должно стоять обязательное ключевое слово `NAME`. Далее может идти произвольное количество операндов, цель которых — перечисление данных (значение, атрибуты и элементы), которые содержит данный элемент. `XMLAttributes` служит для создания списка XML-атрибутов соответствующего XML-элемента. Пример:

```
SELECT
    XMLElement(
        NAME phone,
            XMLAttributes(p.product_id AS id)
            XMLElement(NAME price, p.product_price)
        )
FROM ViewPhone AS p
```

В результате выполнения данного выражения будет создана совокупность из элементов `<phone>` (по одному на каждую строку представления `ViewPhone`), содержащих атрибуты `@id` и по одному вложенному элементу `<price>`, каждый из которых в свою очередь содержит в виде значения цену.

XMLForest. Функция создает последовательность XML-элементов из последовательности перечисленных в качестве операндов имен столбцов. Пример:

```
SELECT
    XMLElement(
        NAME phone,
            XMLAttributes(p.product_id AS id)
            XMLForest(p.product_name AS name, p.product_price AS price)
        )
FROM ViewPhone AS p
```

XMLConcat. Производит операцию конкатенации, объединяя различные XML-элементы в лес элементов, образуя единый документ.

XMLAgg. Данная конструкция действует над совокупностью значений, производя лес XML-элементов. Например, для получения XML-документа, подобного тому, что изображен на рис. 1, можно использовать следующее выражение:

```
SELECT
    XMLElement(
        NAME vendor,
```

```
            XMLAttributes(v.vendor_id AS id),
            XMLAgg(
                XMLElement(
                    NAME products
                    XMLElement(
                        NAME phone,
                        XMLForrest(p.product_name AS name, p.product_price AS price)
                    )
                )
            )
        )
    FROM
        vendor AS v
        JOIN product AS p ON p.product_vendor_id = v.vendor_id
    GROUP BY
        vendor_id
```

Правила построения деревьев запросов на основе SQL/XML-выражений. Опишем правила построения деревьев запросов на основе выражений SELECT, использующих приведенные выше конструкции. Но сначала необходимо сделать несколько замечаний. Во-первых, необходимо отметить, что в общем случае такие выражения возвращают не один документ, а совокупность XML-элементов. Это значит, что простая конкатенация даст лес элементов, что не является корректным XML-документом. Эта проблема легко может быть разрешена введением корневого элемента-контейнера. Во-вторых, использование такого расширенного оператора SELECT для определения представлений (`CREATE VIEW viewname AS SELECT` ...) стандартом не предусматривается. Но, тем не менее, это представляется возможным, если ввести дополнительные допущения. Будем рассматривать только те операторы SELECT, которые в списке выборки содержат одну или несколько функций `XMLElement` верхнего уровня (иными словами, результат выполнения запроса не содержит не-XML данные). Кроме того, будем считать, что после выполнения оператора лес XML-элементов подвергается операции конкатенации и включается в элемент-контейнер с названием, совпадающим с названием представления.

При указанных допущениях правила построения дерева запросов просты. Построение происходит в два этапа. Первый этап состоит в разборе списка выборки и построения структуры дерева запросов без задания дескрипторов источника, ограничения и сортировки. Второй этап — разбор разделов FROM, WHERE и ORDER BY оператора SELECT, что дает возможность определить дескрипторы и завершить построение дерева запросов.

Корневая вершина дерева имеет название, совпадающее с названием XML-представления. От нее исходит *-ребро к вершине, описывающей получение XML-элемента верхнего (для оператора SELECT) уровня. Далее, для каждого описываемого элемента и атрибута определяем вершину в дереве запросов, сохраняя отношения вложенности. Заметим, что конструкции `XMLAttributes` и `XMLForrest` в общем приводят к созданию нескольких вершин. Конструкция `XMLAgg` (и только она!) приводит к появлению дополнительных *-ребер. Для всех листовых вершин необходимо указание источника (при этом дескрипторы на первом этапе не определяются, но начинают использоваться альтернативные имена источников).

На следующем этапе производится анализ разделов FROM (для построения дескрипторов источника), WHERE (для построения дескрипторов ограничения) и ORDER BY (дескрипторы сортировки). Для каждой таблицы-источника в разделе FROM создается дескриптор источника и помещается в вершине, которая является общим предком всех вершин, содержащих обращения к данному источнику. Аналогичные действия производятся с содержимым разделов WHERE и ORDER BY.

5. Применение техники деревьев запросов. Как уже отмечалось, задача экспорта реляционных данных в формате XML сегодня является чрезвычайно актуальной. Конструкции, определенные в 14-й части стандарта SQL : 2003 призваны обеспечить такую возможность в реляционных СУБД.

Деревья запросов позволяют осуществить поддержку XML-представлений в реляционных СУБД. Задачи таких представлений аналогичны задачам реляционных представлений: интеграция данных и защита данных. Интеграция достигается за счет объединения информации из различных таблиц (кроме того, если реляционная СУБД поддерживает встроенный XML-тип, то возможна интеграция данных различного рода — реляционного и XML), а решение вопросов защиты информации может быть достигнуто с помощью наложения ограничений на данные. Именно поэтому задача поддержки обновления таких представлений является актуальной. Возможно построение систем, поставляющих данные в виде XML-представлений

удаленным агентам, от которых поступает полная информация о природе данных и которым доступна только часть информации. Как частный случай такой системы можно рассматривать универсальное средство для создания web-интерфейсов работы с данными, речь о котором пойдет в следующем разделе.

5.1. ***XViews*****: система динамического создания интерфейсов работы с данными.** Изложенные выше приемы работы с XML-представлениями над реляционными СУБД были использованы при разработке приложения *XViews*.

Идея этого приложения заключается в следующем. Большая часть интерфейсов работы с данными (например, редакторские интерфейсы порталов, интерфейсы для работы с данными о сотрудниках предприятия и т.п.) по своей структуре представляют собой иерархически организованное множество текстовых полей ввода, списков выбора, radio-кнопок и других элементов. В то же время, как правило, вся информация хранится в реляционных базах данных (причины этого обсуждались во введении к данной работе). Поэтому возникла идея использования теории обновления XML-представлений над реляционными базами данных (БД) для упрощения и ускорения разработки интерфейсов для работы с данными.

В качестве теоретической основы была взята теория деревьев запросов, впервые предложенная в [8]. В качестве реляционной СУБД использовалась *Microsoft SQL Server 2000*, основного языка программирования — *PHP5*, что дало возможность применения достаточно мощной библиотеки для работы с XML *libxml2*, а также стандарта объектного представления XML-документов *DOM XML*.

Были созданы следующие основные классы.

1. **QueryTree** — класс, реализующий понятие дерева запросов. Само дерево запросов задается в виде XML-файла, DTD которого представлен ниже в приложении B. После загрузки XML-файла строится DOM-дерево, описывающее дерево запросов, и определяются типы вершин (τ, τ_{C}, τ_{S}, τ_{N} или τ_{T}). После этого процесс создания объекта класса завершается и он готов к дальнейшей работе. В дополнительные возможности класса входит графическое представление (визуализация с помощью библиотеки *gd2*) в виде единого png-файла полученного дерева (пример — рис. 3 – 6).

2. **QueryTree_XMLConstructor** — класс для создания XML-представления (алгоритм eval).

3. **QueryTree_Mapper** — класс для отображения XML-представления, созданного с помощью дерева запросов с одной вершиной типа τ_{N}, в одно соответствующее ему реляционное представление (алгоритм mapping).

4. **QueryTree_Splitter** расщепляет дерево запросов с несколькими вершинами типа τ_{N} и τ_{T} в несколько простых деревьев (алгоритм split).

5. **QueryTree_Translator** — класс для трансляции обновления XML-представления в набор SQL-выражений обновления соответствующих реляционных представлений.

Таким образом, была построена основа (прототип) для системы динамического создания web-интерфейсов с древовидной структурой для работы с данными, хранящимися в реляционной БД.

На данный момент поставлены следующие задачи дальнейшей разработки системы.

Этап построения дерева запросов. Процесс построения дерева запросов заключается в создании XML-файла экспертом, который должен знать систему, схему БД и конечный вид представления. Необходимо создать инструментарий, решающий задачу визуального создания дерева запросов. Предполагаемые языки программирования и технологии: *Java, libxml2*.

Этап создания интерфейса на основе XML-представления. Создание библиотеки автоматического построения front-end систем. Предполагаемые языки программирования и технологии: *PHP5, DHTML, XSLT, Macromedia Flash MX 2004*. (На данный момент реализовано на основе Flash MX 2004 для простых видов представлений).

Теоретические основы. Расширение класса возможных представлений (увеличение поддерживаемых видов описаний ограничений, разрешение простых операций с получаемыми из БД значениями констант), введение системы дополнительных связей (для организации списка выбора и т.п.). Исследование возможности использования ссылочных ограничений целостности для автоматического построения шаблонов представлений (мастера построения представлений).

Использование XMLHttpRequest [21] для создания динамичных интерфейсов (т.е. интерфейсов, позволяющих вызывать операции обновления без перегрузки страницы).

Конечная цель разработки данной системы — создание приложения XViews, универсального (независимого от платформы) инструмента создания удобных web-интерфейсов для работы с данными.

5.2. Поддержка обновляемых XML-представлений в СУБД PostgreSQL. PostgreSQL — это свободно распространяемая объектно-реляционная система управления базами данных, наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.

Именно поэтому развитие поддержки XML в целом и XML-представлений в данной СУБД представляется актуальной задачей. На настоящий момент XML-поддержка заключается в возможности сохранения XML-данных в виде CLOB-значений и последующего применения XPath-выражений к таким данным (модуль contrib/xml2). Очевидно, что это не обеспечивает решение всей совокупности задач, описанных выше. Для расширения поддержки XML планируется осуществление следующих шагов.

1) Реализация функций SQL/XML. К сожалению, стандартный синтаксис оказывается очень "неудобным" для реализации, в основном по двум причинам: требование ключевого слова NAME в конструкции XMLElement и возможность указания произвольного числа операндов в некоторых конструкциях. Именно поэтому реализация с использованием стандартного API для написания пользовательских функций и операторов представляется невыгодной и было предложено (и частично реализовано) дополнение в виде patch'a. Возможно, данное дополнение будет включено в версию 8.2.

2) Создание встроенного XML-типа. Основные, проблемы возникающие при реализации данной задачи, таковы: создание индекса для доступа к частям документов; поддержка методов манипуляции с такими данными.

3) Обеспечение возможности создания обновляемых XML-представлений.

6. Заключение. Решение задачи обновления XML-представлений открывает новые перспективы работы с данными на стыке двух совершенно разных технологий — XML и реляционных баз данных. Сочетание преимуществ реляционных СУБД и широких возможностей XML в универсальной системе позволяет достичь хороших результатов. Проблема обновления XML-представлений на данный момент не является хорошо изученной. Поэтому многое еще предстоит исследовать в этой области.

Описанная методика может служить основой для создания интеграционной платформы, обеспечивающей работу с двумя моделями данных одновременно. Представляя собой простую для понимания конструкцию, деревья запросов позволяют создавать XML-представления, приводить их ко второй нормальной форме, отображать на множество реляционных представлений, осуществлять преобразование обновлений XML-документа в набор SQL-выражений.

В данной работе был описан подход к решению проблемы обновления XML-представлений, основы которого изложены в [8]. Предложено расширение класса поддерживаемых представлений за счет поддержки более широкого множества ограничений и за счет введения дополнительного типа описаний вершин — описаний сортировки. На примере приложения XViews была продемонстрирована возможность эффективного применения деревьев запросов для решения задачи автоматизации, упрощения и ускорения построения интерфейсов работы с данными. В заключение приведен краткий анализ возможности реализации поддержки обновляемых XML-представлений в СУБД PostgreSQL.

Приложение А. Схема БД web-каталога. На рис. 7 показана часть схемы базы данных сайта-каталога. Информация о продуктах, содержащихся в каталоге, находится в таблице product, о производителях — в таблице vendor. Таблица tree служит для категоризации продуктов и построения каталога древовидного типа (каждый продукт относится к одной из категорий, каждая категория может принадлежать другой категории).

Приложение В. DTD для описания деревьев запросов в виде XML-файлов.

```
<?xml version=''1.0'' encoding=''UTF-8''?>
<!ELEMENT root (children)>
<!ATTLIST root
        name CDATA #IMPLIED
        atype (T | TS | TC | TN | TT) #IMPLIED
>
<!ELEMENT node (
        source-annotation*,
        where-annotation*,
        sortby-annotation*,
        children
)>
<!ATTLIST node
        name CDATA #REQUIRED
        edgetype (simple | starred) #REQUIRED
        atype (T | TS | TC | TN | TT) #IMPLIED
>
```

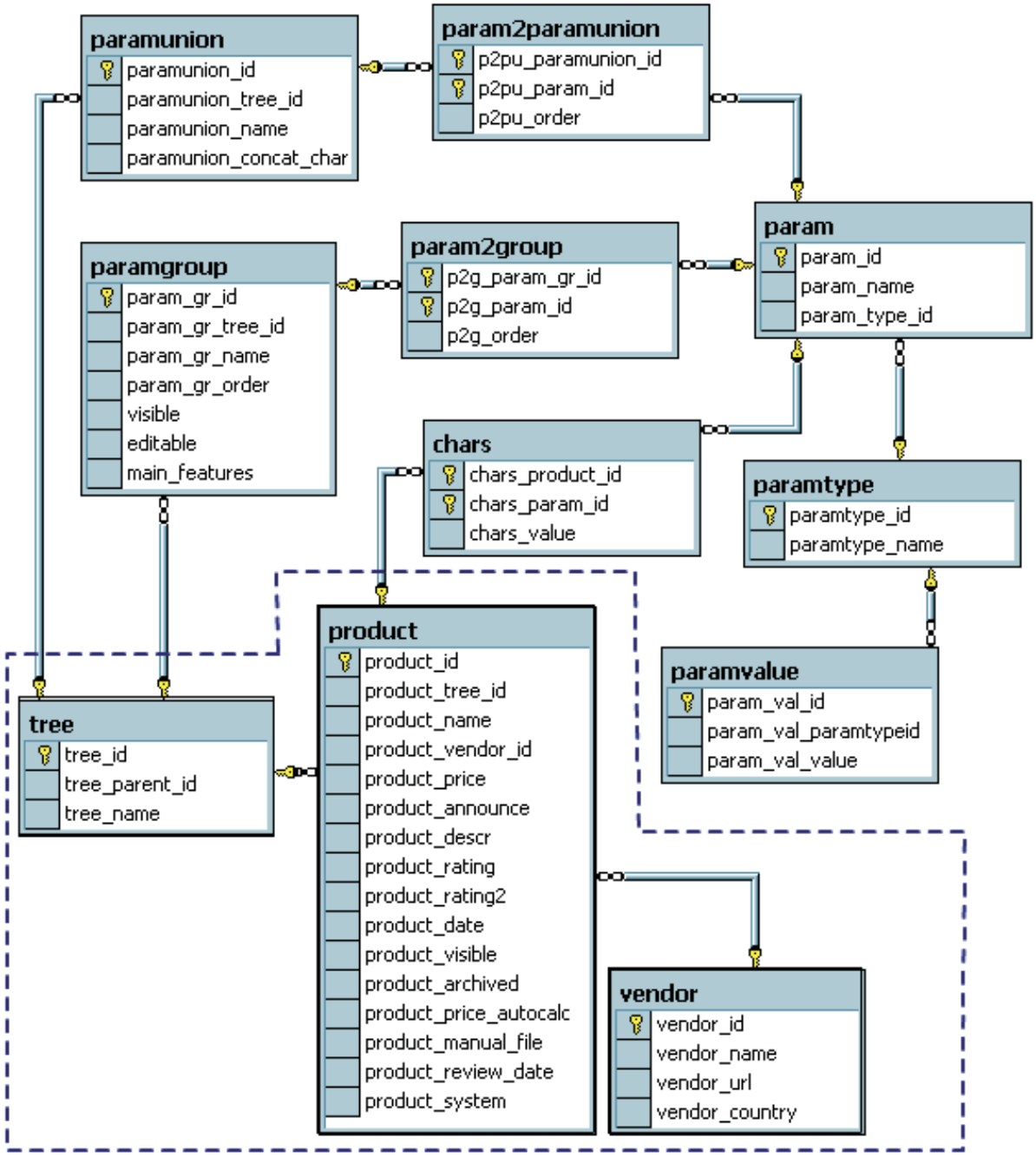

Рис. 7. Схема БД web-каталога

```
<!ELEMENT children (node | leafnode)*>
<!ELEMENT leafnode EMPTY>
<!ATTLIST leafnode
        name CDATA #REQUIRED
        edgetype (simple | starred) #REQUIRED
        value CDATA #REQUIRED
        atype (T | TS | TC | TN | TT) #IMPLIED
>
<!ELEMENT source-annotation EMPTY>
<!ATTLIST source-annotation
        var CDATA #REQUIRED
```

```
        table CDATA #REQUIRED
>
<!ELEMENT where-annotation (#PCDATA)>
<!ELEMENT sortby-annotation>
<!ATTLIST sortby-annotation
        var CDATA #REQUIRED
        desc CDATA #IMPLIED
```

Все вершины дерева запросов задаются элементами root, node и leaf-node. Ребра дерева описываются посредством промежуточных элементов children, которые могут быть дочерними у root или node. Изначально в XML-файле дерева запросов не задается атрибут "atype" у элементов root, node и leaf-node, соответствующий абстрактному типу вершины (поэтому он не является обязательным), но обязательно атрибут edgetype, отвечающий за тип входящего ребра в дереве запросов.

СПИСОК ЛИТЕРАТУРЫ

1. Extensible Markup Language (XML) 1.0 // W3C Recommendations. W3C, 2004 (http://www.w3.org/TR/2004/REC-xml-20040204/).
2. SQL:2003. ISO/IEC JTC 1/SC 32. / Ed.: J. Melton. International Standard. ISO, ANSI, 2003.
3. Microsoft SQLXML Project (http://www.sqlxml.org/).
4. W3C DTD Specification // W3C Recommendations. W3C, 1998 (http://www.w3.org/XML/1998/06/xmlspec-report-v21.htm).
5. XML Path Language (XPath) Version 1.0 // W3C Recommendations. W3C, 1999 (http://www.w3.org/TR/xpath).
6. XML Schema Part 2: Datatypes // W3C Recommendations. W3C, 2001 (http://www.w3.org/TR/xmlschema-2/).
7. *Braganholo V., Davidson S., Heuser C.* On the updatability of XML Views over relational databases // Proceedings of WEBDB 2003. San Diego, 2003.
8. *Braganholo V., Davidson S., Heuser C.* Propagating XML View Updates to a relational database. Technical Report RP-341. Universidade Federal do Rio Grande do Sul, Instituto de Informatica. Porto Alegre, 2004.
9. *Braganholo V., Davidson S., Heuser C.* Reasoning about the updatability of XML Views over relational databases. Technical Report MS-CIS-03-13. University of Pennsylvania. Philadelphia, 2003.
10. *Carey M., Florescu D., Ives Z., Lu Y.* XPERANTO: Publishing object-relational data as XML // Proceedings of the 25th International Conference on Very Large Data Bases. Edinburgh: Morgan Kaufmann, 1999.
11. *Date C., McGoveran D.* Updating Views (6 Parts) // Database Programming & Design. 6 – 8. San Mateo: Miller Freeman Publications, 1994.
12. *Florescu D., Kossmann D.* Storing and querying XML data using an RDMBS // IEEE Data Engineering Bulletin 22. IEEE Computer Society, 1999.
13. *Lee D., Mani M., Chiu F., Chiu W.* Nesting-based Relational-to-XML schema translation // International Workshop on the Web and Databases (WebDB). Santa Barbara: ACM SIGMOD, 2001.
14. *Lee D., Mani M., Chiu F., Chiu W.* NeT & CoT: Translating relational schemas to XML schemas using semantic constraints. Technical report. UCLA Computer Science Dept. University of California. Los Angeles, 2002.
15. *Lee D., Chiu W.* Schema conversion methods between XML and relational models. University of California. Los Angeles, 2002.
16. *Malcolm G.* Programming Microsoft SQL Server 2000 with XML. Redmond: Microsoft Press, 2002.
17. *Mani M., Lee D.* XML to relational conversion using theory of regular tree grammars. University of California. Los Angeles, 2002.
18. *Murata M., Lee D., Mani M.* Taxonomy of XML schema languages using formal language theory. University of California. Los Angeles, 2002.
19. *Shanmugasundaram J., Kiernan J., Fan C., Funderburk J.* XPERANTO: Querying XML Views of relational data. IBM Almaden Research Center. San Jose, 2001.
20. *Новак Л.Г., Кузнецов С.Д.* Свойства данных XML // Труды ИСП РАН. **4**. М.: ИСП РАН, 2003.
21. Dynamic HTML and XML: The XMLHttpRequest Object. Apple Computer, Inc., 2004 (http://developer.apple.com/-internet/webcontent/xmlhttpreq.html).

Поступила в редакцию
12.05.2006