Построение баз данных взаимосвязанных документов

А. Н. Белова, А. В. Соловьев

Аннотация. Статья посвящена проблеме построения баз данных взаимосвязанных документов с контролем целостности связей документов между собой. В данной статье предлагается возможное решение проблемы построения БД взаимосвязанных структурированных документов, созданных на основе формата XML, при хранении их в реляционных СУБД.

Ключевые слова: *электронный документ, СУБД, информационная система, язык запросов.*

Введение

В этой статье рассматривается проблема представления конечному пользователю связанных данных при работе с базой данных документов, данные в которых записаны в соответствии с W3C XML Schema language [6].

Под документом мы будем понимать структурированный текст как совокупность взаимосвязанных между собой частей (блоков) документа [2]. Документ можно условно разделить на неизменную часть (форма) и переменную информацию (содержание, т. е. заполнение документа переменными данными, которые в наборе одинаковых по форме документов различаются).

Теория электронного документа, разработанная в ИСА РАН в 80-е годы [9, 10, 11, 12], находит в настоящее время свое подтверждение в развитии программных средств работы с электронными документами. Особенно бурное развитие электронных документов произошло в связи с разработкой стандарта XML (*eXtensible Markup Language*). В настоящее время все крупные разработчики СУБД (Oracle, MS SQL Server, IBM DB2) включили поддержку работы с XML [13], не только выделив его в отдельный тип данных, но и добавив возможность работы с содержимым XML-данных на основе навигационных языков запросов (XQuery, XPath) [6, 14].

Поэтому в настоящее время можно говорить о переходе от теоретических изысканий к практике, т. е. о повороте электронных средств обработки данных, в том числе и реляционных СУБД, в сторону работы с электронными документами, что с появлением стандартов описания электронных документов (XML) стало намного проще.

Схема XML-документа (XSD) определяет его структуру, порядок элементов, правила, которым он должен соответствовать. База данных, с которой будет работать пользователь, есть набор документов, соответствующих совокупности нескольких таких схем, которые могут быть связаны между собой.

Исходя из определения документа, любой из них может быть представлен иерархией, реже — графом (сетью). Если в документе содержатся перекрестные ссылки, то он представляет собой сеть, иначе — граф. Архитектура иерархических структур данных в общем случае определяется как дерево с бесконечным числом уровней и бесконечным числом потомков на каждом из таких уровней [1]. Это позволяет представить документ практически любой структуры и грамотно отразить практически любые связи между его частями.

Такой архитектуре соответствует формат XML, в настоящее время стандартизованный Консорциумом Всемирной Паутины (англ. World Wide Web Consortium, W3C) [6] как достаточно универсальный формат описания данных. Мы используем этот формат и для создания серии однотипных документов по заранее определенной нами схеме.

Пусть у нас уже есть готовая база данных документов XML, созданная описанным выше способом. Каждый из ее документов может быть соединен с каким-либо другим документом из этой же базы или с самим собой, то есть структура базы может быть сколь угодно сложной (граф с циклами). Вообще, в любой момент работы с базой пользователю может понадобиться любая ее часть. Эта часть может быть цельным документом, может быть объединением нескольких атрибутов разных документов, объединением документов, просто одиночным атрибутом и др., что зависит от потребностей самого пользователя в информации. Это значит, что необходимо обеспечить правильный выбор запрашиваемых данных из базы, избежать их потерь, корректно

отобразить на экране при просмотре, обеспечить надежное хранение с защитой от несанкционированного доступа (НСД). Эти задачи будут основными в нашей работе. Средствами формата XML их решить невозможно, поэтому нужно искать альтернативное решение.

1. Выбор решения

Основным признаком, по которому различаются СУБД, является тип структуры данных, который они поддерживают. Основу рынка промышленных СУБД составляют Реляционные СУБД, кратко — РСУБД. Это неслучайно. В реляционной модели данные разбиваются на наборы, составляющие табличную структуру [1]. Эта структура таблиц состоит из индивидуальных элементов данных, называемых полями. Табличная структура довольно стандартна и универсальна для восприятия информации, и ее понимания. Плата за все эти плюсы, а также за удобство ее использования — огромное количество атомарных данных, сведенное в большое число таблиц, которые в свою очередь связаны множеством ссылок. Такой способ хранения удобен для выборки конкретных данных из БД, но не для наглядного представления документа как связанного целого. При необходимости через запрос можно выбрать практически любую комбинацию конкретных данных, соответственно довольно просто получить нужную выборку.

Расширениями реляционной модели стали объектно-ориентированные (ОО), объектно-реляционные (ОР) и постреляционные СУБД, в основе которых как правило также табличное представление информации.

Остальные модели данных: иерархические, сетевые, файловые, документные в настоящее время не доведены до такого уровня как РСУБД или ОРСУБД. Во всяком случае, по распространенности на рынке решений из данных моделей нет явных конкурентов РСУБД и их объектных реализаций. По производительности также РСУБД являются лидерами (см. [15]).

Исходя из вышесказанного, самое очевидное и приемлемое решение — это перенос базы данных документов в среду промышленной РСУБД, средствами которой возможно решить обозначенные выше проблемы, а также постараться оптимизировать временные и ресурсные затраты на процесс решения поставленной задачи. Таким образом, нам нужна СУБД, с которой будет несложно работать и, одновременно с этим, она должна обладать достаточно универсальным функционалом и обеспечивать надежное хранение и защиту данных от НСД.

2. Дополнительные понятия и определения

Для дальнейшей иллюстрации решения, нам необходимо расширить понятийный аппарат введением еще одного важного понятия. Это понятие **отчета**. Под отчетом будем понимать объект базы данных, который предназначен для вывода информации из БД, прежде всего на печать [2]. В нашей работе отчет будет представлять собой результат запроса к базе данных, и будет представлен в виде отдельного объекта-таблицы, предназначенной для просмотра ответа на запрос. Сама эта таблица будет временным (виртуальным) объектом БД, который впоследствии храниться в БД не будет. Отчеты позволяют выбрать из баз данных нужную пользователю информацию, оформить ее в виде документа, перед выводом на печать просмотреть на экране.

Каждый отчет, как и документ, состоит из формы и содержания. Если к базе посылается какой-либо запрос, то ответом на запрос будет отчет. Данные, извлекаемые из БД и имеющие переменные значение в зависимости от экземпляра объекта БД, мы назовем **содержанием** документа. **Формой** отчета назовем состав и структуру семантических блоков документа-отчета (фрагменты, выделенные по смысловому содержанию), все постоянные тексты (заголовки как общие, так и промежуточные и уровневые), опорные линии, шрифты, постоянные тэги и символы разметки.

3. Практический пример

Ниже приведен пример схемы документа, в котором содержится ссылка на схему документа другого типа:

```
<?xml version="1.0" encoding="Windows-1251" ?>

// Пример описания схемы документа «Клиент банка» с помощью XML

- <FORM_DESCRIPTION
xmlns="http://www.cognitive.ru/XML/2001/ctform"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.cognitive.ru/XML/2001/ctform ctform.xsd"
name="Клиент">
    // тэг, означающий начало описания схемы (модели содержания) документа
    // в схеме определена экранная форма для пользователя
    // указаны вершины схемы, в которых предполагаются приложенные файлы
    // указана ключевая вершина схемы документа
    - <CONTENT_MODEL use="ФормаКлиента" attachment="|СвязанныеДокументы" key="Код_клиента">
```

```
// реквизиты клиента с указанием типа, индексируемости,
<E name="Код_клиента" datatype="integer" index="true" displayname="Код клиента" />
<E name="НазваниеИлиФИО" datatype="text" index="true" />
...
// пример агрегатной вершины «Связанные документы» (файлы)
- <A name="СвязанныеДокументы" use="table">
- <S name="СвязанныйДокумент">
<E name="Документ" datatype="file" index="true" />
...
// пример ссылки на другую схему связанного документа
// связь с другим документом по реквизиту «Код агента»
<E name="Код_агента" datatype="integer" index="true" displayname="Код агента"
// пример реализации: ссылка на схему документа «Агент клиента банка»
link="Агент" />
...
</CONTENT_MODEL>
</FORM_DESCRIPTION>\
```

Рассмотрим данную XML-схему. Она состоит из элементов и атрибутов этих элементов. В качестве данных, с которыми мы будем работать, могут выступать как сами элементы, так и эти атрибуты [4]. У некоторых элементов возникают дочерние элементы, уже со своими атрибутами [5]. В общем случае, количество дочерних элементов и глубина вложенности неограниченны. Первое, на чем мы остановимся, это преобразование структуры из иерархической XML в реляционную БД. После преобразования и переноса этой схемы в другую СУБД, мы должны получить одну таблицу или набор таблиц. При этом мы должны не только сохранить, но и иметь возможность восстановить путь между любыми двумя вершинами средствами СУБД. Это будет необходимо при разрешении ссылок и вообще выборок при запросах к базе. В целом, мы хотим построить удобную схему для более быстрого и легкого заполнения базы данных и работы с ней.

Пусть каждая вершина дерева имеет свой идентификатор. Для восстановления пути между двумя различными вершинами дерева, достаточно знать идентификаторы самих вершин и идентификаторы их родителей. Теперь составим план разбора данной схемы:

1. Выделяем в схеме уровни по глубине вложенности. То есть, чем больше у вершины предков, тем выше степень вложенности, соответственно, тем больше уровень.
2. Заводим отдельную таблицу для каждого уровня.
3. Заполняем каждую из таблиц. При этом, двигаясь сверху вниз по документу, заполняем имена тегов, а слева направо — атрибуты текущего тега.
4. Учитываем, что если в каком-то теге появился дополнительный атрибут, то в таблице появится дополнительная запись.

Для получения каждой вершиной своего идентификатора, заводим отдельную запись, в которой генерируем стандартный идентификатор GUID. Также для поддержания ссылочной ценности и возможности быстрой выборки данных, мы будем использовать составной первичный ключ. Чем выше уровень вложенности, тем первичный ключ длиннее. Предположим, что мы обрабатываем первый уровень. Тогда выделяем одну запись под идентификатор вершины на этом уровне, пока мы находимся на этой глубине, просто присваиваем каждой новой вершине свой идентификатор. Если в какой-то момент мы спустились на уровень ниже текущего, то мы начинаем заполнять таблицу следующего уровня вложенности, для которой генерируем свой первичный ключ. К каждому идентификатору родительского первичного ключа добавляем новый идентификатор ID, показывающий, какое место эта вершина занимает на этом уровне вложенности. То есть с увеличением уровня вложенности на единицу, длина первичного ключа увеличивается на эту же единицу, а к таблице просто добавляется еще одна запись. Также нам понадобится одна главная таблица для хранения списка названий различных схем и одна таблица для идентификаторов вершин, между которыми заключена ссылка.

Вполне возможно было бы сохранить полностью весь документ в одной таблице, здесь нет существенной разницы с предлагаемым вариантом. Но наше представление чуть более наглядно и привычно для конечных пользователей.

Вот как будет выглядеть после всех преобразований схема, которую мы рассматривали:

В XML-схеме документа 5 уровней вложенности, соответственно, в итоге в базе будут храниться 5 таблиц преобразованной иерархической XML-схемы к табличному виду (уровни нумеруются с 1 — верхний — заголовок документа, до 5 — листы дерева иерархии) (рис. 1).

На следующем этапе обработки исходной базы, мы будем заполнять базу данными. Для этого для каждой записи таблицы каждого уровня заведем дочернюю таблицу, в которой постепенно будут накапливаться данные из разных документов, соответствующих одной схеме. При этом отметим, что не в каждом документе задан каждый атрибут и элемент, который есть в начальной схеме, поэтому соответствующие таким «пробелам» поля просто оставим пустыми. Если на этом этапе получится так, что пустыми окажутся данные, связанные между собой

Уровень 1:

КЛИЕНТ0 : таблица

	UID	TAGNAME	ID_0	ATTR_XMLNS	ATTR_XMLNS_	ATTR_XSI_SCH	ATTR_NAME
+	b0e92534-8189-	XML	0				
+	b0ad6edf-f0cd-4	FORM_DESCR	1	http://www.cogr	http://www.w3.c	http://www.cogr	Клиент

Уровень2:

КЛИЕНТ1 : таблица

	UID	TAGNAME	ID_0	ID_1	ATTR_USE	ATTR_ATTACH	ATTR_KEY
+	dd8ab835-4f7e-	CONTENT_MO	1	0	ФормаКлиента	СвязанныеДок	Код_клиента
+	b4f24a77-e910-	PLUGINS	1	1			

Уровень 3:

КЛИЕНТ2 : таблица

	UID	TAGNAME	ID_0	ID_1	ID_2	ATTR_NAME	ATTR_DATATY	ATTR_INDEX	ATTR_DISPLA	ATTR_USE	ATTR_PROGID	ATTR_INITTIME	ATTR_INITSTRI
+	a0bf6400-3ffa-4	E	1	0	0	Код_клиента	integer	true	Код клиента				
+	04f1ecef-5b98-4	E	1	0	1	НазваниеИлиФ	text	true					
+	0705e43e-8447-	E	1	0	2	ТипДокумента	dictionary(ТипД	true					
+	d6226077-c67f-	E	1	0	3	Серия	text	true					
+	7dde165b-8772-	E	1	0	4	Номер	text	true					
+	f9a60f6a-77c8-4	E	1	0	5	ДопСведения	text	true					
+	577d6938-93f8-	A	1	0	6	СвязанныеДок				table			
+	ffed8841-61b0-4	A	1	0	7	Агенты				table			
+	5fae5d04-051b-	A	1	0	8	Каталоги				table			
+	02b5ae7e-bbaa-	E	1	0	9	Тип_клиента	dictionary(ТипК	true	Тип клиента				
+	9d0a166c-4cb0-	DICTIONARY	1	1	10	ТипДокумента					FrmpIgn.OnFile	once	stored:ТипДоку
+	f2ab660b-8c54-	DICTIONARY	1	1	11	Агенты					Frmtools.OnDB	once	collection:Агент
+	0b8e8477-caf5-	DICTIONARY	1	1	12	ТипКлиента					FrmpIgn.OnFile	once	stored:ТипКлие

Уровень 4:

КЛИЕНТ3 : таблица

	UID	TAGNAME	ID_0	ID_1	ID_2	ID_3	ATTR_NAME
+	9a1caaa1-521c	S	1	0	6	0	СвязанныйДок
+	191a2e5b-9f51-	S	1	0	7	1	Агент
+	fbf4bd29-3bec-4	S	1	0	8	2	Каталог

Уровень 5:

КЛИЕНТ4 : таблица

	UID	TAGNAME	ID_0	ID_1	ID_2	ID_3	ID_4	ATTR_NAME	ATTR_DATATY	ATTR_INDEX	ATTR_DISPLA	ATTR_LINK	ATTR_EDIT
	37a9dcaf-3f02-4	E	1	0	6	0	0	Документ	file	true			
	971d8868-59d0-	E	1	0	7	1	1	Код_агента	integer	true	Код агента	Агент	textbox,tool(LIN
	51727add-e526-	E	1	0	7	1	2	СведенияОбАг	text	true			
	6616bdbd-5a31-	E	1	0	8	2	3	Название	url	true			

Рис. 1

из разных документов, например, есть ссылка, но нет адресата ссылки или есть адресат, но нет адресанта, то сообщим об ошибке также средствами СУБД.

4. Поддержка ссылок в реляционной СУБД при отображении иерархических схем на реляционные

Перевести иерархию XML-схемы (или XML-документа) к табличному виду не составило большого труда. Однако в нашем примере в документе содержится ссылка на другой документ, таких ссылок может быть много, тем самым наша дерево-иерархия превращается в сеть, что уже может отображаться в реляционной модели неоднозначно, или, по крайней мере, приводить к зацикливанию функций преобразования графов в реляционные отношения. Чтобы избежать трудностей в функциях преобразования можно выделить обработку ссылок в отдельную функцию и в начале разбирать иерархию XML-схем, затем обрабатывать ссылки и строить связи между элементами иерархий. Как это можно сделать практически показано ниже.

Для начала, рассмотрим более подробно поддержку ссылок[1] между данными в РСУБД [3]. В качестве примера базы данных, содержащей ссылки между отдельными объектами хранения — документами, возьмем книжный каталог. Каждой вершине дерева иерархии конкретного документа соответствует свой уникальный идентификатор «ID», естественно, как

[1] Мы намеренно используем термин *ссылка* вместо термина *связь* (Link, Constraint), чтобы подчеркнуть независимость от платформы реализации сетевых (графовых) и «документных» структур данных.

корневому элементу, так и всем остальным. Для определения наличия или отсутствия ссылки в каждой конкретной вершине введем дополнительное поле с названием «ID_LINK».

Рассмотрим, как же должно быть устроено наше новое поле. При описании ссылки, нам нужно точно знать о наличие ссылки и типе связи, который соответствует ссылке. Если ссылка отсутствует, то значение поля «ID_LINK» будет, например, равно «0». Тогда при запросе мы сможем двигаться дальше по данным. Если же это значение отлично от «0», то договоримся присваивать каждому типу связи свое значение.

В реляционных базах данных встречаются 3 типа связи между сущностями [1]: 1 : 1 (один-к-одному), 1 : M (один-ко-многим), M : M (многие-ко-многим). Эти связи по своей природе как раз характеризуют ссылки в документах исходной базы.

Связь M : M встречается довольно редко. Пусть, например, есть город, в котором работает какое-то количество предприятий. У одного предприятия может быть множество сотрудников, но и какой-либо сотрудник может работать на нескольких предприятиях. В этом случае мы можем заменить эту связь двумя связями 1 : M с использованием промежуточной таблицы хранения идентификаторов. В одной таблице будут храниться все предприятия, в другой все сотрудники, а в промежуточной только пара идентификаторов, отражающая связи между предприятиями и сотрудниками.

Связь 1 : 1 будет служить, например, как заменитель какого-либо атрибута XML-документа или сущности, которая часто встречается в базе. Такой атрибут достаточно один раз присвоить, а в дальнейшем использовать. Такому типу связи присвоим идентификатор «1».

Связи 1 : M. Например, в оглавлениях книг из каталога часто используется связь 1 : M, чтобы показать, что содержание данной книги состоит из соответствующих глав. Идентификатором этой связи назначим «2».

После того, как мы установили тип связи, необходимо узнать, из какого документа брать данные. Для этого нужен идентификатор документа. За значение этого идентификатора примем значение идентификатора корневого элемента документа (**IDENTIFIER_LINK),** идентификатор объекта данных документа будем обозначать полем **IDENTIFIER_DATA**. Итоговая структура таблицы нашего каталога для связи 1 : 1 будет иметь следующий вид:

```
CREATE TABLE «CATALOG» (
«ID» INTEGER NOT NULL PRIMARY KEY,
«NAME» VARCHAR(200) CHARACTER SET WIN1251 NOT NULL, // Имя ссылки
«PARENT_ID» INTEGER, // Идентификатор родителя ссылки
«ID_LINK» INTEGER, // Идентификатор, отвечающий за ссылку
«IDENTIFIER_LINK» INTEGER, // идентификатор документа
«IDENTIFIER_DATA» INTEGER // идентификатор связанного данного
);
```

В какой-либо момент работы с базой данных может возникнуть необходимость просмотреть весь путь от корня документа до его листа. В этом случае его будет легко восстановить по идентификаторам. Даже если на этом пути встретится ссылка, то идентификаторы данных, которые она содержит, нам будут известны.

Теперь рассмотрим задачу выборки в новой базе данных. Пусть у нас есть уже построенный каталог и нужно просмотреть все связи 1 : 1 в нем. Это можно сделать с помощью оператора выборки SELECT. Код выборки в этом случае будет выглядеть так:

```
SELECT* // выбираем идентификатор каталога и имя
CATALOG.ID, CALALOG.NAME, CATALOG. IDENTIFIER_LINK,
(SELECT CATALOG.MANE // а также идентификатор и имя каталога,
FROM CATALOG на который мы ссылаемся
WHERE CATALOG.ID = «IDENTIFIER_LINK»;
)
FROM CATALOG
WHERE ID_LINK = 1; // если этот документ связан с другим посредством связи 1 : 1
```

Если нужно оперировать данными, в которых участвуют ссылки другого типа, то код будет выглядеть немного по-другому. В запросе придется явно перечислить значения нужных полей. Отметим, что выборки, аналогичные SELECT пригодны как для внутридокументных, так и междокументных ссылок. Так как в поле «IDENTIFIER_LINK» мы поместили идентификатор документа, то если ссылка направлена на сам документ, мы просто выберем необходимые значения из того же самого документа.

Если в процессе работы в базе данных нужно что-либо изменить и при изменении будет изменена также ссылка между данными документа, то возможны варианты. Первый — связь просто разрывается, ссылку нужно удалить. То есть в кортеже таблицы CATALOG, соответствующему адресату данной ссылки, поле ID_LINK нужно присвоить равным нулю. Второй — связь меняется. Например, клиент будет обслуживаться другим агентом. Тогда нужно поменять идентификаторы ссылки в измененных вершинах — адресате и адресанте ссылки и удалить старые. Ниже приведен пример триггера, который осуществляет такую замену:

```
CREATE TRIGGER UP BEFORE UPDATE ON CATALOG
FOR EACH ROW
    BEGIN
        INSERT INTO «CATALOG» SET CATALOG.ID = NEW.ID,
        CATALOG.NAME = NEW.NAME;
        CATALOG.PARENT_ID = NEW. PARENT_ID,
        CATALOG.IDENTIFIER_LINK = NEW.IDENTIFIER_LINK,
        CATALOG.IDENTIFIER_DATA = NEW.IDENTIFIER_DATA,
    END;
```

Из примера понятно, что при изменении новыми станут все поля, кроме «ID_LINK». Все остальные операции с базой будут аналогичны операциям с обычной реляционной базой данных.

Преимущество предлагаемого подхода заключается в его универсальности, однако, рассматривается довольно общий случай. При реализации для конкретной задачи потребуется и конкретизация решения, уточнение и оптимизация самой структуры и, соответственно, подходов к отображению данных. Возможно, что решение будет проще, чем в приведенном в статье примере, так и сложнее.

Выводы

Проведя данную работу, мы нашли решение задачи хранения документов, описанных с помощью XSD-схемы и представленных в формате XML, в реляционной БД, а также задачи представления связанных данных и документов конечному пользователю.

В качестве основных результатов исследования можно перечислить следующие:

1. Разработана методология отображения документов формата XML в реляционную БД с сохранением ссылок между объектами (данными) хранения документа.
2. Разработан метод решения задачи хранения и обработки связанных (ссылками) документов, в том числе применительно к большим документоориентированным информационным системам, в которых схемы данных могут быть сложно организованы.
3. Разработанный метод применен на практике в виде программной реализации.

Литература

1. *Дейт К. Дж.*. Введение в системы баз данных. М.: Вильямс, 2005.
2. *Соловьев А. В.* Разработка методов и средств взаимодействия объектно-ориентированных систем управления базами данных с электронными издательскими комплексами. М.: URSS, 2000. 28 с.
3. *Кузьменко Д.* Древовидные (иерархические) структуры данных в реляционных базах данных. (Электронная публикация). http://www.ibase.ru/devinfo/treedb.htm
4. Описание расширения .xsd. (Электронная публикация). http://open-file.ru/types/xsd
5. Основы XML. Учебный курс. http://www.intuit.ru/department/internet/xml/class/free/status/
6. The World Wide Web Consortium (W3C). XML Technologies including XML, XML Namespaces, XML Schema, XSLT, Efficient XML Interchange (EXI), and other related standards. http://www.w3.org/standards/xml/
7. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы. Построение и анализ. 2-е изд. М.: Вильямс, 2005. 452 с.
8. *Лапис Дж*. Совместимы ли XML и реляционные базы данных? http://khpi-iip.mipk.kharkiv.edu/library/extent/dbms/viper/vip2.html
9. *Емельянов Н. Е. и др.* Система НИКА // Системы управления базами данных и знаний. М., 1991.
10. *Емельянов Н. Е.* Теоретический анализ документного интерфейса: Препринт. М.: Всесоюзный научно-исследовательский институт системных исследований, 1987.
11. *Емельянов Н. Е.* Виды представления структурированных данных // Теоретические основы информационной технологии. Сборник трудов. Вып. 22. М.: ВНИИСИ, 1988.
12. *Богачева А. Н., Емельянов Н. Е., Романов А. П.* Генерация информационных систем по формам входных и выходных документов // PC Magazine. № 1. 1993.
13. *Арлазаров В. Л., Емельянов Н. Е.* Революция 2005 года в реляционных базах данных / Труды Института системного анализа РАН (ИСА РАН) Т. 45. «Технология программирования и хранения данных» М.: Ленанд/URSS, 2009. С. 9–18.
14. *Долгоруков А. Ю.*, *Ерохин В. И.* Язык запросов хранилища документов, построенного на НИКА-технологии / Труды Института системного анализа РАН (ИСА РАН) Т. 58. «Обработка изображений и анализ данных» М.: Ленанд/URSS, 2010. С. 53–67.
15. TPC-H Top Ten Performance Results Version 2 Results . http://www.tpc.org/tpch/results/tpch_perf_results.asp

Белова Анна Николаевна. Студентка 4 курса НИТУ «МИСиС». Область научных интересов: системы управления базами данных. E-mail: anna.belova7@gmail.com

Соловьев Александр Владимирович. Ведущий научный сотрудник ИСА РАН, к. т. н. Окончил МГТУ им. Н. Э. Баумана в 1994 г. Количество печатных трудов: 27. Область научных интересов: системный анализ, системы управления базами данных, теория надежности, влияние человеческого фактора, математическое моделирование, электронный документооборот. E-mail: alexsol@cs.isa.ru