УДК 004.98, 004.652.4

ПОСТРОЕНИЕ ХРАНИЛИЩ ОНТОЛОГИЧЕСКИХ БАЗ ЗНАНИЙ

М.Н. Вехорев; М.Г. Пантелеев, к.т.н.
(Санкт-Петербургский государственный электротехнический университет (ЛЭТИ), mvehorev@gmail.com, mpanteleyev@gmail.com)

Описывается проблема хранения онтологически представленной информации в реляционных БД. Хранилище онтологий рассматривается как особый класс информационных систем. Классифицированы существующие подходы к хранению онтологий в БД. Определены ключевые факторы, влияющие на эффективность хранилищ онтологий, а также возможные направления дальнейших исследований в области организации хранилищ онтологической информации.

Ключевые слова: *онтологии, организация хранилищ онтологий, реляционная СУБД, технология хранения RDF-триплетов, хранилище онтологий, язык запросов семантического Веб, движок запросов RDF.*

Концепция и технологии семантического Веб – это динамично развивающиеся направления информационных технологий [1], которые обеспечивают возможность совместного многократного использования знаний различными приложениями, организациями и сообществами, позволяя компьютерам обрабатывать информацию на семантическом уровне.

Ключевым элементом *информационных систем* (ИС), основанных на технологиях семантического Веб, являются онтологические базы знаний. При создании крупных информационных систем по мере роста объемов используемых онтологий их хранение в плоских файлах оказывается непродуктивным. В связи с этим актуальна проблема организации эффективных хранилищ онтологических баз знаний (*RDF*-хранилищ).

Представление онтологий в семантическом Веб

Под онтологией, согласно общепринятому определению, понимается *явная спецификация концептуализации*.

Для автоматической обработки разделяемых знаний консорциумом *W3C* разработаны единые стандарты их представления: *RDF* (*Resource Description Framework*) и основанный на нем язык веб-онтологий *OWL* (*Ontology Web Language*). Под ресурсом понимается любая сущность, которой сопоставлен универсальный идентификатор *URI* (*Universal Resource Identificator*). Основной конструкцией языка *RDF* является утверждение, задаваемое тройкой <субъект> <предикат> <объект> (*RDF*-триплет), например: <стол> <цвет> <черный>. Использование *URI* для задания субъектов и свойств позволяет связывать отдельные утверждения (*RDF*-триплеты) в сколь угодно сложные семантические сети, имеющие единую интерпретацию в открытой сетевой среде.

Простейшая форма хранения онтологий – *OWL*-файл. При чтении такого файла в оперативной памяти создается модель (набор утверждений), с которой выполняется дальнейшая работа. Однако данный подход имеет недостатки: существенный рост затрат оперативной памяти при работе с большими онтологиями (более 10^6 триплетов) вследствие полной загрузки *OWL*-файла, а также значительное увеличение времени загрузки *OWL*-файлов по мере роста количества используемых онтологий. Это не позволяет использовать данный подход при создании крупных ИС и обусловливает необходимость построения *RDF*-хранилищ на основе реляционных СУБД.

Особенности RDF-хранилищ

Под *RDF*-хранилищем понимается информационная подсистема, предназначенная для хранения *RDF*-триплетов и выполнения запросов к ним.

Основными функциями *RDF*-хранилища являются:

- организация хранения онтологий в реляционной БД с использованием реляционных представлений;
- предоставление программного интерфейса для извлечения информации из хранимых онтологий посредством языка структурированных запросов *SPARQL* или специального *API*;
- поддержка функций администрирования хранимых онтологий: добавление, удаление, модификация и распределение прав доступа.

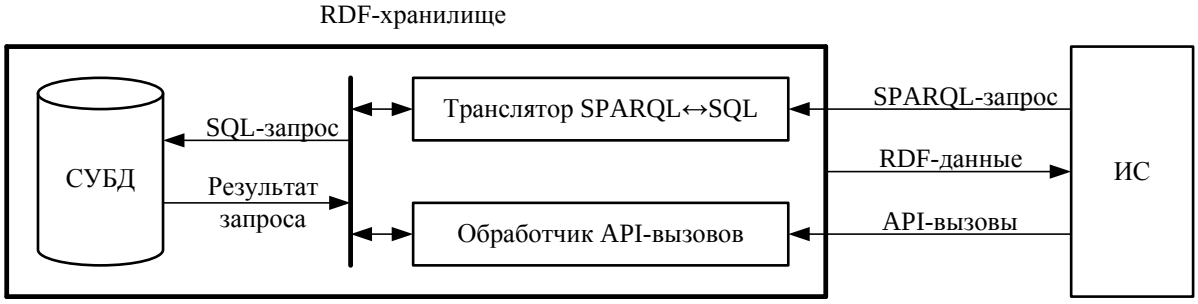
Эффективное *RDF*-хранилище должно удовлетворять следующим требованиям:

- высокая производительность – минимизация времени выполнения запросов;
- минимальные затраты памяти (дискового пространства) для хранения онтологий;
- универсальность подхода – возможность хранения онтологий любой структуры.

При разработке конкретных ИС важны время и сложность развертывания *RDF*-хранилища, а также его стоимость.

Обобщенная архитектура RDF-хранилища

В состав *RDF*-хранилища входят две основные подсистемы: подсистема хранения онтологий на



Архитектура RDF-хранилища

основе реляционной СУБД и подсистема трансляции входных запросов в *SQL*-запросы (см. рис.).

В качестве подсистемы хранения онтологий могут использоваться как коммерческие СУБД (*Oracle, MS SQL Server* и др.), так и свободно доступные (*PostgreSQL, MySQL* и др.).

Прикладные информационные системы могут взаимодействовать с *RDF*-хранилищем посредством специального *API* или структурированных *SPARQL*-запросов. *SPARQL* – это стандартный язык запросов к *RDF*-данным. Например, простой *SPARQL*-запрос для получения имени и номера паспорта некоторой персоны имеет вид:

```
SELECT ?name ?number
WHERE{ ?someone rdf:type :Person.
       ?someone :name ?name.
       ?someone :passportNumber ?number.}
```

Отношение **rdf:type** является стандартным отношением языка *RDF* и означает принадлежность элемента классу. Класс :**Person**, отношения **:name** и **:passportNumber** в данном примере взяты из некоторого пространства имен по умолчанию.

Подсистема трансляции входных запросов в общем случае может включать два компонента:

- обработчик *API*-вызовов, предоставляющий библиотеку классов некоторого языка программирования для работы с онтологиями, включая их загрузку из хранилища;
- транслятор *SPARQL↔SQL*, выполняющий преобразования *SPARQL*-запросов в *SQL*-запросы и обратные преобразования результатов, возвращенных *SQL*-запросами в результат *SPARQL*-запроса.

Недостаток специального *API* состоит в привязке к конкретному языку программирования. Пример такого *API* – библиотека *Jena*. Интерфейс *SPARQL*-запросов является универсальным решением, не зависящим от языка программирования. Поэтому рассмотрим работу с *RDF*-хранилищем с использованием именно этого интерфейса. Преобразования *SPARQL*-запросов в набор *SQL*-запросов будем обозначать *SPARQL→{SQL}*.

Очевидно, что формальная грамматика преобразования *SPARQL→{SQL}* зависит от принятой в *RDF*-хранилище схемы РБД. Вопросы оптимизации функционирования транслятора *SPARQL↔ ↔SQL* рассмотрены, в частности, в [2, 3]. Настоящая статья посвящена определению зависимости производительности *RDF*-хранилища от принятой схемы РБД.

Организация хранения онтологий в БД

Можно выделить два базовых подхода к организации хранения онтологий в РБД:

1) использование единственной таблицы для хранения всех *RDF*-триплетов (подход «вертикальная таблица»);

2) отображение иерархии онтологических сущностей (классов, свойств, экземпляров) в схему РБД.

В соответствии с первым подходом все *RDF*-триплеты хранятся в унифицированной таблице БД, содержащей в общем случае четыре колонки: «граф», «субъект», «объект» и «предикат». Данный подход реализован, в частности, в *Jena SDB* и *3store*. Он характеризуется достаточно высокой временной сложностью выборки *RDF*-триплетов.

Особенностью второго подхода является определение схемы БД в соответствии с конкретной предметной областью, что позволяет оптимизировать выполнение запросов. Реализация данного подхода для больших онтологий предполагает создание большого числа таблиц БД со сложными связями между ними. Известно несколько частных решений, отличающихся способом формирования схемы БД [4, 5].

Анализ факторов, влияющих на эффективность RDF-хранилищ

Эффективность *RDF*-хранилища при использовании *SPARQL*-запросов определяется двумя основными взаимосвязанными факторами: принятой схемой БД и грамматикой преобразования *SPARQL→{SQL}*.

Исходя из представленной на рисунке обобщенной архитектуры *RDF*-хранилища, время **Tsparql** обработки *SPARQL*-запросов в общем виде определяется формулой

$$\mathbf{Tsparql=Ttrans+Tsql\_seq}, \quad (1)$$

где **Ttrans** – время трансляции *SPARQL*-запроса в последовательность *SQL*-запросов в соответствии с принятой трансформационной грамматикой; **Tsql_seq** – время обработки СУБД построенной последовательности *SQL*-запросов.

Введем определения и примем базовые допущения, необходимые для анализа временных затрат на обработку *SPARQL*-запросов.

SQL-запрос в общем случае имеет следующую структуру:

SELECT *<кортеж выбираемых полей>* **FROM** *<кортеж таблиц>* **WHERE** *<кортеж условий>*.

Построение *SQL*-запроса на основе входного *SPARQL*-запроса предполагает формирование *кортежей выбираемых полей* и *условий*. Без существенной потери общности примем равными время формирования одного элемента *кортежа выбираемых полей* и одного элемента *кортежа условий* в запросе. Время формирования *кортежа таблиц* можно считать постоянным и пренебрежимо малым.

Пусть $\mathbf{T}'$ – среднее время формирования элементов *кортежа выбираемых полей* и *кортежа условий* в процессе построения *SQL*-запроса. Тогда время формирования *SQL*-запроса линейно зависит от числа элементов в *кортежах выбираемых полей* и *условий*.

Допустим, что обработка *SQL*-запроса требует последовательного просмотра всех записей таблицы и сравнения анализируемых условий для каждой из них. Тогда время выборки очередной записи из таблицы можно считать равным времени анализа одного условия.

Обозначим $\mathbf{T}''$ среднее время выборки записи из таблицы и анализа условия, необходимое для выполнения *SPARQL*-запроса к СУБД. Будем считать, что $\mathbf{T}''$ не зависит от конкретной СУБД и является постоянным. Можно также считать, что время анализа записей в таблице БД линейно зависит от количества записей таблицы, а время анализа условий линейно зависит от количества условий, проверяемых для каждой записи в таблице БД.

Зависимость времени трансляции *SPARQL*→*{SQL}* от принятой трансформационной грамматики оценивается формулой

$$\mathbf{Ttrans} = \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{Tsql\_def\,i}, \qquad (2)$$

где **N** – количество результирующих запросов *SQL*, получаемых в результате трансляции; **Tsql_def i** – время построения **i**-го *SQL*-запроса.

Зависимость времени формирования очередного *SQL*-запроса от количества полей, содержащихся в кортеже результата, и количества условий будет следующей:

$$\mathbf{Tsql\_def} = \mathbf{Nres} * \mathbf{Tres\_def} + \mathbf{+Ncond} * \mathbf{Tcond\_def}. \qquad (3)$$

Исходя из принятых допущений, можно записать:

$$\mathbf{T}' = \mathbf{Tres\_def} = \mathbf{Tcond\_def}. \qquad (4)$$

Подставив (3) и (4) в (2), получим

$$\mathbf{Ttrans} = \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{Tsql\_def\,i} = \mathbf{=} \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{(Nres\,i + Ncond\,i)} * \mathbf{T}' \qquad (5)$$

Аналогично зависимость времени обработки набора *SQL*-запросов может быть выражена формулой

$$\mathbf{Tsql\_seq} = \mathbf{k} \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{Tsql\,i}, \qquad (6)$$

где **k** – коэффициент схемы БД, численно равный количеству таблиц БД (*Ntbl*); **N** – количество результирующих *SQL*-запросов, построенных при трансляции; **Tsql i** – время выполнения **i**-го *SQL*-запроса к СУБД.

Время, необходимое на выполнение одного *SQL*-запроса, может быть оценено по формуле

$$\mathbf{Tsql} = \left(\mathbf{Trecord} + \mathbf{Tfield}\right), \qquad (7)$$

где **Trecord** – время на анализ записей в таблице; **Tfield** – время на сопоставление условий запроса с текущими значениями в строке таблицы.

Исходя из принятых допущений, можно записать:

$$\mathbf{Trecord} = \mathbf{Ntbl\_rec} * \mathbf{T}'', \qquad (8)$$

где **Ntbl_rec** – количество записей в таблице БД;

$$\mathbf{Tfield} = \mathbf{Ntbl\_fld} * \mathbf{T}'', \qquad (9)$$

где **Ntbl_fld** – количество проверяемых условий в *SQL*-запросе.

Подставив (7–9) в (6), получим:

$$\mathbf{Tsql\_seq} = \mathbf{k} \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{Tsql\,i} = \mathbf{= Ntbl} * \sum_{\mathbf{i=1}}^{\mathbf{N}} \left(\mathbf{Trecord} + \mathbf{Tfield}\right)\mathbf{i} = \mathbf{=\ Ntbl} * \sum_{\mathbf{i=1}}^{\mathbf{N}} \left(\mathbf{\ Ntbl\_rec} + \mathbf{Ntbl\_fld}\right)\mathbf{\ i} * \mathbf{T}'' \qquad (10)$$

Подставив (5) и (10) в (1), получим:

$$\mathbf{Tsparql} = \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{(Nres\,i + Ncond\,i)} * \mathbf{T}' + \mathbf{Ntbl} * * \sum_{\mathbf{i=1}}^{\mathbf{N}} \left(\mathbf{Ntbl\_rec} + \mathbf{Ntbl\_fld}\right)\mathbf{\ i} * \mathbf{T}''. \qquad (11)$$

Формула (11) выражает зависимость времени выполнения *SPARQL*-запроса от особенностей принятой схемы БД и соответствующей трансформационной грамматики.

Рассмотрим влияние схемы БД на трансформационную грамматику для двух вариантов схем – «вертикальная таблица» и «таблицы свойств классов». Схема БД «вертикальная таблица» предполагает создание единой таблицы для хранения *RDF*-триплетов, а схема БД «таблицы свойств

классов» – отдельных таблиц для хранения экземпляров каждого из классов онтологии.

Для сравнения этих вариантов рассмотрим простую онтологию, содержащую четыре класса – *classHuman*, *classMan*, *classWoman*, *classChildren*. Будем считать, что каждый из них имеет два экземпляра, за исключением родительского класса *classHuman*. Каждый экземпляр класса обладает свойствами *propertyName* и *propertyAge*.

Структура хранения данной онтологии на основе схемы «вертикальная таблица» представлена в таблице 1.

Таблица 1

table_Triples

Subject	Predicate	Object
classHuman	subClassOf	classThing
classMan	subClassOf	classHuman
classWoman	subClassOf	classHuman
classChildren	subClassOf	classMan
classChildren	subClassOf	classWoman
classHuman	hasProperty	propertyName
classHuman	hasProperty	propertyAge
instanceMan1	instanceOf	classMan
instanceMan2	instanceOf	classMan
instanceWoman1	instanceOf	classWoman
instanceWoman2	instanceOf	classWoman
instanceChild1	instanceOf	classChildren
instanceChild2	instanceOf	classChildren
instanceMan1	propertyName	Andry
instanceMan2	propertyName	Maxim
instanceWoman1	propertyName	Maria
instanceWoman2	propertyName	Julia
instanceChild1	propertyName	Michael
instanceChild2	propertyName	Regina
instanceMan1	propertyAge	35
instanceMan2	propertyAge	46
instanceWoman1	propertyAge	45
instanceWoman2	propertyAge	33
instanceChild1	propertyAge	16
instanceChild2	propertyAge	8

Реализация хранения данной онтологии на основе схемы «таблицы свойств классов» предполагает создание таблицы иерархии классов онтологии (табл. 2) и отдельных таблиц для каждого из классов (табл. 3–5).

Таблица 2

table_Hierarhy

ID_Hierarhy	subClass	Class	tableName
1	classHuman	classThing	
2	classMan	classHuman	table_classMan
3	classWoman	classHuman	table_classWoman
4	classChildren	classMan	table_classChildren
5	classChildren	classWoman	table_classChildren

Таблица 3

table_classMan

ID_Man	Name	Age
1	Andry	35
2	Maxim	46

Таблица 4

table_classWoman

ID_Woman	Name	Age
1	Maria	45
2	Julia	33

Таблица 5

table_classChildren

ID_children	Name	Age
1	Michael	16
2	Regina	8

Примеры преобразования SPARQL→{SQL}

Рассмотрим возможные варианты трансляции *SPARQL*-запросов к онтологии в набор *SQL*-запросов в зависимости от используемой схемы БД на конкретных примерах и оценим влияние принятой схемы БД на сложность преобразования *SPARQL→{SQL}*.

Пример 1.

SELECT ?x {WHERE ?x a classMan} // *Определение всех экземпляров класса*

Преобразование *SPARQL→{SQL}* для схемы БД «вертикальная таблица» включает следующие шаги.

1. Определение всех подклассов класса **classMan**:

SELECT result1(subject) FROM table_Triples
WHERE (predicate='subClassOf' AND object='classMan') OR (subject='classMan');

2. Выбор экземпляров классов, определенных на шаге 1:

SELECT result2(subject) FROM table_Triples
WHERE predicate='instanceOf' AND object=result1(subject);

3. Выбор значений свойств **propertyName** экземпляров, полученных на шаге 2:

SELECT result3(subject, object) FROM table_Triples
WHERE (subject= result2(subject)) AND (predicate='propertyName');

4. Выбор значений свойств **propertyAge** экземпляров, полученных на шаге 2:

SELECT result4(subject, object) FROM table_Triples
WHERE (subject= result2(subject)) AND (predicate='propertyAge');

5. Объединение результатов, полученных на шагах 3 и 4:

JOIN result3,result4 BY subject;

Аналогичное преобразование *SPARQL→{SQL}* для схемы БД «таблицы свойств классов» включает следующие шаги.

1. Выбор имен таблиц для класса **classMan** и его подклассов:

SELECT result1(tableName) FROM table_Hierarhy