Учебный курс «Делаем фактографию»

Марчук А.Г., д.ф.-м.н., профессор

## Введение

Под фактографической системой мы будем понимать систему фиксации (записи) фактов. Фактами в нашем случае будут сущности реального мира (люди, документы, организационные системы, географические системы) отражаемые через информационные поля или отношения между ними. Например, человек может быть описан датой рождения, организацией, в которой он работает, местом проживания, фотографиями, на которых он отражен. Мы почти полностью выстроим аналог построений Semantic Web [], позволяющих выполнять как структуризацию данных, так и специфицировать эти данные через онтологию.

Следующее относительно независимое построение – хранилище документов. В предлагаемом подходе документ, это файл (в общем случае – файлы), который содержит контент (содержимое) документа и его информационная сущность, как правило, проявляется через просмотр.

Таким образом, появляется задача формирования и использования информационной базы, состоящей из хранилища документов и базы данных. Под формированием мы будем понимать два возможных сценария: автоматическую переработку наборов файлов и второй, редактирование информационной базы. Использование информационной базы предполагается через просмотровые/поисковые интерфейсы и сервисы.

Все указанные элементы информационных и программных построений будут опробованы в простой фактографической системы. От «сложного», комплексного решения построенный вариант будет отличаться заметным упрощением моделей, особенностей программной реализации и интерфейсов.

Курс рассчитан на изучение ряда идей и технологий, ориентированных на создание современных информационных системы. Для студентов необходимо содержательное знание C#/LINQ/.NET/ASP.NET/XML/HTML и опыт их использования.

## Глава 1. База данных

### Тестовый набор данных Фототека

Для опробования разных подходов, нам потребуется простой генератор тестов, в котором по заданному параметру можно породить тестовую базу данных с некоторыми существенными для нас свойствами. Тест сделан намеренно максимально простым и легко воспроизводим в рамках практически любого решения. Тест сделан в парадигме связанных таблиц и приспособлен для максимально эффективной реализации на обычных SQL-платформах.

База данных состоит из трех таблиц: персон, фотографий, отражений. Параметр тестового набора - количество записей в таблице персон или просто количество персон npersons. Количество фотографий - npersons \* 2. Количество отражений npersons \* 6. Все таблицы имеют идентификационное поле id, являющееся первичным ключом для таблицы, все ключи в одной таблице различны и могут быть целочисленными или строковыми. Уникальность идентификаторов поддерживается или на уровне одной таблицы или для всех записей всех таблиц в зависимости от поставленных для экспериментов задач. Таблица персон persons содержит три поля: целочисленный идентификатор id, строковое значение имени name, числовое (возможно целое или длинное целое или вещественное, это зависит от целей тестирования) значение возраста age. Таблица фотографий photos состоит из полей id и name, таблица отражений reflections связывает записи о персонах с записями о фотографиях и содержит поля id, reflected, indoc. Причем два последних поля представляют внешние ссылки (external keys) на соответственно таблицу персон (отражаемое) и таблицу фотографий (отражение в документе).

Тестирование и измерение проводятся по небольшому, но принципиальному спектру вопросов. Это - время загрузки, отдельно или вместе - время вычисления индексов. Это - время выборки записей по задаваемому случайным образом ключу, это - время выборки записей по задаваемому имени или части имени. Это - время вычисления "информационного портрета" персоны. Под этим вычислением понимается получение всех фотографий, на которых отражена персона с случайно задаваемым первичным ключом. Времена измеряются в миллисекундах или в секундах, для больших значений. Для "быстрых" действий по выборкам, измерения производится на подходящем количестве испытаний и приводятся к миллисекундам на 1 тыс. испытаний.

Надо отметить, что тест "фототека" иногда используется в неполном виде. В качестве базы данных можно взять только одну таблицу, мы берем таблицу персон и это будет вполне осмысленная (тестовая) база данных. Эта база данных может рассматриваться как key-value хранилище, где ключом является идентификатор персоны, а значение - какой-то вариант полной записи об этой персоне.

### Основы RDF

RDF (Resource Description Framework) [] – способ структуризации данных, используемый в Semantic Web []. Существует несколько моделей RDF как системы структуризации данных. Чаще всего используется модель ориентированного графа. В этой модели имеются идентифицированные узлы, идентифицированные ребра и строковые константные значения. Исток ребра – всегда узел, сток (цель) ребра – может быть узел, может быть константой.

Изобразительные свойства такого построения достаточны для самых разных задач. Для иллюстрации, рассмотрим фрагмент сети, описывающий типовую ситуацию:

Овалы соответствуют узлам графа, прямоугольники – константам, подписи в овалах и надписи у дуг – их идентификаторы, константные значения изображены внутри прямоугольников. Суть структурирующего построения достаточно понятна из рисунка: есть идентифицированная персона Иванов 1988 года рождения. У Иванова есть отец и Иванов работает в организации с именем НГУ.

Рассмотрим разные виды представления таких графов. Первая – в виде триплетов:

В этом представлении, примитивами являются два вида троек «овал – стрелка – овал» и «овал – стрелка – прямоугольник». Это высказывания. Отдельные высказывания собираются в граф через склеивание одноименных овалов. Оба варианта называются триплетами (тройками) и слева направо их составляющие называются субъект, предикат, объект. В стандартизованном виде триплеты изображаются своими тремя частями, последовательно через пробел расположенными в виде строки:

<id1> <P1> <id2>. или

<id3> <P2> ”data”.

Еще есть XML-форма изображения RDF-графа. Суть ее в том, что множество высказываний разбивается на группы с одинаковым субъектом, а группы изображаются в виде записи специального формата, напр.:

<rdf:RDF>

<person rdf:about=“p3817”>

<name>Иванов Иван Иванович</name>

<birth>1988</birth>

<work-in rdf:resource=“o19302”/>

</person>

<org-sys rdf:about=“o19302”>

<name xml:lang=”ru”>НГУ</name>

<name xml:lang=”en”>NSU</name>

</org-sys>

</rdf:RDF>

Запись помещается в элемент с именем типа и идентификатором в атрибуте rdf:about. Поля записи – это элементы с именем предиката и либо константным значением в виде текста, либо ссылкой на объект через атрибут rdf:resource. База данных будет состоять из одного или нескольких RDF-файлов формата XML. Несколько файлов эквивалентны одному файлу с объединенным набором записей.

Такая форма похожа на традиционные построения средствами XML для реляционных баз данных. Действительно, имя записи соответствует имени таблицы, атрибуты rdf:about и ref:resource соответствуют первичному и внешнему ключу соответственно, имя поля соответствует имени колонки. Но RDF – шире, он позволяет 1) иметь элементы с отсутствующим типом; 2) иметь неопределенный набор полей; 3) иметь множество полей с одинаковым именем, но разными значениями, в частности, на разных языках.

### Задание 1. Построение тестового набора данных

Вот мы и добрались до создания кода. Желательно в этом месте создать решение (Solution) для Visual Studio, проекты в этом решении будут отдельными задачами, некоторые из которых будут использованы в других проектах. Первая задача – создание RDF-файла в формате XML, содержащего базу данных определенного параметром размера. Сделаем это в проекте GenerateRDF. Опуская детали, получающийся код довольно прост. Сначала мы формируем пустой набор записей:

// Загружаем основу

XElement dataset = XElement.Parse(

@"<?xml version='1.0' encoding='utf-8'?>

<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>

</rdf:RDF>

");

Этот фрагмент я сделал экзотически, через парсинг текста, потому что не умею внедрять пространство имен другим образом. Наверное, есть и более естественный способ типа:

XElement dataset = new XElement(“{http://www.w3.org/1999/02/22-rdf-syntax-ns#}RDF”, …);

Далее, мы наполняем этот элемент записями в соответствии с логикой построения «Фототека»:

Random rnd = new Random();

int npersons = 10;

int np = npersons;

int nf = npersons \* 2;

int nr = npersons \* 6;

dataset.Add(

Enumerable.Range(0, np)

.Select(i => new XElement("person", new XAttribute("{http://www.w3.org/1999/02/22-rdf-syntax-ns#}about", "p"+i),

new XElement("name", new XAttribute("{http://www.w3.org/XML/1998/namespace}lang", "ru"), "и"+i),

new XElement("age", "23"))));

dataset.Add(

Enumerable.Range(0, nf)

.Select(i => new XElement("photo", new XAttribute("{http://www.w3.org/1999/02/22-rdf-syntax-ns#}about", "f" + i),

new XElement("name", "dsp" + i))));

dataset.Add(

Enumerable.Range(0, nr)

.Select(i => new XElement("reflection", new XAttribute("{http://www.w3.org/1999/02/22-rdf-syntax-ns#}about", "r" + i),

new XElement("reflected", new XAttribute("{http://www.w3.org/1999/02/22-rdf-syntax-ns#}resource", "p" + rnd.Next(np))),

new XElement("indoc", new XAttribute("{http://www.w3.org/1999/02/22-rdf-syntax-ns#}resource", "f" + rnd.Next(nf))))));

Здесь стоит обратить внимание на внедрение русского спецификатора при задании имени персона (которое начинается с русской «и»). Финальный аккорд – сохранение полученного XML-элемента dataset в файле.

### Первый шаг на пути к базе данных

Созданный файл формата RDF обладает структурирующими свойствами. Кроме того, его можно легко загрузить в оперативную память и использовать для обработки. Если он будет модифицирован, то можно его также легко выгрузить. Посмотрим, как можно использовать эту структуру для нашей тестовой информационной системы. Для этого, надо разобраться в том, как и какие данные мы будем извлекать из будущей базы данных.

Первое требуемое существенное действие – поиск записи или записей по имени или части имени. Логично желать получить из базы данных все записи, совпадающие с заданным образцом имени, напр. задаем «Иванов», получаем всех Ивановых, имеющихся в базе данных. Что такое «получаем»? Получаем набор записей вида:

<person rdf:about="…">

<name xml:lang="ru">Иванов …</name>

…

</person>

У нас в данных Ивановых нет, но можно искать персоны, начинающиеся напр. на «и777». Напишем метод, находящий в наборе записей ту, имя которой похоже на образец:

// Поиск по образцу

Func<XElement, string, IEnumerable<XElement>> Search = (ds, sample) => ds.Elements()

.Where(r => r.Elements("name").Any(f => f.Value.StartsWith(sample)));

Другой способ получения информации из базы данных – выборка по значению идентификатора:

// Выборка записи по идентификатору

Func<XElement, string, XElement> GetById =

(ds, id) => ds.Elements().FirstOrDefault(r => r.Attribute("{http://www.w3.org/1999/02/22-rdf-syntax-ns#}about").Value == id);

На имеющихся тестовых данных можно легко испытать работу этих функций. Но все же это только пробы. Для реальной и даже модельной работы с фактографической системой, нужно создать базу данных. Сначала функциональность базы данных будет минимальной, в дальнейшем будем наращивать функциональность и применять более изощренные решения.

Глядя вперед, мы создадим проекты RDFEngine (библиотека), TesterConsole (консоль) и FactographyView (веб-приложение).

Сначала – движок. Мы определим минимальные требования к движку через интерфейс. Движок – это база данных для будущих приложений. База данных у нас будет формироваться на основе RDF-модели и предоставлять несколько методов.

public interface IEngine

{

public void Clear();

public void Load(IEnumerable<XElement> records);

public IEnumerable<XElement> Search(string searchstring);

public XElement GetRecord(string id, bool addinverse);

}

База данных очищается. Потом – загружается. Внутри загрузки она еще и строится. После этого, база данных готова выполнять поиск и выборку записей. Поясним операции на рисунке и в последующих текстовых конструкциях.

Записью называем узел, напр. p3817 имеющиеся у узла текстовые поля и исходящие объектные ссылки, напр. ссылка на o19302. На рисунке то, что входит в конкретную запись, условно очерчено черным контуром. Запись в промежуточном XML-представлении является XML-элементом с атрибутами id и type, значением которых являются идентификатор узла и тип узла (изображен желтым). Подэлементами записи являются текстовые поля и прямые ссылки. Расширенная запись включает в себя также обратные ссылки, напр. p1, p2. Запись p3817 в промежуточном представлении будет выглядеть:

<record id=’p3817’ type=’person’>

<field prop=’name’>Иванов</field>

<field prop=’birth’>1988</field>

<direct prop=’work-in’><record id=’o19302’/></direct>

<direct prop=’father’><record id=’p2317’/></direct>

<!—Следующие поля действительны для расширенного представления -->

<inverse prop=’p1’><record id=’id1’/></inverse>

<inverse prop=’p2’><record id=’id2’/></inverse>

</record>

Обратите внимание на оформление прямой и обратной ссылок через указания направления, свойства и вырожденной записи цели или источника. Такое оформление позволяет в дальнейшем, расширить запись до более развернутой древовидной композиции за счет расписывания этих ссылок.

Теперь создадим простой XML-движок, реализующий интерфейс IEngine. Движок выполним в виде класса XMLEngine, который конструируется на основе XML-представления RDF-документа. В текста класса XMLEngine есть понятные комментарии о том как класс устроен и реализован. Определенный нюанс имеется в словаре ссылочных элементов subelementsByResource. Это множество всех подэлементов записей, имеющих определенный rdf:resource, ключами выступают значения этого атрибута. Для чего такая структура? На самом деле, как XElement используются ссылки на элементы в структуре rdf (это общее свойство элементов классов при присваивании значений). У этих элементов есть имя, которое соответствует имени предиката и родительский элемент (свойство Parent), который является записью, содержащую данную ссылку. Напр. в записи

<person rdf:about="…">

<name xml:lang="ru">Иванов …</name>

<father rdf:resource="aaa123…" />

…

</person>

В словарь попадет ссылка на подэлемент father, причем попадет под ключом aaa123…

Для чего такая сложность? Так мы во вспомогательных структурах не будем порождать сложных элементов, а обойдемся ссылками.