

XML, Relational Databases, and Ruby on Rails

This paper will explore the relationship between XML and relational databases with a focus on XML interaction with the relational database Ruby on Rails (Rails). First a general discussion of the intersection of XML and databases will be presented as well as descriptions of the different way they structure data. An overview of Ruby on Rails will be given followed by a section on how a Rails application interacts with XML. Both the exporting of database information in an XML document as well as importing data in the XML format to a Rails database will be discussed. The next part of the paper will show parts of a Rails application developed for this paper and how XML was generated from that database. A discussion of the process of setting up the application will be presented followed by a conclusion.

XML and Relational Databases

Gicqueau (2005) makes a case that XML and relational databases are complementary not competitive. Relational databases are strong in their searching capability and in their efficiency of data storage with each unit of information saved in only one place preventing redundancy. Databases are reliable, scalable and have strong management and security features. They have also been around a long time and are deeply embedded in the business structure.

XML is also about data and its strengths include being text-based, human-readable and since it is an open standard, not tied to a particular platform. Because the structure of an XML document is part of the document transmission, it facilitates the transfer and sharing of information with others. As more and more data is exchanged between businesses and across applications, XML gains as the tool to use for this data

exchange. With the complementary strengths of XML and databases it is necessary to find ways to enable them to work together efficiently and in a way that leverages the strengths of each one. To understand how the two can work together it is helpful to understand how each structures data.

A relational database stores data in tables made up of rows and columns. A column holds all the data of a particular kind for all the records of that table. Each row in the table represents a record with the order of the records in the table irrelevant. Each table can contain only the most basic of data so there are usually many tables in a database. The tables are connected through the defining of the logical relationship between tables with the use of “keys”. These keys enable the expression of complex interrelationships between tables.

Unlike databases, XML presents an ordered, hierarchical view of data. The ordering of the data is important and essential for the XML to be well formed. XML data presentation has been likened to a tree structure with the relationship between data expressed mostly by “containment” with the attributes contained within the element (Gicqueau 2005). This makes it difficult to describe in an XML document the kinds of complex relationships that can exist in a relational database. On the other side, the preservation of the ordering in an XML document creates issues for putting the XML data into databases.

This paper will now look at Ruby on Rails, an open source framework for developing web-based, database-driven applications and see how it handles XML.

Ruby on Rails

Ruby on Rails is based on Ruby, an object-oriented scripting language which originated in Japan in the early 1990s. Rails uses the power of Ruby to enable users to

create web based applications quickly and easily. When an application is developed in Rails, there is one place for each piece of code and all the pieces of the application interact in a standard way. Because a lot of the basic structure is built into Rails, an application can be developed with much less coding than would be required using other languages.

Rails uses an architecture called MVC or Model, View, Controller. The Controller orchestrates the application by receiving events from the outside world, such as user input into a web page. The controller calls for various *actions* through interacting with the model and then returns the appropriate view to the user.

The model handles the data processing that takes place in the application. It is responsible for maintaining the state of the application and enforces the business rules that may apply to the data. The model does the data handling work and returns the result.

Views are written in HTML using controls such as text fields, listboxes, and text areas to interact with the user. Views are also responsible for displaying the results of an *action*. There can be many different views (web pages) in an application. The view only displays the data, never handles it. The views are also referred to as templates.

Rails supports 3 types of templates

- `.rhtml` which are a mixture of HTML and embedded Ruby code. This is what is used to generate the HTML pages.
- `.rxml` which uses the Builder library to construct XML responses
- `.rjs` which are used to create JavaScript

How Rails interacts with XML, as an output format and also how it is able to read it will be discussed next.

Generating Output in XML Format

1. Builder

Rails has several ways to interact with a request for XML information. The first uses the .rxml template mentioned above which would reside in the views directory of the application. This approach uses a piece of Ruby code called Builder to generate a well-formed XML document as a side effect of executing a program. Builder is a free standing library, created by Jim Weirich for the purpose of providing a simple way to create XML markup. What follows is an example from Thomas & Hansson (2006) of an .rxml template that outputs a list of product names and prices in XML.

```
xml.div(:class => @productlist) do  
  @products.each do [product]  
    xml.product do  
      xml.productname(product.title)  
      xml.price(product.price, currency => "USD")  
    end  
  end  
end
```

The way Builder works is it takes what is after the xml. and turns it into a tag. In this situation xml.product becomes <product>. Attributes can also be set as in xml.price where a tag <price> was created with a currency attribute. When the appropriate collection of products is sent from the controller, the XML output would look like:

```
<div class="productlist">  
  <product>  
    <productname>Pragmatic Programmer</productname>  
    <price currency="USD">12.34</price>  
  </product>  
  <product>  
    <productname>Rails Recipes</productname>  
    <price currency="USD">23.45</price>  
  </product>  
</div>
```

2. To_xml

A second way Rails handles a request for data in an XML format uses an autogenerating feature through a command called `to_xml`. Unlike `.rxml` templates, using `to_xml` does not allow any control over the order in which the elements are returned. By default everything is dumped with the `to_xml` command. It is possible to exclude or filter on attributes but according to one source (Thomas & Hansson 2006) this quickly becomes messy. A sample code using `to_xml` that lists whatever is in the `product` variable follows. This code would reside in the model part of the application.

```
def products  
  @products = product.find(:all)  
  render :xml => @product.to_xml  
end
```

Input in XML Format

Although there are many articles available about mapping data in XML format to relational databases, Gicqueau (2005), Guardalben & Atre (2004) and Melkumyan (2005) as examples; it is hard to find resources on how and if this can be handled in Rails. There was one source Clark (2005) who showed examples of using Ruby's built-in `net/http` library and `REXML` toolkit. But there was nothing Rails-specific. Clark's example "simply...hack(ed) through the contents of the response body with XPath, pulling out relevant information...". It was unclear what happened after the information was extracted and how it was utilized in a Rails application.

Hands-On Example

Based on the general information above I decided to create a document that would output data from a Ruby on Rails database in XML. In order to do this I:

- Installed Xcode 2.4, Ruby, RubyGems, Ruby on Rails, Mongrel, and MySQL on

my Apple Powerbook which has a Mac OS X 10.4 operating system

- Created a basic database using MySQL
- Created a basic Ruby on Rails application
- Added code to the application controller
- Created an .rxml template
- Generated an XML document

Results

Using MySQL I created a simple table named Material which consisted of 5 variables as follows:

Step 1 – create database and tables using SQL

```
+-----+-----+-----+-----+
Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+
id    | int(11) | NO   | PRI | NULL    | auto_increment |
fiber | varchar(20) | NO   |     |         |               |
weight | varchar(20) | YES  |     | NULL    |               |
brand  | varchar(20) | YES  |     | NULL    |               |
color | varchar(20) | YES  |     | NULL    |               |
+-----+-----+-----+-----+
5 rows in set (0.01 sec)
```

The table is part of a Rails database that will keep track of information for a weaving business called Megunticook Weavings. Once the table was created the Rails database was created. One of the outputs of the initialization of the database is the creation of an input form that looks like:

New material

Fiber

Weight

Brand

Color

Create

[Back](#)



Using this form I entered some raw data to the table Materials. As each “new material” was entered it appears on a screen as follows which allows for the editing and deleting of the item.

Material was successfully created.

Listing materials

Fiber	Weight	Brand	Color	
wool	2720	Maine Line	red	Show Edit Destroy
Wool'stik	2728	Zephyr	sea blue	Show Edit Destroy
cotton	312		yellow	Show Edit Destroy
alpaca	sport	misti	brown	Show Edit Destroy

[New material](#)



Going back to MySQL it is possible to see the populating of the table Materials as data is entered into the Rails application.

Table in SQL

```
mysql> select * from materials;
+----+-----+-----+-----+
| id | fiber | weight | brand | color |
+----+-----+-----+-----+
| 1 | wool | 2/20 | Maine Line | red |
| 2 | Wool/silk | 2/28 | Zephyr | sea blue |
| 3 | cotton | 3/2 | | yellow |
| 4 | alpaca | sport | misti | brown |
+----+-----+-----+-----+
4 rows in set (0.00 sec)
```

The goal was to produce an XML document using the information in the Materials table. Using a text editor I typed the following code and saved it as an .rxml file.

```
xml.instruct! :xml, :version=>"1.0"
  xml.channel {
    for m in @materials
      xml.material do
        xml.title(m.fiber)
        xml.description(m.weight)
        xml.brand(m.brand)
        xml.color(m.color)
      end
    end }
end }
```

I also had to add following lines of code to the controller.

```
def materialsml
  @materials = Material.find(:all)
  render :layout => false
end
```

What resulted when the template was referenced by the browser is the following XML code:



```
-<channel>
-<material>
  <title>wool</title>
  <description>2/20</description>
  <brand>Maine Line</brand>
  <color>red</color>
</material>
-<material>
  <title>Wool/silk</title>
  <description>2/28</description>
  <brand>Zephyr</brand>
  <color>sea blue</color>
</material>
-<material>
  <title>cotton</title>
  <description>3/2</description>
  <brand></brand>
  <color>yellow</color>
</material>
-<material>
  <title>alpaca</title>
  <description>sport</description>
  <brand>misti</brand>
  <color>brown</color>
</material>
</channel>
```



Discussion

There were three resources that were extremely helpful in my downloading, setup and initial work with Rails. The website <http://hivelogic.com/narrative/articles/ruby-rails-mongrel-mysql-osx>, provided a detailed, step by step tutorial that was clear and easy to follow on how to download and setup Rails. The strings to be typed to make the downloads and setups possible were in a format that made it easy to cut and paste them into the command line. I would highly recommend this site for anyone who is new to the Rails environment.

For the first attempt at implementing a Rails application I used a tutorial named *Rolling with Ruby on Rails Revisited* by Walton and Hibbs (2006) at <http://www.onlamp.com>. This walks the reader through a fun “real-world” basic

application. In a short time it is possible to have a Rails application up and running.

For a deeper exploration of Rails I used *Agile Web Development with Rails* by Thomas & Hansson (2006). I tried to follow the example from this book but found it too complicated and involved for what I wanted to do. After several hours of trying to get all parts of the application to work I scrapped it. To my surprise though when I went to develop the simple application I show here, I was able to build it quickly and easily.

An alternative to downloading all the files that I listed above would have been to use an application called Locomotive. This software has in one place all that is needed to develop Ruby on Rails applications. I tried it but did not find it intuitive to use and could not find any helpful tutorials. There also was something fun and a real sense of accomplishment in setting up the whole Rails application on my Apple. I found this whole process challenging but possible to accomplish.

Conclusion

This paper discussed the different ways XML and relational databases handle data. It then described Ruby on Rails, a web-based relational database. Two ways Rails uses of returning data in an XML format were detailed as well as needs an approach on how Rails can handle the input of data in the XML format. An example of a basic Rails application and how XML could be generated from it were presented. Lastly, some thoughts on the learning of Ruby on Rails were given.

Ruby on Rails has the ability to bridge the differences between relational databases and XML in a way that is convenient and consistent with the Rails structure. This makes Rails and XML a strong combination to use in the current web-based environment.

Bibliography

- Benjamin, D. (2007, February 02). *Building Ruby, Rails, Subversion, Mongrel, and MySQL on Mac OS X*. Posted to <http://hivelogic.com/narrative/articles/ruby-rails-mongrel-mysql-osx>.
- Clark, M. (2005, July 12). Producing and Consuming XML over HTTP with Rails. Message posted to: http://clarkware.com/cgi/blosxom/Rails?_start=31.
- Dsb (2006, August 25). How to Generate XML. Message posted to <http://wiki.rubyonrails.com/rails/pages/HowtoGenerateXml>.
- Gicqueau, A. (2005). *Importing XML documents to Relational Databases using Java*. Retrieved March 30, 2007 from <http://www.hitsw.com>.
- Guardalben, G. & Atre, S. (2004). *Integrating XML and Relational Database Technologies: A Position Paper*. Retrieved March 26, 2007 from <http://www.hitsw.com>.
- Holzner, S. (2007). *Beginning Ruby on Rails*. Indianapolis: Wiley Publishing, Inc.
- Holzner, S. (2003). *Sams teach yourself XML in 21 days*. 3rd ed. Indianapolis: Sams.
- Hunter, D., Watt, A., Rafter, J., Duckett, J., Ayers, D., Chase, N., Fawcett, J., Gavin, T., Patterson, B. (2004). *Beginning XML*. 3rd ed. Indianapolis: Wiley Publishing, Inc.
- Melkumyan, D. (2005). *Transferring Data between XML documents and relation database*. Retrieved March 20 from http://www-zeuthen.desy.de/technisches_seminar/texte/TRANSFERRING_DATA_XML_RDB.pdf.
- Thomas, D., & Hansson, D. (2006). *Agile Web Development with Rails*. Raleigh, N.C.: The Pragmatic Bookshelf.
- van der Vlist, E., Vernet, A., Bruchez E., Fawcett, J. and Ayers, D. (2007). *Professional Web 2.0 Programming*. Indianapolis: Wiley Publishing, Inc.
- Walton, B. & Hibbs, C. (2006). *Rolling with Ruby on Rails Revisited*. Retrieved March 26, 2007 <http://www.onlamp.com/pub/a/onlamp/2006/12/14/revisiting-ruby-on-rails-revisited.html>.