

PHẠM THỊ THU THÚY

CÔNG NGHỆ

XML

VÀ ỨNG DỤNG



M

005.13

Ph 104 Th

THƯ VIỆN ĐH NHA TRANG



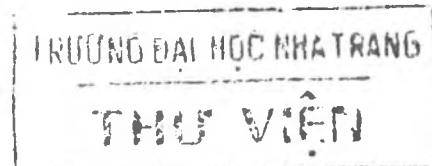
3000038355



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

PHẠM THỊ THU THÚY

CÔNG NGHỆ XML VÀ ỨNG DỤNG



30038355



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

LỜI MỞ ĐẦU

XML (viết tắt của eXtensible Markup Language - ngôn ngữ đánh dấu mở rộng) là một ngôn ngữ đánh dấu mà chúng ta có thể sử dụng để tạo ra thẻ riêng của mình. Nó được tạo nên bởi *Liên minh mạng toàn cầu* nhằm khắc phục những hạn chế của HTML - ngôn ngữ đánh dấu siêu văn bản, là cơ sở của mọi trang Web. Giống như HTML, XML cũng được dựa trên **SGML** – Standard Generalized Markup Language.

XML ra đời không phải để thay thế HTML, mà XML tồn tại vì HTML đã thành công. Như vậy XML kết hợp nhiều tính năng thành công của HTML. XML tồn tại được do HTML không thể đáp ứng được những nhu cầu mới.

Ngày nay hầu hết các nhà cung cấp phần mềm như Microsoft, Oracle, IBM, Sun... và hàng trăm công ty cung cấp phần mềm khác đã đưa XML vào sản phẩm của họ.

Bằng cách tiếp cận vấn đề đơn giản, dễ hiểu cùng với nhiều ví dụ minh họa, cuốn sách có thể cung cấp cho bạn đọc những kiến thức về tài liệu XML như: cách tạo lập, kiểm tra tính hợp lệ và hợp khuôn dạng, truy xuất dữ liệu XML và ứng dụng của XML. Cuối mỗi chương đều có bài tập thực hành, giúp bạn đọc có thể ứng dụng kiến thức đã học và hiểu sâu nội dung trong các chương.

Cuốn sách với mục đích dành cho người học mới bắt đầu tìm hiểu về XML nhưng cũng có thể phù hợp với các bạn đọc có kinh nghiệm trong các lĩnh vực khác nhưng chưa có một nền tảng về XML. Ngoài ra cuốn sách này còn có thể được sử dụng làm nguồn tài liệu tham khảo khi người học cần viết ứng dụng nhỏ liên quan đến XML. Hầu hết các nội dung được trình bày trong cuốn sách đều có một phần ví dụ minh họa liên quan đến chúng nên sẽ giúp người học có thể bắt đầu và thực hiện với một ví dụ đơn giản và sau đó có thể biết cách làm thế nào để nâng cao lượng kiến thức đã có.

Tác giả

Chương 1

TỔNG QUAN VỀ XML

1.1. Giới thiệu

Chương này giới thiệu với bạn đọc về thế giới XML, một ngôn ngữ dùng để quản lý dữ liệu ở hình thức ngắn gọn và dễ quản lý. Trước hết chúng ta sẽ tìm hiểu lịch sử XML và một số ngôn ngữ đã được xây dựng cho phép lưu trữ và quản lý văn bản, gọi chung là các ngôn ngữ đánh dấu. Người đọc sẽ thấy XML khác biệt và mạnh hơn nhiều so với các ngôn ngữ đánh dấu khác.

1.1.1. Đôi nét về lịch sử XML

XML là viết tắt của eXtensible Markup Language (Ngôn ngữ đánh dấu mở rộng), được triển khai nhờ sự đóng góp của rất nhiều người trong mười năm qua. Vào năm 1990, Tim Berners - Lee ở CERN, đã tạo ra HTML (là một phần nhỏ của SGML), nhưng lại rất dễ dùng cho mọi người. Không ngờ sự thành công của HTML vượt quá sức tưởng tượng của chính tác giả.

Nhưng rồi đến một ngày vào năm 1995, người ta bắt đầu thấy sự giới hạn của HTML, mặc dù nó được Netscape, Microsoft cố gắng thêm thắt bởi sự thịnh hành của Web. Do đó, nhiều người có ý định quay trở lại SGML, nhưng lại ái ngại.

Đúng lúc đó vào năm 1996, Jon Bosak ở Sun Microsystem khởi đầu nhóm cộng tác W3C SGML, bấy giờ được gọi là nhóm XML. Mục đích là đơn giản hóa SGML để nó dễ dùng như HTML mà đồng thời có tính mạnh mẽ.

Tim Bray và C.M.Sperberg - McQueen viết hầu hết Specification (bản điều kiện kỹ thuật) nguyên thủy của XML. Trước đó Bray đã có kinh nghiệm nhiều năm quản lý dự án "New Oxford English Dictionary" (Từ điển tiếng Anh Oxford mới). Bray muốn XML hội đủ các điều kiện sau:

- Đơn giản nhằm giúp lập trình viên dễ áp dụng
- Dễ cho các công cụ tìm kiếm (như Yahoo, Infoseek...) phân loại
- Không giới hạn trong tiếng Anh của nước Mỹ

Chính vì lẽ đó, bản Specification đầu tiên của XML được ra đời vào tháng 11 năm 1996. Tháng 5 năm 1997 Microsoft và Inso Corporation xuất bản XSL (eXtensible Style Language) để làm Style Sheet diễn tả cách trình bày một trang XML.

Đến tháng 1 năm 1998, Microsoft cho ra một chương trình miễn phí tên MSXSL để sản xuất ra một trang HTML từ một cặp trang XML và XSL. Vào tháng 2 năm 1998, tổ hợp Web toàn cầu W3C phê chuẩn cho chính thức ban hành phiên bản 1.0 của XML.

1.1.2. XML là gì?

XML có nguồn gốc như một tập hợp con của SGML, trong khi HTML là một ứng dụng của SGML. XML không ảnh hưởng đến các định dạng tổng thể của một tập tin hoặc cách thức thêm vào siêu dữ liệu, nó chỉ cụ thể một vài quy tắc, do đó XML giữ lại rất nhiều tính linh hoạt của SGML.

Các thẻ trong XML thường không được định nghĩa trước mà chúng được tạo theo quy ước của người dùng. XML giới thiệu các tính năng mới bằng việc dựa trên các ưu điểm của HTML.

Có một số đặc trưng làm XML hữu ích hơn trong các hệ thống là:

- **XML là có thể mở rộng:** XML cho phép chúng ta tạo các thẻ theo quy ước của riêng mình để phù hợp với ứng dụng của chúng ta.
- **XML mang dữ liệu, chứ không hiển thị nó:** XML cho phép chúng ta lưu giữ dữ liệu mà không quan tâm đến cái cách nó sẽ được hiển thị.
- **XML là một chuẩn chung:** XML được phát triển bởi tổ chức World Wide Web Consortium (W3C) và có sẵn như là một chuẩn mở.

XML là một ngôn ngữ đánh dấu, bao gồm tập hợp các quy tắc để mã hóa các tài liệu trong một định dạng mà con người và máy tính có thể đọc được. XML có thể được sử dụng để:

- Hỗ trợ tạo các tài liệu HTML cho các website lớn.
- Trao đổi thông tin giữa các tổ chức và hệ thống.
- Lưu trữ và truyền tải các cơ sở dữ liệu.
- Lưu trữ và sắp xếp dữ liệu, có thể tinh chế dữ liệu cần xử lý.
- Hợp nhất với các bảng định kiểu để tạo ra hầu như bất kỳ kết quả mong muốn nào.
- Biểu diễn bất kỳ kiểu dữ liệu nào.

Sự khác nhau giữa XML và HTML

- XML được tạo ra không phải để thay thế HTML mà là để bổ túc cho HTML.
- XML được thiết kế để mô tả dữ liệu và tập trung lên vấn đề đó là dữ liệu gì.
- HTML được thiết kế để hiển thị dữ liệu và tập trung lên vấn đề dữ liệu được hiển thị như thế nào trên trình duyệt.

XML không thực hiện mọi thứ

XML không được thiết kế để thực hiện mọi thứ, nó được tạo ra để cấu trúc, lưu trữ và gửi thông tin. Ví dụ 1.1 dưới đây cho thấy nội dung của một tài liệu XML:

Ví dụ 1.1:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
    <to>Doc gia</to>
    <from>Nha sach Phuong Nam</from>
    <heading>Tu sach de hoc</heading>
    <body>
        XML-Ngon ngu danh dau mo rong
    </body>
</note>
```

Chúng ta thấy ví dụ 1.1 trên là một tập tin văn bản thuần túy, thông tin ghi chú có thông tin về người gửi, người nhận, tiêu đề và phần thông điệp được

bao trong các thẻ XML. Để gửi, nhận hoặc hiển thị nó thì chúng ta phải viết mã chương trình.

XML được tự do mở rộng

Các thẻ được dùng để đánh dấu và cấu trúc tài liệu HTML được định nghĩa sẵn, khi tạo tài liệu HTML chúng ta phải sử dụng các thẻ HTML đã được định nghĩa sẵn trong chuẩn HTML như <H1>, <CENTER>,...

XML cho phép chúng ta định nghĩa và cấu trúc tài liệu XML. Như trong ví dụ 1.1, các thẻ <note>, <to>,... không được định nghĩa trong bất kỳ XML chuẩn nào. Những thẻ này do người tạo ra tài liệu định nghĩa.

1.1.3. XML được dùng như thế nào?

XML được dùng để lưu trữ và trao đổi dữ liệu, XML không được thiết kế để hiển thị dữ liệu.

XML có thể tách rời dữ liệu khỏi HTML

Với XML, dữ liệu của chúng ta được lưu bên ngoài HTML. Khi HTML được dùng để hiển thị dữ liệu, dữ liệu được lưu bên trong HTML. Với XML, dữ liệu có thể được lưu trong tập tin XML riêng biệt. Theo cách này chúng ta có thể tập trung lên việc sử dụng HTML để bố trí và hiển thị dữ liệu nhằm đảm bảo rằng những thay đổi trong dữ liệu bên dưới không yêu cầu bắt cứ thay đổi gì với HTML.

Dữ liệu XML vẫn có thể được lưu bên trong HTML như là “đảo dữ liệu”. Khi đó chúng ta có thể dùng HTML chỉ để định dạng và hiển thị dữ liệu.

XML được dùng để trao đổi dữ liệu

Với XML, dữ liệu có thể được trao đổi giữa các hệ thống không tương thích nhau. Trong thực tế, các hệ thống máy tính và CSDL chứa dữ liệu

1.2. Trình soạn thảo XML (XML Editors)

Để tạo tài liệu XML, chúng ta có thể sử dụng một trình soạn thảo văn bản như Notepad, WordPad (Windows) hay vi, emacs (LINUX/UNIX). Tập tin XML có phần mở rộng là .xml, do đó khi lưu tập tin XML ngoài việc chỉ rõ tên tập tin chúng ta cần nhập thêm phần mở rộng .xml vào tiếp sau tên tập tin. Nếu không chỉ ra phần mở rộng thì các trình soạn thảo tự thêm vào phần mở rộng mặc định, ví dụ Notepad tự động thêm .txt, còn WordPad thì thêm .rtf vào sau tên tập tin.

Mặc dù vậy, có rất nhiều trình soạn thảo phục vụ cho chức năng hiệu chỉnh nội dung dữ liệu XML. Với những trình soạn thảo chuyên dụng này, chúng ta có thể thực hiện được rất nhiều điều như: Các thẻ được hiển thị bằng màu sắc phân biệt, cho phép tìm kiếm, tổ chức, sắp xếp dữ liệu trong tập tin XML,... Một vài trình soạn thảo còn cho phép kiểm tra tính hợp lệ và hợp khuôn dạng của tài liệu.

Dưới đây là một số trình soạn thảo XML nổi tiếng có thể sử dụng:

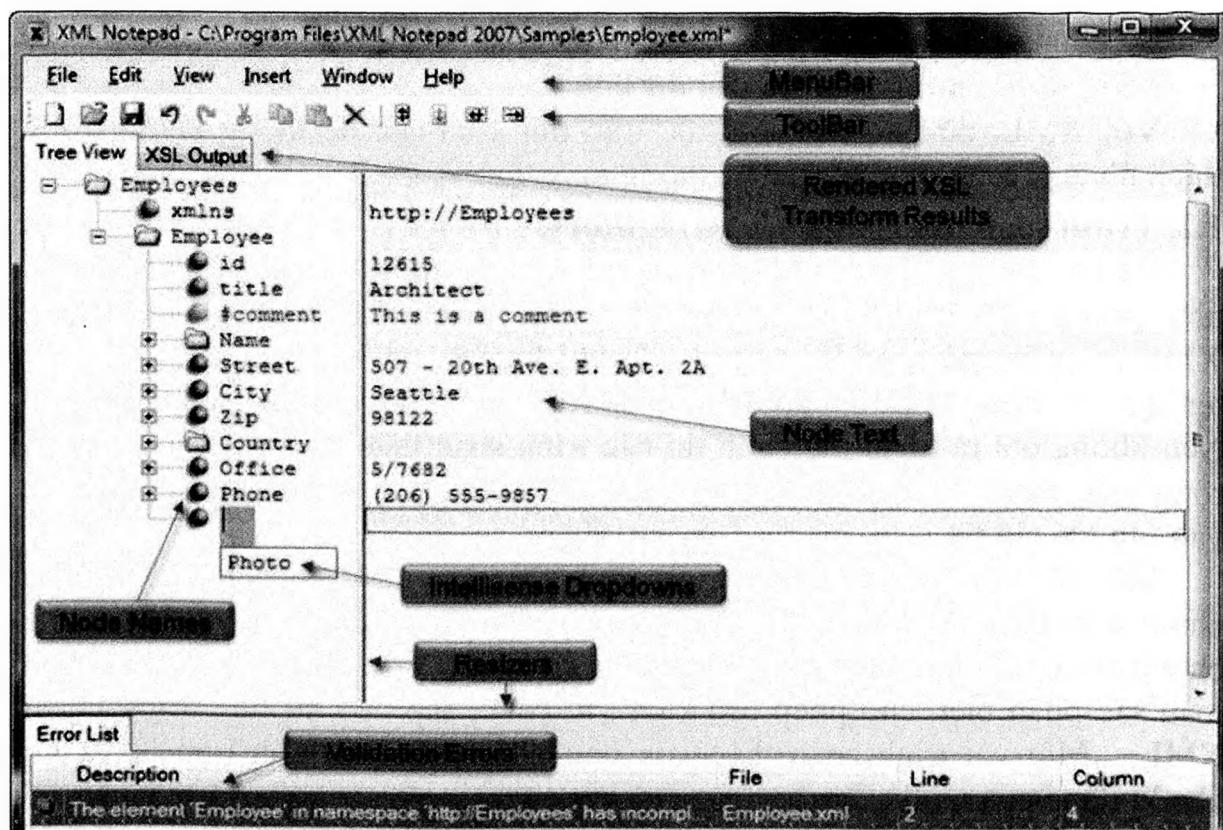
- Adobe FrameMaker (<http://www.adobe.com>)
- XML Pro (<http://www.vervet.com/>)
- XML Writer (<http://xmlwriter.net>)
- XML Notepad (của Microsoft)
- Microsoft's Visual Studio .NET

- XML Spy (<http://www.xmlspy.com>)
- XMLmind (<http://www.xmlmind.com/xmleditor>)
- Stylus Studio (<http://www.stylusstudio.com/>)
- Oxygen XML Editor (<https://www.oxygenxml.com/>)

Nhìn chung, với các trình soạn thảo XML nêu trên, đơn giản và dễ dùng nhất là trình soạn thảo XML Notepad của Microsoft. Hình 1.1 là màn hình soạn thảo XML Notepad của Microsoft. Phần mềm này có thể lấy về tại địa chỉ: <https://xmlnotepad.codeplex.com/>

Nếu muốn dùng trình soạn thảo chuyên nghiệp hơn, chúng ta dùng XML Writer. Đây là phần mềm rất mạnh, nó cho phép soạn thảo và xem tài liệu XML tương tự Explorer của Microsoft.Thêm vào đó, XML Writer còn có thể kiểm tra tính hợp khuôn dạng (well-formed) của tài liệu, kiểm tra tính hợp lệ của định nghĩa DTD và lược đồ XML. Hình 1.2 là hình soạn thảo của phần mềm XML Writer. XML Writer được lấy về từ địa chỉ <http://xmlwriter.net/>

Ngoài ra, XML Spy và Stylus Studio cũng là hai trình soạn thảo XML khá hay. XML Spy mang đầy đủ chức năng kiểm tra định nghĩa DTD và lược đồ XML tương tự như XML Writer. Nếu người sử dụng muốn một công cụ soạn thảo XML có tích hợp XQuery và XPath thì trình soạn thảo Oxygen XML Editor là một trong các lựa chọn tối ưu nhất.



Hình 1.1: Trình soạn thảo XML, XML Notepad

Chúng ta đã có các công cụ soạn thảo XML khá đẹp và tiện dụng. Công cụ mà chúng ta cần tiếp theo sau đây rất cần cho Web, đó là phần mềm trình duyệt (browser).

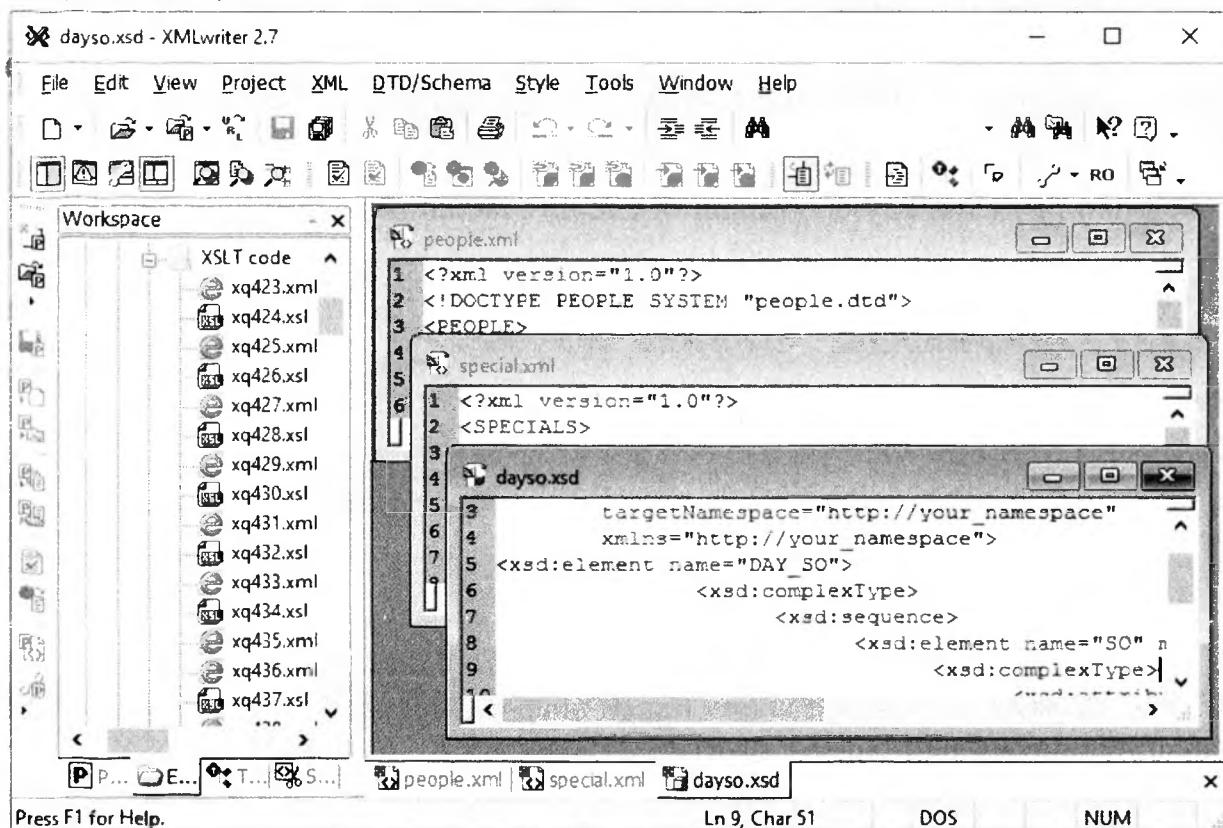
1.3. Trình duyệt XML

Một trình duyệt hỗ trợ cả XML, HTML, CSS, XSL cùng với ngôn ngữ lập trình Script cho trình duyệt như Javascript hiện nay không có nhiều. Hiện nay, những phần mềm trình duyệt hỗ trợ XML (chỉ kiểm tra khuôn dạng XML hợp lệ và cho phép kết hợp CSS, XSL) bao gồm: Internet Explorer, Google Chrome, Jumbo, Firefox... Trong các trình duyệt thì Internet Explorer là trình duyệt hỗ trợ XML mạnh nhất.

Internet Explorer (IE)

Trình duyệt Internet Explorer của Microsoft là trình duyệt hỗ trợ XML mạnh nhất hiện nay. IE có thể hiển thị XML trực tiếp, hoặc dùng ngôn ngữ kịch bản Javascript để truy xuất dữ liệu trong tài liệu XML và cho phép chúng ta quản lý dữ liệu đó bằng mã. Ngoài ra, IE còn hỗ trợ cả Jscript và VBScript. Chúng ta có thể dùng bảng định kiểu CSS hoặc XSL để hiển thị nội dung của XML theo định dạng HTML trong IE đồng thời có thể áp dụng XML vào mô hình ràng buộc và truy xuất đối tượng dữ liệu ADO (ActiveX Data Object).

IE có thể kiểm tra sự hợp lệ của tài liệu XML dùng DTD hoặc lược đồ XML. IE có thể đóng kết dữ liệu XML với các điều khiển HTML như hộp văn bản, nút nhấn,...



Hình 1.2: Trình soạn thảo XML, XML Writer

1.4. Bộ phân tích XML (XML Parser)

Bộ phân tích XML là các gói phần mềm được sử dụng như một phần kèm theo của ứng dụng. Mặc dù vậy, có rất nhiều thư viện cung cấp các hàm, cho phép phân tích nội dung và trích xuất dữ liệu của XML khá đơn giản. Sau đây là một số trình phân tích và thư viện thường dùng nhất:

- *XML for Java (XML4J)*: Thư viện phân tích tài liệu XML phát triển bởi IBM Alpha Works. Thư viện này rất nổi tiếng và được sử dụng rộng rãi trong hầu hết các ví dụ của W3C. Địa chỉ tham khảo: www.alphaworks.ibm.com/tech/cml4j
- *Microsoft XML Parser*: Trình phân tích sử dụng bởi trình duyệt IE. Nếu sử dụng hệ điều hành Windows với Internet Explorer 5 trở lên thì chúng ta đã có sẵn trình phân tích này kèm theo. Có thể download bản tách rời của trình phân tích này tại địa chỉ <http://msdn.microsoft.com/downloads/tools/xmlparser>
- *Java Standard Extension for XML*: Đây là gói thư viện dành cho Java xây dựng bởi SunMicrosystem. Download gói này về dùng chung với các ứng dụng Java tại địa chỉ <http://java.sun.com/products/xml>
- *Python XML Parser*: Trình phân tích tài liệu XML dựa vào ngôn ngữ Python cho Linux và UNIX. Địa chỉ lấy về <http://python.org/topics/xml>

Nhìn chung, các trình phân tích thường tách dữ liệu XML ra thành từng mẫu nhỏ và khiếu chúng có thể truy xuất được bởi mã lệnh của chương trình. Một vài trình phân tích kiêm luôn cả chức năng kiểm tra khuôn dạng hoặc tính hợp lệ của tài liệu XML. Mặc dù vậy, nếu muốn có một công cụ vừa kiểm được lỗi hợp khuôn dạng, vừa kiểm tra được tính hợp lệ của tài liệu XML thì chúng ta có thể cần đến các bộ kiểm tra chuyên dụng hơn.

1.5. Quy tắc cú pháp XML

Phần này trình bày cho chúng ta các quy tắc cú pháp đơn giản để viết một tài liệu XML. Ví dụ dưới là một tài liệu XML đầy đủ:

Ví dụ 1.2:

```
<?xml version="1.0"?>
<contact-info>
    <name>Tanmay Patil</name>
    <company>TutorialsPoint</company>
    <phone>(011) 123-4567</phone>
</contact-info>
```

Trong ví dụ trên có hai loại thông tin:

- Thông tin đánh dấu, như `<contact-info>` và
- Text, hoặc dữ liệu là ký tự, như *Tutorials Point* và *(040) 123-4567*.

Sơ đồ Hình 1.3 minh họa quy tắc cú pháp để viết các kiểu đánh dấu và text khác nhau trong một tài liệu XML.

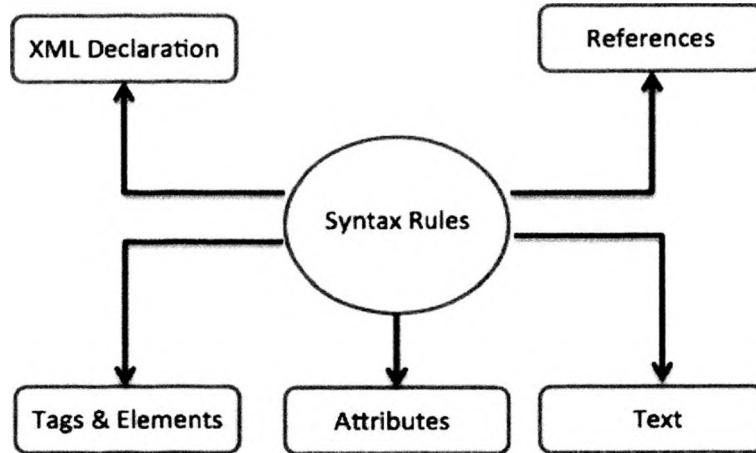
Giải thích chi tiết cho từng phần như sau:

1.5.1. Khai báo định dạng XML (XML Declaration)

Tài liệu XML có thể tùy ý có một phần khai báo XML, được viết như sau:

```
<?xml version="1.0" standalone="yes" encoding="UTF-8"?>
    • version là phiên bản XML, hiện tại là 1.0, sắp có phiên bản 1.1
    • encoding xác định mã hóa ký tự được sử dụng trong tài liệu. W3C
        yêu cầu tất cả trình xử lý XML phải hỗ trợ UTF-8 (Unicode rút gọn
        dùng 1 byte) và UTF-16 (Unicode rút gọn dùng 2 byte). Hiện nay,
        hầu hết trình xử lý XML đều hỗ trợ bộ mã UTF-8 nhưng không hỗ
```

trợ UTF-16. Khi khai báo XML, nếu không đề cập encoding thì trình xử lý mặc định dùng UTF-8.



Hình 1.3: Quy tắc cú pháp của XML

Mặc dù chỉ UTF-8 và UTF-16 được yêu cầu, vẫn có nhiều bộ ký tự mã hóa khác mà một trình xử lý XML có thể hỗ trợ, chẳng hạn như:

- US-ASCII: Bộ mã ASCII
- ISO-10646-UCS-2: Unicode
- ISO-2022-CN: Chinese
- ISO-8859-5: ASCII and Cyrillic

✓ *standalone*: Là “yes” nếu tài liệu không tham chiếu bất kỳ tài liệu hay thực thể bên ngoài nào; “no” nếu ngược lại. Đây là thuộc tính tự chọn

Các quy tắc cú pháp để khai báo XML

- Khai báo XML phân biệt kiểu chữ hoa/thường và phải bắt đầu với "<?xml>", ở đây "xml" viết ở dạng chữ thường.
- Khai báo XML phải là lệnh đầu tiên của tài liệu XML.
- Một HTTP Protocol có thể ghi đè giá trị của *encoding* mà chúng ta đặt trong khai báo XML.

1.5.2. Thẻ và phần tử (Tags and Elements)

Tạo tên thẻ:

Trong XML 1.0, tên thẻ bắt đầu bằng ký tự chữ hoặc dấu gạch dưới hoặc dấu hai chấm. Các ký tự tiếp theo có thể là ký tự số, chữ, gạch dưới, dấu chấm hoặc dấu hai chấm. Ví dụ, các tên thẻ sau là hợp lệ:

```
<DOCUMENT>
<document>
<Chapter15>
<Section-19>
<_text>
```

Tên thẻ có phân biệt chữ hoa và chữ thường, do đó <DOCUMENT> khác với <document>.

Ví dụ sau đây là các tên thẻ không hợp lệ:

```
<2005>
<Loan number>
<.text>
```

```
<*yay*>  
<EMPLOYEE (ID)>
```

Phần tử:

Một XML tập tin được cấu thành bởi một số phần tử XML (Element), còn được gọi là XML-node hoặc XML-tags. Tên các phần tử XML được bao trong các dấu <> như sau:

```
<element>
```

Cú pháp phần tử: Mỗi phần tử XML cần được bao trong hoặc với phần tử bắt đầu hoặc kết thúc như sau:

```
<element>....</element>
```

Hoặc đơn giản theo cách: <element/>

Tạo phần tử rỗng:

Trong XML, phần tử rỗng chỉ có một thẻ, không có thẻ bắt đầu và thẻ kết thúc. Phần tử rỗng là phần tử không chứa dữ liệu hay bất kỳ phần tử nào khác. Trong XML chúng ta dùng dấu /> để đóng phần tử rỗng. Ví dụ sau cho thấy phần tử *heading* là phần tử rỗng:

Ví dụ 1.3:

```
<?xml version="1.0" encoding="UTF-8"?>  
<document>  
    <heading/>  
    <subject>  
        Cong nghe XML va ung dung  
    </subject>  
</document>
```

Phần tử rỗng có thể có thuộc tính. Ví dụ bên dưới chỉ ra rằng phần tử *heading* có thuộc tính là *color*.

```
<?xml version="1.0" encoding="UTF-8"?>  
<document>  
    <heading color="ff0000"/>  
    <subject>  
        Cong nghe XML va ung dung  
    </subject>  
</document>
```

Lồng các phần tử: Một phần tử XML có thể chứa nhiều phần tử XML khác, nhưng các phần tử con này không được đè lên nhau. Thẻ đóng phải có cùng tên như thẻ mở. Ví dụ sau là các thẻ lồng nhau sai cú pháp:

```
<?xml version="1.0"?>  
<contact-info>  
<company>TutorialsPoint  
<contact-info>  
</company>
```

Ví dụ sau minh họa các thẻ lồng nhau đúng cú pháp:

```
<?xml version="1.0"?>  
<contact-info>  
<company>TutorialsPoint</company>  
<contact-info>
```

Phần tử gốc (Root Element): Một tài liệu XML có thể chỉ có một phần tử gốc. Ví dụ sau minh họa một tài liệu XML sai cú pháp, bởi vì cả hai phần tử x và y xuất hiện ở cấp cao nhất mà không phải là một phần tử gốc.

```
<x>...</x>
<y>...</y>
```

Còn đây là ví dụ đúng cú pháp:

```
<root>
  <x>...</x>
  <y>...</y>
</root>
```

Phân biệt kiểu chữ: Tên của các phần tử XML là phân biệt kiểu chữ. Nghĩa là tên của thẻ mở và thẻ đóng phải cùng kiểu.

Ví dụ, `<contact-info>` là khác với `<Contact-Info>`.

Thuộc tính (Attributes)

Thuộc tính XML xuất hiện trong các phần tử, chỉ thị xử lý và khai báo XML, có chức năng như các thuộc tính trong HTML. Thuộc tính thường được sử dụng bằng cặp giá trị *tenthuoctinh* = “*gia tri thuoc tinh*” trong các thẻ mở. Giá trị thuộc tính phải nằm trong dấu nháy kép. Nếu dùng thuộc tính thì nó phải có giá trị. Có thể dùng DTD hoặc giản đồ XML để chỉ định một thuộc tính là bắt buộc hay tùy chọn.

Ví dụ tập tin *Operator.xml* dưới đây có *OperatorID*, *UserName* và *Password* là các thuộc tính của phần tử *Operator*:

```
<?xml version="1.0" encoding="UTF-8"?>
<OperatorList>
  <Operator OperatorID="1" UserName="Vuong" Password="">
    </Operator>
  <Operator OperatorID="2" UserName="Nam" Password="">
    </Operator>
  <Operator OperatorID="3" UserName="Nguyen" Password="h">
    </Operator>
</OperatorList>
```

Khi đọc cuốn sách này, trong các trình duyệt chúng ta sẽ thấy được các thuộc tính và giá trị của nó. Trình xử lý XML có thể nhận giá trị chúng ta gán cho thuộc tính của phần tử. Tuy nhiên nếu dùng thuộc tính để giữ giá trị thì chúng ta không chỉ ra được cấu trúc cho tài liệu. Ví dụ sau đây viết lại phần tử *Operator* thì tài liệu sẽ có một cấu trúc rõ ràng:

```
<operator>
  <OperatorID>3</OperatorID>
  <UserName>Nguyen</UserName>
  <Password>h</Password>
</operator>
```

Lưu ý: Giá trị của thuộc tính phải được đặt trong dấu nháy kép hoặc dấu nháy đơn. Nếu giá trị thuộc tính có chứa dấu nháy kép thì chúng ta đặt giá trị thuộc tính trong dấu nháy đơn. Ví dụ:

```
<poem title='Bai tho "Bien" cua Xuan Dieu'>
```

Ngược lại đặt giá trị thuộc tính trong dấu nháy kép:

```
<poem title="Bai tho 'Bien' cua Xuan Dieu">
```

1.5.3. Quy tắc cú pháp cho thuộc tính trong XML

- Tên thuộc tính trong XML là phân biệt kiểu chữ (không giống như HTML). Tức là, *HREF* và *href* là hai thuộc tính khác nhau.
- Cùng một thuộc tính không thể có hai giá trị. Ví dụ sau là sai cú pháp bởi vì thuộc tính **b** được xác định hai lần:
`....`
- Tên thuộc tính được định nghĩa không có sự trích dẫn, trong khi giá trị thuộc tính phải luôn luôn trong các dấu trích dẫn (dấu nháy kép). Ví dụ sau là sai cú pháp:
`....`

Trong ví dụ này, giá trị thuộc tính không được định nghĩa trong các dấu trích dẫn.

1.5.4. Tham chiếu trong XML

Tham chiếu (References) cho phép chúng ta thêm hoặc bao phần text hoặc phần đánh dấu bổ sung trong một tài liệu XML. Các tham chiếu luôn luôn bắt đầu với biểu tượng "&" , đây là ký tự dành riêng và kết thúc với ký tự ";". XML có hai kiểu tham chiếu:

- Tham chiếu thực thể:** Một tham chiếu thực thể là một nhóm các ký tự được sử dụng trong văn bản thay thế cho một ký tự đặc biệt trùng với khai báo trong XML. Ví dụ: & có amp là tên thay cho dấu &. **Tên** tham chiếu tới một chuỗi văn bản hoặc đánh dấu đã được định nghĩa trước.
- Tham chiếu ký tự:** Chứa các tham chiếu, ví dụ A, chứa một dấu thăng (#) được theo sau bởi một số. Số này luôn luôn tham chiếu tới mã hóa Unicode của ký tự. Ví dụ, 65 tham chiếu tới chữ cái "A".

1.5.5. Text trong XML

- Tên của phần tử XML và thuộc tính XML là phân biệt kiểu chữ, nghĩa là tên của phần tử mở và phần tử đóng phải ở được viết cùng kiểu.
- Để tránh các vấn đề về mã hóa ký tự, tất cả XML tập tin nên được lưu ở dạng Unicode UTF-8 hoặc UTF-16.
- Các ký tự whitespace như khoảng trắng, tab và ngắt dòng giữa các phần tử XML và giữa các thuộc tính XML sẽ bị bỏ qua.
- Một số ký tự được dành riêng trong cú pháp XML. Vì thế chúng không thể được sử dụng một cách trực tiếp. Để sử dụng chúng, một số thực thể thay thế được sử dụng, các thực thể này được liệt kê trong bảng dưới:

Bảng 1.1: Một số ký tự thay thế ký tự đặc biệt trong XML

Ký tự không được phép sử dụng	Thực thể thay thế	Miêu tả
<	<	Nhỏ hơn
>	>	Lớn hơn
&	&	Và
'	'	Dấu nháy đơn

Ghi chú tài liệu

Ghi chú trong tài liệu XML được thực hiện thông qua cặp dấu <!-- và --> giống như trong HTML. Khi đó trình xử lý XML sẽ bỏ qua đoạn văn chúa trong cặp dấu này.

Ví dụ:

```
<?xml version="1.0" encoding="UTF-8"?>
    <!-- Phan khai bao -->
<document>
    <!-- Phan noi dung -->
</document>
```

Lưu ý: Phần ghi chú không được đặt trước khai báo XML. Ví dụ cách dùng sau là không hợp lệ:

```
<!-- Phan khai bao -->
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <!-- Phan noi dung -->
</document>
```

1.6. Các bộ kiểm tra XML (XML Validators)

Làm thế nào để biết được tài liệu XML vừa viết ra là đúng khuôn dạng và hợp lệ? Một trong các cách để biết là dùng bộ kiểm tra XML (XML Validator). Các bộ kiểm tra tính hợp lệ của XML là những gói phần mềm cho phép kiểm tra một số chuẩn trên XML và đưa ra lời nhận xét.

Ví dụ, nếu sử dụng bộ phân tích XML4J của IBM, chúng ta có thể dùng đối tượng DOMWriter để làm bộ kiểm tra Validator, thực hiện kiểm tra tính hợp lệ của tài liệu XML. Nếu tài liệu XML hợp lệ, DOMWriter chỉ hiển thị lại nội dung XML. Nếu phát hiện các lỗi XML không hợp lệ, DOMWriter sẽ chỉ rõ vị trí của thẻ gây lỗi. Tuy nhiên việc dùng DOMWriter đòi hỏi chúng ta phải chạy trên Java. Dưới đây là địa chỉ một số bộ kiểm tra XML để tham khảo, chúng không cần dùng đến Java như DOMWriter. Thông thường các bộ kiểm tra Validator được phân làm hai loại: Loại kiểm tra tài liệu trực tiếp trên Web và loại ứng dụng cài đặt như một chương trình thông thường.

- Kiểm tra tính hợp lệ trực tiếp trên Web

Dưới đây là danh sách một số địa chỉ cung cấp trình kiểm tra XML (các dịch vụ này kiểm tra tài liệu trực tuyến, tức là phải cung cấp đường dẫn URL trong tài liệu của chúng ta):

- <http://validator.w3c.org>: Có thể upload tài liệu từ máy của chúng ta để kiểm tra.
- <http://www.xml.com/pub/a/tools/ruwf/check.html>: Tài liệu phải trực tuyến (tức là tài liệu XML phải được lưu trên một máy chủ web nào đó và được công bố trên internet).
- <http://www.ltg.ed.ac.uk/~richard/xml-check.html>: Tài liệu phải trực tuyến.
- <http://www.stg.brown.edu/service/xmlvalid/>: Có thể upload tài liệu từ máy của chúng ta để kiểm tra. Đây là địa chỉ rất phổ biến dùng để kiểm tra tài liệu XML.

- Kiểm tra tính hợp lệ bằng chương trình

Một số ứng dụng cài đặt có thể dùng để kiểm tra tính hợp lệ của XML như sau:

XML Writer, XML: Đây là trình soạn thảo và kiểm tra tính hợp lệ của tài liệu XML rất đáng giá đã được đề cập ở mục các trình soạn thảo XML.

Giả sử chúng ta dùng XMLwriter để kiểm tra tài liệu XML là *Operator.xml*. Từ thanh trình đơn, bấm chọn trình đơn *File*, chọn mục trình đơn *Open*, hộp thoại *Open File* hiển thị, tìm và chọn tập tin *Operator.xml*, bấm *Open* để mở tập tin.

Từ thanh trình đơn, bấm chọn trình đơn *XML* rồi chọn mục trình đơn *Check If Well-Formed* (hoặc nhấn F6) để kiểm tra tính hợp khuôn dạng. Vùng cửa sổ *Error Message* bên dưới sẽ cho biết kết quả kiểm tra là tài liệu có hợp khuôn dạng hay không? Nếu hợp khuôn dạng thì sẽ hiển thị thông báo *Document is well-formed XML*. Ngược lại thì sẽ xuất hiện dòng thông báo lỗi và có mũi tên xuất hiện tại dòng bị lỗi.

Hình 1.4 dưới đây thể hiện kết quả kiểm tra tính hợp lệ của tài liệu *Operator.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<OperatorList>
    <Operator OperatorID="1" UserName="Vuong" Password="">
    </Operator>
    <Operator OperatorID="2" UserName="Nam" Password="">
    </Operator>
    <Operator OperatorID="3" UserName="Nguyen" Password="h">
    </Operator>
</OperatorList>
```

Hình 1.4: Kết quả kiểm tra tính hợp lệ của tài liệu *Operator.xml*

BÀI TẬP CHƯƠNG 1

1. Tải và cài đặt trình biên tập XML là XMLWriter hoặc Oxygen XML Editor.
2. Trình bày các khái niệm và ý nghĩa sử dụng của tài liệu XML.
3. Trình bày chi tiết về định chuẩn XML: Hệ thống các thẻ đánh dấu, quan hệ lồng nhau giữa các thẻ có nội dung và quy định về thẻ đánh dấu gốc. Ví dụ minh họa cho mỗi quy định.
4. Lỗi gì làm cho các tập tin sau đây không được well-formed (hợp khuôn dạng)?

```
<xmlLibrary>
    <play publicationYear=1898>
        <title>Arms & The Man</title>
        <author>George Bernard Shaw</author>
        <play description> Comedy dealing with the futility of
                           war and the hypocrisy of human nature.
        </play description>
    <play>
        <play publicationYear=1950>
            <title>The Mousetrap</title>
            <author>Agatha Christie</author>
            <play description>A traditional whodunnit with an
                           extraordinary twist.
            </play description>
        <play>
    </xmlLibrary>
```

Chương 2

TẠO TÀI LIỆU XML HỢP KHUÔN DẠNG

2.1. XML hợp khuôn dạng

Tại sao việc tạo một tài liệu XML hợp khuôn dạng lại quan trọng đến thế? Thứ nhất vì tổ chức W3C không xem tài liệu XML là XML trừ khi nó hợp khuôn dạng; Thứ hai là trình xử lý XML không đọc tài liệu XML trừ khi những tài liệu này hợp khuôn dạng. Hai lý do này nói lên việc tạo một tài liệu XML hợp khuôn dạng là điều cần thiết khi tạo tài liệu XML.

Một tài liệu XML được gọi là hợp khuôn dạng khi nó tuân theo các quy tắc và cú pháp của tài liệu XML.

2.1.1. Điều gì tạo nên một tài liệu XML hợp khuôn dạng?

Một tài liệu XML được gọi là hợp khuôn dạng khi nó tuân theo các quy tắc và cú pháp của tài liệu XML. Theo W3C, tính hợp khuôn dạng được xem xét như sau:

- Tổng thể phải phù hợp với cái gọi là sản phẩm tài liệu.
- Thỏa mãn tất cả ràng buộc nêu trong đặc tả về XML.

Mỗi thực thể được phân tích mà được tham khảo trực tiếp hoặc gián tiếp trong tài liệu cũng phải hợp khuôn dạng.

* Phù hợp với một sản phẩm tài liệu:

Một tài liệu phải có 3 phần:

- Phần mở đầu (có thể rỗng)
- Phần tử gốc (có thể chứa các phần tử khác)
- Phần tử còn lại (đây là phần tử tùy chọn)

* Thỏa mãn các ràng buộc

Yêu cầu này hơi khó kiểm tra một chút, bởi vì tài liệu XML hợp khuôn dạng cũng phải thỏa mãn các ràng buộc hợp khuôn dạng trong đặc tả về XML.

* Các thực thể được phân tích cũng phải hợp khuôn dạng

Yêu cầu cuối cùng là bản thân các thực thể cũng phải hợp khuôn dạng. Khi một tài liệu XML được phân tích bởi trình xử lý XML, các tham khảo thực thể ví dụ như (<) được thay bằng thực thể nó thể hiện là (<). Như vậy yêu cầu của các thực thể được phân tích cũng phải hợp khuôn dạng, đơn giản là khi chúng ta thay thế các tham khảo thực thể bằng các thực thể nó thể hiện thì kết quả phải hợp khuôn dạng.

2.1.2. Tạo một tài liệu XML mẫu

Trước hết chúng ta tạo phần khai báo XML:

```
<?xml version="1.0"?>
```

Bởi vì tất cả tài liệu chúng ta thấy là tự chứa (không tham khảo hoặc bao gồm các thực thể bên ngoài) nên chúng ta có thể tạo thuộc tính standalone là “yes” và để biểu diễn được mọi ngôn ngữ chúng ta sử dụng bộ mã UTF-8:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

Tiếp theo thêm phần tử gốc cho tài liệu, giả sử tên phần tử gốc là <document>.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
...
</document>
```

Phần tử gốc chứa các phần tử khác, giả sử có phần tử <employee>:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <employee>
        ...
    </employee>
</document>
```

Mỗi phần tử <employee> chứa phần tử <name>, phần tử <name> chứa 2 phần tử <firstname> và <lastname>:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <employee>
        <name>
            <firstname>Nguyen</firstname>
            <lastname>Son Nam</lastname>
        </name>
        ...
    </employee>
</document>
```

Trong phần tử <employee>, chúng ta thêm các phần tử để lưu ngày bắt đầu làm việc, tên dự án mà nhân viên phụ trách, mỗi dự án gồm có tên sản phẩm, mã số và đơn giá. Sau đây là nội dung đầy đủ cho tài liệu XML ở trên:

Ví dụ 2.1:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <employee>
        <name>
            <firstname>Nguyen</firstname>
            <lastname>Son Nam</lastname>
        </name>
        <hiredate>July 15, 2015</hiredate>
        <projects>
            <project>
                <product>Printer</product>
                <id>HP2300</id>
                <price>$111.00</price>
            </project>
            <project>
                <product>Laptop</product>
                <id>DELL1100</id>
                <price>$989.00</price>
            </project>
        </projects>
    </employee>
</document>
```

```

</projects>
</employee>
<employee>
  <name>
    <firstname>Le Thi</firstname>
    <lastname>Kieu Oanh</lastname>
  </name>
  <hiredate>October 20, 2016</hiredate>
  <projects>
    <project>
      <product>Desktop</product>
      <id>LV333</id>
      <price>$595.00</price>
    </project>
    <project>
      <product>Scanner</product>
      <id>Canon3200</id>
      <price>$200.00</price>
    </project>
  </projects>
</employee>
</document>

```

2.2. Các ràng buộc hợp khuôn dạng

Các ràng buộc hợp khuôn dạng được đề cập rất ít trong đặc tả XML 1.0, một số ràng buộc phải thực hiện với DTD và các tham khảo thực thể. Vì vậy, phần này sẽ chỉ giới thiệu sơ lược về ràng buộc.

Bắt đầu tài liệu là khai báo XML:

Ràng buộc cấu trúc hợp khuôn dạng đầu tiên là phần khai báo XML. Mặc dù một số trình xử lý XML sẽ bỏ qua, tuy nhiên W3C nói rằng chúng ta nên khai báo phần XML.

Chỉ sử dụng các tham khảo ký tự hợp lệ:

Ràng buộc thứ hai là chỉ sử dụng các tham khảo ký tự được hỗ trợ trong đặc tả XML.

Có ít nhất một phần tử

Để được xem là một tài liệu hợp khuôn dạng thì tài liệu XML phải bao gồm một hoặc nhiều phần tử. Phần tử đầu tiên phải là phần tử gốc.

Các phần tử phải có cấu trúc đúng

Trình duyệt khá dễ chịu về cách chúng ta cấu trúc các phần tử HTML, ví dụ khi chúng ta tạo thẻ mở `<h1>` nhưng không có thẻ đóng `</h1>` thì cũng không có vấn đề gì với trình duyệt. Nhưng với XML thì không, chúng ta phải tạo thẻ mở, thẻ đóng cho phần tử có dữ liệu. Ngoài ra, cặp thẻ mở và đóng phải có vị trí phù hợp (đúng cấp) và cùng tên.

Sử dụng phần tử gốc chứa các phần tử khác

Một ràng buộc khác là phần tử gốc phải chứa tất cả các phần tử khác trong tài liệu XML.

Các phần tử lồng nhau

Các phần tử lồng nhau phải đúng cấp. Ví dụ:

```
<employee>
  <name>
    <firstname>Nguyen</firstname>
    <lastname>Son Nam</lastname>
    <hiredate>
      </name>
      July 15, 2015
    </hiredate>
```

Phần tử `<name>` và `<hiredate>` cũng cấp nhưng lồng nhau không đúng.

Tên thuộc tính phải duy nhất

Chúng ta không thể sử dụng cùng tên thuộc tính nhiều hơn một lần trong cùng thẻ mở hoặc thẻ phần tử rỗng. Ví dụ sau cho thấy tài liệu vi phạm ràng buộc này:

```
<message text="Hi there!" text="Hello!">
```

Giá trị thuộc tính

Một ràng buộc khác là giá trị thuộc tính phải được đặt trong dấu nháy kép hoặc nháy đơn. Nếu giá trị có dấu nháy kép ("") và dấu nháy đơn thì chúng ta thoát ký tự ("") bằng `"` và ('') bằng `'`.

Ví dụ, với giá trị số là 50'11"

```
<date time="50&apos;11"/>
```

Không sử dụng tham khảo thực thể và dấu < trong giá trị thuộc tính

Chúng ta không được đặt tham khảo thực thể ngoại hoặc dấu < trong giá trị thuộc tính bởi vì trình xử lý XML không thay thế giá trị thuộc tính với nội dung của thực thể bên ngoài. Ví dụ, thay vì viết:

```
<project note="This is a <project> element.">
```

chúng ta phải viết:

```
<project note="This is a &lt;project&gt; element.">
```

Tránh lạm dụng dấu < và &

Các trình xử lý XML xem dấu < là dấu bắt đầu thẻ và dấu & là dấu bắt đầu tham khảo thực thể vì thế chúng ta nên tránh sử dụng các dấu này ở bất kỳ đâu trong tài liệu.

2.3. Sử dụng không gian tên trong XML

Vì XML cho phép chúng ta tự định nghĩa tên thẻ của riêng chúng ta cho nên dẫn đến khả năng chúng ta sử dụng tên trùng lặp trong hai tài liệu XML khác nhau để mô tả hai kiểu phần tử khác nhau. Để giải quyết vấn đề này, chúng ta phải sử dụng không gian tên. Không gian tên cho chúng ta cách đảm bảo rằng một tập thẻ sẽ không bị xung đột với tập thẻ khác.

Ví dụ, tài liệu sau chứa thông tin trong thẻ `<table>`:

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

Tài liệu sau cũng chứa thông tin trong thẻ `<table>`:

```
<table>
    <name>NTB Coffee Table</name>
    <width>80</width>
    <length>120</length>
</table>
```

Khi gộp hai tài liệu XML này lại sẽ có xung đột về tên phần tử mà cả 2 tài liệu đều chứa là `<table>` với nội dung và định nghĩa khác nhau.

2.3.1. Xử lý xung đột tên phần tử

Để xử lý xung đột về tên cho 2 tài liệu trên, chúng ta dùng tiền tố. Ví dụ đầu, dùng tiền tố v:

```
<v:table>
    <v:tr>
        <v:td>Apples</v:td>
        <v:td>Bananas</v:td>
    </v:tr>
</v:table>
```

Tài liệu sau dùng tiền tố f:

```
<f:table>
    <f:name>NTB Coffee Table</f:name>
    <f:width>80</f:width>
    <f:length>120</f:length>
</f:table>
```

Bây giờ không còn xung đột về tên phần tử vì 2 tài liệu dùng tên khác cho phần tử `<table>` (`<v:table>` và `<f:table>`). Bằng cách sử dụng tiền tố, chúng ta đã tạo ra 2 kiểu phần tử `<table>` khác nhau.

2.3.2. Sử dụng không gian tên

Thay vì chỉ sử dụng tiền tố, tài liệu XML được yêu cầu sử dụng thuộc tính `xmlns` với không gian tên mang ý nghĩa đầy đủ hơn. Để dễ hiểu, chúng ta hãy xem xét lại ví dụ 2.1 ở trên. Giả sử, chúng ta muốn thêm phần tử `<comment>` vào trong dữ liệu của phần tử `<employee>`. Vì dữ liệu này đến từ phòng *Quản lý nguồn lực* và họ chưa tạo phần tử `<comment>`, trong khi chúng ta cần tạo phần tử `<comment>` cho riêng chúng ta. Trước tiên chúng ta phải đưa dữ liệu của phòng *Quản lý nguồn lực* vào không gian tên của nó để nói lên rằng các ghi chú của chúng ta không là thành phần của tập thể XML của phòng *Quản lý nguồn lực*.

Để định nghĩa không gian tên mới, chúng ta dùng thuộc tính `xmlns:prefix="yournamespace"`, prefix là tiền tố chúng ta muốn dùng cho không gian tên, `yournamespace` là không gian tên do chúng ta đặt. Ví dụ chúng ta định nghĩa một không gian tên `http://www.hr.com/human_resources` có tiền tố là `hr` cho phòng *Quản lý nguồn lực* như ví dụ dưới đây:

Ví dụ 2.2:

```
<?xml version="1.0" encoding="UTF-8"?>
<document xmlns:hr="http://www.hr.com/human_resources">
    <employee>
        <name>
```

```

<firstname>Nguyen</firstname>
<lastname>Son Nam</lastname>
</name>
<hiredate>July 15, 2015</hiredate>
<projects>
  <project>
    <product>Printer</product>
    <id>HP2300</id>
    <price>$111.00</price>
  </project>
  <project>
    <product>Laptop</product>
    <id>DELL1100</id>
    <price>$989.00</price>
  </project>
</projects>
</employee> ...
</document>
```

2.3.3. Thuộc tính không gian tên

Thuộc tính không gian tên được đặt trong thẻ mở của phần tử và có cú pháp như sau:

```
xmlns:prefix="yournamespace"
```

Trong ví dụ trên, thuộc tính không gian tên là một địa chỉ internet:

```
xmlns:hr="http://www.hr.com/human_resources"
```

Để định nghĩa không gian tên, chúng ta gán thuộc tính `xmlns:prefix` tới một định danh duy nhất, trong XML thường là URI để chỉ cho trình xử lý XML tham khảo tới DTD cho không gian tên.

2.3.4. Định nghĩa không gian tên với URI

Đặc tả XML mở rộng ý tưởng của URL (Uniform Resource Locator - dùng định vị trí tài nguyên trên internet, chính là địa chỉ xác định tài nguyên trên internet) chuẩn thành URI (Uniform Resource Identifier – định danh tài nguyên trên internet). Trong HTML và web, chúng ta dùng URL, trong XML dùng URI.

Khi định nghĩa không gian tên với thuộc tính `xmlns:prefix`, thường chúng ta gán URI tới thuộc tính đó. Ví dụ sau tạo một không gian tên <http://www.w3.org/1999/xhtml> có tiền tố là `xhtml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/tr/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns:xhtml="http://www.w3.org/1999/xhtml">
...
```

URI của không gian tên có thể chỉ đến DTD hoặc giản đồ XML để định nghĩa cú pháp cho các phần tử XML chúng ta có thể dùng trong không gian tên đó.

Giả sử sau khi tạo không gian tên *hr* chúng ta có thể đặt làm tiền tố cho mỗi thẻ và tên thuộc tính trong không gian tên này với *hr*: như sau:

Ví dụ 2.3:

```
<hr:employee xmlns:hr="http://www.hr.com/humna_resources">
  <hr:name>
    <hr:firstname>Nguyen</hr:firstname>
    <hr:lastname>Son Nam</hr:lastname>
  </hr:name>
  <hr:hiredate>July 15, 2015</hr:hiredate>
  <hr:projects>
    <hr:project>
      <hr:product>Printer</hr:product>
      <hr:id>HP2300</hr:id>
      <hr:price>$111.00</hr:price>
    </hr:project>
    <hr:project>
      <hr:product>Laptop</hr:product>
      <hr:id> ELL1100</hr:id>
      <hr:price>$989.00</hr:price>
    </hr:project>
  </hr:projects>
</hr:employee>
```

Tài liệu này làm rõ một điều là các thẻ này đến từ phòng *Quản lý nguồn lực*. Nói cách khác là sử dụng không gian tên tránh được vấn đề xung đột về tên và thuộc tính. Tất cả tên thuộc tính và thẻ từ không gian tên *hr* thuộc không gian tên của riêng chúng ta, vì thế chúng ta có thể thêm không gian tên của riêng chúng ta vào tài liệu, cho phép chúng ta dùng các phần tử của riêng chúng ta mà không sợ bị xung đột.

Ví dụ tạo không gian tên *boss*, tạo phần tử *<comment>* và các thuộc tính của không gian này trong ví dụ 2.3 như sau:

Ví dụ 2.4:

```
<hr:employee xmlns:hr="http://www.hr.com/humna_resources"
              xmlns:boss="http://www.ht.com/big_boss">
  <hr:name>
    <hr:firstname>Nguyen</hr:firstname>
    <hr:lastname>Son Nam</hr:lastname>
  </hr:name>
  <hr:hiredate>July 15, 2015</hr:hiredate>
  <boss:comment boss:date="10/15/2016">
    Needs much supervision.</boss:comment>
  <hr:projects>
    <hr:project>
      <hr:product>Printer</hr:product>
      <hr:id>HP2300</hr:id>
      <hr:price>$111.00</hr:price>
    </hr:project>
    <hr:project>
      <hr:product>Laptop</hr:product>
```

```

<hr:id> ELL1100</hr:id>
<hr:price>$989.00</hr:price>
</hr:project>
</hr:projects>
</hr:employee>

```

2.3.5. Tạo không gian tên cục bộ

Thuộc tính `xmlns:prefix` có thể dùng cho bất kỳ phần tử nào không chỉ riêng phần tử gốc. Thuộc tính này định nghĩa không gian tên cho phần tử hiện tại và bất kỳ phần tử con nào. Như vậy chúng ta không nên dùng không gian tên làm tiền tố cho đến khi chúng ta định nghĩa nó với thuộc tính `xmlns:prefix`.

Ví dụ sau tạo không gian tên cục bộ `:boss` trong phần tử `<comment>`.

Ví dụ 2.5:

```

<?xml version="1.0" encoding="UTF-8"?>
<hr:employee xmlns:hr="http://www.hr.com/humna_resources">
    <hr:name>
        <hr:firstname>Nguyen</hr:firstname>
        <hr:lastname>Vuong</hr:lastname>
    </hr:name>
    <hr:hiredate>October 15, 2015</hr:hiredate>
    <boss:comment xmlns:boss="http://www.duperbigco.com
                    /big_boss" boss:date="10/15/2016">
        Needs much supervision.</boss:comment>
    <hr:projects>
        <hr:project>
            <hr:product>Printer</hr:product>
            <hr:id>HP2300</hr:id>
            <hr:price>$111.00</hr:price>
        </hr:project>
        <hr:project>
            <hr:product>Laptop</hr:product>
            <hr:id> ELL1100</hr:id>
            <hr:price>$989.00</hr:price>
        </hr:project>
    </hr:projects>
</hr:employee>

```

2.3.6. Không gian tên mặc định

Khi định nghĩa không gian tên mặc định cho một phần tử (gốc) chúng ta sẽ không cần phải dùng tiền tố cho tất cả các phần tử con. Các phần tử và thuộc tính không có tiền tố không gian tên nghĩa là nó có không gian tên mặc định của phần tử cha.

Ví dụ sau định nghĩa không gian tên mặc định cho phần tử gốc `<html>` là `xmlns="http://www.w3.org/1999/xhtml"`, các phần tử `<head>`, `<title>` sẽ có không gian tên mặc định là `xmlns="http://www.w3.org/1999/xhtml"`:

Ví dụ 2.6:

```

<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>

```

```
<title>
    Using XHTML and MathML together
</title>
</head>
....
</html>
```

BÀI TẬP CHƯƠNG 2

Từ dữ liệu sau hãy tạo tài liệu XML và kiểm tra tính hợp khuôn dạng của tài liệu

1. Tạo tập tin OrderDetail.xml

OrderDetail

- OrderDate: 2006-6-6
- Customer: Peter Collingwood
- Item1
 - o ProductID: 3
 - o Quantity: 3
- Item2
 - o ProductID: 1
 - o Quantity: 5

2. Tạo tập tin Customer.xml

Customer

a.

FirstName: Bruce

MiddleInitial:

LastName: Lee

b.

FirstName: Long

MiddleInitial: Tieu

LastName: Ly

...

3.Tạo tập tin QLBanHang.xml

Order

OrderDate: 2006-6-6

Customer: Helen Mooney

ProductID: 2

Quantity: 2

ProductID: 4

Quantity: 3

....

Catalog

CategoryName: Beverages

ProductID: 1

ProductName: Coca-cola

CategoryName: Furniture

ProductID: 2

ProductName: Chair

ProductID: 2

ProductName: Desk

Chương 3

ĐỊNH NGHĨA KIỂU TÀI LIỆU – DTD

Do XML cho phép chúng ta tạo thẻ cho riêng chúng ta, điều đó có nghĩa là trình xử lý XML không thể kiểm tra thẻ của chúng ta trừ khi chúng ta cho nó biết làm cách nào. Trong XML chúng ta chỉ ra cái gì là hợp lệ, cái gì là không hợp lệ bằng cách chỉ ra cú pháp sẽ cho phép đối với tài liệu. Có hai cách để thực hiện điều này là dùng DTD (Document Type Definition - định nghĩa kiểu tài liệu) và dùng lược đồ XML. Chương này sẽ nói về DTD.

3.1. Định nghĩa kiểu tài liệu (DTD)

DTD cung cấp cách để kiểm tra sự hợp lệ của tài liệu. Cú pháp của DTD được xây dựng trong đặc tả XML 1.0. Hầu hết trình xử lý XML sử dụng DTD trong tài liệu XML.

3.1.1. Giới thiệu về DTD

Một tài liệu XML cần phải hợp khuôn dạng mới được xem đúng là tài liệu XML. Chúng ta cần phải cho trình xử lý XML vài cách để kiểm tra cú pháp của tài liệu XML nhằm đảm bảo rằng dữ liệu vẫn còn hợp lệ. Xem lại ví dụ 2.1 (*vidu2.1.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <employee>
        <name>
            <firstname>Nguyen</firstname>
            <lastname>Son Nam</lastname>
        </name>
        <hiredate>July 15, 2015</hiredate>
        <projects>
            <project>
                <product>Printer</product>
                <id>HP2300</id>
                <price>$111.00</price>
            </project>
            <project>
                <product>Laptop</product>
                <id>DELL1100</id>
                <price>$989.00</price>
            </project>
        </projects>
    </employee>
...
</document>
```

Giả sử khi số lượng nhân viên tăng lên đáng kể, việc nhập liệu có thể bị sai, nhưng làm thế nào trình xử lý XML biết phần tử *<project>* phải chứa ít nhất một phần tử *<product>* trừ khi chúng ta cho nó biết; Làm thế nào cho trình xử lý XML biết mỗi phần tử *<employee>* phải chứa phần tử *<name>*? Để

làm những điều này, chúng ta cần sử dụng DTD. DTD chỉ ra cấu trúc của tài liệu XML chứ không phải dữ liệu trong tài liệu đó.

Chúng ta định nghĩa cú pháp của tài liệu XML bằng cách dùng DTD và khai báo định nghĩa đó trong tài liệu bằng cách khai báo kiểu tài liệu. Dùng phần tử `<!DOCTYPE>` để tạo DTD và DTD xuất hiện trong phần tử đó. Phần tử này có thể có một số dạng khác nhau như sau:

- `<!DOCTYPE rootname [DTD]>`
- `<!DOCTYPE rootname SYSTEM URI>`
- `<!DOCTYPE rootname SYSTEM URI [DTD]>`
- `<!DOCTYPE rootname PUBLIC identifier URI>`
- `<!DOCTYPE rootname PUBLIC identifier URI [DTD]>`

Trong cú pháp trên:

- *rootname*: Tên phần tử gốc
- *URI*: Định danh của DTD bên ngoài tài liệu XML hiện thời trên không gian mạng internet.

3.1.2. DTD nội tại trong XML (Internal DTD)

Một DTD được xem như là một DTD nội tại nếu các phần tử được khai báo bên trong XML tập tin. Để xem nó như là DTD nội tại, thuộc tính *standalone* trong khai báo XML phải được thiết lập là *yes*. Nghĩa là, khai báo làm việc độc lập với nguồn ngoại vi.

Cú pháp

Cú pháp của DTD nội tại trong XML như sau:

```
<!DOCTYPE root-element [element-declarations]>
```

Ở đây, *root-element* là tên của phần tử gốc và *element-declarations* là nơi chúng ta khai báo các phần tử.

Ví dụ sau cho thấy một định nghĩa DTD nội tại cho tài liệu quản lý nhân viên.

Ví dụ 3.1: (vidu3-1.xml)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document[
  <!ELEMENT document (employee)*>
  <!ELEMENT employee (name, hiredate, projects)>
  <!ELEMENT name (firstname, lastname)>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT hiredate (#PCDATA)>
  <!ELEMENT projects (project)*>
  <!ELEMENT project (product, id, price)>
  <!ELEMENT product (#PCDATA)>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT price (#PCDATA)>]
<document xmlns:hr="http://www.hr.com/human_resources">
  <employee>
    <name>
      <firstname>Nguyen</firstname>
      <lastname>Son Nam</lastname>
    </name>
  </employee>
</document>
```

```

<hiredate>July 15, 2015</hiredate>
<projects>
    <project>
        <product>Printer</product>
        <id>HP2300</id>
        <price>$111.00</price>
    </project>
    <project>
        <product>Laptop</product>
        <id>DELL1100</id>
        <price>$989.00</price>
    </project>  </projects>
</employee>
</document>

```

Trong đoạn code trên:

Phần đầu khai báo - Bắt đầu khai báo XML với lệnh sau:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

DTD - Ngay sau phần đầu khai báo là phần khai báo kiểu tài liệu, thường là DOCTYPE:

```
<!DOCTYPE document [
```

Khai báo DOCTYPE có một dấu chấm than (!) tại phần đầu của tên phần tử. DOCTYPE thông báo cho XML Parser rằng một DTD được liên kết với tài liệu XML này.

Phần thân DTD - Khai báo DOCTYPE được theo sau bởi phần thân của DTD đó, ở đó chúng ta khai báo các phần tử, thuộc tính, thực thể.

Phần cuối khai báo - Cuối cùng, khu vực khai báo DTD được kết thúc bởi sử dụng một dấu ngoặc mòc vuông đóng và một dấu lớn hơn (]>). Nó kết thúc khai báo DTD và theo sau là tài liệu XML.

Qui tắc cho DTD nội tại trong XML

- DTD phải xuất hiện ở phần bắt đầu của tài liệu (chỉ được đặt trước bởi XML header). Nó không được cho phép xuất hiện ở bất cứ đâu khác bên trong tài liệu.
- Tương tự như khai báo DOCTYPE, khai báo phần tử phải bắt đầu với một dấu chấm than (!).
- Name trong DTD phải kết nối với kiểu phần tử của phần tử gốc.

3.1.3. DTD bên ngoài XML (External DTD)

Một DTD bên ngoài (DTD ngoại vi) được khai báo bên ngoài XML tập tin. Chúng được truy cập bằng việc xác định các thuộc tính hệ thống mà có thể là .dtd tập tin hoặc một URL hợp lệ. Để xem nó như là DTD ngoại vi, thuộc tính *standalone* trong khai báo XML phải được thiết lập là *no*. Nghĩa là, khai báo bao thông tin từ nguồn bên ngoài.

Cú pháp

Sau đây là cú pháp cho DTD bên ngoài (ngoại vi):

```
<!DOCTYPE root-element SYSTEM "tập tin-name">
```

Ở đây, *tập tin-name* là tập tin với đuôi .dtd.

Ví dụ 3.2:

Ví dụ sau viết lại ví dụ 3.1, trong đó tách phần DTD ra thành tập tin .dtd là ví dụ vidu3-2.dtd và liên kết vào tài liệu XML vidu2-1.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE document SYSTEM "vidu3-2.dtd">
<document>
    ...
</document>
```

Nội dung của DTD tập tin là **vidu3-2.dtd** như sau:

```
<!ELEMENT document (employee)*>
<!ELEMENT employee (name, hiredate, projects)>
<!ELEMENT name (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT hiredate (#PCDATA)>
<!ELEMENT projects (project)*>
<!ELEMENT project (product, id, price)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Ví dụ trên giả sử rằng vidu3-2.dtd và vidu2-1.xml nằm cùng thư mục. Vì thế chúng ta chỉ cần chỉ ra tên tập tin dtd trong phần tử <!DOCTYPE> như sau:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE document SYSTEM "vidu3-2.dtd">
```

Chúng ta có thể đặt tập tin dtd ở bất cứ đâu, nếu chúng ta đặt trên một địa chỉ internet thì chúng ta phải chỉ ra đường dẫn URI đầy đủ trong <!DOCTYPE>, ví dụ:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE document SYSTEM
    "http://www.viduxml.com/dtds/vidu3-2.dtd">
```

Chúng ta phải cung cấp đường dẫn giống như vậy cho tập tin .dtd bên ngoài nếu chúng ta muốn dùng bộ kiểm tra XML trực tuyến.

Tạo DTD dùng chung

Muốn tạo DTD dùng chung, trước hết chúng ta dùng từ khóa PUBLIC trong phần tử <!DOCTYPE>. Để dùng từ khóa PUBLIC, chúng ta phải tạo một danh định dùng chung chính thức (Formal Public Identifier - FPI) là một chuỗi văn bản nằm trong dấu nháy có 4 trường phân cách nhau bằng dấu //. Nguyên tắc tạo các trường trong FPI:

- Trường thứ nhất: Chỉ ra rằng DTD dùng cho chuẩn chính thức. Với DTD chúng ta tạo cho riêng chúng ta thì trường này là -, nếu chúng ta tạo DTD có phần thân không là chuẩn chính thức thì trường này là +. Với phần thân có chuẩn chính thức thì trường này tham khảo đến chuẩn của chính nó (ví dụ ISO/IEC 19775:2003).
- Trường thứ 2 lưu tên của nhóm hoặc cá nhân chịu trách nhiệm cho DTD (ví dụ W3C).
- Trường thứ 3 chỉ ra kiểu tài liệu DTD (ví dụ version 1.0).

- Trường thứ 4 chỉ ra ngôn ngữ dùng viết DTD (ví dụ EN là English).

- Khai báo DTD dùng chung như sau:

```
<!DOCTYPE rootname PUBLIC FPI URI>
```

Ví dụ:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document PUBLIC "-//DTD4ALL//Custom DTD version
1.0//EN" "http://www.viduxml.com/dtds/vidu3-2.dtd">
<document>
...
</document>
```

3.1.4. Quản lý không gian tên trong DTD

Một vấn đề quan trọng khác khi chúng ta tạo DTD là cách quản lý không gian tên. Như chúng ta biết, không gian tên là tên tiền tố đặt trước tên phần tử hoặc thuộc tính với dấu hai chấm. Ví dụ trước đây chúng ta dùng phần tử `<employee>` thì nếu chúng ta dùng không gian tên là `emp` thì phần tử được viết lại như sau: `<emp:employee>`.

Ví dụ 3.3: (*vidu3-3.xml*) Minh họa cách tạo và sử dụng không gian tên trong XML với DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document[
<!ELEMENT emp:document (emp:employee)*>
<!ATTLIST emp:document xmlns:emp CDATA#FIXED
                      "http://www.viduxml.com/dtds">
<!ELEMENT emp:employee
          (emp:name, emp:hiredate, emp:projects)>
<!ELEMENT emp:name (emp:firstname, emp:lastname)>
<!ELEMENT emp:firstname (#PCDATA)>
<!ELEMENT emp:lastname (#PCDATA)>
<!ELEMENT emp:hiredate (#PCDATA)>
<!ELEMENT emp: projects (project)*>
<!ELEMENT emp: project (product, id, price)>
<!ELEMENT emp:product (#PCDATA)>
<!ELEMENT emp:id (#PCDATA)>
<!ELEMENT emp:price (#PCDATA)>]>
<emp:document
  xmlns:hr="http://www.hr.com/human_resources">
  <emp:employee>
    <emp:name>
      <emp:firstname>Nguyen</emp:firstname>
      <emp:lastname>Son Nam</emp:lastname>
    </emp:name>
    <emp:hiredate>July 15, 2015</emp:hiredate>
    <emp:projects>
      <emp:project>
        <emp:product>Printer</emp:product>
        <emp:id>HP2300</emp:id>
        <emp:price>$111.00</emp:price>
      </emp:project>
      <emp:project>
```

```

<emp:product>Laptop</emp:product>
<emp:id>DELL1100</emp:id>
<emp:price>$989.00</emp:price>
</emp:project>
</emp:projects>
</emp:employee>
...
</emp:document>

```

3.2. Các thành phần của DTD

3.2.1. Phần tử (Element)

Phần tử trong DTD được khai báo như sau:

```

<!ELEMENT name content_model>
  name: Tên phần tử đang khai báo
  content_model: Mô hình nội dung của phần tử

```

content_model chỉ ra nội dung nào phần tử được phép có, ví dụ như cho phép có phần tử con hoặc dữ liệu văn bản, hoặc chúng ta tạo phần tử rỗng bằng cách dùng từ khóa EMPTY hoặc cho phép bất kỳ nội dung nào bằng cách dùng từ khóa ANY.

Ví dụ sau khai báo phần tử <document>

```

<!DOCTYPE document [
  <!ELEMENT document (employee)*>
  ..
]>

```

Với khai báo trên, phần tử <document> có thể chứa phần tử <employee>

Quản lý nội dung bất kỳ

Nếu chúng ta cho phần tử có mô hình nội dung là ANY thì nó có thể chứa bất kỳ nội dung nào. Ví dụ phần tử <document> sau đây có thể chứa bất kỳ nội dung gì:

```

<!DOCTYPE document [
  <!ELEMENT document ANY>
  ..
]>

```

Chỉ ra phần tử con

Chúng ta có thể chỉ ra một phần tử có thể chứa những phần tử con nào trong mô hình nội dung của phần tử đó. Chỉ ra phần tử con bằng cách liệt kê tên của phần tử con trong dấu ngoặc đơn.

Xét *vidu3-1.xml*, khai báo phần tử <document> chứa phần tử <employee> và dấu * nghĩa là phần tử <document> có thể chứa số phần tử <employee> bất kỳ. Phần tử <employee> chứa các phần tử <name>, <hiredate>, <project>,...

Liệt kê nhiều phần tử trong mô hình nội dung theo cách này gọi là tạo tuần tự, dùng dấu phẩy để phân cách các phần tử. Sau đó các phần tử sẽ xuất hiện theo tuần tự đó trong tài liệu XML. Trong ví dụ *vidu3-1.xml*, khai báo:

```

<!ELEMENT employee (name, hiredate, projects)>

```

thì trong phần tử <employee>, phần tử <name> phải xuất hiện trước tiên, kế đến là <hiredate> và sau cùng là <projects>.

Quản lý nội dung văn bản

Trong DTD, văn bản không phải là thẻ được xem là dữ liệu ký tự phân tích được. Để chỉ ra dữ liệu ký tự này, chúng ta dùng #PCDATA. Đây là cách để tham khảo đến dữ liệu văn bản trong DTD, nó không thể nói bất cứ điều gì về định dạng thật sự của văn bản mặc dù điều này có thể quan trọng nếu chúng ta làm việc với số. Chính vì sự thiếu chính xác này là một trong những lý do lược đồ XML (XML Schema) được giới thiệu như là một cách thay thế DTD.

Ví dụ các phần tử chứa văn bản được khai báo với #PCDATA như sau:

Ví dụ 3.4:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document[
  <!ELEMENT document (employee)*>
  <!ELEMENT employee (name, hiredate, projects)>
  <!ELEMENT name (firstname, lastname)>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT hiredate (#PCDATA)>
  <!ELEMENT projects (project)*>
  <!ELEMENT project (product, id, price)>
  <!ELEMENT product (#PCDATA)>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT price (#PCDATA)>]>
<document> ... </document>
```

Chỉ ra nhiều phần tử con

Có một số cách tùy chọn để khai báo một phần tử có thể chứa các phần tử con.

Ví dụ khai báo phần tử <document> chỉ chứa duy nhất một phần tử <employee>:

```
<!ELEMENT document (employee)>
```

Ví dụ khai báo phần tử <document> chứa danh sách các phần tử con:

```
<!ELEMENT document (employee, contractor, partner)>
```

Chúng ta cũng có thể sử dụng một số ký hiệu mang ý nghĩa đặc biệt. Như dấu * có nghĩa phần tử có thể chứa nhiều phần tử con hoặc không chứa phần tử nào.

Ví dụ phần tử <document> chứa nhiều (hoặc không) phần tử <employee>:

```
<!ELEMENT document (employee)*>
```

Danh sách một số ký hiệu mang ý nghĩa đặc biệt:

- X+: Phần tử X có thể xuất hiện một hoặc nhiều lần.
- X*: Phần tử X có thể xuất hiện nhiều lần hoặc không lần nào.
- X?: Phần tử X có thể xuất hiện một lần hoặc không lần nào.
- X, Y: Phần tử Y xuất hiện sau phần tử X.
- X|Y: Xuất hiện phần tử X hoặc Y, không thể cả 2.

Cho phép phần tử rỗng

Các phần tử có thể không cần có nội dung gì khi đó phần tử được gọi là rỗng. Chúng ta có thể khai báo phần tử rỗng bằng cách dùng DTD với từ khóa EMPTY. Phần tử không thể chứa bất kỳ nội dung gì nhưng nó có thể chứa thuộc tính.

Ví dụ khai báo phần tử `<quantity>` là rỗng như sau:

```
<!ELEMENT quantity EMPTY>
```

Ví dụ sau đây khai báo phần tử `<employee>` có thể chứa phần tử rỗng `<intern>` và chỉ ra đây là phần tử tùy chọn:

Ví dụ 3.5:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document [
  <!ELEMENT document (employee)*>
  <!ELEMENT employee (name, hiredate, projects)>
  <!ELEMENT name (firstname, lastname)>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT hiredate (#PCDATA)>
  <!ELEMENT projects (project)*>
  <!ELEMENT project (product, id, price)>
  <!ELEMENT product (#PCDATA)>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT price (#PCDATA)>
  <!ELEMENT intern EMPTY]>
<document> <employee>
  <name> <firstname>Nguyen</firstname>
    <lastname>Son Nam</lastname>
  </name>
  <hiredate>July 15, 2015</hiredate>
  <projects>
    <project>
      <product>Printer</product>
      <id>HP2300</id>
      <price>$111.00</price>
    </project>
    <project>
      <product>Laptop</product>
      <id>DELL1100</id>
      <price>$989.00</price>
    </project>
  </projects>
</employee>
...
</document>
```

Cách ghi chú trong DTD

DTD có thể rất dài và phức tạp, do đó chúng ta cần dùng ghi chú cho dễ nhớ. Chúng ta có thể dùng cách ghi chú XML chuẩn trong DTD (dùng dấu `<!--` và `-->`).

Ví dụ 3.6:

```
<!DOCTYPE document [
  <!--Khai bao phan tu goc-->
  <!ELEMENT document (employee)*>
  <!--Moi phan tu employee chua phan tu name, hiredate
  va projects -->
  <!ELEMENT employee (name, hiredate, projects)>
```

3.2.2. Thuộc tính (Attribute)

Thuộc tính là cặp tên và giá trị mà chúng ta có thể dùng trong thẻ mở đầu hoặc thẻ rỗng để cung cấp thêm thông tin cho phần tử. Giả sử để thêm thuộc tính *supervisor* vào phần tử *<employee>*, thuộc tính này có thể có giá trị “yes” hoặc “no”, chúng ta viết như sau:

```
<document>
  <employee supervisor="yes">
    <name>
      <firstname>Nguyen</firstname>
      <lastname>Son Nam</lastname>
    </name>...
```

Như vậy việc thêm thuộc tính rất đơn giản, nhưng nếu chúng ta không khai báo nó trong DTD (hoặc giản đồ XML) thì tài liệu của chúng ta không hợp lệ.

Để khai báo một danh sách các thuộc tính cho phần tử, chúng ta dùng phần tử *<!ATTLIST>* trong DTD. Cú pháp tổng quát như sau:

```
<!ATTLIST element_name
  attribute_name type default_value
  attribute_name type default_value ...>
  element_name: Tên phần tử muốn khai báo thuộc tính.
  attribute_name: Tên thuộc tính khai báo.
  type: Kiểu thuộc tính.
  default_value: Giá trị mặc định của thuộc tính.
```

Ví dụ 3.7:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document[
  <!ELEMENT document (employee)*>
  <!ELEMENT employee (name, hiredate, projects)>
  <!ELEMENT name (firstname, lastname)>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT hiredate (#PCDATA)>
  <!ELEMENT projects (project)*>
  <!ELEMENT project (product, id, price)>
  <!ELEMENT product (#PCDATA)>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT price (#PCDATA)>
  <!ATTLIST employee supervisor CDATA #IMPLIED]>>
<document>  <employee supervisor ="no">
  <name>  <firstname>Nguyen</firstname>
          <lastname>Son Nam</lastname>  </name>
  <hiredate>July 15, 2015</hiredate>
  <projects> <project> <product>Printer</product>
              <id>HP2300</id> <price>$111.00</price>
              </project>  <project>
                <product>Laptop</product>  <id>DELL1100</id>
                <price>$989.00</price>
              </project>  </projects>  </employee>...
</document>
```

Nếu chúng ta muốn khai báo nhiều thuộc tính cho phần tử, chúng ta chỉ việc khai báo thêm thuộc tính trong phần tử. Xem ví dụ 3.8, phần tử `<employee>` có các thuộc tính: supervisor, division, fulltime.

Ví dụ 3.8:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document [
<!ELEMENT document (employee)*>
<!ELEMENT employee (name, hiredate, projects)>
<!ELEMENT name (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT hiredate (#PCDATA)>
<!ELEMENT projects (project)*>
<!ELEMENT project (product, id, price)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST employee
                     supervisor CDATA #IMPLIED
                     division CDATA #IMPLIED
                     fulltime CDATA #IMPLIED]>
<document>
  <employee supervisor ="yes" division="finance"
            fulltime="yes">
    <name> <firstname>Nguyen</firstname>
           <lastname>Son Nam</lastname>
    </name>
    <hiredate>July 15, 2015</hiredate>
    <projects>
      <project>
        <product>Printer</product>
        <id>HP2300</id> <price>$111.00</price>
      </project>
      <project>
        <product>Laptop</product>
        <id>DELL1100</id> <price>$989.00</price>
      </project>
    </projects>
  </employee>
  ...
</document>
```

Khi khai báo thuộc tính trong DTD, những thiết lập giá trị mặc định chúng ta có thể dùng trong phần tử `<!ATTLIST>` là:

- *value*: Chỉ ra giá trị văn bản và phải được đặt trong dấu nháy.
 - *#IMPLIED*: Chỉ ra đây là thuộc tính tùy chọn.
 - *#FIXED value*: Thiết lập giá trị cho thuộc tính là *value*.
 - *#REQUIRED*: Chỉ ra đây là thuộc tính bắt buộc và phải có giá trị.
- Các kiểu thuộc tính có thể dùng là:
- *CDATA*: Kiểu dữ liệu văn bản, không có thẻ đánh dấu.

- *ENTITY*: Chỉ ra tên thực thể. Thuộc tính có kiểu ENTITY có nghĩa là thuộc tính được gán giá trị là tên của một thực thể do chúng ta khai báo trước đó.
- *ENTITIES*: Chỉ ra nhiều tên thực thể, các thực thể phân cách nhau bằng ký tự trắng.
- *Enumerated*: Chỉ ra một giá trị từ danh sách giá trị.
- *ID*: Chỉ ra thuộc tính định danh. XML xem kiểu định danh ID với một ý nghĩa khá đặc biệt, giá trị này của thuộc tính dùng làm tên duy nhất để định danh một phần tử nào đó trong tài liệu XML. Vì lý do này, các bộ xử lý XML luôn được giả sử và đảm bảo rằng không được có hai phần tử trùng nhau tên định danh đối với thuộc tính mang kiểu ID. Giá trị gán cho thuộc tính định danh cũng phải phù hợp quy tắc đặt tên của XML. Mặc dù mang kiểu ID nhưng tên thuộc tính không nhất thiết phải là “ID”. Tên thuộc tính được đặt bởi một tên gợi nhớ nào đó rõ ràng hơn bởi vì ID chỉ là kiểu dữ liệu.
- *IDREF*: Kiểu thuộc tính IDREF cho phép xác định thông tin liên quan đến cấu trúc của tài liệu, chính xác hơn là thông tin về quan hệ giữa các phần tử trong tài liệu. Thuộc tính có kiểu IDREF nắm giữ giá trị ID của các phần tử khác.
- *IDREFS*: Giữ nhiều giá trị ID của các phần tử, các giá trị phân cách nhau bằng ký tự trắng.
- *NMTOKEN*: Chỉ ra văn bản tạo thành tên XML. Văn bản này có thể là một hoặc nhiều ký tự chữ, số, gạch ngang, gạch dưới, hai chấm và dấu chấm. Thuộc tính có kiểu dữ liệu này chỉ được gán các giá trị hợp với quy tắc đặt tên của XML và không có khoảng trắng. Sử dụng kiểu NMTOKEN cũng có nghĩa thuộc tính chỉ có thể mang giá trị là một từ đơn bởi khoảng trắng và các ký tự phân cách khác không được phép thể hiện trong giá trị của thuộc tính.
- *NMTOKENS*: Kiểu này cho phép giá trị của thuộc tính là một chuỗi bao gồm nhiều NMTOKEN có tên theo quy tắc của XML và phân cách nhau bằng khoảng trắng.
- *NOTATION*: Chỉ ra tên ký hiệu giữ mô tả định dạng. Có thể coi đây là kiểu dữ liệu ghi nhớ. Chúng ta định nghĩa trước các phần tử ghi nhớ, sau đó dùng chings để gán cho các thuộc tính được khai báo là NOTATION.

Một định nghĩa ghi nhớ thường dùng để xác định dạng thức của các dữ liệu phi XML. Ví dụ “image/gif” là tập tin hình ảnh, “text/html” là tập tin HTML hoặc thuần text, “application/xml” là tập tin tài liệu XML dành cho các ứng dụng...

Chỉ ra giá trị mặc định trực tiếp

Xét ví dụ bên dưới (vidu3-9.xml) chỉ ra giá trị mặc định trực tiếp. Nếu thuộc tính có giá trị mặc định thì khi chúng ta không gán giá trị khác cho thuộc tính đó trong phần tử thì thuộc tính tự động nhận giá trị mặc định. Ví dụ này thiết lập cho thuộc tính *supervisor* giá trị mặc định là “no”, *division* giá trị mặc định là “*humanresource*” và *fulltime* giá trị mặc định là “yes”:

Ví dụ 3.9:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document [
  <!ELEMENT document (employee)*>
  <!ELEMENT employee (name, hiredate, projects)>
  <!ELEMENT name (firstname, lastname)>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT hiredate (#PCDATA)>
  <!ELEMENT projects (project)*>
  <!ELEMENT project (product, id, price)>
  <!ELEMENT product (#PCDATA)>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT price (#PCDATA)>
  <!ATTLIST employee supervisor CDATA "no"
    division CDATA "humanresource" fulltime CDATA "yes">]>
<document>
...
</document>
```

Chỉ ra giá trị mặc định là #REQUIRED

Ví dụ để chỉ ra rằng tất cả các phần tử `<employee>` phải có thuộc tính `supervisor` chúng ta viết như sau:

Ví dụ 3.10:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document [
  <!ELEMENT document (employee)*>
  <!ELEMENT employee (name, hiredate, projects)>
  <!ELEMENT name (firstname, lastname)>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT hiredate (#PCDATA)>
  <!ELEMENT projects (project)*>
  <!ELEMENT project (product, id, price)>
  <!ELEMENT product (#PCDATA)>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT price (#PCDATA)>
  <!ATTLIST employee supervisor CDATA #REQUIRED>
]>  <document>
    <employee supervisor ="no">
        <name>
            <firstname>Lam Thi</firstname>
            <lastname>My Dung</lastname>
        </name>....
    </employee>
</document>
```

Sử dụng kiểu thuộc tính CDATA

Đây là dữ liệu văn bản nằm trong giá trị thuộc tính được đọc và phân tích bởi trình xử lý XML. Không nên sử dụng các ký tự <, “, và & trong giá trị CDATA bởi vì những ký tự này giống với các ký hiệu đánh dấu. Nếu muốn

dùng những ký tự này chúng ta phải dùng tham khảo thực thể định nghĩa trước, ví dụ <, ", và & để thay thế. Định nghĩa CDATA cho thuộc tính tương tự như định nghĩa kiểu PCDATA cho phần tử của DTD. Ví dụ về khai báo và định nghĩa CDATA cho thuộc tính có thể thực hiện như sau:

Ví dụ 3.11:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document[
<!ELEMENT document (employee)*>
<!ELEMENT employee (name, hiredate, projects)>
<!ELEMENT name (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT hiredate (#PCDATA)>
<!ELEMENT projects (project)*>
<!ELEMENT project (product, id, price)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST employee supervisor CDATA #IMPLIED]>
<document>
<employee supervisor ="no">
    <name>
        <firstname>Lam Thi</firstname>
        <lastname>My Dung</lastname>
    . </name>...
</employee>
</document>
```

Kiểu CDATA được dùng rất tổng quát và phổ biến trong hầu hết định nghĩa kiểu cho thuộc tính. Mặc dù vậy, thuộc tính còn được định nghĩa bằng những kiểu đặc thù hơn mà chúng ta sẽ tìm hiểu sau đây.

Sử dụng kiểu liệt kê (enumerated)

Kiểu liệt kê không sử dụng một từ khóa riêng nào cả. Thay vào đó, khai báo kiểu liệt kê được quy định bằng một danh sách giá trị có thể gán cho thuộc tính. Mỗi giá trị của thuộc tính trong danh sách phải phù hợp với cách đặt tên của đặc tả XML. Kiểu này thích hợp cho trường hợp chúng ta muốn giới hạn giá trị được phép của thuộc tính. Ví dụ 3.12 (vidu3-12.xml) chỉ ra rằng thuộc tính *supervisor* có thể có 2 giá trị “yes” và “no” trong đó “no” là giá trị mặc định:

Ví dụ 3.12:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document[
<!ELEMENT document (employee)*>
<!ELEMENT employee (name, hiredate, projects)>
<!ELEMENT name (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT hiredate (#PCDATA)>
<!ELEMENT projects (project)*>
<!ELEMENT project (product, id, price)>
```

```

<!ELEMENT product (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST employee supervisor (yes|no) "no">]
<document> <employee supervisor ="no">
    <name> <firstname>Lam Thi</firstname>
        <lastname>My Dung</lastname>
    </name> ... </employee>
</document>

```

3.2.3. Thực thể (Entity)

Thực thể đơn giản là danh mục dữ liệu, hay nói cách khác thực thể đơn giản là cách XML tham khảo đến mẫu dữ liệu.

Các thực thể có thể đặt bên trong hoặc bên ngoài tài liệu. Thực thể bên trong nằm cùng với tài liệu XML. Thực thể bên ngoài được lưu trong một tập tin riêng.

Khi chúng ta khai báo thực thể trong DTD, sau đó chúng ta tham khảo tới nó với một tham khảo trong tài liệu XML. Có 5 tham khảo thực thể chung được định nghĩa trước trong XML được liệt kê trong bảng sau:

Tham khảo thực thể chung	Ký tự được thay thế
<	<
>	>
&	&
"	"
'	'

Vì thế chúng ta không cần định nghĩa chúng trong DTD mà chỉ việc sử dụng các tham khảo thực thể này trong tài liệu.

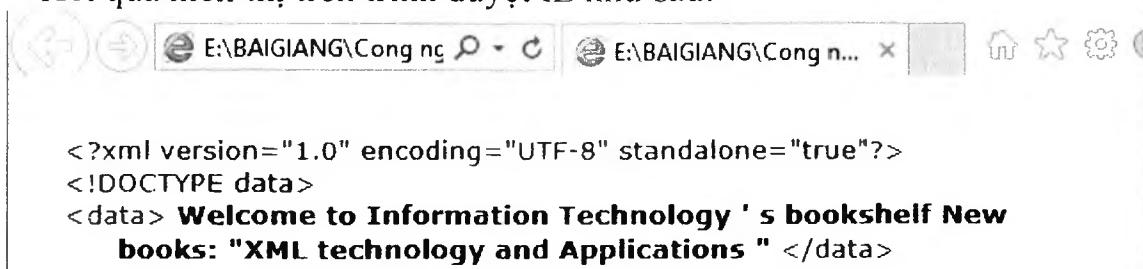
Ví dụ 3.13:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE data [
  <!ELEMENT data (#PCDATA) ]>
  <data> Welcome to Information Technology 's bookshelf New books: "XML technology and Applications" </data>

```

Kết quả hiển thị trên trình duyệt IE như sau:



3.2.3.1. Tạo tham khảo thực thể chung bên trong tài liệu

Dùng phần tử `<!ENTITY>` để khai báo thực thể. Cú pháp:

`<!ENTITY name definition>`

`name:` Tên thực thể

`definition:` Định nghĩa của thực thể

Định nghĩa của thực thể có vài dạng thức khác nhau, dạng đơn giản nhất là văn bản mà trình xử lý XML sẽ thay thế tham khảo thực thể. Ví dụ tham khảo thực thể **copyright** sẽ được thay thế bằng “(c) EnovationSoft Corp. 2005”:

Ví dụ 3.14:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document[
  <!ELEMENT document (employee)*>
  <!ELEMENT employee (copy, name, hiredate, projects)>
  <!ELEMENT copy (#PCDATA)>
  <!ENTITY copyright "(c) EnovationSoft Corp. 2005">
  <!ELEMENT name (firstname, lastname)>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT hiredate (#PCDATA)>
  <!ELEMENT projects (project)*>
  <!ELEMENT project (product, id, price)>
  <!ELEMENT product (#PCDATA)>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT price (#PCDATA)>
  <!ATTLIST employee supervisor (yes|no) "no">]>
<document>
  <employee supervisor ="no">
    <copy>&copyright;</copy>...
  </employee>
</document>
```

3.2.3.2. Tạo tham khảo thực thể chung bên ngoài tài liệu

Dùng từ khóa SYSTEM hoặc PUBLIC khi khai báo tham khảo thực thể bên ngoài. Nếu thực thể bên ngoài dùng riêng thì sử dụng SYSTEM, nếu dùng cho tất cả mọi người thì sử dụng PUBLIC. Cú pháp:

```
<!ENTITY name SYSTEM URI>
<!ENTITY name PUBLIC FPI URI>
```

Ví dụ hãy lưu đoạn văn bản sau như thực thể bên ngoài (*external.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
  "(c) EnovationSoft Corp. 2005"
```

Tạo một tham khảo thực thể bên ngoài có tên là **copyright**, tham khảo tới tài liệu bên ngoài *external.xml* như sau:

```
<!ENTITY copyright SYSTEM "external.xml">
```

Bây giờ chúng ta có thể sử dụng tham khảo thực thể bên ngoài **©right;** như nó là một tham khảo thực thể bên trong. Xem ví dụ 3.15:

Ví dụ 3.15:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE document[
  <!ELEMENT document (employee)*>
  <!ELEMENT employee (copy, name, hiredate, projects)>
  <!ELEMENT name (firstname, lastname)>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT hiredate (#PCDATA)>
  <!ELEMENT projects (project)*>
```

```

<!ELEMENT project (product, id, price)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT copy (#PCDATA)>
<!ATLIST employee supervisor CDATA #IMPLIED>
<!ENTITY copyright SYSTEM "external.xml">]>
<document>
    <employee supervisor ="no">
        <copy>&copyright;</copy>
        <name>
            <firstname>Nguyen</firstname>
            <lastname>Son Nam</lastname>
        </name>
        <hiredate>July 15, 2015</hiredate>
        <projects>
            <project>
                <product>Printer</product>
                <id>HP2300</id>
                <price>$111.00</price>
            </project>...
        </projects>
    </employee>
    ...
</document>

```

3.2.3.3. Tạo thực thể tham số bên trong

Không giống như các tham khảo thực thể chung, các tham khảo tham số không bắt đầu bằng dấu &; mà bắt đầu bằng dấu %. Cú pháp khai báo thực thể tham số bên trong như sau:

```
<!ENTITY %name definition>
```

với *name* là tên thực thể và *definition* là định nghĩa của nó. Chúng ta có thể sử dụng các từ khóa SYSTEM và PUBLIC như đối với tham khảo thực thể bên ngoài:

```
<!ENTITY %NAME SYSTEM URI>
<!ENTITY %NAME PUBLIC FPI URI>
```

Ví dụ sau khai báo thực thể tham số *project* tham khảo đến khai báo phần tử *<project>*:

```
<!ENTITY %project "<!ELEMENT project (product, id,
                                price)>">
```

Khi chúng ta dùng tham khảo thực thể tham số *%project* trong DTD thì nó sẽ được thay thế bằng *<!ELEMENT project (product, id, price)>*.

Ví dụ 3.16: vidu3-16.xml

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document [
    <!ENTITY %project "<!ELEMENT project (product, id,
                                price)>">
    <!ELEMENT document (employee)*>
    <!ELEMENT employee (copy, name, hiredate, projects)>
    <!ELEMENT name (firstname, lastname)>
]
```

```

<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT hiredate (#PCDATA)>
<!ELEMENT projects (project)*>
  %project;
<!ELEMENT product (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST employee supervisor CDATA #IMPLIED]>
<document>
  <employee supervisor ="no">
    <name> <firstname>Nguyen</firstname>
      <lastname>Son Nam</lastname> </name>
    <hiredate>July 15, 2015</hiredate>
    <projects>
      <project> <product>Printer</product>
        <id>HP2300</id>
        <price>$111.00</price>
      </project> ... </projects>
    </employee> ...
  </document>

```

3.2.3.4. Tạo thực thể tham số bên ngoài

Khi chúng ta dùng thực thể tham số bên ngoài trong tập tin DTD, chúng ta có thể tham khảo đến thực thể đó ở bất cứ đâu trong tập tin DTD.

Ví dụ chúng ta có tập tin *vidu3-17.xml* có nội dung như dưới đây, nó có sử dụng tập tin DTD bên ngoài *vidu3-17.dtd*:

Ví dụ 3.17:

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE document SYSTEM "vidu3-17.dtd">
<document>
  <employee supervisor ="no">
    <copy>&copyright;</copy>
    <name> <firstname>Nguyen</firstname>
      <lastname>Son Nam</lastname>
    </name>
    <hiredate>July 15, 2015</hiredate>
    <projects>
      <project> <product>Printer</product>
        <id>HP2300</id> <price>$111.00</price>
      </project>... </projects>
    </employee> ...
  </document>

```

Giả sử trong tập tin *vidu3-17.dtd* chúng ta muốn tạo 3 phần tử trong phần tử *<employee>* là *<supervisorComment>*, *<customerComment>* và *<employeeComment>* để ghi nhận các ghi chú đánh giá về nhân viên. Cả 3 phần tử này có cùng mô hình nội dung là (date, text) với *date* là ngày ghi chú, *text* là nội dung ghi chú. Chúng ta tạo một thực thể tham số mới là *record* cho mô hình nội dung này như sau:

```
<!ENTITY %record "(date, text)">
```

Bây giờ trong tập tin *vidu3-17.dtd* chúng ta tham khảo đến thực thể tham số này khi chúng ta dùng các phần tử *<supervisorComment>*, *<customerComment>* và *<employeeComment>* như sau:

```
<!ENTITY %record "date, text">
<!ELEMENT supervisorComment %record;>
<!ELEMENT customerComment %record;>
<!ELEMENT employeeComment %record;>
<!ELEMENT document (employee)*>
<!ELEMENT employee (name, hiredate, projects,
    supervisorComment*, customerComment*,
    employeeComment*)>
<!ELEMENT name (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT hiredate (#PCDATA)>
<!ELEMENT projects (project)*>
<!ELEMENT project (product, id, price)>
<!ELEMENT product (#PCDATA)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT text (#PCDATA)>
<!ATTLIST employee supervisor CDATA #IMPLIED>
```

Đó là tất cả những gì cần làm để tạo và sử dụng thực thể tham số bên ngoài thực thể tham số *record*.

Tập tin *vidu3-17.dtd* chứa nội dung của DTD có khai báo thực thể tham số bên ngoài. Tập tin *vidu3-17.xml* khai báo sử dụng tập tin *vidu3-17.dtd*.

3.3. Dùng DTD kiểm tra tính hợp lệ của XML

Trong Chương 2, mục IV đã hướng dẫn chúng ta cách sử dụng bộ kiểm tra tính hợp khuôn dạng của tài liệu XML. Tương tự như cách kiểm tra tính hợp khuôn dạng, để kiểm tra tính hợp lệ của tài liệu XML với DTD, chúng ta có thể sử dụng công cụ kiểm tra trực tuyến của W3C tại địa chỉ <http://www.xmlvalidation.com/>

Chúng ta cũng có thể dùng trình soạn thảo XML, chẳng hạn như XMLwriter hoặc Oxygen XML Editor để kiểm tra sự hợp lệ và hợp khuôn dạng của tài liệu. Với trình soạn thảo XMLwriter, chúng ta mở tập tin XML có kèm tập tin DTD và chọn trình đơn XML, chọn *Validate* (hoặc phím F7).

BÀI TẬP CHƯƠNG 3

1. Một chương trình quản lý thông tin sinh viên, và điểm các môn học mà họ đăng ký lưu các thông tin sau: sinh viên gồm mã số sinh viên, họ tên sinh viên, lớp. Một môn học mà sinh viên đăng ký học gồm có thông tin mã môn học, tên môn học, số tín chỉ, với mỗi môn học mà sinh viên đăng ký học thì sẽ có kết quả cuối kì chính là điểm thi mà sinh viên đạt được trong môn đó. Tạo tập tin QLSV.dtd để lưu trữ thông tin trên.
2. Mỗi đơn hàng, người ta cần lưu các thông tin sau: Mã khách hàng, tên khách hàng, địa chỉ liên lạc và một danh sách những mặt hàng người đó mua. Danh sách mặt hàng gồm có nhiều mặt hàng khác nhau, mỗi mặt hàng gồm những thông tin sau: Mã mặt hàng, tên mặt hàng, số lượng, đơn giá. Tạo tập tin QLBH.dtd để lưu trữ thông tin trên.
3. Cho cấu trúc XML lưu trữ thông tin những cuốn sách đã được xuất bản theo từng lĩnh vực. Mỗi lĩnh vực có tên và có thể chưa có sách xuất bản hoặc cũng có thể đã có nhiều cuốn. Mỗi cuốn sách có thông tin một tựa đề duy nhất, một hay nhiều tác giả, mỗi tác giả lại có thông tin mã tác giả, tên tác giả, địa chỉ, số điện thoại với mã tác giả, tên tác giả là duy nhất cho mỗi người, địa chỉ, số điện thoại có thể không có hoặc chỉ có một thông tin độc nhất cho mỗi người. Tạo tập tin QLS.dtd để lưu trữ thông tin trên.
4. Tạo tập tin OrderDetail.dtd mô tả cấu trúc sau:

OrderDetail

- OrderDate: 2006-6-6
- Customer: Peter Collingwood
- Item1
 - o ProductID: 3
 - o Quantity: 3
- Item2
 - o ProductID: 1
 - o Quantity: 5

Kiểm tra tính hợp lệ (valid) của tài liệu OrderDetail.xml ở Bài tập 1, Chương 2

5. Tạo tập tin Customer.dtd mô tả cấu trúc sau:

Customer

- 1. FirstName: Bruce
- MiddleInitial:
- LastName: Lee
- 2. FirstName: Long
- MiddleInitial: Tieu
- LastName: Ly ...

Kiểm tra tính hợp lệ (valid) của tài liệu Customer.xml ở Bài tập 2, Chương 2

6. Tạo tập tin QLBanHang.dtd mô tả cấu trúc sau:

Order

```
OrderDate: 2006-6-6
Customer: Helen Mooney
ProductID: 2
Quantity: 2
ProductID: 4
Quantity: 3
```

...

Catalog

```
CategoryName: Beverages
ProductID: 1
ProductName: Coca-cola
CategoryName: Furniture
ProductID: 2
ProductName: Chair
ProductID: 2
ProductName: Desk
```

Kiểm tra tính hợp lệ (valid) của tài liệu QLBanHang.xml ở Bài tập 3, Chương 2

7. Cho tập tin DatHang.xml như sau:

```
<?xml version="1.0"?>
<DatHang>
    <NgayDat>27-2-2003</NgayDat>
    <KhachHang>Tran Thanh Nhan</KhachHang>
    <SanPham>
        <MaSo>5</MaSo>
        <SoLuong>7</SoLuong>
    </SanPham>
    <SanPham>
        <MaSo>1</MaSo>
        <SoLuong>9</SoLuong>
    </SanPham>
</DatHang>
```

Hãy tạo tập tin DatHang.dtd và đảm bảo rằng DatHang.xml là hợp lệ.

8. Tạo tập tin DSKhachMoi.dtd và đảm bảo rằng tập tin DSKhachMoi.xml là hợp lệ. Tập tin DSKhachMoi.xml như sau:

```
<?xml version="1.0"?>
<DSKhachMoi>
    <!-- Tiec co tu 0 khach moi tro len -->
    <KhachMoi>
        <Ho>Quach</Ho>
        <Ten>Xuan Tam</Ten>
        <CachXungHo>Ban</CachXungHo>
    </KhachMoi>
    <KhachMoi>
        <Ho>Tran</Ho>
        <Ten>Du</Ten>
        <CachXungHo>Co</CachXungHo>
    </KhachMoi>
```

```

<KhachMoi>
    <Ho>Nguyen</Ho>
    <Ten>Tan Son</Ten>
    <CachXungHo>Duong</CachXungHo>
</KhachMoi>
...
</DSKhachMoi>

```

9. Tạo tập tin Bang_don_gia.dtd biểu diễn thông tin về bảng đơn giá thuê phòng của khách sạn (tên khách sạn, địa chỉ, loại phòng, đơn giá).
10. Tạo tập tin Bang_ty_gia.dtd biểu diễn thông tin về bảng tỷ giá các ngoại tệ (tên ngoại tệ, ký hiệu, mua tiền mặt, mua chuyển khoản, bán,...).
11. Tạo tập tin Ket_qua_xo_so.dtd biểu diễn thông tin về kết quả xổ số tỉnh Khánh Hòa (Ngày xổ số, tên giải, số giá trị, tiền thưởng,...)
12. Tạo tập tin Mach_dien.dtd biểu diễn thông tin về mạch điện với các điện trở (tên mạch, giá trị điện trở)
13. Tạo tập tin Tam_giac.dtd biểu diễn thông tin của tam giác ABC với A(1,0), B(-8,3) và C(4,4).
14. Tạo tập tin Da_thuc.xml biểu diễn thông tin của đa thức $P(x) = 4x^5 - 7x^3 + 2x^2 + 4$

Chương 4

LUỢC ĐỒ XML

4.1. Vì sao dùng lược đồ XML?

Lược đồ trong XML được biết đến rộng rãi với tên XML Schema Definition (XSD). Nó được sử dụng để miêu tả và xác nhận cấu trúc và nội dung của dữ liệu XML. Lược đồ trong XML định nghĩa kiểu phần tử, thuộc tính và dữ liệu.

Mặc dù DTD cũng được dùng để miêu tả cấu trúc và nội dung của dữ liệu XML nhưng DTD bộc lộ một số hạn chế như:

- Khi sử dụng DTD, chúng ta phải viết và nghiên cứu một ngôn ngữ mới không tương thích dạng chuẩn không tuân theo qui tắc cách viết của XML.
- DTD chưa hỗ trợ khái niệm tên (namespace) trong khai báo. Việc phối hợp và kế thừa của DTD phải thông qua việc phối hợp giữa DTD bên ngoài với DTD bên trong hay phải sử dụng thực thể tham số. Tuy nhiên, chúng không thể hiện được theo một cách thông thường mà nhìn có vẻ cưỡng ép.
- DTD chưa cung cấp kiểu dữ liệu cụ thể cho phần tử (element) và thuộc tính (attribute) cho dù phần tử có mô hình nội dung và 10 kiểu dữ liệu, nhưng tất cả đều được khai báo với kiểu dữ liệu văn bản PCDATA hoặc CDATA.
- DTD chưa thiết lập được giá trị mặc định của phần tử.

Tổ chức W3C đã đưa ra một giải pháp tổng quát hóa DTD hơn đó là khai báo và định nghĩa các phần tử trong tài liệu XML theo lược đồ XML.

Lược đồ XML tuy không đơn giản hơn định nghĩa DTD nhưng tỏ ra mạnh mẽ và chính xác hơn. Để kiểm tra tính hợp lệ của các định nghĩa DTD trong tài liệu, chúng ta dùng các bộ kiểm tra Validator. Để kiểm tra tính hợp lệ của các khai báo theo lược đồ XML, chúng ta có thể dùng các bộ kiểm tra cú pháp lược đồ (schema checker).

Tuy nhiên lược đồ XML là đặc tả mới phát triển sau này, hiện chưa có nhà phát triển nào hỗ trợ nó mạnh mẽ bằng Microsoft, cụ thể là trình duyệt Internet Explorer. Tuy nhiên Internet Explorer chỉ cài đặt một số đặc tả của lược đồ XML do tổ chức W3C quy định, còn rất nhiều khái niệm và định nghĩa về lược đồ XML chưa được cài đặt. Trong chương này chúng ta sẽ nghiên cứu về định nghĩa lược đồ theo đúng chuẩn XML do tổ chức W3C đề ra.

Chúng ta sẽ tập trung xem xét một ví dụ mang tên *book.xml* và lược đồ kết hợp với nó là *book.xsd* (.xsd là tên quy ước mở rộng được W3C sử dụng cho các tập tin lược đồ). Tài liệu này lưu trữ lại tất cả thông tin về việc thuê sách của một độc giả mang tên Doug Glass và việc mượn sách của một độc giả mang tên Britta Regensburg. Chúng ta lưu lại tên và địa chỉ người mượn cũng như trả sách. Thông tin về quyền sách cũng được lưu lại như tựa sách (title), ngày xuất bản, số ngày mượn,... Tài liệu XML thể hiện nội dung này như sau:

Ví dụ 4.1: (book.xml)

```
<?xml version="1.0"?>
<transaction borrowDate="2016-10-15">
    <Lender phone="607.555.2222">
        <name>Doug Class</name>
        <street>416 Disk Drive</street>
        <city>Medfield</city>
        <state>MA</state>
    </Lender>
    <Borrower phone="310.555.1111">
        <name>Britta Regensburg</name>
        <street>219 Union Drive</street>
        <city>Medfield</city>
        <state>MA</state>
    </Borrower>
    <note>Lender wants these back in two weeks!</note>
    <books>
        <book boodID="123-4567-890">
            <bookTitle>Earthquakes for Breakfast</bookTitle>
            <pubDate>2015-10-20</pubDate>
            <repiacementValue>15.95</replacementValue>
            <maxDaysOut>14</maxDaysOut>
        </book>
        <book boodID="123-4567-891">
            <bookTitle>Avalanches for Lunch</bookTitle>
            <pubDate>2015-10-21</pubDate>
            <replacementValue>19.99</replacementValue>
            <maxDaysOut>14</maxDaysOut>
        </book>
        <book boodID="123-4567-892">
            <bookTitle>Meteor Showers for Dinner</bookTitle>
            <pubDate>2015-10-22</pubDate>
            <replacementValue>11.95</replacementValue>
            <maxDaysOut>14</maxDaysOut>
        </book>
    </books>
</transaction>
```

Tài liệu trên có phần tử gốc là `<transaction>` trong đó chứa nhiều phần tử con khác như `<Lender>`, `<Borrower>`, `<books>`,... Bên trong các phần tử con lại có các phần tử con khác, ví dụ như `<book>` nằm trong `<books>`.

Theo cách nhìn của lược đồ XML, phần tử chứa các phần tử con có thêm thuộc tính được gọi là **kiểu phức hợp** `<complex types>`. Các phần tử chỉ chứa dữ liệu đơn giản như chuỗi, ngày tháng, số - nhưng không được chứa thuộc tính và phần tử con – được gọi là **kiểu giản đơn** (`symply types`). Thêm vào đó, bản thân thuộc tính được xem là kiểu giản đơn bởi vì giá trị gán cho thuộc tính không bao hàm cấu trúc. Nếu như chúng ta nhìn tài liệu ở dạng hình cây (tree) thì cách dễ nhận ra nhất đó là phần tử có kiểu giản đơn thường không có nhánh con (nút lá) trong khi kiểu phức hợp thì có thể có.

Phân biệt sự khác nhau giữa kiểu giản đơn và kiểu phức hợp rất quan trọng, bởi vì chúng ta sẽ khai báo kiểu giản đơn và kiểu phức hợp theo hai cách khác nhau. Chúng ta tự khai báo các kiểu phức hợp của riêng chúng ta trong khi các kiểu giản đơn đã được khai báo và định nghĩa sẵn trong đặc tả của lược đồ XML. Mặc dù vậy, chúng ta cũng có thể tự định nghĩa và khai báo kiểu giản đơn theo cách riêng của mình.

Một điều cần lưu ý nữa là không có phần nào trong tài liệu book.xml chỉ định lược đồ sẽ sử dụng với tài liệu (như chúng ta thấy khi định nghĩa DTD cho tài liệu, chúng ta sử dụng khai báo <!DOCTYPE>). W3C không yêu cầu một cơ chế cứng nhắc nào để kết hợp một lược đồ với nội dung tài liệu. W3C cho phép chúng ta khai báo không gian tên cho tài liệu tham chiếu đến tập tin lược đồ theo địa chỉ URL như sau:

```
<?xml version="1.0"?>
<transaction borrowDate="2016-10-15"
  xmlns="http://www.khoahoc.com.vn/kh-schema">
  <Lender PHONE="607.555.2222">
    <name>Doug Glass</name>
    <street>416 Disk Drive</street>
    <city>Medfield</city>
    <state>MA</state>
  </Lender>
  ...

```

Tiếp đến sẽ tùy thuộc vào cách trình xử lý XML tìm ra lược đồ vào mô tả trên. Dưới đây là nội dung lược đồ cho tài liệu *book.xml*; lược đồ này được lưu trong tập tin *book.xsd*. Tiếp đầu ngữ **xs**: thường được W3C dùng tham chiếu đến không gian tên của lược đồ (chúng ta có thể không cần sử dụng tiếp đầu ngữ này nếu biết chắc rằng lược đồ của mình đang được phân tích bởi bộ kiểm tra tuân theo chuẩn W3C). Mặc dù *book.xsd* là một tập tin lược đồ nhưng nó phải được thiết kế hợp khuôn dạng tuân theo cấu trúc XML.

Ví dụ 4.2: (*book.xsd*)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:documentation>Book borrowing transaction schema
    </xs:documentation>
  </xs:annotation>
  <xs:element name="transaction" type="transactionType"/>
  <xs:complexType name="transactionType">
    <xs:element name="Lender" type="address"/>
    <xs:element name="Borrower" type="address"/>
    <xs:element ref="note" minOccurs="0"/>
    <xs:element name="books" type="books"/>
    <xs:attribute name="borrowDate" type="xs:date"/>
  </xs:complexType>
  <xs:element name="note" type="xs:string"/>
  <xs:complexType name="address">
    <xs:element name="name" type="xs:string"/>
    <xs:element name="street" type="xs:string"/>
```

```

<xs:element name="city" type="xs:string"/>
<xs:element name="state" type="xs:string"/>
<xs:attribute name="phone" type="xs:string"
               use="optional"/>
</xs:complexType>
<xs:complexType name="books">
    <xs:element name="book" minOccurs="0" maxOccurs="10">
        <xs:complexType>
            <xs:element name="bookTitle" type="xs:string"/>
            <xs:element name="pubDate" type="xs:date"
                         minOccurs="0"/>
            <xs:element name="replacementValue"
                         type="xs:decimal"/>
            <xs:element name="maxDaysOut">
                <xs:simpleType base="xs:integer">
                    <xs:maxExclusive value="14"/>
                </xs:simpleType> </xs:element>
            <xs:attribute name="bookID" type="catalogID"/>
        </xs:complexType> </xs:element>
    </xs:complexType>
    <xs:simpleType name="catalogID" base="xs:string">
        <xs:pattern value="\d{3}-\d{4}-\d{3}"/>
    </xs:simpleType>
</xs:schema>

```

Chúng ta lần lượt xem qua từng phần tử của lược đồ trên.

Trước hết chúng ta định nghĩa không gian tên **xs:** cho lược đồ dựa vào định danh duy nhất theo địa chỉ URL <http://www.w3.org/2001/XMLSchema>. Không gian tên này sẽ được dùng cho các khai báo thẻ và thuộc tính, ví dụ như **<xs:element>** hay **<xs:attribute>**. Thêm vào đó khi khai báo, chúng ta chỉ định kiểu cho phần tử và thuộc tính. Để tạo ra một kiểu dữ liệu, chúng ta có thể dùng thẻ **<xs:complexType>** cho kiểu phức hợp, **<xs:simpleType>** cho kiểu giản đơn hay **<xs:annotation>** cho kiểu chú giải trong lược đồ.

Trong ví dụ trên, phần tử gốc của tài liệu *book.xml* là **<transaction>** được định nghĩa là kiểu phức hợp *transactionType* do nó chứa nhiều phần tử và thẻ con bên trong bao gồm kiểu *address* và *books*. Kiểu *address* bản thân nó được định nghĩa có khả năng chứa tên của một cá nhân và địa chỉ liên lạc. Kiểu *books* có khả năng chứa phần tử **<book>** mô tả thông tin về quyển sách... Sử dụng lược đồ theo cách này, chúng ta có thể mô tả toàn bộ cú pháp của nội dung tài liệu *book.xml*.

4.2. Khai báo các phần tử và kiểu dữ liệu

Để chỉ định kiểu cho một phần tử chúng ta phải bảo đảm kiểu đã được định nghĩa trước đó. Như đã nêu, kiểu phức hợp là kiểu chứa các phần tử con trong nó hoặc chứa thêm thuộc tính trong khi kiểu giản đơn thì không. Kiểu giản đơn là các kiểu nội tại đã được định nghĩa sẵn và chúng ta có thể dùng nó tự do trong lược đồ XML của mình. Dưới đây là liệt kê các kiểu giản đơn nội tại (nên nhớ, khi chỉ định kiểu này trong lược đồ XML cũng có nghĩa là chúng ta đang tham chiếu đến không gian tên **xs:** của W3C).

Bảng 4.1: Bảng liệt kê các kiểu giản đơn nội tại

Tên kiểu dữ liệu	Điễn giải
binary	Kiểu dữ liệu nhị phân
boolean	Kiểu luận lý
byte	Kiểu byte
century	Kiểu thế kỷ (ví dụ 20)
date	Kiểu ngày
decimal	Kiểu thập phân
double	Kiểu số thực 64 bit
ENTITY	Kiểu thực thể
ENTITIES	Kiểu đa thực thể
ID	Kiểu định danh
int, integer	Kiểu số nguyên
IDREF	Kiểu tham chiếu định danh
NOTATION	Kiểu ghi chú
NMTOKEN	Kiểu token đơn
NMTOKENS	Kiểu đa token
month	Kiểu tháng (ví dụ 4, 6)
string	Kiểu chuỗi

Để tương thích giữa lược đồ XML và định nghĩa DTD, chỉ nên dùng các kiểu giản đơn như ID, IDREF, ENTITY, ENTITIES, NOTATION, NMTOKEN, NMTOKENS khi khai báo thuộc tính.

Tạo một kiểu phức hợp mới trong lược đồ bằng cách dùng phần tử khai báo `<xs:complexType>`. Bản thân kiểu phức hợp thường chứa các khai báo phần tử con, tham chiếu đến các phần tử khác và khai báo thuộc tính. Khai báo phần tử bằng chỉ thị khai báo `<xs:element>`, khai báo thuộc tính bằng `<xs:attribute>`. Tương tự như DTD, khai báo phần tử sẽ chỉ định cú pháp của phần tử trong lược đồ, chúng ta còn có thể chỉ định kiểu cho phần tử, thậm chí có thể chỉ định cả kiểu cho thuộc tính.

Ví dụ dưới đây trích ra từ lược đồ *book.xsd*. Ở đây chúng ta khai báo *address* là một kiểu phức hợp. Kiểu *address* nằm giữa các phần tử tạo nên thông tin cá nhân của người mượn sách.

```

<xs:complexType name="address">
    <xs:element name="name" type="xs:string"/>
    <xs:element name="street" type="xs:string"/>
    <xs:element name="city" type="xs:string"/>
    <xs:element name="state" type="xs:string"/>
    <xs:attribute name="phone" type="xs:string"
                  use="optional"/>
</xs:complexType>

```

Sau khi định nghĩa cho kiểu *address* trong lược đồ, chúng ta có thể sử dụng kiểu *address* cho `<Lender>` và `<Borrower>` để lưu thông tin cá nhân trong tài liệu như sau:

```

<xs:element name="transaction" type="transactionType"/>
<xs:complexType name="transactionType">

```

```

<xs:element name="Lender" type="address"/>
<xs:element name="Borrower" type="address"/>
<xs:element ref="note" minOccurs="0"/>
<xs:element name="books" type="book"/>
<xs:attribute name="borrowDate" type="xs:date"/>
</xs:complexType>

```

Khi định nghĩa kiểu *address*, cần xác định rằng, bất kỳ phần tử nào mang kiểu này đều phải có 4 phần tử con và một thuộc tính. Các phần tử con là *<name>*, *<street>*, *<city>* và *<state>* và thuộc tính là *phone*.

Trong khi định nghĩa kiểu *transactionType* ở *book.xsd*, chúng ta đã tạo thuộc tính *borrowDate* có kiểu giản đơn *xs:date*. Chúng ta nên nhớ là thuộc tính luôn phải có kiểu giản đơn, không được chứa kiểu phức hợp.

Chúng ta đã xem qua làm thế nào để tạo và khai báo mới một phần tử cũng như thuộc tính trong lược đồ XML. Tuy nhiên chúng ta hãy xem lại khai báo phần tử *<note>*, ở đây chúng ta không định nghĩa mới phần tử *<note>* mà dùng tham chiếu đến phần tử *<note>* đã được khai báo và định nghĩa kiểu trong lược đồ. Khai báo phần tử *<note>* như sau:

```
<xs:element ref="note" minOccurs="0"/>
```

Sử dụng tham chiếu *ref* cho phép ta đưa vào một phần tử đã được định nghĩa trước đó. Mặc dù vậy, không phải phần tử nào cũng có thể đưa vào tham chiếu. Phần tử mà chúng ta tham chiếu đến phải được khai báo là toàn cục. Điều này có nghĩa là bản thân phần tử không được là một phần của định nghĩa kiểu phức hợp. Phần tử hay thuộc tính toàn cục phải được khai báo ngay sau cấp *<xs:schema>*.

Chúng ta hãy tìm hiểu tiếp xem làm cách nào để chỉ định số lần xuất hiện của phần tử trong kiểu phức hợp.

Chỉ định số lần xuất hiện của các phần tử

Trong ví dụ về phần tử *<note>*, bằng cách chỉ định thuộc tính *minOccurs=0* cho phép *<note>* có thể xuất hiện hoặc không cần xuất hiện bên trong thẻ *<transaction>*. Có thể xác định số lần ít nhất mà phần tử phải xuất hiện bên trong một phần tử khác bằng thuộc tính *minOccurs* cùng với số lần nhiều nhất mà phần tử được phép xuất hiện thông qua thuộc tính *maxOccurs*. Ví dụ, để chỉ định *<note>* xuất hiện trong *<transaction>* từ 0-5 lần, chúng ta định nghĩa kiểu *transactionType* như sau:

```

<xs:element name="transaction" type="transactionType"/>
<xs:complexType name="transactionType">
    <xs:element name="Lender" type="address"/>
    <xs:element name="Borrower" type="address"/>
    <xs:element ref="note" minOccurs="0"
                maxOccurs="5"/>
    <xs:element name="books" type="book"/>
    <xs:attribute name="borrowDate" type="xs:date"/>
</xs:complexType>
<xs:element name="note" type="xs:string"/>

```

Nếu không chỉ định thuộc tính *minOccurs* và *maxOccurs* thì mặc định *minOccurs=1* còn *maxOccurs* là không giới hạn. Điều này có nghĩa là phải có ít nhất một phần tử xuất hiện trong tài liệu.

Chỉ định giá trị mặc định cho phần tử

Ngoài thuộc tính *minOccurs* và *maxOccurs*, có thể sử dụng thuộc tính *fixed* và *default* của khai báo *<xs:element>* để chỉ định giá trị có thể gán cho phần tử. Sử dụng thẻ *default* để gán giá trị mặc định cho phần tử thẻ, *fixed* được dùng để gán giá trị cố định và bắt buộc. Mặc dù vậy chúng ta không được sử dụng cả hai thuộc tính *default* và *fixed* cùng lúc trong khai báo. Chẳng hạn, nếu đặt *fixed= "400"* có nghĩa là giá trị của phần tử khi sử dụng trong tài liệu phải luôn là 400. Tuy nhiên nếu phần tử được định nghĩa với *default= "400"* trong tài liệu, người dùng có thể gán một giá trị khác với 400. Nếu người dùng không gán giá trị thì 400 là giá trị của phần tử thẻ. Ví dụ phần tử *trialValue* dưới đây được định nghĩa với giá trị cố định và bắt buộc là 1306.

```
<xs:element name="trialValue" fixed="1306"/>
```

hoặc có thể chỉ định 1306 là một trị mặc định như sau:

```
<xs:element name="trialValue" default="1306"/>
```

Chỉ định ràng buộc và trị mặc định cho thuộc tính

Tương tự như đối với các phần tử trong tài liệu, có thể chỉ định kiểu cho thuộc tính. Và như đã nêu, khác với phần tử thẻ, thuộc tính không có kiểu phức hợp. Chúng ta chỉ có thể khai báo thuộc tính mang kiểu giản đơn nhưng thay vào đó chúng ta được quyền chỉ định các ràng buộc cho thuộc tính. Khi học về DTD ta đã biết qua các khái niệm định nghĩa giá trị thuộc tính như FIXED, REQUIRED,... Lược đồ XML cũng cho phép định nghĩa và ràng buộc giá trị gán cho thuộc tính bằng những tùy chọn sau:

- *required*: Thuộc tính phải hiện hữu trong thẻ và phải được gán giá trị
- *optional*: Thuộc tính có thể có hoặc không và giá trị gán cho thuộc tính là không bắt buộc.
- *fixed*: Giá trị thuộc tính phải được gán cố định và không được thay đổi như đã quy định trong lược đồ.
- *default*: Nếu thuộc tính không được gán giá trị thì giá trị định nghĩa trong lược đồ là giá trị mặc định của thuộc tính.
- *prohibited*: Không cho phép thuộc tính xuất hiện.

Chúng ta sử dụng từ khóa *use* trong khai báo *<xs:element>* để chỉ định ràng buộc cho thuộc tính theo các tùy chọn trên. Ví dụ trong định nghĩa lược đồ dưới đây, thuộc tính *phone* trong kiểu *address* là tùy ý (*optional*) trong khi thuộc tính *name* là bắt buộc (*required*) còn *city* có giá trị mặc định (*default*) là “HCM”:

```
<xs:complexType name="address">
  <xs:element name="name" type="xs:string"
    use="required"/>
  <xs:element name="street" type="xs:string"/>
  <xs:element name="city" type="xs:string" use="default"
    value="HCM"/>
  <xs:element name="state" type="xs:string"/>
  <xs:attribute name="phone" type="xs:string"
    use="optional"/>
</xs:complexType>
```

4.3. Tạo các kiểu đơn giản

Hầu hết các kiểu dữ liệu mà chúng ta sử dụng trong *book.xsd* là các kiểu nội tại (built-in) được quy định sẵn trong đặc tả XML như xs:string, xs:integer, xs:date,... Mặc dù vậy hãy xét thuộc tính *bookID* trong ví dụ dưới đây, thuộc tính này được khai báo có kiểu là catalogID:

```
<xs:complexType name="books">
    <xs:element name="book" minOccurs="0"
                maxOccurs="10">
        <xs:complexType>
            <xs:element name="bookTitle" type="xs:string"/>
            <xs:element name="pubDate" type="xs:date"
                        inOccurs="0"/>
            <xs:element name="replacementValue"
                        type="xs:decimal"/>
            <xs:element name="maxDaysOut">
                <xs:simpleType base="xs:integer">
                    <xs:maxExclusive value="14"/>
                </xs:simpleType> </xs:element>
            <xs:attribute name="boodID" type="catalogID"/>
        </xs:complexType> </xs:element>
    </xs:complexType>
```

Trong trường hợp này, *catalogID* là một kiểu giản đơn không thuộc phần định nghĩa nội tại của XML. Kiểu giản đơn *catalogID* được định nghĩa trong phần cuối của lược đồ là:

```
<xs:simpleType name="catalogID" base="xs:string">
    <xs:pattern value="\d{3}-\d{4}-\d{3}"/>
</xs:simpleType>
```

Chúng ta có thể tự định nghĩa các kiểu giản đơn của riêng mình, tuy nhiên nó phải dựa trên kiểu giản đơn cơ sở nội tại của đặc tả XML. Thuộc tính *base* trong khai báo *<xs:simpleType>* dùng cho mục đích này. Như chúng ta thấy, kiểu giản đơn *catalogID* được định nghĩa dựa trên kiểu giản đơn nội tại *xs:string*. Thế tại sao ta không dùng kiểu nội tại mà lại sinh ra thêm kiểu *catalogID*? Chúng ta hãy chú ý vào phần định nghĩa bên trong của *catalogID*.

```
<xs:pattern value="\d{3}-\d{4}-\d{3}"/>
```

Định nghĩa này giới hạn giá trị của kiểu dữ liệu cơ sở *xs:string* và đó mới chính là mục đích sử dụng của kiểu *catalogID*. Nói cách khác, kiểu giản đơn *catalogID* là một tập con của kiểu cơ sở *xs:string*. Để hiểu rõ vấn đề này chúng ta cần chuyển sang nghiên cứu một chuyên đề mới về giới hạn kiểu (facet).

Tạo kiểu giản đơn thông qua giới hạn kiểu (facet)

Khái niệm *facet* được dùng để chỉ phạm vi giới hạn tập dữ liệu của kiểu giản đơn. Ví dụ, giả sử chúng ta muốn tạo ra kiểu giản đơn mang tên *dayOfMonth* (ngày trong tháng), kiểu này chỉ cần mang giá trị từ 1-31 là đủ. chúng ta định nghĩa kiểu này dựa trên kiểu *xs:integer* cùng với việc sử dụng hai thành phần giới hạn (facet) là *minInclusive* và *maxInclusive* như sau:

```
<xs:simpleType name="dayOfMonth" base="xs:integer">
    <xs:minInclusive value="1">
    <xs:maxInclusive value="31">
</xs:simpleType>
```

Khai báo trên đây có ý nghĩa $1 \leq \text{dayOfMonth} \leq 31$. Chúng ta cũng có thể dùng hai giới hạn *minExclusive* và *maxExclusive* để chỉ giá trị lớn hơn và nhỏ hơn không bao gồm trị biên. Ví dụ về kiểu *catalogID* ở phần trên:

```
<xs:simpleType name="catalogID" base="xs:string">
    <xs:pattern value="\d{3}-\d{4}-\d{3}"/>
</xs:simpleType>
```

Trước hết *catalogID* phải có kiểu chuỗi xs:string. Tiếp đến, *catalogID* chỉ bao gồm các chuỗi so khớp theo mẫu giới hạn facet *<xs:pattern>*. Nội dung so khớp phải thỏa mãn điều kiện:

```
\d{3}-\d{4}-\d{3}
```

có nghĩa là chuỗi phải bao gồm 3 nhóm ký số (digit), mỗi nhóm cách nhau bằng dấu gạch ngang (-). Nhóm thứ nhất gồm 3 ký số, nhóm thứ hai gồm 4 ký số và nhóm thứ ba là 3 ký số. Ví dụ 012-2017-613 là một mẫu catalogID hợp lệ, nhưng AB1-CD23_456 thì không hợp lệ..

Trong tài liệu *book.xml*, chúng ta sử dụng thuộc tính *bookID* có kiểu *catalogID* như sau:

```
<book bookID="123-4567-890">
    <bookTitle>Earthquakes for Breakfast</bookTitle>
    <pubDate>2015-10-20</pubDate>
    <replacementValue>15.95</replacementValue>
    <maxDaysOut>14</maxDaysOut>
</book>
```

Chúng ta đã học qua ba thành phần giới hạn *facet* là *minInclusive*, *maxInclusive* và *pattern*. Một thành phần *facet* khá quan trọng khác là giới hạn kiểu liệt kê *enumeration*. Như khi chúng ta định nghĩa kiểu giản đơn *Weekday*, chúng ta có thể chỉ muốn các giá trị “Sunday”, “Monday”,...”Friday”, “Saturday” chứa trong tập hợp của kiểu dữ liệu. Chúng ta có thể dùng giới hạn liệt kê để định nghĩa kiểu như sau:

```
<xs:simpleType name="Weekday" base="xs:string">
    <xs:enumeration value="Sunday">
    <xs:enumeration value="Monday">
    <xs:enumeration value="Tuesday">
    <xs:enumeration value="Wednesday">
    <xs:enumeration value="Thursday">
    <xs:enumeration value="Friday">
    <xs:enumeration value="Saturday">
</xs:simpleType>
```

Ngoài kiểu số nguyên xs:integer và kiểu chuỗi xs:string ra, còn những kiểu giản đơn nội tại nào cho phép sử dụng giới hạn *facet* khi định nghĩa kiểu giản đơn con? Chúng ta có thể tham khảo các kiểu còn lại trong Bảng 4.1.

Sử dụng các định nghĩa kiểu vô danh (anonymous type definition)

Cho đến bây giờ, tất cả các khai báo sử dụng kiểu trong lược đồ *book.xsd* đều sử dụng thuộc tính *type* để chỉ định kiểu cho phần tử. Mặc dù vậy, chúng ta có thể khai báo kiểu cho phần tử hoặc thuộc tính theo cách vô danh. Với định nghĩa kiểu vô danh đơn giản, chúng ta chỉ cần kèm theo phần tử *<xs:simpleType>* hoặc *<xs:complexType>* bên trong khai báo *<xs:element>*. Trong trường hợp này, chúng ta không cần gán giá trị tường minh cho thuộc

tính type trong `<xs:element>`, bởi vì kiểu vô danh mà chúng ta sử dụng không có tên. Ví dụ để định nghĩa *book* là phần tử có kiểu phức hợp, chúng ta không cần phải định nghĩa kiểu phức hợp riêng biệt, chúng ta có thể kết hợp việc định nghĩa kiểu phức hợp với khai báo kiểu cho *book* như sau:

```
<xs:element name="book" minOccurs="0" maxOccurs="10">
    <xs:complexType>
        <xs:element name="bookTitle" type="xs:string"/>
        <xs:element name="pubDate" type="xs:date"
                    minOccurs="0"/>
        <xs:element name="replacementValue"
                    type="xs:decimal"/>
        <xs:element name="maxDaysOut">
            <xs:simpleType base="xs:integer">
                <xs:maxExclusive value="14"/>
            </xs:simpleType>
        </xs:element>
    </xs:complexType>
```

Kiểu `complexType` bên trong định nghĩa *book* không mang tên nào cả, nó được gọi là kiểu vô danh. Ngoài kiểu vô danh phức hợp, chúng ta có thể định nghĩa kiểu vô danh giản đơn. Chẳng hạn *maxDayOut* ở trên được định nghĩa dựa trên kiểu vô danh đơn giản có cơ sở từ kiểu `xs:integer`.

4.4. Tạo các phần tử

4.4.1. Tạo các phần tử rỗng

Các phần tử rỗng không có nội dung nhưng có thể chứa thuộc tính. Làm thế nào khai báo chúng trong lược đồ XML? Chúng ta thực hiện điều này bằng cách khai báo kiểu phức hợp, sau đó đặt nội dung của thuộc tính *content* trong phần tử chi thi `<xs:complexType>` là “empty”.

Trong ví dụ dưới đây, chúng ta tạo mới một phần tử rỗng mang tên IMAGE. Phần tử này có thể chứa 3 thuộc tính *source*, *width* và *height* như sau:

```
<IMAGE SOURCE="/cover.gif" height="10" width="30">
```

Để định nghĩa phần tử IMAGE trong lược đồ, chúng ta khai báo như sau:

```
<xs:element name="image">
    <xs:complexType content="empty">
        <xs:element name="source" type="xs:string"/>
        <xs:element name="width" type="xs:decimal"/>
        <xs:element name="height" type="xs:decimal"/>
    </xs:complexType>
</xs:element>
```

Giờ thì phần tử IMAGE đã sẵn sàng sử dụng cho tài liệu bởi nó đã được định nghĩa đầy đủ trong lược đồ.

4.4.2. Tạo các phần tử có nội dung hỗn hợp

Trong lược đồ, chúng ta chỉ mới định nghĩa phần tử hoặc chứa văn bản hoặc chứa các phần tử con. Tuy nhiên một phần tử hoàn toàn có thể chứa cùng lúc cả hai nội dung này. Chúng được gọi là nội dung hỗn hợp (mixed-content). Ví dụ, nếu tài liệu của chúng ta có nội dung sau:

```
<?xml version="1.0"?>
<reminder>
    Dear <name> Mr.MK </name>,
```

```

The book <bookTitle> XML Programming </bookTitle>
was supposed to be out for <maxDaysOut>14</maxDaysOut>
<reminder>

```

Phần tử `<reminder>` chứa nội dung hỗn hợp bao gồm vừa văn bản vừa các phần tử con. Chúng ta định nghĩa chúng trong lược đồ như thế nào? Trước hết, trong lược đồ ta sử dụng kiểu vô danh phức hợp để định nghĩa cho phần tử `<reminder>`:

```

<xs:element name="reminder">
  <xs:complexType>
    ...
  </xs:complexType>
</xs:element>

```

Như đã được khai báo ở `book.xsd`, phần tử chỉ thị `<xs:complexType>` cho phép sử dụng thuộc tính `content` để quy định mô hình nội dung cho kiểu dữ liệu cần định nghĩa, trong trường hợp này chúng ta sử dụng chỉ định “*mixed*” gán cho `content` như sau:

```

<xs:element name="reminder">
  <xs:complexType content="mixed"> ... </xs:complexType>
</xs:element>

```

Phần còn lại chỉ cần thêm vào định nghĩa cho các phần tử con:

```

<xs:element name="reminder">
  <xs:complexType content="mixed">
    <xs:element name="name" type="xs:string"/>
    <xs:element name="bookTitle" type="xs:string"/>
    <xs:element name="maxDaysOut">
      <xs:simpleType base="xs:integer">
        <xs:maxExclusive value="14"/>
      </xs:simpleType> </xs:complexType>
  </xs:element>

```

Chúng ta đã biết qua cách dùng thuộc tính `content` để xác định mô hình lưu trữ nội dung cho một phần tử. Vậy nếu không chỉ định thuộc tính `content` cho `<xs:complexType>` thì phần tử sẽ chứa nội dung theo mô hình nào? Mô hình mặc định được áp dụng khi không chỉ định thuộc tính `content` đó là `elementOnly`, có nghĩa là phần tử chỉ chứa các phần tử con. Còn một mô hình khác đó là `textOnly`, mô hình này chỉ cho phép các phần tử chứa văn bản thuần text phi định dạng. Các phần tử chứa nội dung là kiểu giản đơn thường sử dụng mô hình nội dung này.

4.4.3. Lược đồ chú giải (annotation schema)

Lược đồ XML định nghĩa thêm 3 phần tử mới giúp chúng ta đưa chú giải vào lược đồ: `<xs:annotation>`, `<xs:documentation>` và `<xs:appInfo>`.

Các phần tử này làm việc như sau: `<xs:annotation>` là phần tử chú giải chính có thể chứa các phần tử con `<xs:documentation>` và `<xs:appInfo>`. Phần tử `<xs:documentation>` dùng để chứa các chuỗi văn bản chúng ta muốn làm ghi chú hoặc chú thích trong tài liệu. Phần tử `<xs:appInfo>` sẽ chứa các thông tin liên quan đến ứng dụng. Những thông tin này có thể được tổ chức theo cách mà chương trình hiểu được. Ví dụ về khai báo `<xs:annotation>` và `<xs:appInfo>` trong lược đồ có thể thực hiện như sau:

```

<xs:simpleType name="string" base="urSimpleType">
    <xs:annotation>
        <xs:appInfo>
            <has-facet name="length"/>
            <has-facet name="minLength"/>
            <has-facet name="maxLength"/>
            <has-facet name="pattern"/>
            <has-facet name="enumeration"/>
            <has-facet name="maxInclusive"/>
            <has-facet name="maxExclusive"/>
            <has-facet name="minInclusive"/>
            <has-facet name="minExclusive"/>
            <has-property name="ordered" value="true"/>
            <has-property name="bounded" value="false"/>
            <has-property name="numeric" value="false"/>
        </xs:appInfo>
    </xs:annotation>
</xs:simpleType>

```

Ứng dụng của chúng ta có thể dựa vào thông tin trong <xs:appInfo> để biết rõ thêm về giới hạn facet của kiểu string. Chúng ta cũng có thể sử dụng <xs:documentation> để thực hiện chú giải trong lược đồ như sau:

```

<xs:annotation>
    <xs:documentation>XML book description inside
                           schema</xs:documentation>
</xs:annotation>

```

Trong định nghĩa DTD chúng ta có thể tạo các phần tử lựa chọn hoặc chỉ định tuân tự xuất hiện của chúng. Với lược đồ XML, ta cũng có thể thực hiện như vậy.

4.4.4. Tạo các lựa chọn (choices)

Lựa chọn (choice) cho phép chúng ta chỉ định một danh sách các phần tử trong đó chỉ có một phần tử được chọn. Để tạo lựa chọn trong lược đồ XML, chúng ta dùng phần tử khai báo <xs:choice>. Dưới đây là ví dụ- ở đây ta thay đổi kiểu transactionType để cho phép người mượn có thể mượn được một hoặc nhiều quyển sách. Thực hiện điều này bằng cách tạo phần tử <xs:choice> có khả năng nằm giữa một trong hai phần tử <book> và <books> (lưu ý, trong trường hợp này phần tử <book> cần được khai báo toàn cục để ta có thể tham chiếu chúng bên trong danh sách lựa chọn) ta thay đổi lại nội dung của transactionType và book trong lược đồ bằng cách:

```

<xs:complexType name="transactionType">
    <xs:element name="Lender" type="address"/>
    <xs:element name="Borrower" type="address"/>
    <xs:element ref="note" minOccurs="0"/>
    <xs:choice>
        <xs:element name="books" type="books"/>
        <xs:element ref="book">
    </xs:choice>
    <xs:attribute name="borrowDate" type="xs:date"/>
</xs:complexType>

```

```

<xs:complexType name="books">
    <xs:element ref="book" minOccurs="0" maxOccurs="10">
</xs:complexType>
<xs:element name="book">
<xs:complexType>
    <xs:element name="bookTitle" type="xs:string"/>
    <xs:element name="pubDate" type="xs:date"
                minOccurs="0"/>
    <xs:element name="replacementValue"
                type="xs:decimal"/>
    <xs:element name="maxDaysOut">
        <xs:simpleType base="xs:integer">
            <xs:maxExclusive value="14"/>
        </xs:simpleType>
    </xs:element>
    <xs:attribute name="bookID" type="catalogID"/>
</xs:complexType>
</xs:element>

```

Khai báo lựa chọn trong lược đồ XML tương tự như việc sử dụng ký tự | trong khai báo của DTD.

4.4.5. Tạo các khai báo tuần tự (sequence)

Trong lược đồ XML, định nghĩa thứ tự xuất hiện của các phần tử dựa vào `<xs:sequence>`. Để dễ hiểu, giả sử a, b, c, d là các phần tử trong tài liệu. Trong định nghĩa DTD, sự xuất hiện tuần tự của các phần tử trong tài liệu theo biểu thức `(a, b?, (c|d)*)` có thể diễn đạt tương ứng trong lược đồ XML như sau:

```

<sequence>
    <element ref="a">
    <element ref="b" minOccurs="0" maxOccurs="1"/>
    <choice minOccurs="1"> <element name="c"/>
        <element="d"/> </choice>
</sequence>

```

4.4.6. Tạo nhóm và thuộc tính nhóm (attribute group)

Chúng ta có thể định nghĩa và gom các phần tử lại thành một nhóm (group). Chẳng hạn, trong ví dụ về thuê mượn sách, giả sử chúng ta muốn đọc giả có thể vừa mượn sách vừa mượn tạp chí (magazine), ta gom phần tử `<books>` và `<magazine>` lại thành một nhóm với tên `booksAndMagazine` bằng cách sử dụng chỉ thị khai báo `<xs:group>`. Chỉ thị này tương tự như chức năng của biểu thức ngoặc () trong định nghĩa DTD:

```

<xs:group name="booksAndMagazine">
    <xs:element ref="books">
    <xs:element ref="magazine"> </xs:group>

```

Ngoài việc gom nhóm các phần tử lại thành một nhóm, còn có thể gom các thuộc tính lại thành nhóm. Thực hiện công việc này bằng phần tử chỉ thị `<xs:attributeGroup>`. Ví dụ, giả sử chúng ta muốn thêm một số thuộc tính cho phần tử `<book>` nhằm mô tả thông tin của quyển sách. Ta gom các thuộc tính liên quan đến `<book>` lại với nhau thành nhóm mang tên `bookDescription`:

```

<xs:attributeGroup name="bookDescription">
    <xs:attribute name="bookID" type="catalogID"/>

```

```

<xs:attribute name="numberPages"
              type="xs:decimal"/>
<xs:attribute name="coverType">
    <xs:simpleType base="xs:string">
        <xs:enumeration value="leather"/>
        <xs:enumeration value="cloth"/>
        <xs:enumeration value="vinyl"/>
    </xs:simpleType> </xs:attribute>
</xs:attributeGroup>

```

Tiếp đến trong lược đồ, khi định nghĩa phần tử `<book>`, chúng ta tham chiếu đến thuộc tính của nhóm `bookDescription` như sau:

```

<xs:element name=" book">
    <xs:complexType>
        <xs:element name="bookTitle" type="xs:string"/>
        <xs:element name="pubDate" type="xs:date"
                    minOccurs="0"/>
        <xs:element name="replacementValue"
                    type="xs:decimal"/>
        <xs:element name="maxDaysOut">
            <xs:simpleType base="xs:integer">
                <xs:maxExclusive value="14"/>
            </xs:simpleType> </xs:element>
        <xs:attributeGroup ref="bookDescription"/>
    </xs:complexType>
</xs:element>

```

4.4.7. Tạo nhóm `all`

Lược đồ XML còn đưa ra một khái niệm nhóm tập thể khác được gọi là `all`. Các phần tử định nghĩa trong nhóm `all` của lược đồ phải xuất hiện đồng thời trong tài liệu hoặc không được có phần tử nào xuất hiện cả. Thứ tự xuất hiện của các phần tử là không quan trọng.Thêm vào đó, các phần tử chỉ được xuất hiện không quá một lần trong mô hình nội dung được định nghĩa (nghĩa là thành phần giới hạn `maxOccurs` chỉ có thể mang giá trị 0 hoặc 1). Nhóm định nghĩa trong `all` không được chứa lồng các nhóm con khác.

Dưới đây là ví dụ minh họa, ở đây chúng ta chuyển một phần nội dung định nghĩa kiểu `transactionType` trong lược đồ thành nhóm `all`:

```

<xs:complexType name="transactionType">
    <xs:all>
        <xs:element name="Lender" type="address"/>
        <xs:element name="Borrower" type="address"/>
        <xs:element ref="note" minOccurs="0"/>
        <xs:element name="books" type="books"/>
    </xs:all>
    <xs:attribute name="borrowDate" type="xs:date"/>
</xs:complexType>

```

Trong tài liệu XML của chúng ta, các phần tử như `<Lender>`, `<Borrower>`, `<books>`, `<note>` chỉ xuất hiện một lần tối đa chung với nhau hoặc không phần tử nào được xuất hiện cả.

BÀI TẬP CHƯƠNG 4

1. Một chương trình quản lý thông tin sinh viên và điểm các môn học mà họ đăng kí lưu các thông tin sau: sinh viên gồm mã số sinh viên, họ tên sinh viên, lớp. Một môn học mà sinh viên đăng kí học gồm có thông tin mã môn học, tên môn học, số tín chỉ, với mỗi môn học mà sinh viên đăng kí học thì sẽ có kết quả cuối kì chính là điểm thi mà sinh viên đạt được trong môn đó.

Hãy định nghĩa lược đồ XML Schema với yêu cầu như sau:

- a) MSSV: là một chuỗi các kí tự số có chiều dài 7 kí tự.
- b) Lớp của sinh viên là một trong những giá trị của tập hợp gồm các lớp {CTK28, CTK28CD, CTK29, CTK29CD, CTK30, CTK30CD, CTK31, CTK31CD}.
- c) Mã môn học là một chuỗi 4 kí tự, hai kí tự đầu là chữ cái, hai kí tự sau là số.
- d) Số tín chỉ của môn học ≥ 1 và ≤ 5 .
- e) Điểm của sinh viên ≥ 0 và ≤ 10

2. Mỗi đơn hàng, người ta cần lưu các thông tin sau: Mã khách hàng, tên khách hàng, địa chỉ liên lạc và một danh sách những mặt hàng người đó mua. Danh sách mặt hàng gồm có nhiều mặt hàng khác nhau, mỗi mặt hàng gồm những thông tin sau: Mã mặt hàng, tên mặt hàng, số lượng, đơn giá.

Tạo tập tin XML Schema với yêu cầu như sau:

- a) MaKH và MaMH: là một chuỗi 4 kí tự trong đó hai kí tự đầu là chữ cái, hai kí tự sau là kí số.
- b) DiaChi: là một chuỗi các kí tự với định dạng như sau: bắt đầu phải là số nhà, sau đến tên đường.
- c) Số lượng và đơn giá là kiểu số và phải là số dương.

3. Cho cấu trúc XML lưu trữ thông tin những cuốn sách đã được xuất bản theo từng lĩnh vực. Mỗi lĩnh vực có tên và có thể chưa có sách xuất bản hoặc cũng có thể đã có nhiều cuốn. Mỗi cuốn sách có thông tin một tựa đề duy nhất, một hay nhiều tác giả, mỗi tác giả lại có thông tin mã tác giả, tên tác giả, địa chỉ, số điện thoại với mã tác giả, tên tác giả là duy nhất cho mỗi người, địa chỉ email, số điện thoại có thể không có hoặc chỉ có một thông tin độc nhất cho mỗi người. Hãy định nghĩa XML Schema.

- a) Mã sách và mã tác giả là một chuỗi gồm 4 kí tự trong đó hai kí tự đầu là chữ và hai kí tự sau là số.
- b) Địa chỉ email là một chuỗi với định dạng `##@##.##`

- c) Số điện thoại là một chuỗi các kí số với định dạng sau: MaVung.SoDT

4. Đặc tả nội dung và cấu trúc với XSD của tài liệu XML tương ứng ma trận các số nguyên và biểu diễn dưới dạng tập tin XML, kiểm tra tính hợp lệ của tập tin XML.

1	4	12
-9	10	20
0	4	44

5. Biểu diễn thông tin bảng điểm danh sau bằng XSD và tạo tập tin XML tương ứng cho nội dung sau:

Lop: 56TH

Thang: 3 Nam: 2017

MSSV	Ho va ten	So lan vang	Ly do vang
56160342	Khuong Huu Duc	3	Khong ly do
56234980	Le Thi Hien	2	Benh

Đảm bảo tính hợp lệ của tập tin XML vừa tạo.

6. Cho tập tin XML dùng để đóng gói dữ liệu bán hàng online bao gồm các thông tin sau:

- Name – ví dụ như paper, clips, rubber bands, pens, ...
- Inventory ID – ví dụ như “F12333”, “Z55557”,... Thiết lập ràng buộc với yêu cầu là Inventory ID phải bắt đầu là một ký tự in hoa, theo sau là 5 chữ số từ 1 đến 9.
- Unit Price – ví dụ như 4.55, 22.33, ... Yêu cầu không được nhỏ hơn 1.00
- Stock Bin – ví dụ như “B789”, “B123”,... Thiết lập ràng buộc với yêu cầu phải bắt đầu là “B” và theo sau là 3 chữ số từ 0 đến 9.

Yêu cầu:

- Tạo tập tin schema định nghĩa các phần tử, kiểu dữ liệu và thiết lập ràng buộc.
- Tạo tập tin xml và chắc chắn rằng nó hợp khuôn dạng (Well-formed)
- Kiểm tra xem tập tin XML có hợp lệ hay không?

7. Bài tập về quản lý khách sạn

a. Tạo tập tin **KhachSan.xml**, bao gồm các trường thông tin sau:

- DanhSachKhachSan: Thẻ gốc chứa danh sách tất cả thông tin về khách sạn. Mỗi khách sạn chứa tên khách sạn và địa chỉ.
- Mỗi khách sạn có nhiều loại phòng khác nhau. Mỗi loại có các thông tin: Tên loại và đơn giá. Khách sạn chỉ có các loại phòng sau: phòng đơn, phòng đôi
- Mỗi loại phòng có nhiều phòng khác nhau. Mỗi phòng có các thông tin: Mã phòng, tên phòng, lầu, Số điện thoại.
- Khách sạn có nhiều loại dịch vụ khác nhau. Mỗi dịch vụ gồm Mã dịch vụ, tên dịch vụ, đơn giá. Các dịch vụ sinh viên tự nghĩ ra ví dụ: giặt,...
- Khách sạn có nhiều khách đến thuê. Mỗi khách hàng lưu trữ các thông tin sau: Makh, TenKh, Địa chỉ (mỗi khách hàng có thể không có hoặc có 1 địa chỉ), CMND, SoDT(SoDT nhận một trong hai giá trị là di động hoặc cố định).
- Hóa đơn thanh toán cho khách hàng gồm các thông tin sau: Mahd, Ngày đến, Ngày đi, Mã phòng, Mã dịch vụ, Thành tiền. Mỗi khách hàng có thể có 1 hoặc nhiều hóa đơn thanh toán.

b. Tạo tập tin DTD định nghĩa cấu trúc của tài liệu XML trên.

c. Tạo tập tin XML Schema cho tài liệu trên.

8. Bài tập về quản lý đĩa CD

- a. Tạo tập tin **QLDia.xml** với mô tả sau:
 - Phần tử gốc là SPECIALS có chứa các phần tử con: TITLE, MESSAGE, CD
 - TITLE chứa tiêu đề đầu tiên
 - MESSAGE chứa thông điệp nằm bên dưới tiêu đề
 - CD vừa chứa dữ liệu vừa chứa các phần tử con: 1 phần tử ARTIST, PRICEUS, PRICEVN, nhiều phần tử TRACK.
 - Dữ liệu chứa trong CD là tên của CD này
 - ARTIST chứa tên tác giả của CD
 - TRACK chứa tên bài hát, trong TRACK có thêm thuộc tính LENGTH cho biết thời gian bài hát được trình bày.
- b. Tạo File DTD tương ứng
- c. Tạo File XSD

Chương 5

TRUY XUẤT TÀI LIỆU XML

5.1. Mô hình đối tượng tài liệu (DOM – Document Object Model)

DOM là một tiêu chuẩn do tổ chức W3C định nghĩa và mang tính quốc tế. Thực tế thì có rất nhiều ngôn ngữ như C++, C#, ... đều hỗ trợ mô hình DOM. Do đó, DOM không phải thiết kế dành riêng cho Java, tuy nhiên nếu nói về các API cho phép phân tích các tài liệu XML thì chúng ta còn có mô hình SAX (Simple API for XML), một mô hình phân tích XML đơn giản hơn DOM rất nhiều, ban đầu được thiết kế cho Java nhưng hiện nay các ngôn ngữ khác cũng đã bắt đầu hỗ trợ mô hình này.

Các đặc điểm của DOM:

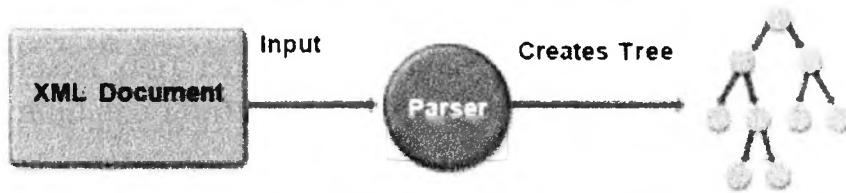
- Truy cập tài liệu XML như là một cấu trúc cây
- Được bao gồm hầu hết các nút element và các nút text
- Có thể "rà soát" (traversing) cây từ sau ra trước
- Đòi hỏi bộ nhớ lớn hơn
- Khá nặng nề trong việc tải và lưu trữ
- Sử dụng nó khi rà soát và hiệu chỉnh cây

Với một tài liệu XML mô hình DOM sẽ duyệt và chuyển nó thành một mô hình cây của các đối tượng (object). Các Object này được tạo ra trong bộ nhớ và có cấu trúc cây. Chính vì vậy mà có thể rà soát tài liệu XML này bằng cách rà soát trên các Object trong bộ nhớ và có thể thêm bớt các thông tin, Object để tạo ra một tài liệu XML mới hoặc một sản phẩm mới.

Như vậy, theo mô hình DOM, tài liệu XML của chúng ta là một cây bao gồm tập hợp các nút (node). Nội dung những nút này có thể chứa phần tử, dữ liệu, thuộc tính,... và các nút con khác. Có tất cả 12 loại nút sau đây trong mô hình DOM:

Nút	Mô tả
Element	Phần tử XML
Attribute	Nút thuộc tính
Text	Nút chứa dữ liệu text
CDATA section	Phân đoạn CDATA trong XML
Entity reference	Thực thể tham chiếu trong XML
Entity	Nút thực thể
Processing instruction	Chỉ thị xử lý
Comment	Ghi chú trong tài liệu XML
Document	Mô tả một nút lớn nhất đó là toàn bộ tài liệu XML
Document type	Kiểu tài liệu - Định nghĩa XML
Document fragment	Một đoạn tài liệu XML
Notation	Chú thích NOTATION trong XML

Một tài liệu XML sau khi được phân tích bởi mô hình DOM thì một cây sẽ được tạo ra trong bộ nhớ mang thông tin của tài liệu đó. Việc phân tích tài liệu XML bây giờ đưa về phân tích, xử lý các nút của cây.



Hình 5.1: Quy trình phân tích tài liệu XML của DOM

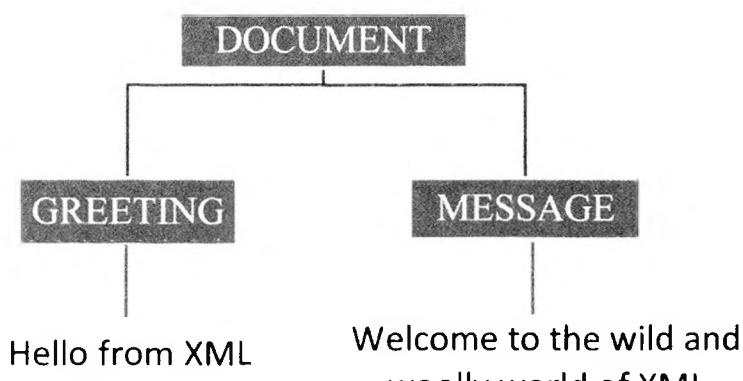
Ví dụ, giả sử chúng ta có tài liệu XML sau:

Ví dụ 5.1:

```

<?xml version="1.0" encoding = "UTF-8"?>
<DOCUMENT>
    <GREETING> Hello From XML </GREETING>
    <MESSAGE>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
  
```

Tài liệu này có thể phân theo cấu trúc hình cây bao gồm các nút đối tượng như sau:



Hình 5.2: Cấu trúc dạng cây của Ví dụ 5.1

Ngoài chỉ thị xử lý ra, phần tử DOCUMENT là một nút bao gồm hai nút con GREETING và MESSAGE. Nút GREETING và MESSAGE lần lượt chứa một nút con khác lưu dữ liệu dạng text với nội dung “Hello from XML” và “Welcome to the wild and woolly world of XML”. Toàn bộ cấu trúc trên chính là mô hình DOM. Khi phân tích cây tài liệu DOM ta coi mỗi nút là một đối tượng (object). DOM cung cấp cho chúng ta các phương thức như nextChild(), lastSibling(),... để lẩn ra toàn bộ các nút con khác.

W3C định nghĩa rất nhiều cấp độ cho mô hình DOM, chẳng hạn:

- Level 0: Là đặc tả DOM không chính thức, khởi đầu của mô hình DOM áp dụng cho các trình duyệt thông dụng trước đây là Netscape navigator 3.0 và IE 3.0
- Level 1: Đặc tả và định nghĩa mô hình DOM ở cấp độ này được cài đặt và sử dụng nhiều nhất. DOM ở cấp độ này tập trung vào sự kết hợp giữa HTML và XML.

- Level 2: Cấp độ nâng cao này chỉ đang là ứng cử viên được W3C xem xét. Mô hình DOM ở cấp độ 2 mở rộng cho phép chúng ta sử dụng không gian tên và kết hợp với mô hình định kiểu như CSS hay XSL.
- Level 3: Cấp độ này chỉ đang ở mức hoạch định. Nó cho phép nạp, xử lý, lưu lại mô hình nội dung tài liệu (như các định nghĩa DTD và lược đồ XML)

Phân cấp cây DOM

- Một nút Document bao gồm.
 - Một nút Element (Đây chính là Element gốc của tài liệu)
 - Một hoặc nhiều các nút chỉ lệnh (Processing Instruction)
 - Một hoặc nhiều nút Comment
- Một nút Element có thể chứa
 - Các nút Element khác
 - Một hoặc nhiều nút Text
 - Một hoặc nhiều nút thuộc tính (Attr)
 - Một nút thuộc tính (Attr) chứa một nút Text

5.2. Nạp dữ liệu XML

Để minh họa cho việc xử lý dữ liệu XML theo mô hình DOM, xét tài liệu *book.xml*, tài liệu này chứa các thông tin về các quyển sách trên quầy sách.

Ví dụ 5.2: (*books.xml*)

```
<?xml version="1.0"?>
<bookstore>
    <book category="cooking">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price> </book>
    <book category="children">
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price> </book>
    <book category="web">
        <title lang="en">XQuery Kick Start</title>
        <author>James McGovern</author>
        <author>Per Bothner</author>
        <author>Kurt Cagle</author>
        <author>James Linn</author>
        <author>Vaidyanathan Nagarajan</author>
        <year>2003</year>
        <price>49.99</price> </book>
    <book category="web" cover="paperback">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year> <price>39.95</price> </book>
    </bookstore>
```

5.2.1. Truy xuất giá trị của một nút

Ví dụ này đọc *books.xml* vào *xmlDoc* và truy xuất các giá trị văn bản của *<title>* phần tử đầu tiên trong *books.xml*:

Ví dụ 5.3:

```
<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);
    }
}
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    document.getElementById("demo").innerHTML =
        xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
}
</script> </body> </html>
```

Kết quả hiển thị trên trình duyệt như sau:

Everyday Italian

Giải thích ví dụ:

- *xmlDoc* - đối tượng DOM XML được tạo ra bởi các phân tích cú pháp.
- *getElementsByTagName ("title") [0]* - nhận *<title>* phần tử đầu tiên
- *childNodes [0]* - nút con đầu tiên của *<title>* (nút văn bản)
- *nodeValue* - giá trị của các nút (các văn bản chính nó)

5.2.2. Nạp một chuỗi XML

Ví dụ này nạp một chuỗi văn bản thành một đối tượng XML DOM và trích xuất các thông tin XML với JavaScript:

Ví dụ 5.4:

```
<html>
<body> <p id="demo"></p>
<script>
    var text, parser, xmlDoc;
    text = "<bookstore><book>" + "<title>Everyday Italian
    </title>" + "<author>Giada De Laurentiis</author>" +
    "<year>2005</year>" + "</book> </bookstore>";
    parser = new DOMParser();
    xmlDoc = parser.parseFromString(text,"text/xml");
    document.getElementById("demo").innerHTML =
        xmlDoc.getElementsByTagName("title")[0].childNodes[0]
            .nodeValue;
</script> </body> </html>
```

Kết quả hiển thị trên trình duyệt cũng là “Everyday Italian” như Ví dụ 5.3.

5.3. Giao diện lập trình

5.3.1. Thuộc tính XML DOM

Một số thuộc tính DOM tiêu biểu:

- x.nodeName - tên của x
- x.nodeValue - giá trị của x
- x.parentNode - node cha của x
- x.childNodes - các nút con của x
- x.attributes - các thuộc tính nút của x

Lưu ý: Trong danh sách trên, x là một đối tượng nút.

5.3.2. Phương pháp XML DOM (XML DOM Methods)

- x.getElementsByTagName – lấy các phần tử với tên thẻ quy định
- x.appendChild (node) - chèn một nút con vào x
- x.removeChild (node) - loại bỏ một nút con từ x

Lưu ý: Trong danh sách trên, x là một đối tượng nút.

5.3.3. Các nút của DOM (DOM nodes)

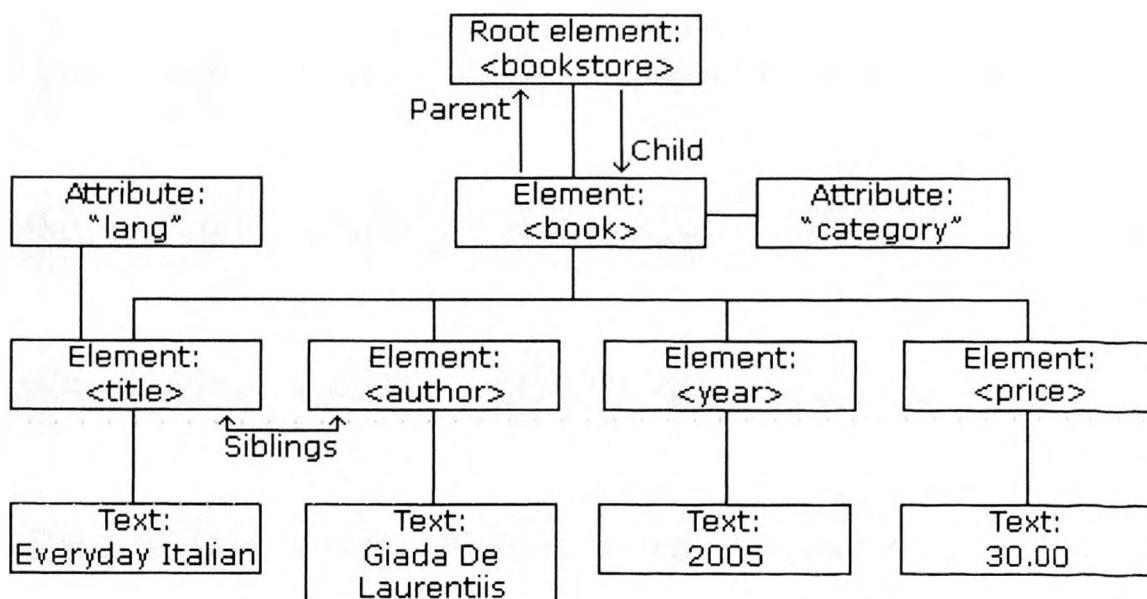
Xét lại Ví dụ 5.2 (books.xml):

Nút gốc trong tập tin XML ở trên được đặt tên là <bookstore>. Tất cả các nút khác trong tài liệu được chứa trong <bookstore>. Nút gốc <bookstore> giữ 4 <book> nút. Đầu tiên nút <book> giữ nút con: <title>, <author>, <năm> và <giá>. Các nút con lần lượt chứa một nút văn bản, "Everyday Ý", "Giada De Laurentiis", "2005", và "30.00".

Cấu trúc cây của XML DOM

XML DOM xem một tài liệu XML như là một cấu trúc cây. Tất cả các nút có thể được truy cập thông qua cây. Nội dung của chúng có thể được sửa đổi hoặc xóa và có thể tạo ra các phần tử mới. Cây nút hiển thị tập hợp các nút và các kết nối giữa chúng. Cây bắt đầu tại nút gốc và các nhánh chỉ ra các nút văn bản ở mức thấp nhất của cây:

Hình dưới đây trình bày thể hiện của books.xml dưới dạng cây:



Hình 5.3: Hiển thị của books.xml dưới dạng cây

Nút Cha mẹ, con cái và anh chị em ruột (Node Parents, Children, and Siblings):

Các nút trong cây có một mối quan hệ thứ bậc với nhau. Nút cha mẹ có các nút con. Các nút con trên cùng một mức độ được gọi là anh chị em ruột (sibling).

Trong một cây nút, nút trên cùng được gọi là gốc. Mỗi nút, trừ nút gốc, có chính xác một nút cha mẹ. Một nút có thể có bất kỳ số lượng con cái. Một nút lá là một nút không có con. Nút anh chị em là các nút có cùng nút cha mẹ.

Bởi vì dữ liệu XML được cấu trúc theo dạng cây, nó có thể được duyệt qua mà không biết cấu trúc chính xác của cây và không biết loại dữ liệu chứa bên trong.

Nút con đầu – nút con cuối (First Child - Last Child)

Nhìn vào đoạn XML sau đây:

```
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>
```

Trong XML trên, `<title>` là nút con đầu tiên của phần tử `<book>` và phần tử `<price>` là nút con cuối cùng của phần tử `<book>`. Hơn nữa, phần tử `<book>` là nút cha của các phần tử `<title>`, `<author>`, `<year>` và `<price>`.

5.4. Truy cập vào các nút XML DOM

Với DOM, chúng ta có thể truy cập vào tất cả các nút trong một tài liệu XML. Các ví dụ dưới đây sử dụng tập tin XML books.xml.

Truy cập một nút sử dụng số chỉ số của nó trong một danh sách nút

Sử dụng phương thức `getElementsByName()` để có được `<title>`, phần tử thứ ba trong "books.xml"

Ví dụ 5.5:

```
<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    myFunction(this);
  }
}
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
  var xmlDoc = xml.responseXML;
  var x = xmlDoc.getElementsByTagName("title");
  document.getElementById("demo").innerHTML =
  x[2].childNodes[0].nodeValue;
}
</script>
</body>
</html>
```

Kết quả hiển thị trên trình duyệt là giá trị “XQuery Kick Start” của phần tử `<title>` thứ ba.

Xem các loại nút của một phần tử

Sử dụng thuộc tính `nodeType` để có được loại nút của phần tử gốc trong "books.xml".

Ví dụ 5.6:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);
    }
};
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    document.getElementById("demo").innerHTML =
        xmlDoc.documentElement.nodeName + "<br>" +
        xmlDoc.documentElement.nodeType;
}
</script>
</body>
</html>
```

Kết quả hiển thị trên trình duyệt là:

```
bookstore
1
```

Vòng qua các nút phần tử bằng cách sử dụng các mối quan hệ nút

Sử dụng thuộc tính `nodeType` và thuộc tính `nextSibling` để xử lý các nút phần tử trong "books.xml".

Ví dụ 5.7:

```
<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200)
        myFunction(this);
};
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var x, y, i, xlen, xmlDoc, txt;
    xmlDoc = xml.responseXML;
    x = xmlDoc.getElementsByTagName("book")[0];
    xlen = x.childNodes.length;
    y = x.firstChild;
    txt = "";
}
```

```

        for (i = 0; i < xlen; i++) {
            if (y.nodeType == 1) {
                txt += i + " " + y.nodeName + "<br>";
            }           y = y.nextSibling;
        }
    document.getElementById("demo").innerHTML = txt; }
</script> </body>
</html>

```

Kết quả hiển thị trên trình duyệt như sau:

```

1 title
3 author
5 year
7 price

```

5.5. Truy cập vào các nút

Chúng ta có thể truy cập vào một nút bằng một trong ba cách:

1. Bằng cách sử dụng các phương thức getElementsByTagName ()
2. Bằng cách duyệt qua (traversing) các nút của cây.
3. Bằng cách điều hướng cây nút, sử dụng các mối quan hệ của nút.

5.5.1. Phương thức getElementsByTagName ()

getElementsByTagName () trả về các yếu tố với một tên thẻ chỉ định.

Cú pháp:

```
node.getElementsByTagName("tagname");
```

Ví dụ 5.8: Ví dụ sau đây trả về tất cả phần tử <title> dưới phần tử x:

```
x.getElementsByTagName("title");
```

Lưu ý rằng ví dụ trên chỉ trả lại phần tử <title> dưới nút x. Để trả lại tất cả phần tử <title> trong tài liệu XML, ta sử dụng câu lệnh sau:

```
xmlDoc.getElementsByTagName("title");
```

trong đó xmlDoc là bản thân tài liệu (nút tài liệu- document node)

Phương thức *getElementsByTagName ()* trả về một danh sách nút. Một danh sách nút là một mảng của các nút.

Phần tử <Title> trong x có thể được truy cập bởi số chỉ mục. Để truy cập vào <title> thứ ba, chúng ta có thể viết:

```
y = x[2];
```

Lưu ý: Các chỉ số bắt đầu từ 0.

5.5.2. Duyệt qua các nút của cây

Các lệnh sau vòng qua các nút con của nút gốc, đó cũng là các nút phần tử:

Ví dụ 5.9:

```

<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this); }
}
xhttp.open("GET", "books.xml", true);
xhttp.send();

```

```

function myFunction(xml) {
    var x, i, xmlDoc, txt;
    xmlDoc = xml.responseXML;
    txt = "";
    x = xmlDoc.documentElement.childNodes;
    for (i = 0; i < x.length; i++) {
        if (x[i].nodeType == 1) {
            txt += x[i].nodeName + "<br>"; }
    }
    document.getElementById("demo").innerHTML = txt; }
</script>
</body>
</html>
```

Kết quả hiển thị trên trình duyệt là:

```

book
book
book
book
```

Giải thích ví dụ trên: Giả sử chúng ta đã tải "books.xml" vào xmlDoc; Nhận các nút con của phần tử gốc (xmlDoc); Đối với mỗi nút con, kiểm tra các loại nút; Nếu các loại nút là "1" thì nó là một nút phần tử; Xuất ra tên của nút nếu nó là một nút phần tử.

5.5.3. Điều hướng cây nút

Các lệnh sau đây điều hướng cây nút bằng cách sử dụng mối quan hệ nút:

Ví dụ 5.10:

```

<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this); } };
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var x, y, i, xlen, xmlDoc, txt;
    xmlDoc = xml.responseXML;
    x = xmlDoc.getElementsByTagName("book")[0];
    xlen = x.childNodes.length;
    y = x.firstChild;
    txt = "";
    for (i = 0; i < xlen; i++) {
        if (y.nodeType == 1) {
            txt += i + " " + y.nodeName + "<br>"; }
        y = y.nextSibling; }
    document.getElementById("demo").innerHTML = txt; }
</script> </body>
</html>
```

Giải thích ví dụ: Giả sử chúng ta đã tải "books.xml" vào xmldoc; Nhận các nút con của phần tử cuốn sách đầu tiên; Thiết lập biến "y" là nút con đầu tiên của cuốn sách đầu tiên; Đối với mỗi nút con: Kiểm tra các loại nút; Nếu các loại nút là "1" thì nó là một nút phần tử; Xuất ra tên của nút nếu nó là một nút phần tử; Thiết lập biến "y" là nút anh chị em tiếp theo và chạy qua các vòng lặp lại.

5.6. Các thuộc tính của một nút trong XML DOM

Trong XML DOM, mỗi nút là một đối tượng. Đối tượng có các phương thức và thuộc tính, có thể được truy cập và thao tác bằng JavaScript. Ba thuộc tính quan trọng của nút là:

- nodeName
- nodeValue
- nodeType

5.6.1. Thuộc tính của nodeName

Thuộc tính nodeName chỉ định tên của một nút.

- nodeName là chỉ đọc
- nodeName của một nút phần tử cũng giống như tên thẻ
- nodeName của một nút thuộc tính là tên thuộc tính
- nodeName của một nút văn bản luôn luôn là #text
- nodeName của nút tài liệu luôn là #document

5.6.2. Thuộc tính của nodeValue

Thuộc tính của nodeValue chỉ định giá trị của một nút.

- nodeValue cho nút phần tử là không xác định
- nodeValue cho các nút văn bản là văn bản chính nó
- nodeValue cho các nút thuộc tính là giá trị thuộc tính

Các mã sau đây lấy lại giá trị nút phần tử <title> đầu tiên:

```
var x =  
xmldoc.getElementsByTagName("title")[0].childNodes[0];  
var txt = x.nodeValue;
```

Giải thích ví dụ: Giả sử chúng ta đã tải "books.xml" vào xmldoc; Nhận nút văn bản của nút phần tử <title> đầu tiên; Thiết lập biến txt chứa giá trị của nút văn bản.

Thay đổi giá trị của một phần tử: Các mã sau đây thay đổi giá trị của phần tử <title> đầu tiên:

Ví dụ 5.11:

```
<!DOCTYPE html>  
<html> <body>  
<p id="demo1"></p>  
<p id="demo2"></p>  
<script>  
var xhttp = new XMLHttpRequest();  
xhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200) {  
        myFunction(this);  
    } };  
xhttp.open("GET", "books.xml", true);
```

```

xhttp.send();
function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    var x;
    x= xmlDoc.getElementsByTagName("title")[0].childNodes[0];
    document.getElementById("demo1").innerHTML =
        x.nodeValue;
    x.nodeValue = "Everyday Spanish";
    x= xmlDoc.getElementsByTagName("title")[0].childNodes[0];
    document.getElementById("demo2").innerHTML =
        x.nodeValue; }

</script> </body>
</html>

```

Kết quả hiển thị trên trình duyệt như sau:

Everyday Italian
Everyday Spanish

Giải thích ví dụ: Giả sử chúng ta đã tải "books.xml" vào xmlDoc; Nhận giá trị của nút phần tử <title> đầu tiên; Thay đổi giá trị của nút văn bản thành "Easy Cooking".

5.6.3. Thuộc tính của.nodeType

Thuộc tính *nodeType* quy định các kiểu của nút; *nodeType* chỉ được đọc.

Các loại nút quan trọng nhất là:

Loại nút	Giá trị
Phần tử (Element)	1
Thuộc tính (Attribute)	2
Văn bản (Text)	3
Chú thích (Comment)	8
Tài liệu (Document)	9

5.7. Danh sách nút của XML DOM

Khi sử dụng các thuộc tính hoặc phương thức như *childNodes* hoặc *getElementsByName()*, một đối tượng danh sách nút được trả về. Một đối tượng danh sách nút đại diện cho một danh sách các nút, theo thứ tự như trong XML. Các nút trong danh sách nút được truy cập với số lượng chỉ số bắt đầu từ 0. Hình 5.4 đại diện cho một danh sách nút của phần tử <title> trong "books.xml".

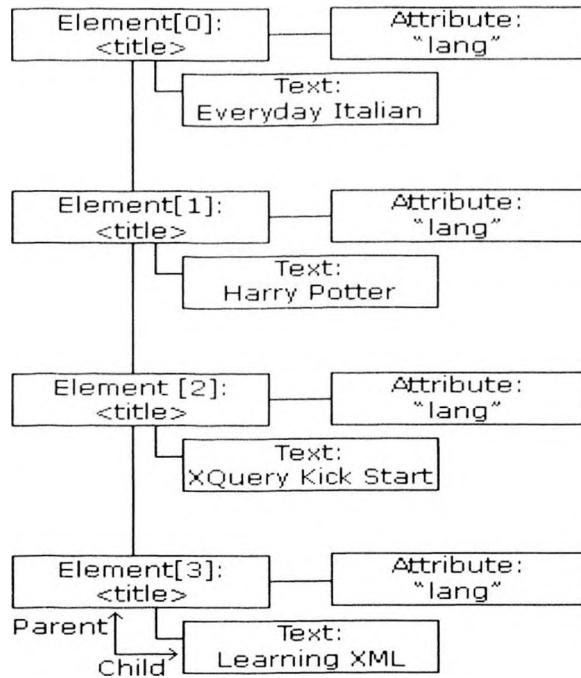
Giả sử "books.xml" được nạp vào biến xmlDoc. Đoạn mã này trả về một danh sách các nút của các phần tử <title> trong "books.xml":

```
x = xmlDoc.getElementsByTagName("title");
```

Sau khi thực hiện câu lệnh trên, x là một đối tượng danh sách nút. Đoạn mã sau trả về giá trị của phần tử <title> đầu tiên trong danh sách nút (x):

```
var txt = x[0].childNodes[0].nodeValue;
```

Sau khi thực hiện các lệnh trên, txt = "Everyday Italian".



Hình 5.4: Danh sách các nút của phần tử <title> trong books.xml

5.7.1. Chiều dài của danh sách nút

Một đối tượng danh sách luôn tự cập nhật. Nếu một phần tử bị xóa hoặc thêm, danh sách được tự động cập nhật. Thuộc tính *length* của một danh sách nút là số lượng nút trong danh sách.

Đoạn mã dưới đây trả về số lượng phần tử <title> trong "books.xml":

```
x = xmlDoc.getElementsByTagName('title').length;
```

Sau khi thực hiện câu lệnh trên, giá trị của x sẽ là 4.

Chiều dài của danh sách nút có thể được sử dụng để lặp qua tất cả các phần tử trong danh sách. Ví dụ sau đây sử dụng thuộc tính *length* để lặp qua danh sách các phần tử <title>:

Ví dụ 5.12:

```

<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);
    }
}
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var x, i, xmlDoc, txt;
    xmlDoc = xml.responseXML;
    txt = "";
    x = xmlDoc.getElementsByTagName('title');
    for (i = 0 ; i <x.length; i++) {
        txt += x[i].childNodes[0].nodeValue + "<br>";
    }
    document.getElementById("demo").innerHTML = txt;
}
</script> </body>
</html>

```

Kết quả hiển thị trên trình duyệt như sau:

Everyday Italian
Harry Potter
XQuery Kick Start
Learning XML

Giải thích ví dụ: Giả sử "books.xml" được nạp vào xmldoc; Thiết lập biến x để giữ một danh sách nút của tất cả các phần tử <title>; Thu thập các giá trị nút văn bản từ các phần tử <title>.

5.7.2. Danh sách thuộc tính DOM

Thuộc tính (attribute) của một nút phần tử trả về một danh sách các nút thuộc tính. Thuộc tính này được gọi là một bản đồ nút được đặt tên và tương tự với một danh sách nút, ngoại trừ một số khác biệt trong phương thức và thuộc tính. Một danh sách thuộc tính luôn tự cập nhật. Nếu một thuộc tính được xóa hoặc thêm, danh sách được tự động cập nhật.

Câu lệnh này trả về một danh sách các nút thuộc tính từ phần tử <book> đầu tiên trong "books.xml":

```
x = xmlDoc.getElementsByTagName('book')[0].attributes;
```

Sau khi thực hiện của lệnh trên, x.length = là số thuộc tính và x.getNamedItem() có thể được sử dụng để trả lại một nút thuộc tính.

Ví dụ dưới đây lấy giá trị của thuộc tính "category" và số lượng các thuộc tính, trong một cuốn sách:

Ví dụ 5.13:

```
<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);
    }
}
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var x, txt, xmlDoc;
    xmlDoc = xml.responseXML;
    x = xmlDoc.getElementsByTagName("book")[0].attributes;
    txt = x.getNamedItem("category").nodeValue + "<br>" +
    x.length;
    document.getElementById("demo").innerHTML = txt;
}
</script>
</body>
</html>
```

Kết quả hiển thị trên trình duyệt như sau:

cooking
1

Giải thích ví dụ: Giả sử "books.xml" được nạp vào xmldoc; Thiết lập biến x để giữ một danh sách của tất cả các thuộc tính của phần tử <book> đầu tiên; Lấy giá trị của thuộc tính "category" và chiều dài của danh sách thuộc tính.

5.8. Duyệt cây nút XML DOM

Thông thường khi truy xuất trên tài liệu XML, có một số công việc chúng ta muốn chúng được lặp đi lặp lại, ví dụ: khi chúng ta muốn trích xuất các giá trị của mỗi phần tử, điều này được gọi là "duyệt qua các cây nút".

Ví dụ dưới đây duyệt qua tất cả các nút con của phần tử `<book>` và hiển thị tên và giá trị của phần tử này:

Ví dụ 5.14:

```
<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var x, i, xmlDoc;
var txt = "";
var text = "<book>" +
"<title>Everyday Italian</title>" + "<author>Giada De
Laurentiis</author>" +
"<year>2005</year>" + "</book>";
parser = new DOMParser();
xmlDoc = parser.parseFromString(text, "text/xml");
// documentElement always represents the root node
x = xmlDoc.documentElement.childNodes;
for (i = 0; i < x.length ;i++) {
    txt += x[i].nodeName + ": " +
        x[i].childNodes[0].nodeValue + "<br>";}
document.getElementById("demo").innerHTML = txt;
</script> </body>
</html>
```

Kết quả hiển thị trên trình duyệt như sau:

```
title: Everyday Italian
author: Giada De Laurentiis
year: 2005
```

Giải thích: Tải các chuỗi XML vào `xmlDoc`; Nhận các nút con của phần tử gốc; Đối với mỗi nút con, xuất ra tên nút và các giá trị nút của nút văn bản.

Một số khác biệt giữa các trình duyệt trong việc biên dịch DOM:

- Tất cả các trình duyệt hiện đại hỗ trợ các đặc điểm kỹ thuật W3C DOM.
- Tuy nhiên, có một số khác biệt giữa các trình duyệt. Một sự khác biệt quan trọng là: Cách xử lý khoảng trắng và dòng mới.

5.8.1. Khoảng trắng và dòng mới trong DOM (White Spaces and New Lines)

Tài liệu XML thường chứa dòng mới hoặc các ký tự khoảng trắng giữa các nút. Điều này thường xảy ra khi các tài liệu được soạn thảo bởi một trình soạn thảo đơn giản, ví dụ như Notepad.

Ví dụ sau đây (được soạn thảo bởi Notepad) chứa CR / LF (dòng mới) giữa mỗi dòng và hai ký tự trắng ở phía trước của mỗi nút con:

```
<book>
    <title>Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price> </book>
```

Internet Explorer 9 (IE9) và các phiên bản trước đó không xem khoảng trắng hoặc dòng mới như các nút văn bản, trong khi các trình duyệt khác có.

Ví dụ sau sẽ xuất ra số lượng các nút con của phần tử gốc (trong books.xml). IE9 và các phiên bản trước đó sẽ ra 4 con nút, trong khi IE10 và các phiên bản sau này và các trình duyệt khác sẽ ra 9 nút con:

Ví dụ 5.15:

```
<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);
    }
}
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    var x = xmlDoc.documentElement.childNodes;
    document.getElementById("demo").innerHTML =
    "Number of child nodes: " + x.length;
}
</script> </body>
</html>
```

Kết quả hiển thị trên trình duyệt IE 11 là:

Number of child nodes: 9

5.8.2. Dữ liệu ký tự được phân tích (Parsed Character Data - PCDATA)

Bộ phân tích cú pháp XML thường phân tích tất cả các văn bản trong một tài liệu XML. Khi một phần tử XML được phân tích cú pháp, các văn bản giữa các thẻ XML cũng được phân tích:

```
<message>This text is also parsed</message>
```

Bộ phân tích cú pháp thực hiện điều này vì các phần tử XML có thể chứa các yếu tố khác, như trong ví dụ này, phần tử `<name>` chứa hai phần tử khác (`first` và `last`):

```
<name><first>Bill</first><last>Gates</last></name>
```

và bộ phân tích cú pháp sẽ phân tách nó ra thành các phần tử như thế này:

```
<name>
  <first>Bill</first>
  <last>Gates</last>
</name>
```

Dữ liệu ký tự được phân tích (PCDATA) là được sử dụng để mô tả cho dữ liệu văn bản sẽ được phân tích bởi bộ phân tích cú pháp XML.

5.8.3. Dữ liệu ký tự không được phân tích (Unparsed Character Data - CDATA)

Thuật ngữ CDATA được sử dụng để biểu diễn cho loại dữ liệu văn bản sẽ không được phân tích bởi bộ phân tích cú pháp XML. Thông thường, các ký tự như "<" và "&" là không hợp lệ trong các phần tử XML. Ký tự "<" sẽ sinh ra một lỗi vì bộ phân tích cú pháp diễn giải nó là ký tự khởi đầu của một phần tử mới. Ký tự "&" sẽ tạo ra một lỗi vì phân tích cú pháp diễn giải nó là khởi đầu

của một thực thể ký tự. Một số văn bản, như mã JavaScript, chứa rất nhiều ký tự "<" hoặc "&". Để tránh các lỗi trên, chúng ta có thể định nghĩa chúng là CDATA. Tất cả mọi thứ bên trong phần CDATA sẽ được bỏ qua bởi trình phân tích.

Một phần CDATA bắt đầu với "<![CDATA[" và kết thúc với "]]>":

Ví dụ 5.16:

```
<script>
<! [CDATA[
function matchwo(a,b) {
    if (a < b && a < 0) {
        return 1;
    } else { return 0; }
} ]>
</script>
```

Trong ví dụ trên, tất cả mọi thứ bên trong phần CDATA được bỏ qua bởi trình phân tích.

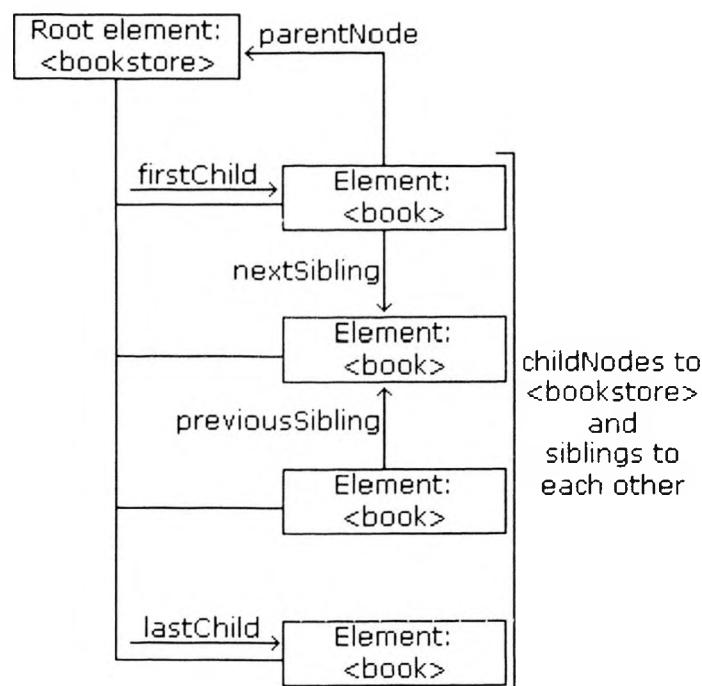
Ghi chú về CDATA:

Nội dung bên trong CDATA không thể chứa chuỗi "]]>"; CDATA không cho phép lồng các CDATA bên trong; Các "]]>" đánh dấu sự kết thúc của phần CDATA không thể chứa dấu cách hoặc ngắt dòng.

5.9. Duyệt qua các nút trong DOM

Truy cập vào các nút trong cây nút thông qua các mối quan hệ giữa các nút, thường được gọi là "duyệt nút". Trong XML DOM, các mối quan hệ nút được xác định là thuộc tính của các nút: parentNode, childNodes, firstChild, lastChild, nextSibling, previousSibling.

Những hình ảnh dưới đây minh họa một phần của cây nút và các mối quan hệ giữa các nút trong books.xml:



Hình 5.6: Một phần của cây nút và các mối quan hệ giữa các nút trong books.xml

5.9.1. Nút cha (parentNode)

Tất cả các nút có chính xác một nút cha. Đoạn lệnh sau đây duyệt các nút cha của phần tử <book>:

Ví dụ 5.17:

```
function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    var x = xmlDoc.getElementsByTagName("book")[0];
    document.getElementById("demo").innerHTML =
    x.parentNode.nodeName;
}
```

Giải thích ví dụ: Nạp "books.xml" vào xmlhttp; Lấy phần tử <book> đầu tiên; Xuất ra tên nút của nút cha của "x".

5.9.2. Tránh các nút rỗng

Firefox và một số trình duyệt khác, sẽ xử lý khoảng trắng hoặc dòng mới như các nút văn bản nhưng Internet Explorer thì không. Điều này gây ra một vấn đề khi sử dụng các thuộc tính: firstChild, lastChild, nextSibling, previousSibling.

Để tránh duyệt đến các nút rỗng (nút chỉ chứa khoảng trắng và ký tự dòng mới giữa các nút phần tử), chúng ta sử dụng một hàm để kiểm tra các loại nút:

Ví dụ 5.18:

```
function get_nextSibling(n) {
    var y = n.nextSibling;
    while (y.nodeType != 1) {
        y = y.nextSibling;    }
    return y;
}
```

Hàm trên cho phép chúng ta sử dụng get_nextSibling (node) thay vì sử dụng thuộc tính node.nextSibling.

Giải thích ví dụ: Nút phần tử là loại 1. Nếu nút anh chị em không phải là một nút phần tử, nó di chuyển đến các nút tiếp theo cho đến khi một nút phần tử được tìm thấy. Bằng cách này, kết quả sẽ là như nhau trong cả trình duyệt Internet Explorer và Firefox.

5.9.3. Lấy phần tử con đầu tiên

Các lệnh sau đây hiển thị các nút phần tử đầu tiên của <book> đầu tiên:

Ví dụ 5.19:

```
<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);    }};
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    var x =
get.firstChild(xmlDoc.getElementsByTagName("book")[0]);
```

```

        document.getElementById("demo").innerHTML =
            ' ' + x.nodeName; }

//check if the first node is an element node
function getFirstChild(n) {
    var y = n.firstChild;
    while (y.nodeType != 1) {y = y.nextSibling; }
    return y;
}
</script> </body>
</html>

```

Kết quả hiển thị trên trình duyệt là:

title

Giải thích ví dụ: Nạp "books.xml" vào xmlhttp; Sử dụng hàm `getFirstChild` trên phần tử `<book>` đầu tiên để có được nút phần tử con đầu tiên; Xuất ra tên nút của nút phần tử con đầu tiên.

5.9.4. Lấy phần tử con cuối cùng

Ví dụ 5.20: Ví dụ này sử dụng phương thức `lastChild()` và một chức năng tùy chỉnh để có được các nút con cuối cùng của một nút.

```

<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);
    }
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    var x =
        get_lastChild(xmlDoc.getElementsByTagName("book")[0]);
    document.getElementById("demo").innerHTML =
        x.nodeName;
}
function get_lastChild(n) {
    var y = n.lastChild;
    while (y.nodeType!=1) {
        y = y.previousSibling;
    }
    return y;
}
</script> </body>
</html>

```

Kết quả hiển thị trên trình duyệt là “price”

5.9.5. Lấy nút anh em tiếp theo

Ví dụ 5.21: Ví dụ này sử dụng phương thức `nextSibling()` và một chức năng tùy chỉnh để có được các nút anh chị em tiếp theo của một nút

```

<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {

```

```

        myFunction(this); }};
 xhttp.open("GET", "books.xml", true);
 xhttp.send();
 function myFunction(xml) {
     var xmlDoc = xml.responseXML;
     var x =
get_nextSibling(xmlDoc.getElementsByTagName("title")[0]);
     document.getElementById("demo").innerHTML =
         x.nodeName; }

function get_nextSibling(n) {
    var y = n.nextSibling;
    //check if the node is an element node
    while (y.nodeType != 1) {y = y.nextSibling; }
    return y; }
</script> </body>
</html>
```

Kết quả hiển thị trên trình duyệt là “author”

5.9.6. Lấy nút anh em trước đó

Ví dụ 5.22: Ví dụ này sử dụng phương thức *previousSibling()* và một chức năng tùy chỉnh để có được các nút anh chị em trước đó của một nút.

```

<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this); }}
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    var x =
get_previousSibling(xmlDoc.getElementsByTagName("price")[0]);
    document.getElementById("demo").innerHTML =
        x.nodeName; }
function get_previousSibling(n) {
```

```

    var y = n.previousSibling;
    //check if the node is an element node
    while (y.nodeType != 1) {
        y = y.previousSibling;
    } return y; }
</script> </body> </html>
```

Kết quả hiển thị trên trình duyệt là “year”

5.10. Lấy giá trị của nút

- Trong DOM, tất cả phần tử được xem là nút, nút phần tử không chứa giá trị văn bản. Các giá trị văn bản của một nút phần tử được lưu trữ trong một nút con. Nút này được gọi là một nút văn bản. Để lấy giá trị văn bản, chúng ta phải lấy giá trị của nút văn bản của các phần tử.

- Thuộc tính *nodeValue* được sử dụng để có được những giá trị văn bản của một nút văn bản.
- Phương thức *getElementsByTagName()* trả về một danh sách nút của tất cả các phần tử, với tên thẻ nhất định, theo thứ tự như chúng xuất hiện trong tài liệu nguồn.
- Thuộc tính *childNodes* trả về danh sách các nút con của một phần tử.

Ví dụ 5.23:

```
<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);
    }
}
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    var x = xmlDoc.getElementsByTagName('title')[0];
    var y = x.childNodes[0];
    document.getElementById("demo").innerHTML =
    y.nodeValue; } </script> </body>
</html>
```

Kết quả hiển thị trên trình duyệt là

Everyday Italian

Lấy giá trị của một thuộc tính

Trong DOM, thuộc tính là các nút. Không giống như các nút phần tử, các nút thuộc tính có giá trị văn bản. Để có được giá trị của một thuộc tính là lấy giá trị văn bản của nó. Điều này có thể được thực hiện bằng cách sử dụng phương thức *getAttribute()* hoặc sử dụng thuộc tính *nodeValue* của các nút thuộc tính.

Phương thức *getAttribute()* trả về giá trị của một thuộc tính.

Ví dụ 5.24: Các mã sau đây lấy giá trị văn bản của thuộc tính "category" của các phần tử *<book>*:

```
<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);
    }
}
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var x, i, xmlDoc, txt;
    xmlDoc = xml.responseXML;
    txt = "";
    x = xmlDoc.getElementsByTagName('book');
```

```

for (i = 0; i < x.length; i++) {
    txt += x[i].getAttribute('category') + "<br>" ;
}
document.getElementById("demo").innerHTML = txt;
}</script> </body>
</html>

```

Kết quả hiển thị trên trình duyệt là:

```

cooking
children
web
web

```

Phương thức `getAttributeNode()` trả về một nút thuộc tính.

Đoạn lệnh sau đây lấy giá trị văn bản của thuộc tính "lang" của phần tử `<title>` đầu tiên:

Ví dụ 5.25:

```

x = xmlDoc.getElementsByTagName("title")[0];
y = x.getAttributeNode("lang");
txt = y.nodeValue;

```

5.11. Thao tác trên giá trị của các nút

5.11.1. Thay đổi giá trị của nút

- Để thay đổi các giá trị văn bản của một phần tử, chúng ta phải thay đổi giá trị của nút văn bản của phần tử đó.

Ví dụ 5.26: Giả sử "books.xml" đã được nạp vào `xmldoc`. Câu lệnh này thay đổi giá trị của phần tử `<title>` đầu tiên:

```

xmldoc.getElementsByTagName("title")[0].childNodes[0].nodeValue = "new content"

```

- Thay đổi một thuộc tính bằng cách sử dụng phương thức `setAttribute()`. Nếu thuộc tính không tồn tại, một thuộc tính mới được tạo ra.

Ví dụ 5.27: Ví dụ sau lấy phần tử `<book>` đầu tiên, sau đó thay đổi giá trị của thuộc tính "category" thành "food"

```

<!DOCTYPE html>
<html> <body> <p id="demo"></p> <script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this);
    }
}
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    var x = xmlDoc.getElementsByTagName('book');
    x[0].setAttribute("category", "food");
    document.getElementById("demo").innerHTML =
    x[0].getAttribute("category");
}
</script> </body>
</html>

```

Chúng ta cũng có thể sử dụng `nodeValue` để thay đổi giá trị của thuộc tính: `xmldoc.getElementsByTagName("book")[0].getAttributeNode("category").nodeValue = "food";`

5.11.2. Gỡ bỏ các nút (Remove nodes)

Phương thức `removeChild()` loại bỏ một nút cụ thể. Khi một nút được lấy ra, tất cả các nút con của nó cũng bị xóa.

Ví dụ 5.28: Câu lệnh sau sẽ loại bỏ phần tử `<book>` đầu tiên của tài liệu `book.xml`:

```
y = xmlDoc.getElementsByTagName("book")[0];
xmlDoc.documentElement.removeChild(y);
```

Phương thức `removeChild()` là cách duy nhất để loại bỏ một nút được chỉ định. Khi chúng ta di chuyển đến các nút chúng ta muốn loại bỏ, chúng ta có thể loại bỏ nút đó bằng cách sử dụng thuộc tính `parentNode` và phương thức `removeChild()`:

Ví dụ 5.29:

```
x = xmlDoc.getElementsByTagName("book")[0];
x.parentNode.removeChild(x);
```

Phương thức `removeChild()` cũng có thể được sử dụng để loại bỏ một nút văn bản:

Ví dụ 5.30:

```
x = xmlDoc.getElementsByTagName("title")[0];
y = x.childNodes[0];
x.removeChild(y);
```

Giải thích ví dụ: Giả sử "books.xml" được nạp vào `xmldoc`; Đặt biến `x` là nút phần tử `title` đầu tiên; Đặt biến `y` là các nút văn bản để loại bỏ; Loại bỏ nút phần tử bằng cách sử dụng các phương thức `removeChild()` từ nút cha.

Phương thức `removeChild()` không được thông dụng trong việc dùng để loại bỏ các văn bản từ một nút, thay vào đó người ta thường dùng thuộc tính `nodeValue`. Xem đoạn tiếp theo.

Ví dụ 5.31: `nodeValue` được sử dụng để thay đổi giá trị của một nút văn bản:

```
xmldoc.getElementsByTagName("title")[0].childNodes[0].nodeValue = "";
```

Câu lệnh trên nhận nút `title` đầu tiên và sử dụng thuộc tính `nodeValue` để xóa nội dung từ nút văn bản. Phương thức `removeAttribute()` loại bỏ một nút thuộc tính bằng tên của nó.

Ví dụ 5.32: `removeAttribute('category')`. Lệnh sau đây loại bỏ thuộc tính "category" trong phần tử `<book>` đầu tiên:

```
x = xmlDoc.getElementsByTagName("book");
x[0].removeAttribute("category");
```

Phương thức `removeAttributeNode()` loại bỏ một nút thuộc tính, sử dụng các đối tượng nút là tham số, ví dụ như `removeAttributeNode()`.

Ví dụ 5.33: Loại bỏ tất cả các thuộc tính của phần tử `<book>`:

```
<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    myFunction(this);
  }
};
```

```

xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var x, i, attnode, old_att, xmlDoc, txt;
    xmlDoc = xml.responseXML;
    txt = "";
    x = xmlDoc.getElementsByTagName('book');
    for (i = 0; i < x.length; i++) {
        while (x[i].attributes.length > 0) {
            attnode = x[i].attributes[0];
            old_att =
                x[i].removeAttributeNode(attnode);
            txt += "Removed: " + old_att.nodeName +
                ":" + old_att.nodeValue + "<br>"; } }
    document.getElementById("demo").innerHTML = txt; }
</script> </body>
</html>
```

Kết quả hiển thị trên trình duyệt là:

```

Removed: category: cooking
Removed: category: children
Removed: category: web
Removed: category: web
Removed: cover: paperback
```

5.11.3. Thay thế các nút (Replace nodes)

Phương thức `replaceChild()` được sử dụng để thay thế một nút. Đoạn mã sau thay thế phần tử `<book>` đầu tiên:

Ví dụ 5.34:

```

<!DOCTYPE html>
<html>
    <body>
        <p id="demo"></p>
        <script>
            var xhttp = new XMLHttpRequest();
            xhttp.onreadystatechange = function() {
                if (this.readyState == 4 && this.status == 200)
                    { myFunction(this); } };
            xhttp.open("GET", "books.xml", true);
            xhttp.send();
            function myFunction(xml) {
                var x, y, z, i, newNode, newTitle, newText,
                xmlDoc, txt;
                xmlDoc = xml.responseXML;
                txt = "";
                x = xmlDoc.documentElement;
// Create a book element, title element and a text node
                newNode = xmlDoc.createElement("book");
                newTitle = xmlDoc.createElement("title");
                newText = xmlDoc.createTextNode("A Notebook");
                // Add a text node to the title node
```

```

        newTitle.appendChild(newText);
        // Add the title node to the book node
        newNode.appendChild(newTitle);
        y = xmlDoc.getElementsByTagName("book")[0];
// Replace the first book node with the new book node
        x.replaceChild(newNode, y);
        z = xmlDoc.getElementsByTagName("title");
        // Output all titles
        for (i = 0; i < z.length; i++) {
            txt += z[i].childNodes[0].nodeValue + "<br>";
        }
        document.getElementById("demo").innerHTML = txt;
    </script>  </body>
</html>

```

Kết quả hiển thị trên trình duyệt như sau:

```

A Notebook
Harry Potter
XQuery Kick Start
Learning XML

```

Phương thức *replaceData()* được sử dụng để thay thế dữ liệu trong một nút văn bản.

Phương thức *replaceData()* có ba thông số:

- offset - Nơi bắt đầu thay thế các ký tự, giá trị offset bắt đầu là không
- length - Có bao nhiêu ký tự cần thay thế
- string - chuỗi để chèn

Ví dụ 5.35:

```

xmlDoc=loadXMLDoc("books.xml");
x=xmlDoc.getElementsByTagName("title")[0].childNodes[0];
x.replaceData(0,8,"Easy");

```

Giải thích ví dụ: Nạp "books.xml" vào xmlDoc; Lấy nút văn bản của phần tử *<title>* đầu tiên; Sử dụng phương thức *replaceData()* để thay thế tám ký tự đầu tiên từ nút văn bản với "Easy"

Một cách khác để thay thế dữ liệu trong một nút văn bản là sử dụng thuộc tính *nodeValue*.

5.11.4. Tạo các nút (Create nodes)

Phương thức *createElement()* tạo ra một nút phần tử mới.

Ví dụ 5.36: Tạo nút phần tử mới *<edition>* bên trong phần tử *<book>*

```

<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this); } };
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var x, y, i, newElement, txt, xmlDoc;

```

```

xmlDoc = xmlDoc.responseXML;
newElement = xmlDoc.createElement("edition");
x = xmlDoc.getElementsByTagName("book")[0]
x.appendChild(newElement);
// Display all elements
xlen = x.childNodes.length;
y = x.firstChild;
txt = "";
for (i = 0; i < xlen; i++) {
    if (y.nodeType == 1) {
        txt += y.nodeName + "<br>"; }
    y = y.nextSibling; }
document.getElementById("demo").innerHTML = txt;
</script> </body>
</html>

```

Kết quả hiển thị trên trình duyệt như sau:

```

title
author
year
price
edition

```

Phương thức *createAttribute()* được sử dụng để tạo ra một nút thuộc tính mới:

Ví dụ 5.37: Tạo thuộc tính mới có tên “Edition” và giá trị của thuộc tính là “first”, sau đó gán thuộc tính và giá trị mới này cho phần tử *<title>* đầu tiên trong book.xml:

```

<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this); } };
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var x, newatt, xmlDoc;
    xmlDoc = xml.responseXML;
    newatt = xmlDoc.createAttribute("edition");
    newatt.nodeValue = "first";
    x = xmlDoc.getElementsByTagName("title");
    x[0].setAttributeNode(newatt);
    document.getElementById("demo").innerHTML =
    "Edition: " + x[0].getAttribute("edition"); }
</script> </body>
</html>

```

Kết quả hiển thị trên trình duyệt như sau:

```

Edition: first

```

Lưu ý: Nếu thuộc tính đã tồn tại, nó được thay thế bằng thuộc tính mới.

- Phương thức *createTextNode()* tạo ra một nút văn bản mới:

Ví dụ 5.38: Tạo ra nút phần tử mới `<edition>`; Tiếp tục tạo ra nút văn bản “first”; Nối nút văn bản vào nút phần tử; Nối nút phần tử vào nút phần tử `<book>` đầu tiên trong tập tin book.xml:

```
<!DOCTYPE html>
<html> <body> <p id="demo"></p>
    <script>
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                myFunction(this);
            }
        }
        xhttp.open("GET", "books.xml", true);
        xhttp.send();
        function myFunction(xml) {
            var xmlDoc = xml.responseXML;
            var x, y, i, newEle, newText, txt;
            // add an edition element
            newEle = xmlDoc.createElement("edition");
            newText = xmlDoc.createTextNode("first");
            newEle.appendChild(newText);
            x = xmlDoc.getElementsByTagName("book")[0];
            x.appendChild(newEle);
            // display all elements
            xlen = x.childNodes.length;
            y = x.firstChild;
            txt = "";
            for (i = 0; i < xlen; i++) {
                if (y.nodeType == 1) {
                    txt += y.nodeName + "<br>";
                }
                y = y.nextSibling;
            }
            document.getElementById("demo").innerHTML = txt;
        }
    </script> </body>
</html>
```

Kết quả hiển thị trên trình duyệt như sau:

```
title
author
year
price
edition
```

- Phương thức *createCDATASEction()* tạo ra một nút CDATA mới.

Ví dụ 5.39: Tạo một nút CDATA mới có tên “Special Offer & Book Sale”; Nối nút CDATA mới cho phần tử `<book>` đầu tiên:

```
<!DOCTYPE html>
<html> <body> <p id="demo"></p>
    <script>
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200)
                { myFunction(this); };
        }
        xhttp.open("GET", "books.xml", true);
        xhttp.send();
        function myFunction(xml) {
            var xmlDoc = xml.responseXML;
            var x, y, i, newEle, newCData;
            x = xmlDoc.getElementsByTagName("book")[0];
            newCData = xmlDoc.createCDATASection("Special Offer & Book Sale");
            x.appendChild(newCData);
        }
    </script> </body>
</html>
```

```

xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var x, newCDATA, xmlDoc;
    xmlDoc = xml.responseXML;
newCDATA = xmlDoc.createCDATASection("Special Offer & Book Sale");
    x = xmlDoc.getElementsByTagName("book")[0];
    x.appendChild(newCDATA);
    document.getElementById("demo").innerHTML =
        x.lastChild.nodeValue; }
</script> </body>
</html>

```

Kết quả hiển thị trên trình duyệt như sau:

Special Offer & Book Sale

- Phương thức *createComment()* tạo ra một nút chú thích mới.

Ví dụ 5.40: Tạo một nút chú thích mới có tên "Revised April 2016"; Nối nút chú thích mới cho phần tử <book> đầu tiên:

```

<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this); }
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var x, newComment, xmlDoc;
    xmlDoc = xml.responseXML;
newComment = xmlDoc.createComment("Revised April 2016");
    x = xmlDoc.getElementsByTagName("book")[0];
    x.appendChild(newComment);
    document.getElementById("demo").innerHTML =
        x.lastChild.nodeValue; }
</script> </body>
</html>

```

- Phương thức *insertBefore()* chèn một nút trước nút con quy định. Phương thức này rất hữu ích khi vị trí của nút được thêm vào là quan trọng.

Ví dụ 5.41: Tạo một nút phần tử mới <book>; Chèn nút mới ở phía trước của nút phần tử <book> cuối cùng:

```

<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        myFunction(this); }
xhttp.open("GET", "books.xml", true);
xhttp.send();

```

```

function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    var newNode = xmlDoc.createElement("book");
    var x = xmlDoc.documentElement;
    var y = xmlDoc.getElementsByTagName("book");
    document.getElementById("demo").innerHTML =
    "Book elements before: " + y.length + "<br>";
    x.insertBefore(newNode, y[3]);
    document.getElementById("demo").innerHTML +=
    "Book elements after: " + y.length;
}
}</script></body>
</html>

```

Kết quả hiển thị trên trình duyệt là:

```

        Book elements before: 4
        Book elements after: 5

```

Lưu ý: Nếu tham số thứ hai của `insertBefore()` là rỗng, nút mới sẽ được thêm vào ngay sau nút con cuối cùng.

Cả hai phương thức `x.insertBefore(newNode, null)` và `x.appendChild(newNode)` đều nối thêm một nút con mới cho x.

- Thêm văn bản vào một nút văn bản hiện có bằng phương thức `insertData()`. Phương thức `insertData()` có hai tham số:

- offset – Nơi bắt đầu chèn ký tự (bắt đầu từ số không)
- string - chuỗi để chèn

5.11.5. Sao chép các nút (Copy nodes)

Fương thức `cloneNode()` tạo ra một bản sao của một nút được chỉ định. Phương thức `cloneNode()` có một tham số (`true` hoặc `false`). Giá trị `true` yêu cầu sao chép toàn bộ các thuộc tính và nút con của nút được sao chép, giá trị `false` thì chỉ sao chép nút được chỉ định.

Ví dụ 5.42: Ví dụ sau sao chép nút `<book>` đầu tiên và gắn nó vào nút gốc của tài liệu:

```

<!DOCTYPE html>
<html> <body> <p id="demo"></p>
<script>
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200)
        {   myFunction(this);   } ;
        xhttp.open("GET", "books.xml", true);
        xhttp.send();
    function myFunction(xml) {
        var x, y, cloneNode, i, xmlDoc, txt;
        xmlDoc = xml.responseXML;
        txt = "";
        x = xmlDoc.getElementsByTagName('book')[0];
        cloneNode = x.cloneNode(true);
        xmlDoc.documentElement.appendChild(cloneNode);
        // Output all titles
        y = xmlDoc.getElementsByTagName("title");

```

```
        for (i = 0; i < y.length; i++) {
            txt += y[i].childNodes[0].nodeValue + "<br>";
        }
        document.getElementById("demo").innerHTML = txt;
    }
</script> </body>
</html>
```

Kết quả hiển thị trên trình duyệt là:

Everyday Italian
Harry Potter
XQuery Kick Start
Learning XML
Everyday Italian

BÀI TẬP CHƯƠNG 5

1. Sử dụng DOM, tạo một form nhập liệu đăng ký. Nếu không nhập gì vào form thì nó sẽ hiển thị “Chúng ta chưa nhập tên”, “Chúng ta chưa nhập email”,...và nếu nhập vào thì nó sẽ đổi sang chữ OK, không cho phép tải lại trình duyệt.

Hướng dẫn thực hiện:

- Tạo form.
- Viết Javascript truyền dữ liệu nhập vào form vào một biến. Có nhiều biến thì ta sẽ dùng mảng. Ở bước này ta sẽ dùng `document.getElementById("id").value` để truyền dữ liệu.
- Viết hàm Javascript kiểm tra dữ liệu nhập vào và xuất text ra vị trí mong muốn và trả dữ liệu về một ID nào đó. (Dùng element `.innerHTML` để truyền)
- Gán hàm ở bước 3 vào form với event `onkeyup` để nó thực hiện kiểm tra thi mỗi khi ta gõ một cái gì đó vào form.

2. Xây dựng GiaoVien.xml tập tin

Mã GV	Họ GV	Tên GV	Nhập Sinh	Giới Tính	Địa Chỉ	Mã MII
GV001	Nguyễn Tuấn Hưng		01/01/1976	Nam	An Giang	MH01
GV002	Trần Nhã	Linh	01/03/1979	Nữ	An Giang	MH02
GV003	Phạm Thế	Tùng	04/03/1976	Nam	Cà Mau	MH03
GV004	Lâm Thành	Quang	05/07/1980	Nam	Đồng Nai	MH04
GV005	Hoàng Ngọc	Lan	01/01/1984	Nữ	Bến Tre	MH05

Yêu cầu:

- Sử dụng mô hình DOM truy xuất các dữ liệu theo dạng như trên.
- In danh sách những giáo viên là nữ.

Hướng dẫn:

Câu a: Tạo tập tin XML: Mở asp.net → add new item → xml tập tin → đặt tên → add → soạn tập tin XML.

Truy xuất dữ liệu “DOM”: Sử dụng dataset đọc file XML

Gridview = dataset hoặc dataGrid = dataset

ví dụ: ĐỌC DỮ LIỆU XML TỪ DATASET

tạo trang aspx → design → kéo gridview → add → đặt tên header text và data Fidle thêm thư viện: using System.Xml;

protected void Page_Load(object sender, EventArgs e)

```
{    DataSet ds = new DataSet();
    ds.ReadXml(Server.MapPath("qlygiaoVien.xml"));
    GridView1.DataSource = ds;
    GridView1.DataBind(); }
```

→ khi chạy hiện hai cột giống nhau vào properties → autochannergata colum → false

Câu b:

In danh sách giáo viên là nữ.

Add new item → đặt tên →

```

using System.Xml;
public partial class giaoviennua : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        XmlDocument doc = new XmlDocument();
        doc.Load(Server.MapPath("qlygiaoVien.xml")); // trang xml mình muốn load lên.
        XmlNodeList list = doc.GetElementsByTagName("giaoVien"); // thẻ giáo viên
        string tb = "<Table>";
        foreach (XmlNode nut in list)
        {
            if (nut.ChildNodes[4].InnerText == "Nữ")// số 4 là cột thứ tư trong bảng
                mình thiết kế nhưng mình đếm từ 0
            {
                tb = tb + "<tr>";
                tb = tb + "<td>" + nut.ChildNodes[0].InnerText + "</td>";
                tb = tb + "<td>" + nut.ChildNodes[1].InnerText + "</td>";
                tb = tb + "<td>" + nut.ChildNodes[2].InnerText + "</td>";
                tb = tb + "<td>" + nut.ChildNodes[3].InnerText + "</td>";
                tb = tb + "<td>" + nut.ChildNodes[4].InnerText + "</td>";
                tb = tb + "<td>" + nut.ChildNodes[5].InnerText + "</td>";
                tb = tb + "</tr>"; }
            tb = tb + "</Table>";
        Label1.Text = tb; // bên trang aspx phải kéo một table thì mới có Label1
    }
}

```

3. Cho một đoạn code HTML sau:

```

<!DOCTYPE html>
<html> <head>
    <title>Tìm kiếm - freeTuts.net</title>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
    <style> #search_advance{display: none;} </style>
</head>
<body>
    <h1>Tìm kiếm dữ liệu</h1>
    <table border="1" cellspacing="0" cellpadding="5">
        <tr><td>Nhập tiêu đề</td>
            <td><input type="text" id="title" value="" /> <br/>
<a href="#" id="show_search_advance">Tìm kiếm nâng cao</a>
            </td> </tr>
            <tr id="search_advance">
                <td>Chọn chuyên mục</td>
                <td> <select id="category">
                    <option value="1">PHP</option>
                    <option value="2">Javascript</option>
                </select> </td> </tr>
                <tr> <td></td>
                    <td>
                        <input type="button" id="btn_search" value="Tìm kiếm" />
                    </td> </tr> </table>
    </body>
</html>

```

Khi chạy đoạn code này lên thì nội dung thẻ `select#category` sẽ không hiển thị là vì đây là một chương trình tìm kiếm và cho người dùng chọn thêm chức năng tìm kiếm nâng cao.

Yêu cầu bài tập như sau:

- Chúng ta hãy viết một đoạn mã Javascript xử lý khi người dùng click vào chữ "tìm kiếm nâng cao" thì sẽ **hiển thị category** và đồng thời đổi nội dung của nó lại thành "bỏ tìm kiếm nâng cao".
- Lúc này nếu người dùng click tiếp vào "bỏ tìm kiếm nâng cao" thì sẽ **ẩn category** và đồng thời chuyển nội dung của nó lại thành "tìm kiếm nâng cao".

Hướng dẫn giải:

- *Ta sẽ viết một hàm show_search_advance() và gán nó vào sự kiện onclick của thẻ a tìm kiếm nâng cao.*
- *Dựa vào nội dung của thẻ a để biết khi nào thì ẩn category và khi nào thì hiển thị category*

4. Tạo các trang tài liệu .html chứa mã Script đọc và phân tích tài liệu **KhachSan.xml (bài tập 7a, chương 4)** với các yêu cầu sau:

- Các phần tử có tag là Khachhang
- Các phần tử có tag là Khachhang đầu tiên
- Các phần tử có tag là Khachhang thứ hai
- Các phần tử có tag TenKH của khách hàng cuối cùng
- Thay đổi Tên của Khachhang đầu tiên là Hồ Khánh
- Cho biết cha của nút Phòng
- Trả về thuộc tính SoDt của các Khachhang theo các thứ tự đã cho
- Thay đổi thuộc tính SoDt thành Didong của nút Khachhang đầu tiên (Dùng SetAttribute và NodeValue)
- Xóa phần tử khách hàng đầu sử dụng RemoveChild và ParentNode
- Tạo chú thích cho node HoaDon

Chương 6

XPATH

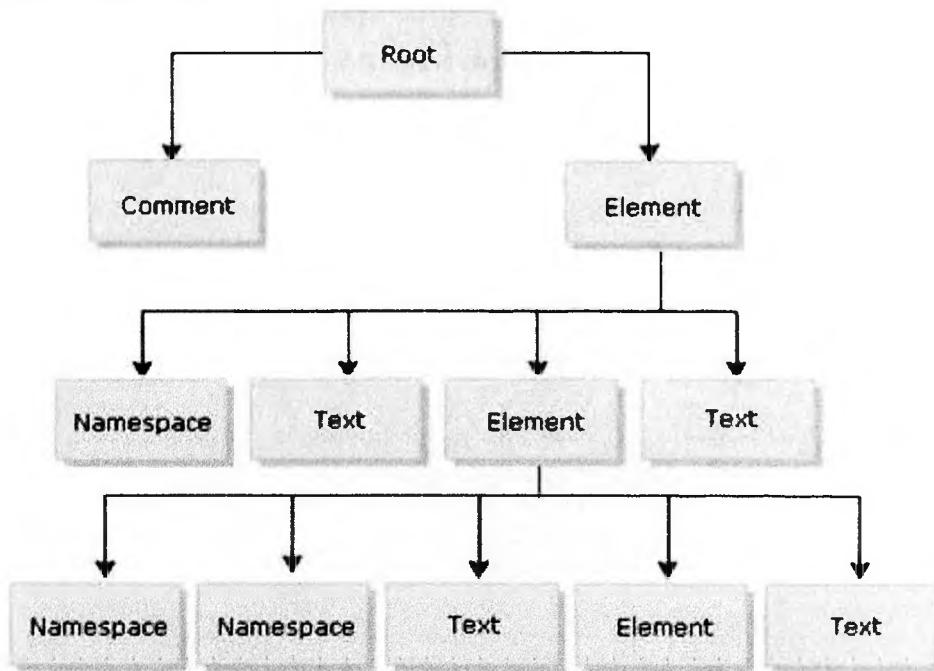
6.1. Giới thiệu XPath

XPath là viết tắt của XML Path Language. XPath là một thành phần hỗ trợ giúp truy xuất thông tin trong tập tin XML làm tiền đề cho việc áp dụng stylesheet kết hợp XML để tạo ra kết xuất tùy theo yêu cầu. XPath hỗ trợ nền tảng để tạo ra XQuery áp dụng trong truy vấn dữ liệu tương tự như truy vấn SQL trên cơ sở dữ liệu.

XPath chứa hơn 200 hàm được xây dựng. Có những hàm cho các giá trị chuỗi, giá trị số, các phép toán luận lý, so sánh thời gian, thao tác nút, thao tác chuỗi và nhiều hơn nữa. Hiện nay XPath cũng có thể được sử dụng trong JavaScript, Java, XML Schema, PHP, Python, C, C ++ và rất nhiều ngôn ngữ khác. Phiên bản hiện tại là XPath 3.0, được tổ chức W3C đưa ra vào ngày 08 tháng 4 năm 2014.

XPath là một trong ba thành phần trong ngôn ngữ XSL – **Extensible Style Language**. XPath là ngôn ngữ hỗ trợ tìm kiếm thông tin trong tài liệu XML thông qua việc sử dụng biểu thức XPath để định hướng tìm kiếm dữ liệu trên XML thay vì phải thực hiện tìm kiếm đệ qui để duyệt cây XML.

XPath định nghĩa 7 loại nodes theo mô hình thể hiện bên dưới (cấu trúc tài liệu XML) từ root, element, attribute, text, namespace, processing-instruction và comment:



Hình 6.1: Mô hình cấu trúc tài liệu XML

6.2. Các nút của XPath

Như trên đã giới thiệu, trong XPath có bảy loại nút, bao gồm: phần tử, thuộc tính, văn bản, không gian tên (namespace), chỉ thị xử lý, chú thích và nút tài liệu. Tài liệu XML được coi như là cây của các nút. Phần tử trên cùng của cây được gọi là phần tử gốc.

Ví dụ 6.1: Xét tài liệu **book.xml** sau đây:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book>
        <title lang="en">Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book>
        <title lang="en">Learning XML</title>
        <price>39.95</price>
    </book>
</bookstore>
```

Trong ví dụ trên, các nút như sau:

<bookstore> (**nút phần tử gốc**)
<author>J K. Rowling</author> (**nút phần tử**)
lang="en" (**nút thuộc tính**)

Giá trị nguyên tử: Giá trị nguyên tử là các nút không có con hoặc cha mẹ.
Ví dụ về các giá trị nguyên tử: *J K. Rowling, en*

Mối quan hệ của các nút: Nút cha: Mỗi phần tử và thuộc tính có một cha.

Ví dụ, *book* là cha của *title*, *author*, *year* và *price*. Mỗi nút phần tử có thể không có hoặc có nhiều nút con. Nút anh chị em là các nút có cùng nút cha. Ví dụ, các phần tử *title*, *author*, *year* và *price* là anh chị em với nhau. Nút tổ tiên (ancestor) bao gồm nút cha, nút cha của nút cha, ... Ví dụ, tổ tiên của nút *title* là phần tử *book* và *bookstore*. Nút hậu duệ (descendants) bao gồm nút con, con của nút con,... Ví dụ, hậu duệ của *bookstore* là *book*, *title*, *author*, *year* và *price*.

6.3. Cú pháp của XPath

Lựa chọn các nút

XPath sử dụng biểu thức đường dẫn để lựa chọn các nút hoặc bộ nút trong một tài liệu XML. Các nút được chọn bằng cách làm theo các bước sau.

- Để truy vấn với đường dẫn tuyệt đối nghĩa là đi từ root của tài liệu XML đến các thành phần cần truy cập, sử dụng dấu /
- Để truy vấn với đường dẫn tương đối để có thể truy cập đến thành phần bất kỳ thỏa điều kiện, XPath sử dụng với dấu //
- Để truy vấn đến một thành phần bất kỳ mà không cần biết tên của nó là gì, XPath sử dụng dấu *
- Để truy cập thuộc tính của một nút, XPath qui định cú pháp bắt đầu là @. Ví dụ @tênThuộcTính
- Để truy cập đến giá trị của một biến được định nghĩa, XPath qui định cú pháp bắt đầu là \$. Ví dụ \$tênBiến
- Điều kiện khi truy vấn được đặt trong dấu []

Chọn nút: XPath sử dụng biểu thức đường dẫn để lựa chọn các nút trong tài liệu XML. Các biểu thức đường dẫn thông dụng nhất được liệt kê dưới đây:

Bảng 6.1: Biểu thức đường dẫn thông dụng trong XPath

Biểu thức	Mô tả
<i>nodename</i>	Chọn tất cả các nút có tên là " <i>nodename</i> "
/	Chọn từ nút gốc
//	Chọn nút trong tài liệu từ nút hiện hành
.	Chọn nút hiện tại
..	Chọn nút cha của nút hiện hành
@	Chọn các thuộc tính

Bảng dưới liệt kê một số biểu thức đường dẫn và kết quả của biểu thức:

Bảng 6.2: Biểu thức đường dẫn và kết quả

Biểu thức đường dẫn	Kết quả
bookstore	Chọn tất cả các nút có tên là "bookstore"
/bookstore	Chọn nút phần tử gốc bookstore Lưu ý: Nếu đường dẫn bắt đầu với một dấu gạch chéo (/) nó luôn luôn đại diện cho một đường dẫn tuyệt đối.
bookstore/book	Chọn tất cả các phần tử book là con của bookstore
//book	Chọn tất cả các phần tử book không quan tâm chúng ở vị trí nào trong tài liệu
bookstore//book	Chọn tất cả phần tử book là hậu duệ của phần tử bookstore, không quan tâm phần tử book này nằm ở vị trí nào bên dưới phần tử bookstore
//@lang	Chọn tất cả các thuộc tính có tên là "lang"

Vị từ (Predicate): Vị từ được sử dụng để tìm thấy một nút cụ thể hoặc một nút có chứa một giá trị cụ thể. Vị từ luôn được đặt vào trong dấu ngoặc vuông. Bảng dưới đây liệt kê một số biểu thức đường dẫn với vị từ và kết quả của biểu thức:

Bảng 6.3: Đường dẫn có vị từ và kết quả

Biểu thức đường dẫn	Kết quả
/bookstore/book[1]	Chọn phần tử book đầu tiên của bookstore. Lưu ý: Trong IE 5-9 nút đầu tiên là [0], nhưng theo W3C, nút đầu tiên là [1]. Để giải quyết vấn đề này trong IE, thiết lập <i>SelectionLanguage</i> cho XPath: <i>In JavaScript</i> : <code>xml.setProperty("SelectionLanguage","XPath");</code>
/bookstore/book[last()]	Chọn phần tử book cuối cùng con của bookstore
/bookstore/book[last()-1]	Chọn phần tử book kế phần tử book cuối cùng con của bookstore
/bookstore/book[position()<3]	Chọn 2 phần tử book đầu tiên của bookstore

//title[@lang]	Chọn các title có một thuộc tính là <i>lang</i>
//title[@lang='en']	Chọn tất cả phần tử title có một thuộc tính "lang" với giá trị là "en"
/bookstore/book[price>35.00]	Chọn tất cả phần tử book của phần tử bookstore có phần tử price > 35.00
/bookstore/book[price>35.00]/title	Chọn tất cả phần tử title của phần tử book con bookstore có phần tử price > 35.00

Các phép toán được sử dụng trong XPath

- Đại số: +, -, * (nhân), div (chia thập phân), mod (chia lấy dư)
- So Sánh hay quan hệ: =, != (khác), <, <=, >, >=
- Luật lý: true, false, and, or, not
- Kết hợp: | (hội)

Các thành phần hỗ trợ truy vấn mối quan hệ giữa các nút trong tài liệu XML (Axes). Cú pháp: TênAxis::TênNode

Bảng 6.4: Các thành phần hỗ trợ truy vấn mối quan hệ giữa các nút trong XML

Axis	Định nghĩa
ancestor	Chọn tất cả các nút trên của nút hiện hành
ancestor-or-self	Chọn tất cả các nút trên của nút hiện hành và chính nó
attribute	Chọn tất cả các thuộc tính của nút hiện hành
child	Chọn nút con của nút hiện hành
descendant	Chọn tất cả các nút dưới của nút hiện hành
descendant-or-self	Chọn tất cả các nút dưới của nút hiện hành và chính nó
following	Chọn tất cả các nút sau thẻ đóng của nút hiện hành
following-sibling	Chọn các nút ngang cấp sau thẻ đóng của nút hiện hành
namespace	Chọn tất cả namespace của nút hiện hành
parent	Chọn tất cả nút cha của nút hiện hành
preceding	Chọn các thành phần trước thẻ mở của nút hiện hành
preceding-sibling	Chọn các nút ngang hàng trước nút hiện hành
self	Chọn nút hiện hành

Bảng 6.5: Các hàm trong XPath

Hàm	Định nghĩa
node-name(node)	Trả về tên nút của nút được đưa vào hàm
null(\$node)	Trả về true nếu nút là null
data(item, item, ...)	Lấy các trị tuân tự của các items đưa vào
base-uri()	Lấy trị thuộc tính base-uri của nút hiện hành
base-uri(\$node)	Lấy trị thuộc tính base-uri của nút
document-uri()	Lấy trị thuộc tính document-uri của tài liệu XML
number(arg)	Trả về kiểu số
abs(\$num)	Lấy trị tuyệt đối
ceiling(\$num)	Lấy trị số nguyên nhỏ nhất lớn hơn num
floor(\$num)	Lấy trị số nguyên lớn nhất nhỏ hơn num
string(arg)	Trả về kiểu chuỗi

compare(comp1, comp2)	So sánh giá trị cho comp1 với comp2, -1 là nhỏ hơn, 0 là bằng nhau và 1 là lớn hơn
concat(string, string, ...)	Nối chuỗi
substring(string, start [,len])	Lấy chuỗi con trong string đưa vào
string-length([string])	Lấy chiều dài chuỗi của nút hiện hành
normalize-space([string])	Loại bỏ khoảng trắng thừa trong và ngoài chuỗi string của nút hiện hành hay string
normalize-unicode	Tương tự như normalize-space nhưng dùng cho Unicode
translate(string1, string2, string3)	Convert chuỗi string1 bằng cách thay thế chuỗi string3 thay thế chuỗi string 2
contains(string1, string2)	Trả ra true nếu string1 chứa string2
match(string1, pattern)	Trả ra true nếu string1 tuân thủ đúng pattern
replace(string1, pattern, replace)	Thay thế chuỗi replace với thành phần trong string1 đúng pattern
tokenize(string, pattern)	Tương tự hàm split của ngôn ngữ java
boolean(arg)	Chuyển giá trị arg về kiểu luận lý true hay false
count([node])	Trả về số lượng nút của nút
last()	Trả về nút cuối cùng trong nút hiện hành
local-name([node])	Trả về tên nút nằm sau phần dấu :
name()	Trả về namespace qualified prefix:tênNút
namespace-uri([node])	Trả về uri của namespace của nút hiện hành
text()	Trả về chuỗi trị của nút
position()	Trả về vị trí của nút

Đánh giá các biểu thức XPath trên Windows

Một trong các công cụ tốt nhất để làm việc với XPath trên hệ điều hành Windows là trình soạn thảo Oxygen XML Editor¹. Sau khi cài đặt thành công, cửa sổ soạn thảo Oxygen XML Editor sẽ có thanh công cụ XPath 2.0, chúng ta chỉ cần gõ câu lệnh XPath vào ô trống bên phải thanh công cụ XPath 2.0 và Enter thì kết quả như Hình 6.2.

Đánh giá biểu thức XPath trên hệ điều hành Mac OS X

Các công cụ để làm việc với XPath trên hệ điều hành Mac OS X – mà không sử dụng lớp Java – ít có cải tiến và ít tinh tế hơn so với các công cụ làm việc trên hệ điều hành Windows. Hầu hết các công cụ thường được sử dụng là AquaPath², là phần mềm mã nguồn mở và tải miễn phí. Tải AquaPath dưới dạng ảnh của đĩa và đơn giản chỉ cần thực hiện di chuyển ứng dụng AquaPath từ hình ảnh thể hiện vào thư mục ứng dụng Web của chúng ta.

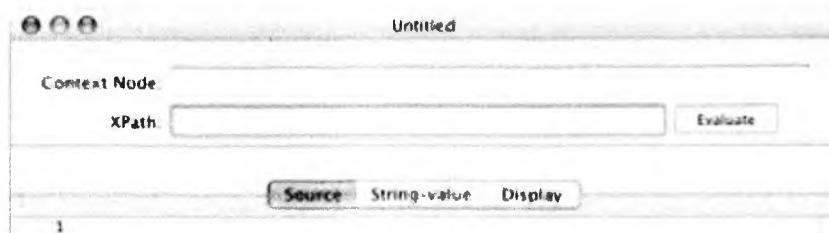
¹ <https://www.oxygenxml.com/>

² <http://aquapath-project.eu/>

Nháy đúp chuột vào ứng dụng AquaPath mới cài đặt, chúng ta sẽ nhìn thấy màn hình có dạng như Hình 6.3.



Hình 6.2: Giao diện của Oxygen XML Editor
khi truy vấn phần tử *book* đầu tiên trong tài liệu *book.xml*



Hình 6.3: Giao diện của phần mềm AquaPath

Giao diện ban đầu của AquaPath có thể trông khá đơn giản, tuy nhiên khi chúng ta bắt đầu tải vào tài liệu XML và gõ vào biểu thức XPath thì công cụ AquaPath này giống như công cụ truy vấn XPath của Oxygen XML Editor.

6.4. Các ví dụ minh họa XPath

6.4.1. XPath – Cú pháp cơ bản

- Đường dẫn tuyệt đối: Nếu đường dẫn XPath bắt đầu bằng / thì có nghĩa đây là một đường dẫn tuyệt đối bắt nguồn từ phần tử gốc.

Ví dụ 6.2:

Tài liệu XML	Câu lệnh XPath	Kết quả
<code><AAA></code>	<code>/AAA</code>	Phần tử được chọn là AAA
<code><BBB/> <CCC/></code>	<code>/AAA/BBB</code>	Chọn tất cả phần tử BBB là con của AAA
<code><DDD> <CCC/></code>		
<code></DDD></code>		
<code></AAA></code>	<code>/AAA/DDD/CCC</code>	Chọn tất cả phần tử CCC là con của /AAA/DDD

- Đường dẫn tổng thể: Sử dụng ký tự // để tham chiếu đến tất cả các phần tử trong tài liệu (ở độ sâu bất kỳ)

Ví dụ 6.3:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB> <CCC/> <DDD> <BBB/> </DDD> <CCC> <DDD> <BBB/> <BBB/> </DDD> </CCC> </AAA>	//BBB	Chọn tất cả phần tử BBB ở độ sâu bất kỳ
<AAA> <BBB> <CCC/> <BBB> <CCC/> </BBB> </BBB> </AAA>	//DDD/BBB	Chọn tất cả phần tử BBB trong tài liệu là con của DDD

- Chọn các phần tử bằng ký tự đại diện *: Sử dụng ký tự * để chọn tất cả các phần tử cùng thỏa mãn đường dẫn chỉ định.

Ví dụ 6.4:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB> <CCC/> <BBB> <CCC/> </BBB> </BBB> </AAA>	/AAA/*	Chọn tất cả phần tử là nút con trực tiếp của AAA
	/*/BBB	Chọn tất cả phần tử là các nút BBB ở cấp thứ 2 (nút con trực tiếp của nút gốc)
	//*	Chọn tất cả các phần tử trong tài liệu

- Chọn các phần tử theo vị trí bằng hoặc []: Sử dụng ngoặc vuông [] để chỉ định vị trí của một phần tử XML cần chọn.

Ví dụ 6.5:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB> <BBB/> <BBB> <CCC> content </CCC> </BBB> <BBB/> </AAA>	/AAA/BBB[1]	Chọn nút BBB thứ 1 (con của AAA)
	/AAA/BBB[last()]	Chọn nút BBB cuối cùng (con của AAA)
	/AAA/BBB[CCC='content']	Chọn nút BBB có nút con CCC = content

- Chọn thuộc tính của một phần tử trong tài liệu: Sử dụng ký hiệu @ để chọn hay chỉ đường dẫn đến một thuộc tính bên trong phần tử XML.

Ví dụ 6.6:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB id="b1"/> <BBB id="b2"/> <CCC name="ccc"/> <CCC/> </AAA>	/AAA/CCC[@name='ccc']	Chọn các nút CCC có giá trị thuộc tính name = ccc
	/AAA/CCC[@*]	Chọn những nút CCC có thuộc tính
	/AAA/CCC[not(@*)]	Chọn những nút CCC không có thuộc tính

	/AAA/BBB [normalize-space(@id)='b1']	Chọn BBB có thuộc tính id="b1" (không phân biệt khoảng trắng)
--	---	---

- Chọn phần tử trong tài liệu dựa trên số đếm: Dùng hàm count() để xác định tiêu chuẩn lựa chọn dựa trên tổng số phần tử có mặt trong tài liệu.

Ví dụ 6.7:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <DDD/> <DDD/> <CCC> <DDD/> <CCC/> </AAA>	//*[count(DDD)=2]	Chọn các phần tử có 2 thẻ con DDD
	//*[count(*)=1]	Chọn các phần tử có một thẻ con (tên bất kỳ)

- Chọn các phần tử dựa theo tên của phần tử: Sử dụng hàm name() để lấy tên phần tử. Hàm name() còn kết hợp với một số hàm như: starts-with() – các phần tử có tên bắt đầu với; contains() – tên phần tử có chứa ký tự được chỉ định; string-length() – chiều dài của tên phần tử.

Ví dụ 6.8:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB> <DDD/> <BBB/> <ECC> <EEE/> <ECC/> <CFB/> </AAA>	//*[name()='DDD']	Chọn các phần tử có tên là DDD
	//*[starts-with(name(), 'E')]	Chọn các phần tử có tên bắt đầu bằng chữ E
	//*[contains(name(), 'B')]	Chọn các phần tử có tên chứa B
	//*[string-length(name())=3]	Chọn những phần tử có chiều dài tên thẻ là 3
	//*[string-length(name())>3]	Chọn phần tử có chiều dài tên thẻ lớn hơn 3

- Chọn nhóm các phần tử dựa trên điều kiện logic: Chúng ta có thể kết hợp nhiều nhóm lựa chọn hay đường dẫn XPath với nhau thông qua phép hợp băng toán tử hoặc, ký hiệu |.

Ví dụ 6.9:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB/> <CCC/> <EEE/> </AAA>	//CCC //BBB	Chọn tất cả các phần tử CCC hoặc tất cả các phần tử BBB.
	//AAA/EEE //BBB	Chọn phần tử EEE là con của AAA và mọi phần tử BBB.

6.4.2. XPath_Axis

- Axis: Dùng để chọn các phần tử trong mối quan hệ với phần tử hiện thời.
- Chỉ định phần tử con trong đường dẫn: Sử dụng `child::` để chỉ định chọn phần tử chứa phần tử con.

Ví dụ 6.10:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB/> <CCC/> </AAA>	/AAA ↔ /child::AAA /child::AAA/child::BBB ↔ /AAA/BBB ↔ /child::AAA/BBB	Chọn tất cả các phần tử AAA Chọn phần tử BBB

- Chỉ định phần tử con cháu: Sử dụng `descendant::*` để chỉ định chọn phần tử cấp dưới (con cháu).

Ví dụ 6.11:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB> <DDD> <CCC> <DDD/> <EEE/> </CCC> </DDD> </BBB> <CCC> <DDD> <EEE/> </DDD> </CCC> </AAA>	/descendant::* /AAA/BBB/descendant::* //CCC/descendant::* //CCC/descendant::DDD	Chọn các phần tử là con của phần tử gốc Chọn các phần tử là con cháu của AAA/BBB Chọn mọi phần tử là con cháu của CCC Chọn các phần tử DDD là con cháu của CCC

- Chọn phần tử cha: Toán tử `parent::*` được dùng để chọn phần tử cha trên phần tử chỉ định một cấp.

Ví dụ 6.12:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB> <DDD/> <EEE/><CCC/> <DDD/> </BBB> <CCC> <DDD> <EEE/> </DDD> </CCC> </AAA>	//DDD/parent::*	Chọn các phần tử là cha của phần tử DDD.

- Chọn các phần tử cấp trên (cha ông): Toán tử `ancestor::*` được dùng để chọn các phần tử trên phần tử chỉ định một hoặc nhiều cấp (và do đó bao gồm luôn cả phần tử gốc).

Ví dụ 6.13:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB> <CCC/> </BBB> <CCC> <DDD> <EEE/> </DDD><FFF/> </EEE>	/AAA/BBB/CCC/ancestor::*	Chọn mọi phần tử nằm trên các phần tử thuộc về đường dẫn /AAA/BBB/CCC/

<code></DDD></CCC> </AAA></code>	<code>//FFF/ancestor::*</code>	Chọn các phần tử nằm trên phần tử FFF
--	--------------------------------	---------------------------------------

- Chọn các phần tử cùng cấp kế tiếp: Toán tử *following-sibling::* được dùng để yêu cầu chọn các phần tử cùng cấp sau phần tử chỉ định.

Ví dụ 6.14:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB> <CCC/> <DDD/> </BBB> <XXX> <DDD> <FFF> <GGG/></FFF> </DDD> </XXX> <CCC> <DDD/> </CCC> </AAA>	<code>/AAA/BBB/following-sibling::*</code>	Chọn các phần tử cùng cấp và tiếp theo sau /AAA/BBB/
	<code>//CCC/following-sibling::*</code>	Chọn mọi phần tử trong tài liệu cùng cấp và theo sau phần tử mang tên CCC

- Chọn các phần tử cùng cấp trước đó: Toán tử *preceding-sibling::* để chọn các phần tử cùng cấp trước phần tử chỉ định.

Ví dụ 6.15:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB> <CCC/> <DDD/> </BBB> <XXX> <DDD> <EEE/> <DDD/> <FFF/> <FFF> <GGG/> </FFF> </DDD> </XXX> <CCC> <DDD/> </CCC> </AAA>	<code>/AAA/XXX/preceding-sibling::*</code>	Chọn các phần tử cùng cấp nhưng đứng trước /AAA/XXX/
	<code>//CCC/preceding-sibling::*</code>	Chọn tất cả các phần tử cùng cấp và nằm trước phần tử CCC trong tài liệu

- Chọn tất cả các phần tử theo sau phần tử chỉ định: Toán tử *following::* để chọn tất cả các phần tử theo sau một phần tử nào đó.

Ví dụ 6.16:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB> <CCC/> </BBB> <XXX> <DDD> <CCC/> </DDD> </XXX> </AAA>	<code>/AAA/XXX/following::*</code>	Chọn tất cả các phần tử theo sau /AAA/XXX/
	<code>//ZZZ/following::*</code>	Chọn tất cả các phần tử theo sau phần tử ZZZ trong tài liệu

- Chọn tất cả các phần tử đứng trước phần tử chỉ định: Toán tử *preceding::* được dùng để chọn ngược lại với *following* đó là chọn tất cả các phần tử đứng trước phần tử chỉ định.

Ví dụ 6.17:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB> <CCC/> <ZZZ/> <FFF> <GGG/> <XXX> <DDD> <CCC/> </DDD> </XXX> </AAA>	/AAA/XXX/preceding::*	Chọn tất cả các phần tử đứng trước /AAA/XXX/

- Chọn phần tử cấp dưới và phần tử chỉ định: Toán tử *descendant-or-self::* được dùng để chọn các phần tử nằm dưới phần tử chỉ định, kể cả phần tử chỉ định cũng sẽ được chọn.

Ví dụ 6.18:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB> <CCC/> <ZZZ> <DDD/> <ZZZ/> </BBB> <XXX> <DDD> <EEE/> <DDD/> <CCC/> <FFF/> <FFF><GGG/> </FFF> </DDD> </XXX> </AAA>	/AAA/XXX/descendant-or-self::*	Chọn tất cả các phần tử nằm dưới phần tử /AAA/XXX/ kể cả phần tử /AAA/XXX/
	//CCC/descendant-or-self::*	Chọn tất cả các phần tử nằm dưới phần tử CCC và kể cả CCC

- Chọn phần tử cấp trên và phần tử chỉ định: Toán tử *ancestor-or-self::* để chọn các phần tử nằm trên phần tử chỉ định, kể cả phần tử chỉ định cũng sẽ được chọn.

Ví dụ 6.19:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA> <BBB/> <XXX><DDD> <EEE/> <DDD/> <CCC/> <FFF/> </DDD> </XXX> </AAA>	/AAA/XXX/DDD/EEE/ancestor-or-self::*	Chọn các phần tử nằm trên cấp của /AAA/XXX/DDD/EEE/ kể cả phần tử /AAA/XXX/DDD/EEE/ cũng sẽ được chọn.

- Chọn phần tử dựa trên biểu thức tính toán: Chúng ta có thể sử dụng hàm *position()* để tính toán vị trí phần tử được chọn hoặc dùng hàm *floor* để lấy giá trị nhỏ nhất gần với giá trị chỉ định; hàm *ceiling()* để lấy giá trị lớn nhất gần

với giá trị chỉ định; ngoài ra có thể kết hợp hàm *floor()* và *ceiling()* với hàm *last()*.

Ví dụ 6.20:

Tài liệu XML	Câu lệnh XPath	Kết quả
<AAA>	/BBB[position() mod 2=0]	Chọn tất cả các phần tử BBB có vị trí là số chẵn
<BBB/> <BBB/>	//BBB[position()=floor(last() div 2 +0.5) or position() = ceiling(last() div 2+ 0.5)]	Chọn tất cả các phần tử BBB có vị trí thỏa biểu thức đề ra
<BBB/> <BBB/>		
<BBB/> <CCC/>		
<CCC/>		
<CCC/>	/AAA/BBB[position()=2]	Chọn phần tử BBB nằm dưới /AAA và vị trí là 2
</AAA>		

Kết luận

XPath là một công cụ tốt khi chúng ta muốn tìm thông tin đặc biệt trong một tài liệu XML. Không có một công cụ nào có thể nhanh bằng và hữu dụng bằng XPath khi chúng ta muốn tìm dữ liệu liên quan đến các phần tử, thuộc tính hoặc dữ liệu nguyên bản.

XPath là lý tưởng khi chúng ta nắm vững cấu trúc tài liệu XML. Có sự khác nhau rất lớn về hiệu quả của XPath `/descendant-or-self::*[contains(.,'Fred')]` và XPath `//ACTOR[name[contains(.,'Fred')]]`. Khi thực hiện câu lệnh XPath đầu tiên chúng ta sẽ không thể đoán trước được tốc độ xử lý sẽ như thế nào. Còn XPath thứ hai chứng tỏ sự hiểu biết cẩn kẽ về cấu trúc của tài liệu nguồn và được xử lý một cách nhanh chóng.

BÀI TẬP CHƯƠNG 6

1. Biểu thức XPath/html/body//div [1] trả về kết quả gì?
2. Viết một biểu thức XPath để tìm tất cả các phần tử <div>
3. Sử dụng lại **QLDia.xml** (bài tập 8a, chương 4), tạo các câu lệnh XPath theo các điều kiện sau:
 - a. Đếm xem phần tử gốc có bao nhiêu nút con
 - b. Chọn các phần tử có tên CD, ARTIST, PRICEVN
 - c. Chọn các phần tử là con của CD
 - d. Chọn CD đầu tiên
 - e. Chọn CD tại vị trí thứ 2
 - f. Chọn CD cuối cùng
 - g. Chọn phần tử cha của CD
 - h. Chọn các Track có length <5 phút
 - i. Tìm các đĩa CD có tên bắt đầu là A
 - j. Tìm các đĩa CD có tên chứa chữ “Uớc Mơ”;
 - k. Tìm các đĩa CD tên là Xuân 1 và Xuân 2
 - l. Tìm các đĩa CD có tên không chứa chữ Xuân
 - m. Chọn các phần tử cấp trên của CD
 - n. Tìm các phần tử cùng cấp trước phần tử CD đầu tiên
 - o. Chọn các phần tử cùng cấp sau phần tử CD đầu tiên
 - p. Chọn các phần tử theo sau phần tử Message và các phần tử con của nó
 - q. Chọn các phần tử đứng trước phần tử Message và các con của nó
 - r. Chọn phần tử là tổ tiên của Title
 - s. Chọn tất cả các phần tử trước CD ngoại trừ phần tử gốc
 - t. Chọn tất cả các phần tử CD và con của nó
 - u. Chọn tất cả các phần tử Track và tổ tiên của nó
 - v. Chọn phần tử đi sau phần tử Message
 - w. Đếm xem có bao nhiêu phần tử track
 - x. Có bao nhiêu phần tử Track trong đĩa CD cuối cùng

Chương 7

ĐỊNH DẠNG TÀI LIỆU XML VỚI XSLT

XSLT là viết tắt của eXtensible Stylesheet Language Transformations là một ngôn ngữ dựa trên XML dùng để biến đổi các tài liệu XML. Trong chương này, chúng ta sẽ khám phá ra cách biến đổi tài liệu XML sang một định dạng XML khác hoặc định dạng HTML. Chúng ta sẽ thấy XSLT như một ngôn ngữ khai báo như thế nào? XSLT khác với đa số các ngôn ngữ lập trình phổ biến như Java và C # vì chúng là thủ tục. Sau đó chúng ta sẽ được giới thiệu về cú pháp của XSLT, mẫu (template) và học cách sử dụng chúng.

7.1. Giới thiệu XSLT

Mục đích chính của XSLT là được dùng để lấy một tài liệu XML hiện có và chuyển nó sang một định dạng khác. Định dạng mới có thể là XML, HTML hoặc chỉ văn bản thuần túy, chẳng hạn như tệp tin CSV (Comma Separated Values). XSLT được sử dụng trong hai tình huống phổ biến:

- Để chuyển đổi từ XML thành định dạng đặc trưng để hiển thị, chẳng hạn như HTML.
- Để chuyển đổi từ định dạng được hiểu bởi một ứng dụng vào cấu trúc yêu cầu của người khác. Điều này đặc biệt phổ biến khi trao đổi dữ liệu giữa các tổ chức khác nhau.

Với sự phát triển không ngừng, hiện tại XSLT có khả năng xử lý các tệp không phải là XML, vì vậy chúng ta có thể lấy một tệp văn bản thuần túy và chuyển đổi nó sang định dạng XML hoặc bất kỳ định dạng nào khác.

XSLT khác với nhiều ngôn ngữ lập trình chính như C # hoặc Java bởi hai lý do chính. Thứ nhất, XSLT là một ngôn ngữ khai báo (*declarative language*) và thứ hai, nó là một ngôn ngữ chức năng (*functional language*).

7.1.1. XSLT như là một ngôn ngữ khai báo

Hầu hết các ngôn ngữ lập trình chủ đạo được xem là thủ tục. Dữ liệu được đưa vào phần mềm, sau đó thao tác từng bước. Mỗi câu lệnh hoặc một đoạn lệnh có một nhiệm vụ được xác định rõ ràng là chịu trách nhiệm về những thay đổi nhỏ cho dữ liệu; Những thay đổi cá nhân này được kết hợp để tạo ra sự chuyển đổi dữ liệu tổng thể theo yêu cầu.

Ví dụ: Chúng ta có một bộ sưu tập của các đối tượng Tác giả, mỗi trong số đó có một thuộc tính FirstName và một thuộc tính LastName. Chúng ta được yêu cầu hiển thị tên đầy đủ của mỗi Tác giả trong bộ sưu tập. Bộ sưu tập Tác giả ban đầu là rỗng vì vậy Tác giả đầu tiên có chỉ số bằng không và tác giả cuối cùng có chỉ số là tổng số Tác giả trừ đi một.

Đoạn mã lệnh của chúng ta có thể sẽ giống như thế này:

```
int index;
for (index = 0; index < allAuthors.Count; index++)
{
    Author thisAuthor = allAuthors[index];
    Console.WriteLine(thisAuthor.FirstName + " " +
        thisAuthor.LastName);
}
```

Đây là một đoạn mã chuẩn. Chúng ta lặp lại tất cả các Tác giả bằng cách sử dụng một chỉ mục được tăng dần từ số không đến số đối tượng Tác giả trừ một. Tại mỗi lần lặp thông qua vòng lặp, chúng ta chỉ định Tác giả hiện tại cho một biến, *thisAuthor* và sau đó truy cập hai thuộc tính mà chúng ta quan tâm, FirstName và LastName. Sau đó chúng ta phải xác định tổng số đối tượng Tác giả sử dụng thuộc tính *Count* và theo dõi Tác giả chúng ta đang xử lý bằng một chỉ mục, *Index*. Một số ngôn ngữ khác cho phép chúng ta viết mã này ở mức độ khai báo cao hơn. Ví dụ, trong C# chúng ta có thể sử dụng câu trúc *foreach* như sau:

```
foreach (Author thisAuthor in allAuthors)
{
    Console.WriteLine(thisAuthor.FirstName + " " +
        thisAuthor.LastName); }
```

Đoạn mã ở trên có cấp độ khai báo cao hơn. Chúng ta không phải lo lắng về việc theo dõi các đối tượng Tác giả. Thay vào đó, chúng ta chỉ cần hỏi từng Tác giả và hiển thị các chi tiết của nó. Một ví dụ khác của lập trình khai báo là SQL, được sử dụng để truy vấn cơ sở dữ liệu quan hệ. Trong SQL, nếu chúng ta muốn xem tên của tất cả các tác giả trong một bảng thì chúng ta có thể sử dụng lệnh như thế này:

```
SELECT FirstName, LastName FROM Authors;
```

Trong câu lệnh trên, chúng ta không cần theo dõi các dòng trong bảng Tác giả. Chúng ta để cho công cụ truy vấn thực hiện các hoạt động cấp thấp.

XSLT lấy ý tưởng này để cho bộ xử lý xem xét các chi tiết cấp thấp. XSLT được thiết kế từ nền tảng như là một ngôn ngữ khai báo, vì vậy chúng ta không cần phải quan tâm đến việc làm thế nào để một cái gì đó được thực hiện. Thay vào đó, chúng ta tập trung vào việc mô tả những gì chúng ta muốn thực hiện. Ví dụ: nếu chúng ta muốn thực hiện một hoạt động tương tự để đưa ra tất cả các tên tác giả từ một tài liệu XML có chứa nhiều phần tử *<author>*, chẳng hạn như đoạn dữ liệu XML sau đây:

```
<authors>
    <author> <firstName>Danny</firstName>
        <lastName>Ayers</lastName> </author>
    <author> <firstName>Joe</firstName>
        <lastName>Fawcett</lastName> </author>
    <author> <firstName>William</firstName>
        <lastName>Shakespeare</lastName> </author>
</authors>
```

Chúng ta sẽ sử dụng một mẫu XSLT như sau:

```
<xsl:template match="author" />
    <xsl:value-of select="firstName" />
    <xsl:value-of select="lastName" />
</xsl:template>
```

Như đã thấy, chúng ta không phải khai báo một biến để theo dõi các phần tử *<author>* hoặc viết bất kỳ mã nào để lặp qua nó. Chúng ta chỉ cần cho bộ xử lý XSLT xuất giá trị của phần tử *<firstName>* và *<lastName>* bất cứ khi nào chúng ta gặp một phần tử *<author>*.

7.1.2. XSLT là một ngôn ngữ chức năng

Nếu chúng ta đã từng sử dụng các ngôn ngữ như Java, C++, C#, PHP, hoặc những ngôn ngữ khác thì chúng ta đã sử dụng ngôn ngữ lập trình bắt buộc. Bắt buộc theo nghĩa đen có nghĩa là chúng ta đặt máy tính chính xác những gì chúng ta muốn nó làm. Sử dụng lại ví dụ ở trên, nếu một tác giả thay đổi *lastName* của mình, thì mô hình tiêu chuẩn để diễn tả nó bằng mã sẽ là sử dụng một tham chiếu đến đối tượng Tác giả đó và sửa đổi thuộc tính *LastName*. Mã giả cho việc này sẽ như sau:

```
Author authorToDelete = getAuthor(12345); //Lấy ID của tác giả  
authorToDelete.LastName = "Marlowe"; //Thay đổi lastName
```

XSLT cũng thực hiện theo mô hình này. **Ưu điểm** chính của mô hình này là thứ tự thực hiện việc chuyển đổi hoàn toàn không liên quan, bộ xử lý tự tối ưu hóa việc thực hiện.

7.1.3. Quá trình thực hiện chương trình XSLT

Quá trình thực hiện chương trình XSLT bao gồm 3 bước: Bước 1: Chuẩn bị dữ liệu nguồn là tập tin XML; Bước 2: Soạn thảo chương trình XSLT; Bước 3: Cho thực hiện chương trình.

Bước 1: Dữ liệu nguồn được chuẩn bị thông qua một trong các cách sau:

- Cách 1: Sử dụng trình soạn thảo văn bản bất kỳ.
- Cách 2: Sử dụng trình soạn thảo XML Editor

Bước 2: Chương trình XSLT có thể được chuẩn bị thông qua một trong các cách sau:

- Cách 1: Sử dụng trình soạn thảo văn bản bất kỳ
- Cách 2: Sử dụng trình soạn thảo XML Editor hoặc Oxygen XML Editor
- Cách 3: Sử dụng trình soạn thảo chương trình XSLT (XSLT Editor)

Bước 3: Tùy theo mục tiêu của việc thực hiện có thể tiến hành 1 trong 3 cách sau:

Cách 3.1: Sử dụng môi trường lập trình: Cho thực hiện trực tiếp bên trong môi trường lập trình. Cách này thích hợp cho việc học tập và thử nghiệm chương trình XSLT.

Cách 3.2: Sử dụng trình duyệt Web: Cho thực hiện trực tiếp với sự hỗ trợ của trình duyệt Web. Cách này cho phép ứng dụng trực tiếp XSLT trong việc thể hiện thông tin trên Web.

Cách 3.3: Tự viết chương trình: Cho thực hiện thông qua việc viết một ứng dụng trong ngôn ngữ lập trình khác (ví dụ C#). Ứng dụng này sẽ:

- Nạp chương trình XSLT vào bộ nhớ
- Chuẩn bị dữ liệu nguồn (nếu cần thiết)
- Cho thực hiện
- Xử lý kết xuất được tạo ra (nếu cần thiết)

Cách này thích hợp khi cần "nhúng" chương trình XSLT vào một ứng dụng để có thể thực hiện nhanh, dễ bảo trì, chuẩn một số xử lý biến đổi nào đó liên quan tài liệu XML.

Chi tiết của việc thực hiện ba cách trong bước 3 như sau:

Cách 3.1: Sử dụng môi trường lập trình: Thực hiện chương trình XSLT với môi trường lập trình Visual Studio.NET được tiến hành qua ba bước:

- Bước 1: Tạo tập tin XML nguồn: Chọn Project - Add New Item với loại tập tin là XML, xuất hiện cửa sổ cho phép soạn thảo tập tin XML.
- Bước 2: Tạo chương trình XSLT: Chọn Project - Add New Item với loại tập tin XSLT, xuất hiện cửa sổ cho phép soạn thảo chương trình XSLT.
- Bước 3: Cho thực hiện:
 - o Bước 3.1: Chọn cửa sổ Properties để xác định tập tin XML nguồn và tập tin kết xuất.
 - o Bước 3.2: Quay về cửa sổ soạn thảo chương trình XSLT (Click và cửa sổ) và sau đó chọn chức năng XML, chọn Debug XSLT.

Lưu ý: Bước 3.1 chỉ cần thực hiện một lần nếu không thay đổi tập tin nguồn. Chúng ta có thể đánh dấu điểm ngắt bên trong chương trình XSLT tương tự khi Debug ứng dụng với ngôn ngữ lập trình khác.

Cách 3.2: Sử dụng trình duyệt Web thực hiện chương trình XSLT

Bước 1: Chèn vào tập tin XML nguồn chỉ thị yêu cầu thực hiện chương trình XSL. Lưu ý, chèn sau chỉ thị khai báo tài liệu XML.

```
<?xml-stylesheet type="text/xsl" href='đường dẫn đến
tập tin chương trình XSLT' ?>
```

Ví dụ:

```
<?xmlversion="1.0"encoding="utf-8" ?>
<?xml-stylesheettype="text/xsl" href='Hello.xslt' ?>
<authors>
```

Bước 2: Tạo chương trình XSLT: Chọn Project - Add New Item với loại tập tin XSLT, xuất hiện cửa sổ cho phép soạn thảo chương trình XSLT

Bước 3: Cho thực hiện: Mở trình duyệt Web và sau đó chọn URL là đường dẫn đến tập tin XML. Hoặc mở My Computer, nhấp chuột phải vào tập tin XML (đã chèn chỉ thị thực hiện XSL) chọn Open With, chọn Internet Explorer. Đối với trình soạn thảo XML Writer, chúng ta có thể mở tập tin XML và chọn F4 hoặc F5 để hiển thị kết quả. Lưu ý, tất cả các tập tin XML và XSLT đều phải được đặt trong cùng một thư mục.

Cách 3.3: Tự viết chương trình cho thực hiện XSLT

Bước 1: Tạo tập tin XML nguồn.

Bước 2: Tạo chương trình XSLT: Chọn Project - Add New Item với loại tập tin XSLT, xuất hiện cửa sổ cho phép soạn thảo chương trình XSLT.

Bước 3: Cho thực hiện

- Khai báo đối tượng Bo_thuc_hien
- Đọc tập tin chương trình *.XSL vào Bo_thuc_hien
- Yêu cầu Bo_thuc_hien thực hiện chương trình XSLT với dữ liệu nguồn và kết xuất

Với Visual Studio.NET 2005 VB.NET: Đoạn chương trình sau sẽ cho thực hiện chương trình Hello.xslt

- Dữ liệu nguồn là tập tin Nguoi_dung.xml
- Kết xuất là tập tin văn bản Loi_chao.txt

```
Imports System.Xml Imports System.Xml.Xsl Module
Thuc_hien_XSLT
Public Sub Main()
```

```

Dim Duong_dan_Xml As String = "...\\...\\Nguoi_dung.xml"
Dim Duong_dan_Xslt As String = "...\\...\\Hello.xslt"
Dim Duong_dan_Kq As String = "...\\...\\Loi_chao.txt"
Dim Thuc_hien As New XslCompiledTransform(True)
Thuc_hien.Load(Duong_dan_Xslt)
Thuc_hien.Transform(Duong_dan_Xml, Duong_dan_Kq)
End Sub
End Module

```

7.2. Các phần tử cơ bản của XSLT

XSLT dựa trên ý tưởng về các mẫu (*templates*). Ý tưởng cơ bản là chúng ta đưa ra một số mẫu sao cho khớp với XML trong tài liệu nguồn. Khi mẫu so khớp XML được tìm thấy, mẫu được kích hoạt và nội dung của nó được thêm vào tài liệu đầu ra.

Sử dụng các mẫu để xử lý các phần khác nhau của tài liệu nguồn là một trong những tính năng mạnh mẽ nhất của XSLT. Một số cấu trúc XSLT cơ bản sau đây cho phép chúng ta viết các phép biến đổi cơ bản:

- **<xsl: stylesheet>**: Đây là thành phần tài liệu bao gồm tất cả các mẫu của chúng ta. Chúng ta cũng sử dụng nó cho một số cấu hình, chẳng hạn như cài đặt phiên bản XSLT chúng ta muốn sử dụng.
- **<xsl: template>**: Đây là nền tảng của XSLT và có hai tính năng chính. Nó mô tả chi tiết các mục từ tài liệu nguồn cần xử lý và sử dụng nội dung của nó để chỉ định những gì nên được thêm vào đầu ra khi nó được thực hiện.
- **<xsl: apply-templates>**: Phần tử này chịu trách nhiệm quyết định xem mục nào trong tài liệu nguồn cần được xử lý; Sau đó chúng được xử lý bởi mẫu thích hợp.
- **<xsl: value-of>**: Phần tử này được sử dụng để đánh giá một biểu thức và thêm kết quả vào đầu ra. Ví dụ: chúng ta có thể đang xử lý một phần tử **<Person>** và sử dụng **<xsl: value-of>** để thêm nội dung của phần tử **<Name>** vào đầu ra.
- **<xsl: for-each>**: Thỉnh thoảng chúng ta cần xử lý một số mặt hàng theo cách tương tự nhưng sử dụng **<xsl: template>** không phải là một lựa chọn tốt. Trong trường hợp đó chúng ta có thể sử dụng phần tử **<xsl: for-each>** để gom nhóm các đối tượng và xử lý theo từng nhóm.

Trước khi chúng ta bắt đầu học về các mẫu này, chúng ta sẽ cần một tài liệu đầu vào XML cho các phép biến đổi của chúng ta. Ví dụ 7.1 sau đây là một tài liệu khá đơn giản về chi tiết một số chính trị gia nổi tiếng.

Ví dụ 7.1: People.xml

```

<?xml version="1.0"?>
<People>
    <Person bornDate="1874-11-30" diedDate="1965-01-24">
        <Name>Winston Churchill</Name>
        <Description> Winston Churchill was a mid-20th
                    Century British politician who became famous as
                    Prime Minister during the Second World War.
        </Description>
    </Person>

```

```

<Person bornDate="1917-11-19" diedDate="1984-10-31">
    <Name>Indira Gandhi</Name>
    <Description> Indira Gandhi was India's first female
        prime minister and was assassinated in 1984.
    </Description>
</Person>
<Person bornDate="1917-05-29" diedDate="1963-11-22">
    <Name>John F. Kennedy</Name>
    <Description> JFK, as he was affectionately known,
        Was a United States president who was
        assassinated in Dallas, Texas.
    </Description>
</Person>
</People>

```

XSLT đầu tiên sau đây tập trung vào một trường hợp sử dụng phổ biến: chuyển đổi XML thành một trang HTML.

7.2.1. Phần tử *xsl:stylesheet*

Ví dụ 7.2 cho thấy phần tử *xsl:stylesheet* được sử dụng bởi tất cả các biến đổi XSL:

Ví dụ 7.2: Shell.xslt

```

<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
    <!--các phần tiếp theo của XSLT được đặt ở đây -->
</xsl:stylesheet>

```

Khai báo trên cho thấy rằng các phần tử XSLT nằm trong không gian tên <http://www.w3.org/1999/XSL/Transform>. Số phiên bản được khai báo là 1.0. Mặc dù Saxon là một bộ xử lý phiên bản 2.0 nó sẽ chạy các tập tin được đánh dấu là phiên bản 1.0 trong chế độ tương thích ngược. URI không gian tên này không thay đổi giữa hai phiên bản, do đó để thay đổi phiên bản chúng ta chỉ cần thay đổi thuộc tính này.

Mặc dù Ví dụ 7.2 là đúng cú pháp, nhưng nó thực sự không làm bất cứ điều gì hữu ích. Để thực sự tạo một đầu ra mới, chúng ta cần làm hai việc: chúng ta phải chọn một số phần tử hoặc các thuộc tính để xử lý, đồng thời chúng ta cần phải mô tả yêu cầu cho đầu ra dựa trên các mục này. Thành phần được sử dụng để mô tả kết quả đầu ra tạo ra là *<xsl:template>*

7.2.2. Phần tử *xsl:template*

Phần tử *<xsl:template>* là nền tảng của toàn bộ công nghệ XSL. Vì vậy việc nắm vững cơ chế hoạt động của phần tử này là chìa khóa cho toàn bộ quá trình. Nếu chúng ta thêm một phần tử *<xsl:template>* vào ví dụ chuyển đổi chúng ta sẽ có được Ví dụ 7.3.

Ví dụ 7.3: PeopleToHtml-Basic.xslt

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/"> <!-- basic output here -->
    </xsl:template>
</xsl:stylesheet>

```

Chỉ thị này yêu cầu bộ vi xử lý: Thực hiện mã trong mẫu này bắt cứ khi nào nó gặp một mục phù hợp với những gì được chỉ định. Ký hiệu / là chỉ nút gốc trong tài liệu XML.

Việc thêm đầu ra vào một thành phần `<xsl: template>` rất dễ dàng. Bất cứ thứ gì xuất hiện giữa thẻ bắt đầu và thẻ kết thúc sẽ được gửi đến cây kết quả, thuật ngữ kỹ thuật cho đầu ra từ một biến đổi. Chúng ta sẽ bắt đầu bằng cách sử dụng mẫu để tạo một vỏ HTML như trong Ví dụ 7.4:

Ví dụ 7.4: PeopleToHtml-BasicStructure.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
        <head> <title>Famous People</title> </head>
        <body> <h1>Famous People</h1><hr/> </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

Bây giờ chúng ta đã thêm một số phần tử HTML cơ bản vào mẫu. Lưu ý rằng vì một tài liệu XSLT là XML, nội dung phải hợp khuôn dạng XML. Điều này có nghĩa là chúng ta phải sử dụng các cấu trúc như `<hr />` thay vì chỉ `<hr>`. Bộ vi xử lý XSLT có các quy tắc đặc biệt về HTML được nhúng vào chúng, vì vậy chúng sẽ tự động xuất ra các phần tử này một cách chính xác nếu họ có thể nhận ra rằng chúng ta đang tạo một tệp tin HTML.

Tiếp theo, chúng ta cần được hướng dẫn biến đổi để thực sự xử lý các phần tử này và vì thế chúng ta sẽ cần một loại phần tử mới.

7.2.3. Phần tử `<xsl: apply-templates>`

Phần tử `<xsl: apply-templates>` sử dụng một thuộc tính `select` để chọn các nút nào cần xử lý. Bộ vi xử lý sau đó tìm kiếm XSLT cho một phần tử `<xsl: template>` có thuộc tính đối sánh phù hợp với các nút đó. Để buộc công cụ XSLT xử lý nút `<Person>`, cần thêm phần tử `<xsl: apply-templates>` vào mã chuyển đổi và đặt nó bên trong thẻ danh sách không có thứ tự HTML (``), như được hiển thị trong Ví dụ 7.5.

Ví dụ 7.5: PeopleToHtml-ProcessPerson.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html> <head> <title>Famous People</title> </head>
    <body> <h1>Famous People</h1> <hr/>
        <ul><xsl:apply-templates select="People/Person"/>
    </ul> </body> </html>
</xsl:template>
<xsl:template match="Person">
    <li><!-- Person details here --></li>
</xsl:template>
</xsl:stylesheet>
```

Trong Ví dụ 7.5, chúng ta đã sử dụng XPath là People/Person, để chọn các nút mà chúng ta muốn để hiển thị. Thao tác này sẽ chọn ba phần tử <Person> và chuyển cho mẫu phù hợp với chúng.

Cuối cùng, chúng ta cần trích xuất một số dữ liệu. Có nhiều cách để trích xuất thông tin từ các nút trong một tài liệu XML, nếu nội dung cần trích xuất chỉ là văn bản thông thường thì sự lựa chọn tối ưu là sử dụng <xsl: value-of>.

7.2.4. Phần tử <xsl: value-of>

Phần tử <xsl: value-of> sử dụng rất đơn giản. Nó có một thuộc tính, được đặt tên là *select*, nó đưa XPath đến nút mà chúng ta cần. Nếu chúng ta chỉ định một phần tử đích chúng ta sẽ nhận được tất cả các văn bản bên trong phần tử đó. Nếu chúng ta chỉ định một thuộc tính chúng ta sẽ nhận được giá trị của thuộc tính dưới dạng chuỗi. Xét lại Ví dụ 7.5, yêu cầu trích xuất thông tin bên trong phần tử <Person>, do đó, chúng ta chỉ cần thêm *Name* như trong Ví dụ 7.6.

Ví dụ 7.6: PeopleToHtml-PersonName.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html><head><title>Famous People</title> </head>
        <body> <h1>Famous People</h1> <hr/> <ul>
            <xsl:apply-templates select="People/Person"/>
        </ul> </body>
    </html> </xsl:template>
    <xsl:template match="Person">
        <li> <xsl:value-of select="Name"/> </li>
    </xsl:template>
</xsl:stylesheet>
```

Khi gán tập tin XSLT của Ví dụ 7.6 vào tập tin People.xml, kết quả trên trình duyệt như sau:

Famous People

-
- Winston Churchill
 - Indira Gandhi
 - John F. Kennedy

Hình 7.1: Kết quả khi thực hiện gán tập tin XSLT
của Ví dụ 7.6 vào tập tin People.xml

Bây giờ chúng ta đã thực hiện chuyển đổi hoàn chỉnh bằng cách sử dụng sự kết hợp giữa <xsl: apply-templates> để chỉ định các nút được xử lý với <xsl: template> để xử lý chúng. Phương pháp này thường được gọi là xử lý đầy, bởi bộ xử lý đi qua XML nguồn và đầy các nút được lựa chọn bởi <xsl: apply-templates> vào <xsl: template> có liên quan. Thông thường, để thuận tiện hơn người ta thường sử dụng phương pháp xử lý đầy bằng cách lấy các

nút trực tiếp và sử dụng các nội dung của chúng. Đối với loại xử lý này, chúng ta cần phần tử `<xsl: for-each>`.

7.2.5. Phần tử `<xsl: for-each>`

Phần tử `<xsl: for-each>` cho phép chúng ta chọn một nhóm các nút và áp dụng một thao tác cho mỗi nút. Nó hoạt động khác với vòng lặp `for` trong các ngôn ngữ khác, nơi mà vòng lặp `for` thường được sử dụng để lặp qua một mảng hoặc bộ sưu tập. Như đã đề cập trước đó, XSLT là một ngôn ngữ chức năng và trong quá trình xử lý, không có sự bảo đảm về thứ tự xử lý trong nhóm các nút được chọn. Tương tự, chúng ta có thể thoát khỏi vòng lặp bằng cách sử dụng câu lệnh `break`.

Ví dụ 7.7 cho thấy XSLT sẽ thực hiện như thế nào nếu chúng ta thay thế `<xsl: apply-templates>` bằng một lệnh `<xsl: for-each>`.

Ví dụ 7.7: `PeopleToHtml-ForEach.xslt`

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html> <head> <title>Famous People</title> </head>
        <body> <h1>Famous People</h1> <hr />
            <ul> <xsl:for-each select="People/Person">
                <li> <xsl:value-of select="Name" />
            </li> </xsl:for-each>
        </ul> </body> </html>
    </xsl:template>
</xsl:stylesheet>
```

Phần tử `<xsl: for-each>` có một thuộc tính `select` trỏ đến các nút chúng ta muốn xử lý. Đối với mỗi nút trong nhóm, nội dung của lệnh `<xsl: for-each>` được thực thi. Do đó, thuộc tính `select` sử dụng XPath là `People/Person` như ví dụ trước và đối với mỗi phần tử `<Person>` một danh sách sẽ được tạo ra. Kết quả của XSLT này giống với Hình 7.1.

Tóm lại, bây giờ chúng ta đã có hai cách để xử lý các nút: Xử lý đầy các nút bằng cách sử dụng `<xsl: template>` hoặc xử lý kéo với `<xsl: for-each>`. Cái nào tốt nhất? Mặc dù không có quy tắc nào là tốt nhất, tuy nhiên chúng ta nên bắt đầu bằng cách sử dụng `<xsl: template>`. Phần tử `<xsl: template>` thường linh động hơn và dễ bảo trì hơn `<xsl: for-each>`. Phần tử `<xsl: for-each>` chỉ nên sử dụng khi muốn thực hiện các mã nhanh chóng và khi chúng ta không quan tâm đến đoạn mã đó trong tương lai.

Vai trò của XPath trong XSLT

Ngoài thuộc tính `select`, có rất nhiều hướng dẫn XSLT sử dụng XPath và phần này xem xét các hướng dẫn thay thế này. Chúng ta có thể bắt đầu bằng cách mở rộng ví dụ trong Ví dụ 7.7 theo hai cách: đầu tiên chúng ta sẽ làm cho trang đầu ra thú vị hơn bằng cách sử dụng một bảng HTML và hiển thị ngày sinh và ngày mất của những người nổi tiếng. Chúng ta sẽ sử dụng phần tử `<xsl: apply-templates>` và cơ chế xử lý đầy. Điều này có nghĩa là chúng ta chỉ cần sửa đổi các mẫu chính để bảng HTML cơ bản được tạo ra và sau đó thay đổi mẫu phù hợp với các phần tử `Person`.

Ví dụ 7.8: PeopleToHtml-WithTable.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html> <head> <title>Famous People</title>
      </head> <body> <h1>Famous People</h1> <hr/>
        <table> <caption>Famous People</caption>
          <thead>
            <tr> <th>Name</th> <th>Born</th>
              <th>Died</th> <th>Description</th>
            </tr> </thead> <tbody>
              <xsl:apply-templates select="People/Person"/>
            </tbody> </table>
          </body> </html> </xsl:template>
  <xsl:template match="Person">
    <tr>
      <td> <xsl:value-of select="Name" /> </td>
      <td> <xsl:value-of select="@bornDate" /> </td>
      <td> <xsl:value-of select="@diedDate" /> </td>
      <td> <xsl:value-of select="Description" /> </td>
    </tr> </xsl:template>
  </xsl:stylesheet>
```

Kết quả của việc chuyển đổi sang HTML được thể hiện ở Hình 7.2.

Famous People

Famous People			
Name	Born	Died	Description
Winston Churchill	1874-11-30	1965-01-24	Winston Churchill was a mid 20th century British politician who became famous as Prime Minister during the Second World War.
Indira Gandhi	1917-11-19	1984-10-31	Indira Gandhi was India's first female prime minister and was assassinated in 1984.
John F. Kennedy	1917-05-29	1963-11-22	JFK, as he was affectionately known, was a United States president who was assassinated in Dallas, Texas.

Hình 7.2: Kết quả khi thực hiện gán tập tin XSLT
của Ví dụ 6.7 vào tập tin People.xml

Chúng ta có thể thấy rằng định dạng kết quả hiển thị các giá trị ngày tháng không thân thiện. Kết quả vẫn hiển thị theo định dạng ngày tháng của XML. Nếu chúng ta muốn thay đổi định dạng hiển thị thì chúng ta cần phải xử lý các giá trị đó trước khi cho hiển thị. Chúng ta cũng có thể loại bỏ sự mơ hồ về giá trị biểu thị tháng bằng cách sử dụng tên của tháng thay vì mã hai chữ số. Điều này sẽ cung cấp cho chúng ta một cơ hội để sử dụng một mẫu có tên và cũng chỉ ra cách sử dụng các hàm XPath để thao tác một giá trị chuỗi. Cách xử lý được thể hiện trong đoạn mã dưới đây:

```
<xsl:template name="iso8601DateToDisplayDate">
  <xsl:param name="iso8601Date" />
  <xsl:variable name="yearPart"
    select="substring($iso8601Date, 1, 4)" />
  <xsl:variable name="monthPart"
    select="substring($iso8601Date, 6, 2)" />
  <xsl:variable name="datePart"
```

```

        select="substring($iso8601Date, 9, 2)" />
    <xsl:value-of select="concat($datePart, '/',
        $monthPart, '/', $yearPart)" />
</xsl:template>
```

Đoạn mã trên cho phép chúng ta chiết xuất từng thành phần của năm, tháng và ngày. Chúng ta trích xuất các phần khác nhau của ngày, tháng và năm và đặt chúng vào ba biến có tên *datePart*, *monthPart* và *yearPart*. Để làm điều này, chúng ta sử dụng hàm *substring* của XPath. Hàm này có ba tham số, trong đó tham số thứ ba là tùy chọn: (1) Chuỗi để vận hành; (2) Vị trí bắt đầu để xử lý; (3) Chiều dài của chuỗi kết quả.

Nếu tham số thứ ba bị bỏ qua, toàn bộ chuỗi, bắt đầu từ ký tự trong tham số thứ hai, sẽ được trả về. Vì vậy, để truy cập phần tháng, *monthPart*, chúng ta thao tác trên giá trị ngày đầy đủ và bắt đầu từ ký tự thứ 6. Sau đó chúng ta lặp lại thao tác này với *datePart* bằng cách lấy hai ký tự, bắt đầu từ ký tự thứ 9. Khi chúng ta đã có ba phần riêng biệt, chúng ta sử dụng một hàm XPath khác, *concat()*, để nối chúng lại với nhau bằng dấu phân cách thích hợp.

Phần tử *<xsl: variable>* được sử dụng để khai báo biến cục bộ hoặc toàn cục. *<xsl: variable>* cho phép trích rút thông tin từ tập tin XML và đưa vào một biến (đúng ra là hằng vì nội dung biến này không thể thay đổi được). Ngoài cách sử dụng phần tử *<xsl: variable>*, chúng ta có thể hai cách khác để khai báo biến: (1) sử dụng thuộc tính *select*; (2) sử dụng nội dung của phần tử. Phương pháp thứ hai này sẽ như sau:

```

<xsl:variable name="myVariable">
    <myElement>Some content</myElement>
</xsl:variable>
```

Nói chung, nếu có thể chúng ta nên sử dụng thuộc tính *select* để chỉ định những gì chúng ta muốn trong biến. Cách thứ hai (sử dụng nội dung phần tử) có thể dẫn đến các trở ngại bởi vì một cây mới phải được xây dựng và một nút bên ngoài được thêm vào. Điều này có thể dẫn đến các vấn đề khi sử dụng biến. Một khi chúng ta đã thiết lập nội dung của một biến, chúng ta có thể truy cập nó bằng cách sử dụng tên của biến trước bởi dấu \$.

Mặt khác, chúng ta nên chú ý phạm vi của biến. Nếu chúng ta khai báo một biến như là một phần tử cấp trên, một con trực tiếp của *<xsl: stylesheet>*, thì nó có thể được sử dụng ở bất cứ đâu trong tài liệu. Nếu chúng ta tạo nó trong một *<xsl: template>*, nó chỉ có thể được sử dụng ở đó, và nó chỉ có thể được sử dụng trong nội bộ nơi mà nó được tạo ra.

Ví dụ: Đoạn mã sau đây chứa hai lần sử dụng biến *\$demo*. Biến *\$demo* xuất hiện lần đầu thì không có vấn đề gì vì nó nằm bên trong phần tử cha, *<xsl:for-each>* và được sử dụng trong phần tử đó. Biến *\$demo* thứ hai (sau thẻ đóng *</xsl:for-each>*) sẽ tạo ra lỗi vì bộ xử lý phải cố thực hiện truy cập lại biến *\$demo* bên ngoài cha mẹ của nó (phần tử *<xsl:for-each>*).

```

<xsl:template name="usingVariables">
    <xsl:for-each select="someElements/someElement">
        <xsl:variable name="demo" select="'Some text'"/>
        <!-- this next line is okay as $demo is in scope-->
        <xsl:value-of select="concat(someElement, $demo)" />
    </xsl:for-each>
```

```

<!-- next line is an error as $demo is out of scope-->
    <xsl:value-of select="$demo" />
</xsl:template>

```

Để sử dụng đoạn mã này chúng ta cần phải sửa đổi lệnh bằng cách thêm lệnh tạo bảng và tận dụng chức năng của phần tử `<xsl: call-template>`. Kết quả hoàn chỉnh được trình bày trong Ví dụ 7.9.

Ví dụ 7.9: *PeopleToHtml-FriendlyDate.xslt*

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html> <head> <title>Famous People</title> </head>
    <body> <h1>Famous People</h1> <hr/>
      <table> <caption>Famous People</caption>
        <thead> <tr> <th>Name</th>
          <th>Born</th> <th>Died</th>
          <th>Description</th> </tr> </thead>
        <tbody>
          <xsl:apply-templates select="People/Person" />
        </tbody> </table> </body> </html>
    </xsl:template>
    <xsl:template match="Person">
      <tr> <td> <xsl:value-of select="Name" /> </td>
      <td>
        <xsl:call-template name="iso8601DateToDisplayDate">
          <xsl:with-param name="iso8601Date"
            select="@bornDate" />
        </xsl:call-template> </td> <td>
        <xsl:call-template name="iso8601DateToDisplayDate">
          <xsl:with-param name="iso8601Date"
            select="@diedDate" />
        </xsl:call-template> </td>
      <td> <xsl:value-of select="Description"/>
      </td> </tr> </xsl:template>
    <xsl:template name="iso8601DateToDisplayDate">
      <xsl:param name="iso8601Date" />
      <xsl:variable name="yearPart"
        select="substring($iso8601Date, 1, 4)" />
      <xsl:variable name="monthPart"
        select="substring($iso8601Date, 6, 2)" />
      <xsl:variable name="datePart"
        select="substring($iso8601Date, 9, 2)" />
      <xsl:value-of select="concat($datePart, '/',
        $monthPart, '/', $yearPart)" />
    </xsl:template>
  </xsl:stylesheet>

```

7.2.6. Phần tử `<xsl: call-template>`

Phần tử `<xsl: call-template>` có một thuộc tính, `name`, dùng để xác định mẫu nào được gọi. Bên trong phần tử `<xsl: call-template>` có thể có nhiều phần tử `<xsl: with-param>` với giá trị thỏa mãn `<xsl: template>`. Các phần tử

`<xsl:with-param>` có một thuộc tính `select` để lấy ra bất cứ giá trị nào được yêu cầu. Kết quả của sự chuyển đổi mới này được thể hiện trong Hình 7.3.

Famous People

Famous People			
Name	Born	Died	Description
Winston Churchill	30/11/1874	24/01/1965	Winston Churchill was a mid 20th century British politician who became famous as Prime Minister during the Second World War
Indira Gandhi	19/11/1917	31/10/1984	Indira Gandhi was India's first female prime minister and was assassinated in 1984
John F. Kennedy	29/05/1917	22/11/1963	JFK, as he was affectionately known, was a United States president who was assassinated in Dallas, Texas

Hình 7.3: Kết quả khi thực hiện gán tập tin XSLT
của Ví dụ 7.9 vào People.xml

Xét định dạng hiển thị ngày tháng ở Hình 7.3, mặc dù đã rõ ràng phân biệt ngày/tháng/năm, tuy nhiên cách hiển thị này dễ gây nhầm lẫn hiểu ngày thành tháng và ngược lại. Bởi vì định dạng ở Hình 7.3 tuân theo tiêu chuẩn Châu Âu là ngày/tháng/năm, trong khi ở Hoa Kỳ mặc định tháng/ngày/năm. Để loại bỏ sự mơ hồ này, chúng ta có thể sửa đổi mẫu để hiển thị tháng theo tên chứ không phải là con số. XSLT cho phép thực hiện điều này bằng cách nhúng và lấy thông tin tra cứu từ một nguồn bên ngoài thông qua việc sử dụng hàm `document()`.

Hàm `document()` là một trong những hàm hữu ích nhất trong thư viện XSLT. Hàm này khá đơn giản vì chỉ có một tham số, là một chuỗi chỉ đường link đến một tài liệu bên ngoài, thường là dưới dạng một URL. Nếu chúng ta có một phép biến đổi XSL để xử lý tệp XML nhưng chúng ta cũng muốn kết hợp thông tin từ tài liệu được tổ chức tại <http://www.wrox.com/books.xml>, chúng ta nên sử dụng mã như sau:

```
<xsl:variable name="books" select=
    "document('http://www.wrox.com/books.xml')"/>
```

Giả sử URL <http://www.wrox.com/books.xml> trả đến một tài liệu XML hợp khuôn dạng, biến `$books` sẽ giữ một tham chiếu đến nút gốc của tài liệu và các nút khác có thể được truy cập theo cách thông thường bằng XPath. Ví dụ: mỗi sách có thể được tìm thấy bằng cách sử dụng biểu thức:

```
$books/Books/Book
```

Bây giờ chúng ta sẽ thấy hàm `document()` làm thế nào để thực hiện nhiệm vụ biến tháng được thể hiện dưới dạng một số thành thể hiện dưới dạng ký tự chữ cái.

- Đầu tiên, chúng ta cần xây dựng một bảng tra cứu, đó là một tập tin XML cho phép chúng ta ánh xạ số tháng với tên của nó như được hiển thị trong Ví dụ 7.10.

Ví dụ 7.10: Months.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Months>
    <Month index="1">January</Month>
    <Month index="2">February</Month>
    <Month index="3">March</Month>
    <Month index="4">April</Month>
    <Month index="5">May</Month>
```

```

<Month index="6">June</Month>
<Month index="7">July</Month>
<Month index="8">August</Month>
<Month index="9">September</Month>
<Month index="10">October</Month>
<Month index="11">November</Month>
<Month index="12">December</Month>
</Months>

```

2. Bước thứ hai là sử dụng hàm *document()* để truy cập tập tin *Months.xml*. Sử dụng một biến *\$allMonths* để giữ kết quả như sau:

```
<xsl:variable name="allMonths" select="document('months.xml')"/>
```

3. Biến *\$allMonths* là một phần tử cấp cao, là một nút con trực tiếp của *<xsl:stylesheet>*. Tiếp theo chúng ta thay đổi câu lệnh thao tác trên giá trị kiểu ngày tháng để lấy giá trị văn bản của *month* bằng cách sử dụng biến *\$monthPart*:

```

<xsl:template name="iso8601DateToDisplayDate">
  <xsl:param name="iso8601Date"/>
  <xsl:variable name="yearPart"
    select="substring($iso8601Date, 1, 4)"/>
  <xsl:variable name="monthPart"
    select="substring($iso8601Date, 6, 2)"/>
  <xsl:variable name="monthName" select=
    "$allMonths/Month[@index= number($monthPart)]"/>
  <xsl:variable name="datePart" select=
    "substring($iso8601Date, 9, 2)"/>
  <xsl:value-of select="concat($datePart, ' ', $monthName,
    ' ', $yearPart)"/>
</xsl:template>

```

4. Tiếp theo, loại bỏ dấu gạch chéo trong câu cuối *<xsl: value-of>* và thay thế bằng một khoảng trắng. XSLT mới như trong Ví dụ 7.11.

Ví dụ 7.11: PeopleToHtml-MonthNames.xml

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="allMonths"
    select="document('months.xml')"/>
  <xsl:template match="/">
    <html> <head> <title>Famous People</title> </head>
    <body> <h1>Famous People</h1> <hr /> <table>
      <caption>Famous People</caption>
      <thead> <tr> <th>Name</th> <th>Born</th>
        <th>Died</th> <th>Description</th>
      </tr> </thead> <tbody>
        <xsl:apply-templates select="People/Person" />
      </tbody> </table> </body> </html> </xsl:template>
  <xsl:template match="Person">
    <tr> <td> <xsl:value-of select="Name" /> </td>
      <td>
        <xsl:call-template name="iso8601DateToDisplayDate">

```

```

<xsl:with-param name="iso8601Date"
    select="@bornDate"/>
</xsl:call-template> </td> <td>
<xsl:call-template name="iso8601DateToDisplayDate">
    <xsl:with-param name="iso8601Date"
        select="@diedDate" />
</xsl:call-template> </td> <td>
    <xsl:value-of select="Description" /> </td> </tr>
</xsl:template>
<xsl:template name="iso8601DateToDisplayDate">
    <xsl:param name="iso8601Date" />
    <xsl:variable name="yearPart"
        select="substring($iso8601Date, 1, 4)" />
    <xsl:variable name="monthPart"
        select="substring($iso8601Date, 6, 2)" />
    <xsl:variable name="monthName"
        select="$allMonths/Months/Month[@index =
            number($monthPart)]" />
    <xsl:variable name="datePart"
        select="substring($iso8601Date, 9, 2)" />
    <xsl:value-of select="concat($datePart, ' ',
        $monthName, ' ', $yearPart)" />
</xsl:template>
</xsl:stylesheet>

```

Kết quả của việc phép chuyển đổi này được thể hiện trong Hình 7.4.

Famous People

Famous People			
Name	Born	Died	Description
Winston Churchill	30 November 1874	24 January 1965	Winston Churchill was a mid 20th century British politician who became famous as Prime Minister during the Second World War.
Indira Gandhi	19 November 1917	31 October 1984	Indira Gandhi was India's first female prime minister and was assassinated in 1984.
John F. Kennedy	29 May 1917	22 November 1963	JFK, as he was affectionately known, was a United States president who was assassinated in Dallas, Texas.

Hình 7.4: Kết quả khi thực hiện gán tập tin XSLT
của Ví dụ 7.11 vào People.xml

Trong Ví dụ 7.11, XSLT tham chiếu giá trị tháng từ một tập tin XML bên ngoài. Chúng ta cũng có thể nhúng trực tiếp XML vào trong XSLT. Để thực hiện việc này, chúng ta cần gom nhóm các phần tử dưới dạng một không gian tên như sau:

```

<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform"
    xmlns:myData="http://wrox.com/namespaces/embeddedData">
<xsl:variable name="thisDocument" select="document('')"/>
<myData:Months>
    <Month index="1">January</Month>
    <Month index="2">February</Month>
    <Month index="3">March</Month>
    <Month index="4">April</Month>

```

```

<Month index="5">May</Month>
<Month index="6">June</Month>
<Month index="7">July</Month>
<Month index="8">August</Month>
<Month index="9">September</Month>
<Month index="10">October</Month>
<Month index="11">November</Month>
<Month index="12">December</Month>
</myData:Months>
<!-- rest of stylesheet -->
</xsl:stylesheet>

```

Để truy cập vào dữ liệu XML bên trong XSLT, chúng ta cũng sử dụng hàm *document()* nhưng tham số là rỗng, như sau:

```
<xsl:variable name="thisDocument" select="document('')"/>
```

Xét phần tử *<myData:Months>*. Bởi vì phần tử này nằm trong một không gian tên, nên chúng ta cần phải thêm tiền tố của không gian tên này vào trong đường dẫn:

`$thisDocument/xsl:stylesheet/myData:Months/Month[@index = number($monthPart)]`. Hãy nhớ rằng chỉ có phần tử *<myData: Months>*

Tập tin XSLT mới này được trình bày đầy đủ trong Ví dụ 7.12. Kết quả của sự chuyển đổi này sẽ hoàn toàn giống như Hình 7.4.

Ví dụ 7.12: *PeopleToHtml-LocalDocument.xml*

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform"
    xmlns:myData="http://wrox.com/namespaces/embeddedData">
    <xsl:variable name="thisDocument"
        select="document('')"/>
    <myData:Months>
        <Month index="1">January</Month>
        <Month index="2">February</Month>
        <Month index="3">March</Month>
        <Month index="4">April</Month>
        <Month index="5">May</Month>
        <Month index="6">June</Month>
        <Month index="7">July</Month>
        <Month index="8">August</Month>
        <Month index="9">September</Month>
        <Month index="10">October</Month>
        <Month index="11">November</Month>
        <Month index="12">December</Month>
    </myData:Months>
    <xsl:template match="/">
        <html>
            <head> <title>Famous People</title> </head>
            <body> <h1>Famous People</h1> <hr />
                <table> <caption>Famous People</caption>
                    <thead> <tr> <th>Name</th>
                        <th>Born</th> <th>Died</th>

```

```

<th>Description</th> </tr>  </thead>
<tbody> <xsl:apply-templates
        select="People/Person" />  </tbody>
</table></body> </html>
</xsl:template>
<xsl:template match="Person"> <tr>
    <td> <xsl:value-of select="Name" /> </td> <td>
        <xsl:call-template name="iso8601DateToDisplayDate">
            <xsl:with-param name="iso8601Date"
                select="@bornDate"/>
        </xsl:call-template> </td>
        <td>
            <xsl:call-template name="iso8601DateToDisplayDate">
                <xsl:with-param name="iso8601Date"
                    select="@diedDate"/>
            </xsl:call-template> </td>
            <td> <xsl:value-of select="Description"/> </td>
        </tr> </xsl:template>
<xsl:template name="iso8601DateToDisplayDate">
    <xsl:param name="iso8601Date" />
    <xsl:variable name="yearPart" select=
        "substring($iso8601Date, 1, 4)" />
    <xsl:variable name="monthPart" select=
        "substring($iso8601Date, 6, 2)" />
    <xsl:variable name="monthName" select=
        "$thisDocument/xsl:stylesheet/myData:Months/
        Month[@index = number($monthPart)]" />
    <xsl:variable name="datePart"
        select="substring($iso8601Date, 9, 2)" />
    <xsl:value-of select="concat($datePart, ' ',
        $monthName, ' ', $yearPart)" />
</xsl:template>
</xsl:stylesheet>

```

7.2.7. Phần tử logic có điều kiện (conditional logic)

Có hai cách chính để sử dụng logic có điều kiện trong phiên bản 1.0 của XSLT. Cách thứ nhất là sử dụng một phần tử `<xsl:if>`; cách thứ hai là sử dụng `<xsl:choose>`. Phần tử `<xsl:if>` cho phép chúng ta thực hiện các kiểm tra đơn giản nhưng không cung cấp cho chúng ta tùy chọn khác, cú pháp là:

```

<xsl:if test="test condition goes here">
    <!-- instructions if the condition is true -->
</xsl:if>

```

Phần tử `<xsl:if>` có thuộc tính được là *test*. Giá trị của thuộc tính này là một biểu thức XPath dùng để tạo ra một giá trị Boolean là *true* hoặc *false*. Nếu điều kiện đánh giá là *true*, các lệnh tiếp theo sau phần tử được thực hiện. Các ví dụ kiểm tra có thể là:

- Person: Kết quả là *true* nếu có ít nhất một phần tử *Person*.
- *Name* = ‘*Indira Gandhi*’: Kết quả là *true* nếu phần tử *Name* chứa giá trị ‘*Indira Gandhi*’.

- $number(substring(Person/@bornDate, 1, 2)) = 19$: Lấy hai ký tự đầu tiên của thuộc tính *bornDate* và trả về *true* nếu giá trị bằng 19.

Quay lại Ví dụ 7.12, chúng ta có thể đánh dấu tên của những người sinh ra trong thế kỷ 20 bằng một màu khác. Để làm như vậy, thêm một số câu lệnh vào bên dưới phần tử <Person> như sau:

- Khai báo một phần tử <xsl:variable> có tên *nameCSS* để giữ thông tin kiểu có liên quan.
- Sau đó kiểm tra thuộc tính *bornDate* có giá trị là 19 hay không (thế kỷ 20). Nếu điều này là đúng thì đặt giá trị của biến thành màu đỏ, nếu không nó sẽ vẫn để màu mặc định (đen).
- Tiếp theo, thêm thuộc tính *style* vào phần tử <td> để lấy giá trị của *\$NameCSS*.

Kết quả cuối cùng giống như đoạn mã sau đây:

```
<xsl:template match="Person">
  <xsl:variable name="nameCSS">
    <xsl:if test="number(substring(@bornDate, 1, 2)) =
      19">color:red; </xsl:if> </xsl:variable>
    <tr> <td style="{{$nameCSS}}>
      <xsl:value-of select="Name" /> </td> .....
  </xsl:template>
```

Khi chúng ta tiến hành chuyển đổi, chúng ta sẽ có được kết quả thể hiện trong Hình 7.5, nơi chính trị gia đầu tiên, Winston Churchill, màu đen và những người khác có màu đỏ.

Famous People

Famous People			
Name	Born	Died	Description
Winston Churchill	30 November 1874	24 January 1965	Winston Churchill was a mid 20th century British politician who became famous as Prime Minister during the Second World War.
Indira Gandhi	19 November 1917	31 October 1984	Indira Gandhi was India's first female prime minister and was assassinated in 1984.
John F. Kennedy	29 May 1917	22 November 1963	JFK, as he was affectionately known, was a United States president who was assassinated in Dallas, Texas.

Hình 7.5: Kết quả khi thay đổi đoạn lệnh ở Ví dụ 7.12

Phần tử <xsl:if> khá hạn chế. Vì thế chúng ta cần một cái gì đó mạnh mẽ hơn để xử lý nhiều điều kiện hơn, đó là <xsl:choose>. Cú pháp như sau:

```
<xsl:choose>
  <xsl:when test="test condition goes here">
  </xsl:when>
  <!--mệnh đề otherwise không bắt buộc -->
  <xsl:otherwise>
    <!--các câu lệnh -->
  </xsl:otherwise>
</xsl:choose>
```

Về cơ bản, chúng ta có thể có bất kỳ số phần tử <xsl:when> bên trong <xsl:choose>. Chúng ta cũng có thể có tùy chọn <xsl:otherwise> được thực hiện nếu tất cả các kiểm tra trước đó đã thất bại.

Ví dụ, giả sử chúng ta muốn cải thiện giao diện của tài liệu đầu ra của chúng ta bằng cách cho mỗi hàng số lẻ có màu nền khác với hàng số chẵn. Chúng ta có thể thực hiện việc này bằng cách sử dụng hàm *position()*, hàm này đưa ra vị trí của nút được xử lý. Vì vậy, bên trong nút *Person*, chúng ta có thể thêm những dòng lệnh sau đây:

```

<xsl:template match="Person">
  <xsl:variable name="rowCSS">
    <xsl:choose>
      <xsl:when test="position() mod 2 = 0">
        color:blue; </xsl:when>
        <xsl:otherwise>color:black;</xsl:otherwise>
      </xsl:choose> </xsl:variable>
  <xsl:variable name="nameCSS">
    <xsl:if test="number(substring(@bornDate, 1, 4)) >
      $targetYear"> color:red; </xsl:if>
  </xsl:variable>
  <!--Gan bien rowCSS cho thuoc tinh Style cua tr-->
  <tr style="${rowCSS}">
    ...
</xsl:template>

```

Tập tin XSLT mới này được trình bày đầy đủ trong Ví dụ 7.13.

Ví dụ 7.13: PeopleToHtml-BornAfter.xslt

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform"
  xmlns:myData="http://wrox.com/namespaces/embeddedData">
  <xsl:param name="targetYear" select="1900" />
  <xsl:variable name="thisDocument"
    select="document('')"/>
  <myData:Months>
    <Month index="1">January</Month>
    <Month index="2">February</Month>
    <Month index="3">March</Month>
    <Month index="4">April</Month>
    <Month index="5">May</Month>
    <Month index="6">June</Month>
    <Month index="7">July</Month>
    <Month index="8">August</Month>
    <Month index="9">September</Month>
    <Month index="10">October</Month>
    <Month index="11">November</Month>
    <Month index="12">December</Month>
  </myData:Months>
<xsl:template match="/">
  <html>
    <head> <title>Famous People</title> </head>
    <body> <h1>Famous People</h1> <hr />
    <table> <caption>Famous People</caption>
    <thead> <tr> <th>Name</th> <th>Born</th>

```

```

        <th>Died</th> <th>Description</th> </tr>
    </thead> <tbody>
        <xsl:apply-templates select="People/Person" />
    </tbody> </table> </body> </html> </xsl:template>
<xsl:template match="Person">
    <xsl:variable name="rowCSS">
        <xsl:choose>
            <xsl:when test="position() mod 2 = 0">
                color:blue;</xsl:when>
            <xsl:otherwise>color:black;</xsl:otherwise>
        </xsl:choose> </xsl:variable>
    <xsl:variable name="nameCSS">
        <xsl:if test="number(substring(@bornDate, 1, 4)) >
                      $targetYear">color:red;</xsl:if>
    </xsl:variable>
    <!--Gan bien rowCSS cho thuoc tinh Style cua phan tu tr-->
    <tr style="{{$rowCSS}}>
        <td style="{{$nameCSS}}>
            <xsl:value-of select="Name" /> </td>
        <td> <xsl:call-template name="iso8601DateToDisplayDate">
            <xsl:with-param name="iso8601Date"
                            select="@bornDate" />
            </xsl:call-template> </td>
        <td> <xsl:call-template name="iso8601DateToDisplayDate">
            <xsl:with-param name="iso8601Date"
                            select="@diedDate" />
            </xsl:call-template> </td>
        <td> <xsl:value-of select="Description" /> </td>
    </tr> </xsl:template>
<xsl:template name="iso8601DateToDisplayDate">
    <xsl:param name="iso8601Date" />
    <xsl:variable name="yearPart"
                  select="substring($iso8601Date, 1, 4)" />
    <xsl:variable name="monthPart"
                  select="substring($iso8601Date, 6, 2)" />
    <xsl:variable name="monthName"
                  select="$thisDocument/xsl:stylesheet/
myData:Months/Month[@index = number($monthPart)]"/>
    <xsl:variable name="datePart"
                  select="substring($iso8601Date, 9, 2)" />
    <xsl:value-of select="concat($datePart, ' ',
                                $monthName, ' ', $yearPart)" />
</xsl:template>
</xsl:stylesheet>

```

Kết quả chuyển đổi được thể hiện ở Hình 7.6.

Famous People

Famous People			
Name	Born	Died	Description
Winston Churchill	30 November 1874	24 January 1965	Winston Churchill was a mid 20th century British politician who became famous as Prime Minister during the Second World War.
Indira Gandhi	19 November 1917	31 October 1984	Indira Gandhi was India's first female prime minister and was assassinated in 1984.
John F. Kennedy	29 May 1917	22 November 1963	JFK, as he was affectionately known, was a United States president who was assassinated in Dallas, Texas.

Hình 7.6: Kết quả chuyển đổi Ví dụ 7.13

Theo kết quả ở Hình 7.6, hai tên cuối cùng có màu đỏ, do có năm sinh lớn hơn 1900.

7.2.8. Phần tử `<xsl:param>`

Phần tử `<xsl:param>` được sử dụng để khai báo một tham số cục bộ hoặc toàn cục. Lưu ý: Tham số là toàn cục nếu nó được khai báo là một phần tử cấp cao nhất, thường được đặt bên dưới ngay sau phần tử `<xsl:stylesheet>`, và cục bộ nếu nó được khai báo bên trong một khuôn mẫu.

Cú pháp:

```
<xsl:param name="name" select="expression">
    <!-- Content:template -->
</xsl:param>
```

Trong cú pháp trên, thuộc tính `name` là bắt buộc, dùng để lưu tên của tham số; thuộc tính `select` là tùy chọn, dùng để chỉ định đường dẫn XPath hoặc giá trị mặc định cho tham số.

Trong Ví dụ 6.13, khai báo `<xsl:param name="targetYear" select="1900"/>` cho tham số `targetYear` và được đặt giá trị mặc định là 1900. Sau đó sử dụng thuộc tính `bornDate` để kiểm tra năm sinh có lớn hơn 1900 hay không, nếu đúng thì hiển thị tên của người đó là màu đỏ:

```
<xsl:variable name="nameCSS">
    <xsl:if test="number(substring(@bornDate, 1, 4)) >
        $targetYear">color:red;
    </xsl:if>
</xsl:variable>
```

7.2.9. Phần tử `<xsl:sort>`

Phần tử `<xsl:sort>` khá đơn giản. Nó có thể được sử dụng như là con của `<xsl:apply-templates>` hoặc `<xsl:for-each>`. Nó có các thuộc tính sau để kiểm soát quá trình sắp xếp:

- `select`: Một biểu thức XPath chỉ tới các nút để sắp xếp.
- `data-type`: Làm thế nào để xử lý các giá trị nút, thường là văn bản hoặc số.
- `order`: Gồm 2 thuộc tính: tăng (asc) hoặc giảm (desc), mặc định là asc.

Giả sử chúng ta muốn sắp xếp những người trong bảng kết quả thuộc Hình 7.6 có năm sinh tăng dần. Điều này sẽ cần hai bước: bước thứ nhất chúng ta cần chuyển đổi giá trị của `bornDate`, hiện tại ở định dạng yyyy-mm-dd, thành một số có thể được sắp xếp; và thứ hai, chúng ta cần sử dụng phần tử `<xsl:sort>`.

Đối với bước đầu tiên chúng ta cần sử dụng hàm `translate()`. Hàm này có ba đối số. Đầu tiên là một biểu thức trả đến dữ liệu để làm việc, tham số thứ hai là tìm kiếm, và thứ ba là giá trị để thay thế cho ký tự tìm thấy. Ví dụ:

```
translate('The first of the few ', 'fiw', 'wot')
```

Câu lệnh này sẽ tìm các ký tự *f*, *i*, và *w* sau đó thay đổi chúng thành *w*, *o*, và *t*, tương ứng. Kết quả của chuỗi sau khi biến đổi là:

```
The worst of the wet
```

Bây giờ trở lại trong ví dụ với *people.xml*, chúng ta sử dụng hàm `translate()` để loại bỏ các dấu gạch nối trong giá trị của *bornDate*:

```
translate(@bornDate, ' - ', '')
```

Kết quả của dòng lệnh trên mang lại cho chúng ta một chuỗi tám chữ số. Sau đó chúng ta có thể dùng hàm `<xsl:sort>` để sắp xếp tăng dần dựa trên giá trị của *bornDate*. Phần tử `<xsl:sort>` là con trực tiếp của `<xsl:apply-templates>`, như thể hiện dưới đây:

```
<xsl:apply-templates select="People/Person">
  <xsl:sort select="translate(@bornDate, ' - ', '')"
    data-type="number"/>
</xsl:apply-templates>
```

Sau khi thay đoạn lệnh trên cho dòng `<xsl:apply-templates select="People/Person"/>` trong Ví dụ 7.13, kết quả thực hiện chuyển đổi được thể hiện ở Hình 7.7.

Famous People

Famous People				
Name	Born	Died	Description	
Winston Churchill	30 November 1874	24 January 1965	Winston Churchill was a mid 20th century British politician who became famous as Prime Minister during the Second World War.	
John F. Kennedy	29 May 1917	22 November 1963	JFK, as he was affectionately known, was a United States president who was assassinated in Dallas, Texas.	
Indira Gandhi	19 November 1917	31 October 1984	Indira Gandhi was India's first female prime minister and was assassinated in 1984.	

Hình 7.7: Kết quả chuyển đổi Ví dụ 6.73 với lệnh sắp xếp cột Born

So sánh với thể hiện ở Hình 7.6, kết quả ở Hình 7.7 có thay đổi, dòng thứ 2 trong Hình 7.6 được chuyển xuống dòng thứ 3 do giá trị ở cột Born nhỏ hơn.

7.2.10. Phần tử `<xsl:copy>` và `<xsl:copy-of>`

Cả hai phần tử này đều có thể được sử dụng để sao chép nội dung từ nguồn đến cây đầu ra. Phần tử `<xsl:copy>` thực hiện một bản sao nông cạn vì nó chỉ sao chép các nút hiện tại mà không sao chép bất kỳ nút con nào bên trong. Phần tử `<xsl:copy-of>` thực hiện bản sao chép sâu - nó sao chép nút đã được xác định với tất cả các con và thuộc tính của nó.

Ví dụ 7.14 dưới đây cho phép chúng ta kiểm tra sự khác nhau giữa `<xsl:value-of>`, `<xsl:copy>`, và `<xsl:copy-of>` trong việc xử lý từng phần tử `<Person>`.

Ví dụ 7.14: CopyingNodes.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" />
```

```

<xsl:template match="/">
  <People>
    <xsl:apply-templates select="People/Person" />
  </People>
</xsl:template>
<xsl:template match="Person">
  <valueOf>  <xsl:value-of select=".." />  </valueOf>
  <copy>    <xsl:copy /> </copy>
  <copyOf>   <xsl:copy-of select=".."/>   </copyOf>
</xsl:template>
</xsl:stylesheet>

```

Giải thích Ví dụ 7.14: Phần tử `<xsl:output>` được đặt sau phần tử `<xsl:stylesheet>` nhằm xác định định dạng đầu ra trong thuộc tính `method`. Có các tùy chọn của thuộc tính `method` là `xml` (mặc định), `html` và `text` (văn bản). Phiên bản XSLT 2.0 còn có tùy chọn thứ tư là `xhtml`. Nếu chúng ta muốn kết quả ở đầu ra được thực hiện để làm cho nó dễ đọc hơn thì chúng ta sử dụng thuộc tính `indent` được thiết lập là `yes`. Một thuộc tính khác thường thấy trong `<xsl:output>` là `encoding` (mã hóa). Thuộc tính này cho phép chúng ta biết liệu đầu ra phải ở trong một bộ mã hóa khác với `utf-8`, ví dụ `iso-8859-1`, hay không.

Trong Ví dụ 7.14, đầu tiên `<xsl:template>` khớp với nút gốc (/), và điều này chỉ đơn giản là thêm một phần tử `<People>` ở tài liệu đầu ra. Sau đó `<xsl:apply-templates>` được gọi bên trong `<People>` và các phần tử `<Person>` được chọn để xử lý. Các phần tử `<Person>` bị xử lý bởi phần tử `<xsl:template>` thứ hai. Bên trong phần tử `<xsl:template>` thứ hai này có ba chế độ xem khác nhau của mỗi phần tử `<Person>`.

Üng với mỗi phần tử `<Person>` cách thể hiện thứ nhất được chứa trong một phần tử `<valueOf>`, và được xử lý bởi `<xsl:value-of select="..">`, nghĩa là bộ xử lý chỉ chiết xuất các giá trị văn bản và bỏ qua tên các thuộc tính và các phần tử con. Ở cách thể hiện thứ hai, phần tử `<Person>` này được chứa trong phần tử `<copy>` và được xử lý bởi `<xsl:copy>`, nghĩa là bộ xử lý chỉ đưa ra phần tử `<Person>` mà không có thuộc tính hoặc con của nó. Cách thể hiện thứ ba được chứa bên trong phần tử `copyOf` với xử lý `<xsl:copy-of select=".."/>`, nghĩa là tạo ra một bản sao chép sâu bao gồm phần tử `<Person>` cùng với các thuộc tính và tất cả các con, bao gồm các nút văn bản. Kết quả chuyển đổi được hiển thị trong Ví dụ 7.15.

Ví dụ 7.15: CopyingNodes.xml

```

<?xml version="1.0" encoding="utf-8"?>
<People>
  <valueOf> Winston Churchill
    Winston Churchill was a mid 20th century British
    politician who became famous as Prime Minister
    during the Second World War. </valueOf>
  <copy> <Person /> </copy>
  <copyOf>
    <Person bornDate="1874-11-30" diedDate="1965-01-24">
      <Name>Winston Churchill</Name>
      <Description>
        Winston Churchill was a mid 20th century British

```

```

        politician who became famous as Prime Minister
        during the Second World War.
    </Description> </Person> </copyOf>
    <valueOf> Indira Gandhi
        Indira Gandhi was India's first female prime
        minister and was assassinated in 1984.
    </valueOf> <copy> <Person /> </copy>
    <copyOf>
        <Person bornDate="1917-11-19" diedDate="1984-10-31">
            <Name>Indira Gandhi</Name>
            <Description>
                Indira Gandhi was India's first female prime
                minister and was assassinated in 1984.
            </Description> </Person> </copyOf>
    <valueOf> John F. Kennedy
        JFK, as he was affectionately known, was a United
        States President who was assassinated in Dallas,
        Texas. </valueOf>
    <copy> <Person /> </copy>
    <copyOf>
        <Person bornDate="1917-05-29" diedDate="1963-11-22">
            <Name>John F. Kennedy</Name>
            <Description> JFK, as he was affectionately known,
                was a United States president who was
                assassinated in Dallas, Texas. </Description>
        </Person> </copyOf>
    </People>

```

Đối với bất kỳ ngôn ngữ nào, một vấn đề quan trọng khác là việc sử dụng lại các đoạn mã. Đối với XSLT, chúng ta có hai cách để sử dụng lại các mã là bằng khai báo: `<xsl:include>` và `<xsl:import>`.

7.2.11. Phản tử `<xsl:include>` và `<xsl:import>`

Phản tử `<xsl:include>` cho phép chúng ta đưa một bảng định kiểu (*style sheet*) vào bên trong một cái khác. Điều này có tác dụng tương tự như cách sao chép và dán mã từ các mẫu định kiểu bên ngoài vào bảng định kiểu chính, nhưng cho phép chúng ta xây dựng thêm các module hữu ích khác.

Ví dụ sau trích xuất mã được sử dụng để thay thế giá trị ngày được định dạng XML bằng một phiên bản thân thiện với người dùng. Sau đó, chúng ta sẽ kết hợp biểu định kiểu này vào một tập tin định kiểu khác bằng cách sử dụng phản tử `<xsl:include>`.

Ví dụ 7.16: DateTemplates.xslt

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform"
    xmlns:myData="http://wrox.com/namespaces/embeddedData">
    <xsl:variable name="thisDocument"
                  select="document('')"/>
    <myData:Months>
        <Month index="1">January</Month>
        <Month index="2">February</Month>
        <Month index="3">March</Month>

```

```

<Month index="4">April</Month>
<Month index="5">May</Month>
<Month index="6">June</Month>
<Month index="7">July</Month>
<Month index="8">August</Month>
<Month index="9">September</Month>
<Month index="10">October</Month>
<Month index="11">November</Month>
<Month index="12">December</Month>
</myData:Months>
<xsl:template name="iso8601DateToDisplayDate">
<xsl:param name="iso8601Date" />
<xsl:variable name="yearPart"
    select="substring($iso8601Date, 1, 4)" />
<xsl:variable name="monthPart"
    select="substring($iso8601Date, 6, 2)" />
<xsl:variable name="monthName"
    select="$thisDocument/xsl:stylesheet/myData:Months/
Month[@index = number($monthPart)]" />
<xsl:variable name="datePart"
    select="substring($iso8601Date, 9, 2)" />
<xsl:value-of select="concat($datePart, ' ', $monthName,
' ', $yearPart)" />
</xsl:template>
</xsl:stylesheet>

```

Tiếp theo, chúng ta tạo một XSLT thứ hai, *PeopleToHtml-UsingIncludes.xslt*, với mã trong Ví dụ 7.17.

Ví dụ 7.17: *PeopleToHtml-UsingIncludes.xslt*

```

<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform"
    xmlns:myData="http://wrox.com/namespaces/embeddedData">
    <xsl:include href="DateTemplates.xslt" />
    <xsl:param name="targetYear" select="3000" />
    <xsl:template match="/">
        <html>
            <head> <title>Famous People</title> </head>
            <body> <h1>Famous People</h1> <hr />
                <table> <caption>Famous People</caption>
                    <thead> <tr>
                        <th>Name</th> <th>Born</th>
                        <th>Died</th> <th>Description</th>
                    </tr> </thead>
                    <tbody>
                        <xsl:apply-templates select="People/Person">
                            <xsl:sort select="translate(@bornDate, '-', '')" data-type="number"/>
                        </xsl:apply-templates>
                    </tbody> </table>
                </body>
            </html>
        </xsl:template>
        <xsl:template match="Person">
            <xsl:variable name="rowCSS">

```

```

<xsl:choose>
    <xsl:when test="position() mod 2 = 0">
        color:#0000aa;</xsl:when>
    <xsl:otherwise>color:#006666;</xsl:otherwise>
</xsl:choose>
</xsl:variable>
<xsl:variable name="nameCSS">
<xsl:if test="number(substring(@bornDate, 1, 4))>
    $targetYear">color:red;</xsl:if>
</xsl:variable>
<tr style="{$rowCSS}">
    <td style="{$nameCSS}">
        <xsl:value-of select="Name" />
    </td> <td>
        <xsl:call-template name="iso8601DateToDisplayDate">
            <xsl:with-param name="iso8601Date"
                select="@bornDate" />
        </xsl:call-template>
    </td> <td>
        <xsl:call-template name="iso8601DateToDisplayDate">
            <xsl:with-param name="iso8601Date"
                select="@diedDate" />
        </xsl:call-template>
    </td> <td> <xsl:value-of select="Description" />
    </td> </tr>
</xsl:template>
</xsl:stylesheet>

```

Phần tử `<xsl:include>` sử dụng thuộc tính `href` để trỏ tới tệp tin XSLT khác. Bộ xử lý lấy mã từ tệp tham chiếu, loại bỏ phần tử `<xsl:stylesheet>` bên ngoài và tạo ra một biểu diễn trong bộ nhớ bao gồm bảng định kiểu của trang chính và của trang được thêm vào. Sau đó, sự chuyển đổi được thực hiện theo cách thông thường.

Phần tử `<xsl:import>` hoạt động theo cách rất giống với `<xsl:include>` chỉ khác là nếu các bảng mã tham chiếu có xung đột với bất kỳ mã đã có trong XSLT chính thì chúng sẽ có ưu tiên thấp hơn. Ví dụ, nếu chúng ta có hai đoạn mã cùng tác động lên phần tử `<Person>`, thì đoạn mã nằm trong XSLT chính sẽ được thực hiện. Đối với các đoạn mã không có bất kỳ mẫu nào khớp với cùng một nút thì `<xsl:import>` và `<xsl:include>` hoạt động giống nhau.

Tóm lại, trong chương này chúng ta đã học được:

- Tiết lộ cơ bản đầu sau XSLT để chuyển đổi một tài liệu XML sang một XML khác, sang một HTML hoặc văn bản thuần túy.
- Phần tử `<xsl: template>` cơ bản tương ứng với XML cụ thể và xuất ra nội dung mới
- Các phần tử `<xsl: apply-templates>` kết hợp với `<xsl: template>`.
- XPath được sử dụng trong suốt XSLT để chỉ định các nút xử lý và trích xuất các mục dữ liệu cụ thể.
- Các thành phần nâng cao hơn `<xsl: include>` và `<xsl: import>` cho phép chúng ta viết các mô đun mã có thể tái sử dụng.

BÀI TẬP CHƯƠNG 7

1. Tạo tập tin LopHoc.xml chứa mã lớp và tên lớp. Tạo tập tin LopHoc.xsl để truy vấn và hiển thị thông tin các lớp học như sau:

MaLop	TenLop
CNTT16	Công nghệ thông tin 2017
TCKT17	Tài chính kế toán 2017
TKDH15	Thiết kế đồ họa

2. Tạo tập tin **SinhVien.xml** có định nghĩa kiểu dữ liệu sử dụng XML Schema (**SinhVien.xsd**). Giới tính chỉ nhập Nam hoặc Nữ. Mã lớp nhập theo qui ước: 4 ký tự đầu là chữ cái, 2 ký tự kế tiếp là số. Tạo tập tin **SinhVien.xsl** truy vấn và hiển thị thông tin sinh viên tăng dần theo tên như sau:

STT	MaSv	TenSv	GioiTinh	NgaySinh	MaLop
1	sv01	Phan Tuấn Cường	Nam	03/03/1993	CNTT16
2	sv02	Huỳnh Chí Tâm	Nam	08/03/1993	CNTT16
3	sv03	Vương Chí Dũng	Nam	14/04/1991	CNTT16
4	sv04	Phạm Đức Nguyên	Nam	27/07/1993	TCKT15
5	sv05	Trương Mỹ Yên Nhi	Nữ	03/10/1993	TCKT15

3. Tạo tập tin **MonHoc.xml** (Mã môn học, tên môn học, số giờ) có định nghĩa kiểu dữ liệu sử dụng XMLSchema (**MonHoc.xsd**) với yêu cầu: Số giờ tối thiểu là 4, tối đa là 48. Tạo tập tin **MonHoc.xsl** truy vấn và hiển thị các môn học có số giờ từ 30 trở lên như sau:

MaMh	TenMH	SoGio
CSDL	Cơ sở dữ liệu	30
MCB	Mạng căn bản	30
THDC	Tin học đại cương	45
THVP	Tin học văn phòng	45
XLA	Xử lý ảnh	45

4. Tạo tập tin **KetQua.xml** (Mã sinh viên, mã môn học, điểm thi) có định nghĩa kiểu dữ liệu sử dụng XMLSchema (**KetQua.xsd**) với yêu cầu: Điểm thi ≥ 0 và điểm thi ≤ 10 . Tạo tập tin **KetQua.xsl** truy vấn và hiển thị điểm thi của những sinh viên học môn THVP và có điểm thi ≥ 5 . Hiển thị giảm dần theo điểm thi. như sau:

MaSv	MaMh	DiemThi
sv01	THDC	10
sv01	THVP	9.5
sv02	THDC	9
sv02	THVP	8
sv03	THDC	7
sv03	THVP	4.5

5. Sử dụng lại tập tin **KhachSan.xml** (*bài tập 7a, chương 4*) thực hiện các yêu cầu sau:

- a. Sử dụng XSLT chuyển XML thành trang web như sau: Có dòng chữ “Chương Trình Quản Lý Khách Sạn” ở giữa màn hình, kiểu **<H1>** màu đỏ, Họ tên sinh viên thực hiện ở giữa màn hình, kiểu **<H3>** màu xanh

- b. Tạo tập tin **HoaDon.xsl** xuất theo dạng sau:

HÓA ĐƠN THANH TOÁN

Mã khách hàng:

Tên khách hàng:

CMND:

Ngày lập:

ST T	Ngày đến	Ngày Đi	Mã Phòng	Đơn giá Phòng	Mã Dịch Vụ	Đơn giá DV	Thành Tiền

Yêu cầu:

- Dòng chữ “HÓA ĐƠN THANH TOÁN” màu đỏ, căn giữa. Tiêu đề các cột trong bảng màu XANH, bảng CĂN GIỮA.
- Cột **SoTT** được tạo ra một cách tự động.
- Tô màu nền xanh cho các ô CÓ LOẠI PHÒNG LÀ ĐÔI.
- Cột Thành tiền được tính theo công thức: (NGÀY ĐI–NGÀY ĐÉN)*ĐƠN GIÁ PHÒNG + ĐƠN GIÁ DỊCH VỤ
- c. Sử dụng lại tập tin xml, tạo tập tin xslt hiển thị bảng cho khách hàng tên “Nguyen Tran”

- d. Sử dụng lại tập tin xml, tạo tập tin hiển thị bảng cho khách hàng tên “Nguyen Tran” và khách hàng “Tran Tuan Anh”
6. Sử dụng lại **QLDia.xml** (bài tập 8a, chương 4), tạo các tập tin .xslt theo các điều kiện sau:
- a. Đếm xem phần tử gốc có bao nhiêu nút con
 - b. Chọn các phần tử có tên CD, ARTIST, PRICEVN
 - c. Chọn các phần tử là con của CD
 - d. Chọn CD đầu tiên
 - e. Chọn CD tại vị trí thứ 2
 - f. Chọn CD cuối cùng
 - g. Chọn phần tử cha của CD
 - h. Chọn các Track có length <5 phút
 - i. Tìm các đĩa CD có tên bắt đầu là A
 - j. Tìm các đĩa CD có tên chứa chữ “Uớc Mơ”;
 - k. Tìm các đĩa CD tên là Xuân 1 và Xuân 2
 - l. Tìm các đĩa CD có tên không chứa chữ Xuân
 - m. Chọn các phần tử cấp trên của CD
 - n. Tìm các phần tử cùng cấp trước phần tử CD đầu tiên
 - o. Chọn các phần tử cùng cấp sau phần tử CD đầu tiên
 - p. Chọn các phần tử theo sau phần tử Message và các phần tử con của nó
 - q. Chọn các phần tử đứng trước phần tử Message và các phần tử con của nó
 - r. Chọn phần tử là tổ tiên của Title
 - s. Chọn tất cả các phần tử trước CD ngoại trừ phần tử gốc
 - t. Chọn tất cả các phần tử CD và con của nó
 - u. Chọn tất cả các phần tử Track và tổ tiên của nó
 - v. Chọn phần tử đi sau phần tử Message
 - w. Đếm xem có bao nhiêu phần tử track
 - x. Có bao nhiêu phần tử Track trong đĩa CD cuối cùng

Chương 8

XQUERY

XQuery là một ngôn ngữ dùng để tìm kiếm và thao tác bất cứ thứ gì có thể được biểu diễn dưới dạng cây sử dụng mô hình XQuery và XPath Data Model (XDM). Các chương trình XQuery (hoặc các biểu thức XQuery) có thể truy cập nhiều tài liệu, hoặc thậm chí nhiều cơ sở dữ liệu và trích xuất các kết quả rất hiệu quả.

XQuery xây dựng dựa trên XPath và là mở rộng của XPath. Điều này có nghĩa là cú pháp của XQuery giống như XPath và không dựa trên phần tử XML như XSLT.

Trong chương này, bạn sẽ học được tất cả về ngôn ngữ XQuery này: XQuery là gì và cách sử dụng nó. Bạn cũng sẽ tìm hiểu một số hướng dẫn sơ bộ để sử dụng XQuery khi sử dụng XSLT. Nói tóm lại, sử dụng XSLT thường là tốt nhất nếu bạn mong đợi xử lý toàn bộ tài liệu XML từ đầu đến cuối. Trong khi đó XQuery thường là tốt nhất nếu bạn muốn xử lý một phần của một tài liệu, nếu bạn muốn làm việc với cùng một tài liệu nhiều lần, hoặc nếu bạn đang xử lý một số lượng lớn các tài liệu.

Ngày nay, XQuery được sử dụng rộng rãi và có rất nhiều trình xử lý XQuery khác nhau, như Saxon, MarkLogic (bản thương mại), eXist, Qizx, oXygen, BaseX,... Các ví dụ trong chương này tập trung này vào trình xử lý BaseX, vì BaseX là một ứng dụng độc lập mã nguồn mở, có thể được chạy độc lập hoặc như một máy chủ, và cũng có một giao diện đồ họa dễ sử dụng.

8.1. Cài đặt BaseX và thực hiện truy vấn

Các bước cụ thể để cài đặt phần mềm BaseX như sau:

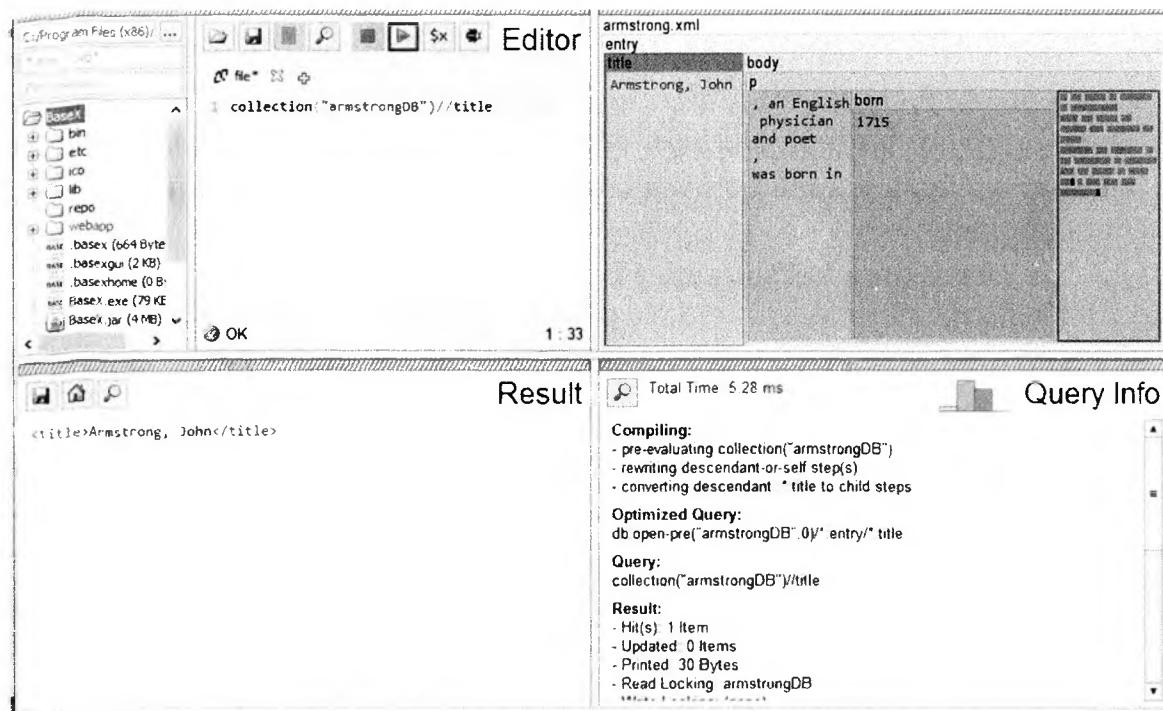
1. Vào www.baseX.org và tìm đường link Tải xuống. Thông thường ở cuối văn bản giới thiệu sản phẩm hoặc ngay ở trang đầu.
2. Chọn phiên bản chính thức. BaseX phát hành thường xuyên các phiên bản - tại thời điểm viết giáo trình này, BaseX có phiên bản BaseX 8.6.4.exe. Có một vài tệp tin để lựa chọn: trình cài đặt Windows cũng như tệp .dmg dành cho người dùng Mac OS X và tệp nén ZIP cho những người dùng Linux. Tải xuống tệp tin nào phù hợp với hệ điều hành của bạn.
3. Tải tập tin, và tiến hành cài đặt BaseX
4. Tạo tài liệu XML đơn giản sau đây (với bất kỳ trình soạn thảo văn bản hoặc XML) và đặt tên là *arm strong.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<entry id="armstrong-john">
    <title>Armstrong, John</title>
    <body><p>, an English physician and poet, was born in
        <born>1715</born> in the parish of Castleton in
        Roxburghshire, where his father and brother were clergymen;
        and having completed his education at the university of
```

```

Edinburgh, took his degree in physic, Feb. 4, 1732, with
much reputation.</p>
</body>
</entry>
```

5. Bây giờ trở lại cửa sổ BaseX, và từ trình đơn *Database*, chọn *Open & Manage*. Tạo cơ sở dữ liệu mới được gọi là "armstrongDB" bằng cách chỉ đường dẫn đến tập tin *armstrong.xml*. Bạn sẽ thấy Hình 8.1, mặc dù bộ cục thực tế có thể thay đổi nếu bạn có phiên bản BaseX mới hơn 8.6.4. Trong vùng Editor, trong tab được đánh dấu File, gõ truy vấn ngắn sau đây: *collection("armstrongDB")//title*



Hình 8.1: Kết quả thực hiện truy vấn trên BaseX

Khởi động truy vấn bằng cách nhấp vào biểu tượng hình tam giác màu xanh lá cây ở khu vực Editor, gần giữa cửa sổ BaseX. Bạn sẽ thấy kết quả xuất hiện trong vùng bên dưới và cũng còn có một số thông kê về khoảng thời gian truy vấn đã thực hiện câu lệnh. XQuery có thời gian thực hiện rất nhanh.

Trong câu lệnh ở trên, bạn đã thấy một biểu thức XQuery rất đơn giản, với chỉ là một dòng. Trên thực tế, XQuery có rất nhiều biểu thức với các khối câu lệnh dài hơn và phức tạp hơn.

8.2. Câu lệnh FLWOR

FLWOR (được đọc là "flower") là viết tắt của *For*, *Let*, *Where*, *Order by*, và *Return*. FLWOR tương tự như câu lệnh truy vấn Select trong SQL. Mỗi câu lệnh FLWOR **phải luôn bắt đầu với** mệnh đề *for* hay mệnh đề *let* (hay có thể là cả hai). Các câu lệnh FLWOR có thể được XQuery xử lý song song.

Cú pháp:

```

(ForClause | LetClause) +
[WhereClause]
[OrderByClause]
return ExprSingle
```

8.2.1. Mệnh đề For

Chức năng: Tạo ra các bộ dữ liệu tuần tự, mỗi bộ được gán cho một giá trị biến điều khiển, được gọi là tuple stream. Cơ chế hoạt động của mệnh đề for khi XQuery xử lý như sau:

- Thực hiện tạo ra các bộ dữ liệu theo thứ tự
- Kiểm tra từng bộ dữ liệu với điều kiện nếu có trong câu lệnh FLWOR
- Đưa ra kết quả trả về có thể là một bộ dữ liệu đang có hay trích xuất hay kết xuất của quá trình xử lý dựa trên bộ dữ liệu tùy theo biểu thức kết quả trong mệnh đề return
- 3 bước trên được thực hiện cho đến khi các bộ dữ liệu được duyệt qua hết

Mệnh đề for có thể xử lý nhiều hơn một biểu thức với nhiều biến. Khi điều này diễn ra, XQuery thực hiện tương tự như cơ chế thực hiện vòng lặp chèn trong vòng lặp với tiêu chí biến với biểu thức được khai báo là vòng lặp trong cùng, biến khai báo đầu tiên với biểu thức là vòng lặp ngoài cùng.

Cú pháp:

for \$varName [as sequenceType] [at \$pos] in expr [, ...]

Trong đó:

- **sequenceType**: các kiểu dữ liệu được định nghĩa như element(), atomic schema type, attribute(), node(), item() ...
- **pos**: thể hiện vị trí – thứ tự tương ứng với bộ dữ liệu đang được duyệt
- **expr**: biểu thức xpath

8.2.2. Mệnh đề Let

Chức năng: Tương tự như mệnh đề for nhưng các biến với biểu thức chỉ được thực hiện duy nhất một lần. Nếu như trong mệnh đề Let có nhiều hơn một biến khai báo thì XQuery chỉ tạo ra bộ dữ liệu duy nhất chứa tất cả biến đó.

Cú pháp:

let \$varName [as sequenceType] := expr [, ...]

Lưu ý cú pháp trên có phép toán gán thể hiện bằng :=

8.2.3. Mệnh đề where

Chức năng: Thể hiện điều kiện để lọc các kết quả của các bộ dữ liệu có được trong mệnh đề for hay let, tương tự như mệnh đề Where trong câu lệnh SQL nhưng biểu thức trong where ở XQuery chính là câu lệnh Xpath.

Mệnh đề where sử dụng được tất cả các biến được khai báo trong mệnh đề for và let

Cú pháp: **where expr**

8.2.4. Mệnh đề return

Chức năng: Kết xuất từ bộ dữ liệu, kết xuất hay xử lý từ các bộ dữ liệu hay định dạng theo kết xuất mong muốn.

Cú pháp: **return expr**

8.2.5. Mệnh đề order by

Chức năng: Tương tự mệnh đề order by trong SQL nhưng sử dụng Xpath.

Cú pháp:

order by expr [ascending|descending]

Lưu ý:

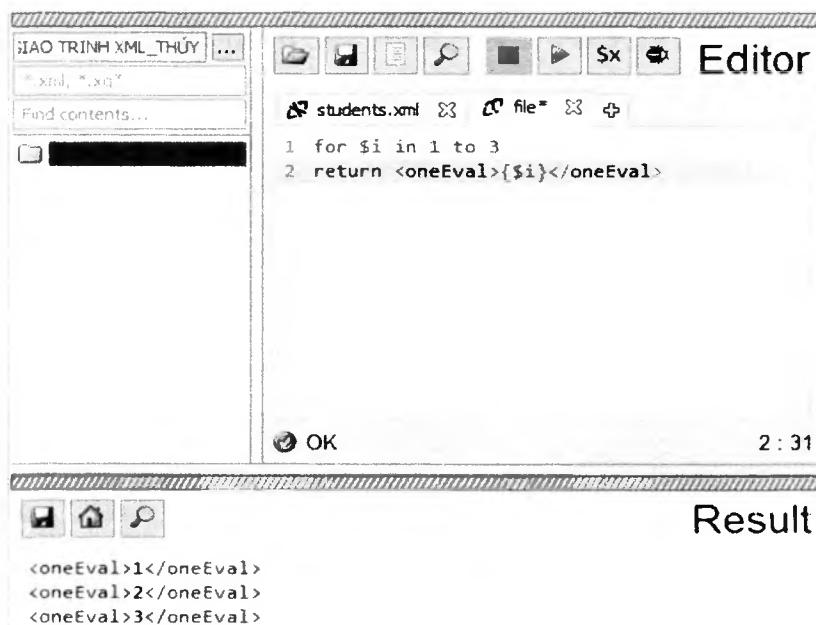
1. Một trong những lưu ý trong XQuery đó là tầm vực của biến: Biến được khai báo trong *for* hay trong *let*, sau đó được sử dụng trong *where*, *return*, hay *order by*. Tuy nhiên, biến sau khi được khai báo không được sử dụng trực tiếp ngay trong mệnh đề *for* hay *let* – theo nghĩa là không bao giờ vừa khai báo mà vừa lại dùng.
2. Ngoài ra, khi khai trùng biến trong XQuery thì kết quả có thể không như mong đợi vì biến thứ 2 có thể ghi đè biến thứ nhất, XQuery không báo lỗi trong trường hợp này.
3. Tương tự như trường hợp khai báo biến trùng như đã nêu ở trên thì XQuery vẫn thực thi mà không báo lỗi và đưa về kết quả mà chúng ta không biết được trước.

8.3. Một số ví dụ XQuery

Ví dụ 8.1:

```
for $i in 1 to 3
return <oneEval>{$i}</oneEval>
```

Câu lệnh trên đang duyệt các giá trị từ 1 đến 3, mỗi lần duyệt một số sẽ gán số đó vào biến *i*. Trong mỗi lần duyệt sẽ trả ra kết quả là một nút *oneEval* với giá trị biến *i* đang được duyệt.



Hình 8.1: Kết quả của Ví dụ 8.1 trên BaseX

Ví dụ 8.2:

```
for $i in reverse(1 to 3)
return <oneEval>{$i}</oneEval>
```

Câu lệnh trên tương tự Ví dụ 8.1 nhưng chúng ta sử dụng hàm *reverse* để duyệt ngược từ 3 đến 1.

The screenshot shows the BaseX Editor interface. On the left is a file tree with XML files like '0_Mục lục.docx', '1_Chuong 1_Tan...', '2_Chuong 2_XML...', etc. The main window shows an XML file named 'students.xml' with the following XQuery code:

```

1 for $i in reverse(1 to 3)
2 return <oneEval>{$i}</oneEval>
3

```

Below the editor is a 'Result' pane displaying the output of the query:

```

<oneEval>3</oneEval>
<oneEval>2</oneEval>
<oneEval>1</oneEval>

```

Hình 8.2: Kết quả của Ví dụ 8.2 trên BaseX

Ví dụ 8.3:

```

for $i in (1 to 10)[. mod 2 = 0]
return <oneEval>{$i}</oneEval>

```

Trong ví dụ này chúng ta đang sử dụng một biểu thức xử lý tính toán để tạo ra tập dữ liệu là các giá trị chia hết cho 2 trong đoạn từ 1 đến 10.

The screenshot shows the BaseX Editor interface. The file tree is identical to the previous one. The 'students.xml' file contains the following XQuery code:

```

1 for $i in (1 to 10)[. mod 2 = 0]
2 return <oneEval>{$i}</oneEval>

```

The 'Result' pane shows the output:

```

<oneEval>2</oneEval>
<oneEval>4</oneEval>
<oneEval>6</oneEval>
<oneEval>8</oneEval>
<oneEval>10</oneEval>

```

Hình 8.3: Kết quả của Ví dụ 8.3 trên BaseX

Ví dụ 8.4:

```

for $i in (1, 2)
for $j in ('a', 'b')
return <oneEval>i is {$i}
and j is {$j}</oneEval>

```

Trong ví dụ này chúng ta đang duyệt tương tự như lập trình là cho i chạy từ 1 đến 2, khi i đang chạy thực hiện duyệt j với hai giá trị a và b để đưa ra kết quả tổng hợp từ i và j trong kết xuất trả về.

```

<oneEval>i is 1
and j is a</oneEval>
<oneEval>i is 1
and j is b</oneEval>
<oneEval>i is 2
and j is a</oneEval>
<oneEval>i is 2
and j is b</oneEval>

```

Hình 8.4: Kết quả của Ví dụ 8.4 trên BaseX

Thay vì viết như lập trình, trong khái niệm và cú pháp XQuery đã hỗ trợ duyệt lồng nhiều cấp mà không cần sử dụng nhiều lần for mà chỉ cần khai báo các biến với các biểu thức trong mệnh đề for thì kết quả cũng tương đương về kết quả lẩn ngữ nghĩa. Ví dụ 8.4 có thể được viết lại như sau:

```

for $i in (1, 2),
  $j in ('a', 'b')
return <oneEval>i is {$i}
      and j is {$j}</oneEval>

```

Ví dụ 8.5:

```

for $a in (1 to 50),
  $b in (2, 3, 5, 7)
  where $a mod $b eq 0
  return $a * $b

```

Trong ví dụ này chúng ta áp dụng ngữ nghĩa của Ví dụ 8.4 và áp dụng thêm mệnh đề *where* để chọn những giá trị a chia hết cho b và đưa ra kết xuất là tích của a và b.

The screenshot shows the BaseX interface. The top window is titled 'Editor' and contains an XQuery script named 'students.xml':

```

1 for $a in (1 to 50),
2 $b in (2, 3, 5, 7)
3 where $a mod $b eq 0
4 return $a * $b

```

The bottom window is titled 'Result' and displays the output of the query:

```

4
9
8
25
12
18
49
16
27
20
50
24
36
28
98
45
75

```

Hình 8.5: Kết quả của Ví dụ 8.5 trên BaseX

Ví dụ 8.6:

```

for $a for $i in (1 to 3)
return (<one>{$i}</one>,
<two>{$i}</two>)

```

Trong ví dụ này chúng ta trả về nhiều hơn một kết quả cho mỗi lần duyệt của for, cụ thể là 02 node.

The screenshot shows the BaseX interface. The top window is titled 'Editor' and contains an XQuery script named 'students.xml':

```

1 for $i in (1 to 3)
2 return (<one>{$i}</one>,
3 <two>{$i}</two>)

```

The bottom window is titled 'Result' and displays the output of the query:

```

<one>1</one>
<two>1</two>
<one>2</one>
<two>2</two>
<one>3</one>
<two>3</two>

```

Hình 8.6: Kết quả của Ví dụ 8.6 trên BaseX

Ví dụ 8.7:

```
for $i at $j in 7 to 11  
return <oneEval>{$j}. {$i}</oneEval>
```

Trong ví dụ này chúng ta thực hiện lấy luôn vị trí hay thứ tự của các bộ dữ liệu trong quá trình duyệt và đưa vào trong kết quả.

The screenshot shows the BaseX Editor interface. The left pane displays a file tree with several XML files. The right pane has two tabs: 'Editor' and 'Result'. In the 'Editor' tab, the code for the XML query is shown. In the 'Result' tab, the output of the query is displayed, showing the numbers 1 through 5, each enclosed in a <oneEval> tag.

```
1 for $i at $j in 7 to 11  
2 return <oneEval>{$j}. {$i}</oneEval>
```

```
<oneEval>1. 7</oneEval>  
<oneEval>2. 8</oneEval>  
<oneEval>3. 9</oneEval>  
<oneEval>4. 10</oneEval>  
<oneEval>5. 11</oneEval>
```

Hình 8.7: Kết quả của Ví dụ 8.7 trên BaseX

Ví dụ 8.8:

```
for $s  
in (<one/>, <two/>, <three/>)  
return <out>{$s}</out>
```

Trong ví dụ này chúng ta đang duyệt trong một tập nút và trả ra tập nút mới theo khái niệm tạo ra cấu trúc lưu trữ dữ liệu mới.

The screenshot shows the BaseX Editor and Query Info interface. The left pane shows a file tree. The right pane has three main sections: 'Editor', 'Result', and 'Query Info'. The 'Editor' section contains the XML query. The 'Result' section shows the output of the query, which is a repeating set of <out> tags. The 'Query Info' section provides performance statistics and a summary of the query execution.

```
1 for $s  
2 in (<one/>, <two/>, <three/>)  
3 return <out>{$s}</out>
```

```
<out>  
  <one/>  
/<out>  
  <two/>  
/<out>  
  <three/>  
/<out>
```

Query:
for \$s in (<one/>, <two/>, <three/>)
return <out>{\$s}</out>

Result:
- Hit(s): 3 Items
- Updated: 0 Items
- Printed: 75 Bytes
- Read Locking: (none)
- Write Locking: (none)

Hình 8.8: Kết quả của Ví dụ 8.8 trên BaseX

Ví dụ 8.9:

```
avg(  
    for $x at $i in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
    where $i mod 2 = 0  
    return $x)
```

Ví dụ này hợp hàm Xpath với câu truy vấn XQuery để tính kết quả dựa trên tập kết quả của XQuery truy vấn, cụ thể là tính trung bình tất cả các giá trị mà chia hết cho 2 trong khoảng từ 1 đến 10.

The screenshot shows the BaseX Editor interface. The left pane displays a file tree with XML and XQuery files. The right pane contains the XQuery code:

```
1 avg(  
2 for $x at $i in (1, 2, 3, 4, 5, 6, 7, 8  
, 9, 10)  
3 where $i mod 2 = 0  
4 return $x)
```

Below the editor are tabs for 'Result' and 'Query Info'. The 'Query Info' tab shows compilation details:

- Compiling: - rewriting (1, 2, 3, 4, 5, 6, ...)

Hình 8.9: Kết quả của Ví dụ 8.9 trên BaseX

Ví dụ 8.10:

```
for $boy at $bo ypos in ('Simon', 'Nigel', 'David'),  
$game at $gamepos in ('pushups', 'situps')  
return <tuple n='{$bo ypos}'>  
boy {$bo ypos} is {$boy}, item {$gamepos}: {$game}  
</tuple>
```

Ví dụ này là truy vấn phức tạp phối hợp tất cả để tạo ra cấu trúc tại liệu theo ý muốn.

The screenshot shows the BaseX Editor interface. The left pane displays a file tree with XML and XQuery files. The right pane contains the XQuery code:

```
1 for $boy at $bo ypos in ('Simon', 'Nigel', 'David'),  
2 $game at $gamepos in ('pushups', 'situps')  
3 return <tuple n='{$bo ypos}'>  
4 boy {$bo ypos} is {$boy}, item {$gamepos}: {$game}  
5 </tuple>
```

Below the editor are tabs for 'Result' and 'Query Info'. The 'Query Info' tab shows compilation and optimization details:

- Compiling:
 - rewriting ("Simon", "Nigel", ...)
 - type check removed: \$bo ypos_1 as xs:integer
 - rewriting ("pushups", "situps")
 - type check removed: \$gamepos_3 as xs:integer
- Optimized Query:

```
for $boy_0 at $bo ypos_1 in ("Simon", "Nigel", ...) for  
$game_2 at $gamepos_3 in ("pushups", "situps")  
return element tuple { (attribute n { ($bo ypos_1), "#A:boy ", $bo ypos_1, " is ", $boy_0, ", item ", $  
gamepos_3, " ", $game_2) ) }
```

Hình 8.10: Kết quả của Ví dụ 8.10 trên BaseX

Trên đây là 10 ví dụ đơn giản về XQuery, tiếp theo chúng ta áp dụng XQuery trên tài liệu XML.

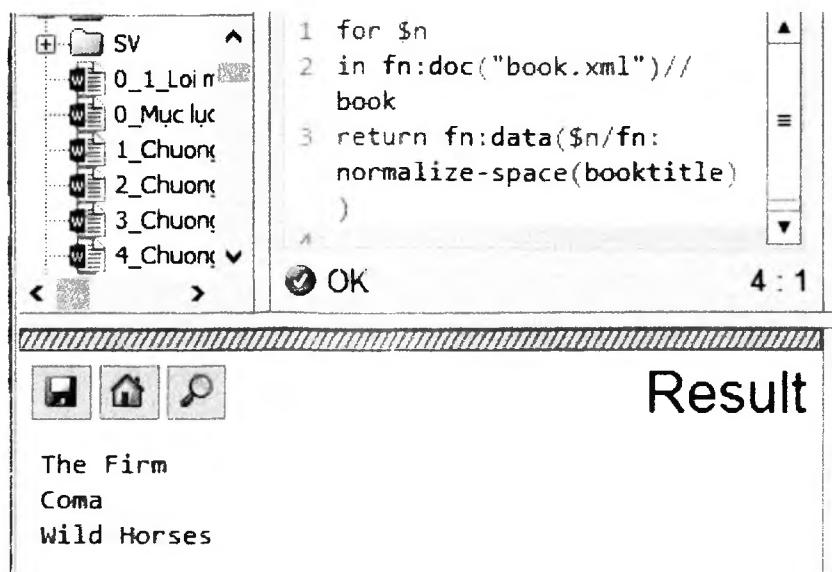
Để thực hiện được truy vấn trên tài liệu XML, chúng ta phải sao chép tập tin *.xml vào thư mục cài đặt BaseX. Ví dụ xét tập tin book.xml như sau:

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
    <book id="123" saleof="50">
        <booktitle>The Firm</booktitle>
        <author>John Grisham</author>
        <price>99</price>    </book>
    <book id="618">
        <booktitle>Coma</booktitle>
        <author>Robin Cook</author>
        <price>99.5</price>  </book>
    <book id="189">
        <booktitle>Wild Horses</booktitle>
        <author>Rebecca Moore</author>
        <price>35.5</price>
    </book>
</library>
```

Ví dụ 8.11:

```
for $n
in fn:doc("book.xml")//book
return fn:data($n/fn:normalize-space(booktitle))
```

Trong ví dụ này chúng ta duyệt các node book và n được gán đến từng node book trong quá trình duyệt và đưa ra tựa sách – booktitle trong kết quả.



Hình 8.11: Kết quả của Ví dụ 8.11 trên BaseX

```
for $n
as element(book) at $i
in fn:doc("book.xml")//book
return fn:concat($i, ' ', fn:data($n/fn:normalize-
space(booktitle)))
```

Trong ví dụ này chúng ta truy vấn kết hợp với lấy vị trí để đưa ra kết quả:

The screenshot shows a software interface with a sidebar on the left containing a file tree with files like 'SV', '0_1_Loi mo dau.', '0_Mục lục.docx', etc. On the right, there is an XSLT code editor with the following content:

```
1 for $n
2 as element(book) at $i
3 in fn:doc("book.xml")//book
4 return fn:concat($i, '. ', fn:data($n
5 /fn:normalize-space(booktitle)))
```

At the bottom right of the code editor is the number '3 : 9'. Below the code editor is a toolbar with icons for file operations.

Result

- 1. The Firm
- 2. Coma
- 3. Wild Horses

Hình 8.12: Kết hợp lấy vị trí để đưa ra kết quả

```
for $n
as element(book) at $i
in fn:doc("book.xml")//book
return fn:concat($i, '. ', fn:data($n/fn:normalize-
space(booktitle)))
```

Trong ví dụ này chúng ta truy vấn kết hợp với lấy vị trí để đưa ra kết quả:

The screenshot shows a software interface with a sidebar on the left containing a file tree with files like 'SV', '0_1_Loi mo dau.', '0_Mục lục.docx', etc. On the right, there is an XSLT code editor with the same XSLT code as in Figure 8.12:

```
1 for $n
2 as element(book) at $i
3 in fn:doc("book.xml")//book
4 return fn:concat($i, '. ', fn:data($n
5 /fn:normalize-space(booktitle)))
```

At the bottom right of the code editor is the number '3 : 9'. Below the code editor is a toolbar with icons for file operations.

Result

- 1. The Firm
- 2. Coma
- 3. Wild Horses

Hình 8.13: Kết hợp lấy vị trí để đưa ra kết quả

```
for $b in fn:doc("book.xml")//book[price < 99.5]
order by $b/price descending
return $b
```

Trong ví dụ này chúng ta truy vấn kết hợp Xpath có điều kiện để lọc dữ liệu ngay trong mệnh đề *for* và thực hiện sắp xếp kết quả trả về theo price giảm dần:

The screenshot shows the BaseX interface. On the left is a tree view of XML documents: 'HOC XML' (with 'SV' expanded) and 'book.xml'. The right pane displays XQuery code and its execution results.

```

1 for $b in fn:doc("book.xml")
  ")//book[price < 99.5]
2 order by $b/price
3 descending
4 return $b
    OK
  3 : 10

```

Result

```

<book id="123" saleof="50">
  <booktitle>The Firm</booktitle>
  <author>John Grisham</author>
  <price>99</price>
</book>
<book id="189">
  <booktitle>Wild Horses</booktitle>
  <author>Rebecca Moore</author>
  <price>35.5</price>
</book>

```

Hình 8.14: Kết quả của việc lọc dữ liệu trong mệnh đề *for*

Ví dụ 8.12:

```

let $i := (1 to 3)
return <oneEval>{$i}</oneEval>

```

Trong ví dụ này chúng ta đang chỉ sử dụng mệnh đề *let*, do vậy *let* chỉ thực hiện một lần nên cho dù duyệt giá trị từ 1 đến 3 nhưng *let* không duyệt từ giá trị như *for* mà *let* kết hợp với cả ba giá trị và đưa ra luôn kết xuất.

The screenshot shows the BaseX interface. On the left is a tree view of XML documents: 'HOC XML' (with 'SV' expanded) and 'book.xml'. The right pane displays XQuery code and its execution results.

```

1 let $i := (1 to 3)
2 return <oneEval>{$i}</oneEval>
    OK
  2 : 31

```

Result

```

<oneEval>1 2 3</oneEval>

```

Hình 8.15: Kết quả của Ví dụ 8.12 trên BaseX

Ví dụ 8.13:

```

let $salary as xs:decimal := 5.0
return $salary * 2

```

Trong ví dụ này chúng ta thực hiện tương tự khái niệm khai báo biến với kiểu thập phân, gán trị khởi tại và đưa ra kết quả với biến nhân 2.

Hình 8.16: Kết quả của Ví dụ 8.13 trên BaseX

Ví dụ 8.14: Trong ví dụ này, chúng ta sẽ thực hiện các câu lệnh truy vấn trên một tài liệu XML phức tạp hơn, *catalog.xml*, tài liệu này như sau:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
    <product dept="WMN">
        <number>557</number>
        <name language="en">Fleece Pullover</name>
        <colorChoices>navy black</colorChoices>
    </product>
    <product dept="ACC">
        <number>563</number>
        <name language="en">Floppy Sun Hat</name>
    </product>
    <product dept="ACC">
        <number>443</number>
        <name language="en">Deluxe Travel Bag</name>
    </product>
    <product dept="MEN">
        <number>784</number>
        <name language="en">Cotton Dress Shirt</name>
        <colorChoices>white gray</colorChoices>
        <desc>Our <i>favourite</i> shirt! </desc>
    </product>
</catalog>
```

Câu lệnh truy vấn như sau:

```
for $prod in doc('catalog.xml')//product
let $prodDept := $prod/@dept
where $prodDept = 'ACC' or $prodDept = 'WMN'
return $prod/name
```

Trong ví dụ này chúng ta thực hiện duyệt *product* trong tài liệu và khai báo biến *prodDept* để dùng biến đó trong mệnh đề *where* để lọc dữ liệu. Trong một lần duyệt, nếu dữ liệu nào thỏa thì sẽ trả ra *name* của nút *product*.

The screenshot shows the BaseX interface. On the left, there's a file tree with XML files like 'HOC XML Tieng viet.xml', 'SV', '0_1_Loi moi dau.docx', '0_Muc luc.docx (28 K)', '1_Chuong 1_Tong quan', and '2_Chuong 2_XML hop'. In the center, XQuery code is displayed:

```

1 for $prod in doc('catalog.xml')//product
2 let $prodDept := $prod/@dept
3 where $prodDept = 'ACC' or $prodDept = 'WMN'
4 return $prod/name
5

```

An 'OK' button is at the bottom left, and the time '3:45' is in the top right.

Result

```

<name language="en">Fleece Pullover</name>
<name language="en">Floppy Sun Hat</name>
<name language="en">Deluxe Travel Bag</name>

```

Hình 8.17: Kết quả của Ví dụ 8.14 trên BaseX

Để đơn giản câu lệnh, chúng ta có thể khai báo biến thông qua câu lệnh let để làm tường minh câu lệnh với cùng ngữ nghĩa và kết quả như Hình 8.17.

```

let $doc := doc('catalog.xml')
for $prod in $doc//product
let $prodDept := $prod/@dept
let $prodName := $prod/name
where $prodDept = 'ACC' or $prodDept = 'WMN'
return $prodName

```

Ví dụ 8.15:

```

for $prod in doc('catalog.xml')//product
let $prodDept := $prod/@dept
where $prod/number > 100
and starts-with($prod/name, 'F')
and exists($prod/colorChoices)
and ($prodDept = 'ACC' or $prodDept = 'WMN')
return $prod

```

Ví dụ này xử lý với điều kiện phức tạp kết hợp and tương tự như truy vấn với mệnh đề điều kiện phức tạp trong SQL.

The screenshot shows the BaseX interface. On the left, there's a file tree with XML files like 'HOC XML Tieng viet.xml', 'SV', '0_1_Loi moi dau.docx', '0_Muc luc.docx (28 K)', '1_Chuong 1_Tong quan', '2_Chuong 2_XML hop', '3_Chuong 3_DTD va XML Schema', '4_Chuong 4_Luoc de', and '5_Chuong 5_Truy van'. In the center, XQuery code is displayed:

```

1 for $prod in doc('catalog.xml')//product
2 let $prodDept := $prod/@dept
3 where $prod/number > 100
4 and starts-with($prod/name, 'F')
5 and exists($prod/colorChoices)
6 and ($prodDept = 'ACC' or $prodDept = 'WMN')
7 return $prod
8

```

An 'OK' button is at the bottom left, and the time '6:45' is in the top right.

Result

```

<product dept="WMN">
<number>557</number>
<name language="en">Fleece Pullover</name>
<colorChoices>navy black</colorChoices>
</product>

```

Hình 8.18: Kết quả của Ví dụ 8.15 trên BaseX

```

let $prods := doc('catalog.xml')//product
for $d in distinct-values($prods/@dept),
$n in distinct-values($prods[@dept = $d]/number)
return <result dept='{$d}' number='{$n}'/>

```

Trong ví dụ này chúng ta đang thực hiện truy vấn dữ liệu nhưng loại bỏ những giá trị trùng trong kết quả tương tự sử dụng *distinct* trong câu lệnh SQL

The screenshot shows the Oxygen XML Editor interface. On the left, there's a file tree with a folder 'HOC XML Tieng viet' containing 'SV' and several documents like '0_1_Loi moi dau.doc', '0_Mục lục.docx', etc. In the center, XQuery code is displayed:

```

1 let $prods := doc('catalog.xml')//product
2 for $d in distinct-values($prods/@dept),
3 $n in distinct-values($prods[@dept = $d]/
4 number)
5 return <result dept='{$d}' number='{$n}'/>

```

Below the code is an 'OK' button and a timestamp '4 : 43'. To the right, a 'Result' pane displays the output XML:

```

<result dept="WMN" number="557"/>
<result dept="ACC" number="563"/>
<result dept="ACC" number="443"/>
<result dept="MEN" number="784"/>

```

Hình 8.19: Kết quả khi sử dụng *distinct*

Ví dụ 8.16: Trong ví dụ này, chúng ta sẽ thực hiện kết hợp thêm một tài liệu XML (*order.xml*) với Ví dụ 8.15 để thể hiện khái niệm tương tự như *join* bảng trong câu lệnh SQL.

```

<?xml version="1.0" encoding="UTF-8"?>
<order num="00299432" date="2017-05-15" cus="0221A">
    <item      dept="WMN"      num="557"      quantity="1"
color="navy"/>
    <item dept="ACC" num="563" quantity="1"/>
    <item dept="ACC" num="443" quantity="2"/>
    <item dept="MEN" num="784" quantity="1"
color="white"/>
    <item      dept="MEN"      num="784"      quantity="1"
color="gray"/>
    <item dept="WMN" num="557" quantity="1"
color="black"/>
</order>

```

Câu lệnh truy vấn như sau:

```

for $item in doc('order.xml')//item,
$product   in   doc('catalog.xml')//product[number =
$item/@num]
    return <item num='{$item/@num}'
name='{$product/name}'
quan='{$item/@quantity}'/>

```

Ví dụ này đang kết giữa bảng *catalog* và *order* dựa trên giá trị *number* và đưa ra kết xuất là lấy thông tin 02 tài liệu phối hợp lại với nhau.

```

for $item in doc('order.xml')//item,
$product in doc('catalog.xml')//product[number = $item/@num]
return <item num='{$item/@num}' name='{$product/name}' quan='{$item/@quantity}'/>

```

Result

```

<item num="557" name="Fleece Pullover" quan="1"/>
<item num="563" name="Floppy Sun Hat" quan="1"/>
<item num="443" name="Deluxe Travel Bag" quan="2"/>
<item num="784" name="Cotton Dress Shirt" quan="1"/>
<item num="784" name="Cotton Dress Shirt" quan="1"/>
<item num="557" name="Fleece Pullover" quan="1"/>

```

Compiling:

- pre-evaluating doc("order.xml")
- rewriting descendant-or-step(s)
- converting descendant::*

Hình 8.20: Kết quả của Ví dụ 8.16 trên BaseX

```

for $item in doc('order.xml')//item,
$product in doc('catalog.xml')//product
where $item/@num = $product/number
return <item num='{$item/@num}' name='{$product/name}' quan='{$item/@quantity}'/>

```

Ví dụ này cho kết quả tương tự như Hình 8.20 nhưng thay vì viết câu lệnh điều kiện sử dụng XPath thì câu lệnh này đưa điều kiện thành mệnh đề *Where* để thể hiện tính rõ ràng trong câu lệnh XQuery.

8.4. Một số khái niệm XQuery nâng cao

Thay vì viết lệnh trực tiếp trong quá trình xử lý, XQuery cho phép viết ra các file riêng có đuôi dạng *xq* như các tập tin batch lệnh của SQL để có thể thực hiện và đưa vào các chương trình thực hiện linh động hơn. Các file thường bắt đầu với phần prolog (ngữ nghĩa như XML) chứa thông tin cho phép XQuery xử lý như khai báo version XQuery, khai báo hàm, biến, khai báo namespace, import các file XQuery khác vào. Phần khai báo version cụ thể như sau:

```
XQuery version "1.0" encoding "utf-8";
```

Khai báo namespace với cú pháp:

```
declare namespace prefix="URI namespace"
```

Ví dụ:

```
declare namespace fobo = "http://www.fromoldbooks.org/ns/";
```

Mặc định XQuery có một số namespace mặc định mà không cần khai báo cụ thể:

§ *xml* = <http://www.w3.org/XML/1998/namespaces> - hỗ trợ XML

§ *xs* = <http://www.w3.org/2001/XMLSchema> - hỗ trợ XML schema

§ *xsi* = <http://www.w3.org/2001/XMLSchema-instance> -

hỗ trợ XML schema instance

§ *fn* = <http://www.w3.org/2005/xpath-functions> - hỗ trợ function xpath

local = <http://www.w3.org/2005/XQuery-local-functions> - hỗ trợ function XQuery

Hỗ trợ import module với 02 cú pháp:

import module namespace prefix="URI namespace" at "file/url";

import module "file|url"; - sử dụng chỉ cho file mà không cần quan tâm namespace

Khai báo biến với cú pháp:

declare variable \$name [as type] := [external] value;

Khai báo biến tương tự như khai báo biến trong XSL:

Ví dụ:

```
declare variable $socks := "black";
declare variable $sockprice as xs:decimal := 3.6;
declare variable $argyle as element(*) :=
    <sock>argyle</sock>;
declare variable $places as xs:string :=
    doc("places.xml");
```

Khai báo functions: **declare function func_Name (param_list) as sequence_type {**

<function body> }

Phần function body, có thể gọi function khác, câu lệnh XPath, XQuery, FLWOR, ... Các thành phần khác tương tự như tạo function trong lập trình.

Phần thứ 2 sau prolog là nội dung truy vấn chính thức hay câu lệnh XQuery. Ngoài ra, XQuery còn hỗ trợ chia module và lưu trữ dưới file có đuôi *xqm*.

Ví dụ 8.17:

```
declare function local:discountPrice(
    $price as xs:decimal?, $discount as xs:decimal?,
    $maxDiscountPct as xs:integer?) as xs:double? {
    let $newDiscount:= if ($discount) then $discount else 0
    let $maxDiscount := if ($maxDiscountPct)
        then ($price * $maxDiscountPct) div 100 else 0
    let $actualDiscount := min(($maxDiscount, $discount))
    return ($price - $actualDiscount);
    for $prod in doc('catalog.xml')//product
        return local:discountPrice($prod/number,
            $prod/discount, 15)
```

Ví dụ trên chúng ta đang khai báo hàm ở phần in đậm và sau đó sử dụng hàm ngay trong truy vấn. Kết quả chúng ta nhận được như Hình 8.21.

The screenshot shows the BaseX XML editor interface. On the left, there's a tree view of XML files under a folder named 'HOC XML Tieng viet'. The 'SV' folder contains several files like '0_1_Loi moi dau.docx', '0_Muc lục.docx', etc. In the main window, an XQuery script is displayed:

```

1 declare function local:discountPrice(
2   $price as xs:decimal?, $discount as xs:decimal?,
3   $maxDiscountPct as xs:integer?) as xs:double? {
4   let $newDiscount := if ($discount) then $discount else 0
5   let $maxDiscount := if ($maxDiscountPct)
6     then ($price * $maxDiscountPct) div 100 else 0
7   let $actualDiscount := min(($maxDiscount, $discount))
8   return ($price - $actualDiscount)
9 };
10 for $prod in doc('catalog.xml')//product
11 return local:discountPrice($prod/number, $prod/discount, 15)

```

At the bottom right of the code area, there's an 'OK' button. Below the editor, there are three small icons: a document, a house, and a magnifying glass. To the right of these icons, the word 'Resi' is partially visible.

On the far left, below the tree view, there are some numerical values: 473.45, 478.55, 376.55, and 666.4.

Hình 8.21: Kết quả của Ví dụ 8.17 trên BaseX

Tóm lại, XQuery là một ngôn ngữ được chuẩn hóa của W3C để truy vấn tài liệu và dữ liệu XML. XQuery và XSLT đều được xây dựng trên XPath; XSLT là một ngôn ngữ cú pháp XML chứa các biểu thức XPath bên trong các thuộc tính và XQuery sử dụng cú pháp XPath mở rộng với nhiều từ khóa.

Hai khối xây dựng quan trọng nhất của XQuery là biểu thức FLWOR và các chức năng. XQuery FLWOR là viết tắt của for-let-where-order by-return. Các hàm được xác định bởi người sử dụng có thể được đệ quy, và có thể được tập hợp lại cùng với các biến "chỉ đọc" do người dùng xác định thành các tập tin thư viện riêng biệt gọi là các mô-đun.

BÀI TẬP CHƯƠNG 8

- 1) Viết một biểu thức truy vấn để lấy chuỗi các con số từ 1 đến 100 và tạo ra một danh sách là bình phương các giá trị tương ứng của chúng.
- 2) Sử dụng tập tin *people.xml* (Chương 7), dựa vào ngày sinh và ngày mất, tính tuổi thọ của từng người. Hiển thị những người người sống lâu hơn 70 tuổi. Sắp xếp danh sách theo tuổi thọ giảm dần (người sống lâu nhất đứng đầu đầu danh sách, ...)
- 3) Sử dụng tập tin *khachsan.xml* (bài tập 7a, chương 4), viết câu lệnh XQuery thực hiện các lệnh sau:
 - a. Hiển thị các dịch vụ của khách sạn có đơn giá lớn hơn 100.000
 - b. Hiển thị danh sách khách hàng có địa chỉ chứa tên ‘HCM’
 - c. Sắp xếp kết quả hiển thị của câu b) theo tên khách hàng
- 4) Tạo một dãy các thuộc tính được chuyển đổi theo mẫu sau:

```
<property idProperty="PROPERTY-ID">  
  <owner>OWNER-NAME</owner>  
  <address>STRUCTURED-PROPERTY-ADDRESS</address>  
</property>
```

...

Lưu ý: Sử dụng hàm tạo trực tiếp và truy vấn lồng nhau

- 5) Sử dụng XQuery tạo một đoạn XHTML với một bảng thuộc tính như sau:

```
<table>  
  <tr>  
    <th>Id</th><th>Name</th><th>Features</th>  
  </tr>  
  ...  
  <tr>  
    <td>PROPERTY-ID</td>  
    <td>PROPERTY-NAME</td>  
    <td>COMMA-SEPARATED-LIST-OF-FEATURES</td>  
  </tr>  
  ...  
</table>
```

Chương 9

XML VÀ CƠ SỞ DỮ LIỆU QUAN HỆ

9.1. Cơ sở dữ liệu quan hệ và XML

Nếu như trước kia, chúng ta thường có hai lựa chọn chính khi quyết định nơi lưu trữ dữ liệu của chúng ta: Hệ quản trị cơ sở dữ liệu (CSDL) quan hệ truyền thống như Oracle, SQL Server, DB2, Access; hoặc sử dụng phần mềm mã nguồn mở được sử dụng phổ biến là MySQL. Ngày nay, chúng ta có thêm một lựa chọn khác, đó là sử dụng XML.

Ưu điểm của CSDL quan hệ là việc lưu trữ dữ liệu dưới dạng các bảng biểu, trực quan và dễ phân biệt. Tuy nhiên, nhược điểm về hiệu suất khi chúng ta cần là phải kết hợp nhiều bảng với nhau để truy xuất dữ liệu.

XML có lợi thế là lưu trữ tất cả dữ liệu trong cùng tài liệu, cho phép các dữ liệu lồng nhau và các tài liệu không thể dễ dàng bị phá vỡ. Để phát huy ưu thế của CSDL và XML, chúng ta cần một lai ghép của hai hệ thống này: Một hệ thống có thể lưu trữ dữ liệu dạng bảng cùng với các tài liệu XML, cho phép người dùng truy vấn và sửa đổi XML cũng như thực hiện các hoạt động thông thường trên CSDL quan hệ.

Những hệ quản trị CSDL truyền thống khác nhau có các bộ tính năng khác nhau, nhưng nhìn chung chúng có hai điểm chung:

- Thứ nhất, chúng sử dụng một ngôn ngữ đặc biệt gọi là Structured Query Language (SQL) để truy vấn dữ liệu.
- Thứ hai, dữ liệu trong các hệ quản trị CSDL này có thể được chia nhỏ và lưu trữ dưới dạng bảng, trong đó các mục dữ liệu thường được biểu diễn dưới dạng hàng trong một bảng. Các thuộc tính khác nhau của các mục này được thể hiện bằng các trường khác nhau hoặc các cột của các hàng này.

Tiếp theo, chúng ta sẽ xem xét một số ứng dụng triển khai các chức năng của XML.

9.2. Sử dụng MySQL với XML

MySQL là một trong những CSDL mã nguồn mở hàng đầu hiện nay. MySQL được sử dụng rất nhiều trong việc xây dựng các website dựa trên PHP và Java. MySQL có hỗ trợ một vài tính năng XML. Trước khi xem xét các tính năng này, chúng ta tiến hành cài đặt MySQL.

9.2.1. Cài đặt MySQL

Chúng ta có thể tải xuống MySQL từ địa chỉ <https://www.mysql.com/downloads/>. Chúng ta chọn mục Community và chọn download MySQL Community Server và chọn phiên bản 5.7.18 (hoặc mới hơn nếu có). Các phiên bản ổn định của MySQL có sẵn cho cả hệ điều hành Windows và Unix/Linux. Chọn phiên bản phù hợp nhất cho nền tảng hệ điều hành của chúng ta và làm theo hướng dẫn.

Đối với chương này, chúng ta cần phải cài đặt máy chủ và các chương trình khách hàng. Chúng ta cũng có thể cài đặt các công cụ front-end, cung cấp

cho chúng ta một giao diện người dùng đồ họa. Tuy nhiên, đối với những ví dụ này chúng ta chỉ có thể sử dụng trình soạn thảo giao diện lệnh của Windows.

Chúng ta có thể sử dụng một công cụ GUI để tương tác với MySQL, nhưng nếu chúng ta thực sự muốn hiểu những gì đang xảy ra đằng sau thì sử dụng tiện ích dòng lệnh mysql là tốt nhất. Chúng ta mở cửa sổ lệnh Windows và điều hướng đến thư mục *bin* của tiến trình cài đặt, ví dụ, *C:\Program Files\MySQL\MySQL Server 5.7\bin*. Sau đó gõ *mysql -u root -p*. Nếu mọi thứ hoạt động bình thường, chúng ta sẽ thấy dấu nhắc mysql hỏi mật khẩu chúng ta đã chọn trong quá trình cài đặt.

```
C:\Program Files\MySQL\MySQL Server 5.7\bin>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.18-log MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Hình 9.1: Tiện ích dòng lệnh của MySQL

Tiếp theo sau đây, chúng ta sẽ tạo một CSDL mới trên MySQL và thêm thông tin cho CSDL.

9.2.2. Tạo và nạp một CSDL MySQL

Trước khi chúng ta có thể thêm thông tin vào MySQL, chúng ta phải tạo một CSDL. CSDL hoạt động như một vùng chứa trong đó chúng ta nhóm các thông tin liên quan đến một dự án.

1. Để tạo một cơ sở dữ liệu có tên Blog với UTF-8 như là một bộ ký tự, chúng ta nhập vào các lệnh sau:

```
mysql> create database Blog    DEFAULT CHARACTER SET
'utf8';
Query OK, 1 row affected (0.00 sec)
mysql>
```

2. Di chuyển vào CSDL vừa được tạo bằng cách gõ lệnh sau:

```
mysql> use Blog;
Database changed
mysql>
```

Sự khác biệt lớn nhất giữa CSDL XML và CSDL quan hệ là CSDL quan hệ có cấu trúc cao. Một CSDL XML có thể tìm hiểu cấu trúc của các tài liệu của chúng ta khi chúng ta tải chúng, mà không cần bất kỳ định nghĩa trước. Điều này là không thể với một CSDL quan hệ. Trong một CSDL quan hệ, thông tin được lưu trữ trong bảng với các hàng và cột. Các bảng này tương tự

như bảng bảng tính, ngoại trừ tên và loại cột cần phải được xác định trước khi chúng ta có thể sử dụng chúng.

3. Tạo một trong các bảng này để chứa thông tin cần thiết cho các bài tập trong chương này. Gọi tên nó là *BlogPost* và để đơn giản, cho nó chỉ có hai cột:

- Một cột có tên *PostId* sẽ được sử dụng như một khóa chính khi chúng ta muốn lấy một mục nhập blog cụ thể.

- Một cột có tên *Post* để giữ mục nhập blog trong XML

Tất nhiên, đây là một giàn đồ rất nhỏ. Chúng ta có thể muốn thêm nhiều cột vào bảng này để đơn giản hóa và tối ưu hóa các truy vấn cụ thể. Để tạo ra bảng này, chúng ta nhập lệnh như sau:

```
mysql> create table BlogPost(PostId INT PRIMARY KEY, Post LONGTEXT);  
Query OK, 0 row affected (0.29 sec)  
mysql>
```

4. CSDL bây giờ đã sẵn sàng để chấp nhận các mục blog của chúng ta. Trong một ứng dụng thực tế, các mục này sẽ được bổ sung bởi một ứng dụng web tốt, nhưng chương này vẫn tiếp tục sử dụng tiện ích dòng lệnh mysql để bổ sung chúng. Trong SQL, chúng ta thêm thông tin thông qua một câu lệnh chèn. Nhập một vài mục blog của riêng chúng ta theo mẫu này:

```
mysql> INSERT BlogPost (PostId, Post) SELECT 1,'<post  
xmlns:x= "http://www.w3.org/1999/xhtml" id="1"  
author="Joe Fawcett" dateCreated="2011-09-18">  
    <title>A New Book</title>  
    <body> <x:p>  
        <x:b>I've been asked to co-author a new edition  
of <x:a href="http://www.wrox.com/WileyCDA/WroxTitle/  
productCd-470114878.html"> Beginning XML </x:a>  
by Wrox </x:b> </x:p>  
        <x:p>It's incredible how much has changed since  
the book was published nearly five years ago. XML  
is now a bedrock of many systems, contrarily you see  
less of it than previously as it's buried under more  
layers.</x:p>  
        <x:p>There are also many places where it has  
stopped being an automatic choice for data transfer,  
JSON has become a popular replacement where the  
data is to be consumed directly by a JavaScript  
engine such as in a browser.</x:p>  
        <x:p>The new edition should be finished towards  
the end of the year and be published early in  
2012.</x:p>  
</body> </post>';
```

Sau khi Enter, kết quả: Query OK, 1 row affected (0.17 sec). Chúng ta có thể nhập thêm bản ghi thứ 2, 3, và 4 cho BlogPost bằng cách tương tự như trên.

9.2.3. Truy vấn MySQL

Bây giờ chúng ta đã có dữ liệu cho CSDL Blog. Bởi vì MySQL là một CSDL SQL, vì vậy chúng ta có thể sử dụng tất cả sức mạnh của SQL để truy vấn nội dung của CSDL. Để hiển thị tất cả các mục, chỉ cần gõ như sau:

```
mysql> select * from BlogPost;
```

Kết quả của truy vấn trên là tất cả những dữ liệu vừa chèn vào BlogPosst, nếu chúng ta muốn một cái gì đó súc tích hơn, chúng ta có thể lựa chọn các ký tự đầu tiên của mỗi mục:

```
mysql> select PostId, substring(Post, 1, 60) FROM BlogPost;
```

PostId	substring(Post, 1, 60)
1	<post xmlns:x="http://www.w3.org/1999/xhtml" id="1" author=
2	<post xmlns:x="http://www.w3.org/1999/xhtml" id="2" autho
3	<post xmlns:x="http://www.w3.org/1999/xhtml" id="3" author="
4	<post xmlns:x="http://www.w3.org/1999/xhtml" id="2" author="

```
4 rows in set (0.00 sec)
```

Hoặc, nếu chúng ta chỉ muốn biết số lượng bản ghi:

```
mysql> select COUNT(*) FROM BlogPost;
```

COUNT(*)
4

```
1 row in set (0.00 sec)
```

Đây là SQL thuần túy và có thể được thực hiện với bất kỳ CSDL SQL nào mà không có hỗ trợ XML. Nhưng giả sử nếu chúng ta muốn hiển thị nội dung của phần tử tiêu đề? Lúc này chúng ta cần chức năng hỗ trợ XML trong MySQL, đó là hàm *ExtractValue*.

Hàm *ExtractValue* đánh giá kết quả của biểu thức XPath trên một đoạn XML được truyền vào dưới dạng chuỗi.

```
mysql> SELECT PostId, ExtractValue(Post, '/post/title') Title FROM BlogPost;
```

PostId	Title
1	A New Book
2	A New Book
3	Size of the Solar System
4	Go, Neutrino, Go!

```
4 rows in set (0.00 sec)
```

Chúng ta không bị giới hạn sử dụng hàm *ExtractValue* trong câu lệnh SELECT, chúng ta có thể sử dụng nó trong mệnh đề WHERE. Ví dụ, để lấy ID của mục nhập blog với một tiêu đề cụ thể, chúng ta sử dụng như sau:

```
mysql> SELECT PostId FROM BlogPost WHERE ExtractValue(Post, '/post/title')='A New Book';
+-----+
| PostId |
+-----+
|      1 |
|      2 |
+-----+
2 rows in set (0.06 sec)
```

Nếu chúng ta đã quen với hành vi XPath là dịch các phần tử thành các chuỗi bằng cách nối các nút văn bản từ tất cả các con cháu của chúng, chúng ta có thể cho rằng *ExtractValue* sẽ làm như vậy, nhưng thực tế không phải vậy: *ExtractValue* chỉ nối các nút văn bản trực tiếp nhung trong các phần tử. Trong trường hợp này, các nút văn bản duy nhất là con trực tiếp từ các phần tử mô tả là khoảng trắng. Để có chức năng như XPath, chúng ta cần phải nêu rõ trong *ExtractValue* rằng chúng ta muốn các nút văn bản ở bất kỳ mức nào:

```
mysql> SELECT PostId, ExtractValue(Post, 'post/body//text()') Body FROM BlogPost;
```

Làm thế nào chúng ta chọn các mục có chứa hình ảnh? Trong XPath, chúng ta sử dụng *//img* trực tiếp trong câu lệnh và điều này sẽ được coi là đúng khi và chỉ khi có ít nhất một phần tử ** ở đâu đó trong tài liệu. Nếu chúng ta đã quen thuộc với XPath, chúng ta có thể viết như thế này:

```
mysql> SELECT PostId, ExtractValue(Post, '/post/title') Title FROM BlogPost WHERE ExtractValue(Post, '//x:img') != '';
```

Empty set (0.00 sec)

Tuy nhiên, câu lệnh trên không được thực hiện vì các phần tử ** trống rỗng: chúng không có bất kỳ nút văn bản con nào, và *ExtractValue* chuyển chúng thành các chuỗi rỗng. Để làm cho truy vấn đó hoạt động, chúng ta cần phải chọn một nút mà sẽ có một giá trị (như *//x:img/@src*) hoặc đếm số phần tử ** và kiểm tra kết quả lớn hơn không. Phương pháp này được hiển thị trong đoạn mã sau đây:

```
mysql> SELECT PostId, ExtractValue(Post, '/post/title') Title From BlogPost where ExtractValue(Post, '//x:img/@src') != ' ';
+-----+-----+
| PostId | Title          |
+-----+-----+
|      3 | Size of the Solar System |
+-----+-----+
1 row in set (0.03 sec)
```

```
mysql> SELECT PostId, ExtractValue(Post, '/post/title') Title From BlogPost Where ExtractValue(Post, 'count(/x:img)') >0;
+-----+-----+
```

```

| PostId | Title
+-----+-----+
|      3 | Size of the Solar System |
+-----+-----+
1 row in set (0.06 sec)

```

Tuy nhiên, hầu hết các hàm xử lý chuỗi của XPath không được thực hiện trên MySQL. Ví dụ: nếu chúng ta muốn tìm mục có liên kết đến các URI từ www.wrox.com, thông thường chúng ta viết như sau:

```

mysql> SELECT PostId, ExtractValue(Post, '/post/title')
Title          From          BlogPost          Where
ExtractValue(Post, 'count(//x:a[starts-
with(@href, "http://www. wrox.com ")]))') >0;
ERROR 1105 (HY000): XPATH syntax error:
'(@href, "http://www. wrox.com "))'

```

Thật không may, hàm *starts-with* không được thực hiện, vì vậy chúng ta sẽ nhận được một thông báo lỗi. Nó sẽ báo rằng có một lỗi cú pháp, lúc này chúng ta cần phải sử dụng SQL để làm những gì chúng ta không thể làm với XPath:

```

mysql> SELECT PostId, ExtractValue(Post, '/post/title')
Title From BlogPost Where ExtractValue(Post, '//x:a/@href')
Like 'http://www.wrox.com/%';
+-----+-----+
| PostId | Title
+-----+-----+
|      1 | A New Book
|      2 | A New Book
+-----+-----+
2 rows in set (0.00 sec)

```

Bây giờ chúng ta đã biết làm thế nào sử dụng được các chức năng truy vấn dữ liệu XML trên MySQL, tiếp theo chúng ta cần phải cập nhật một tài liệu XML.

9.2.4. Cập nhật XML trong MySQL

Hàm XML thứ hai được MySQL 5.7 giới thiệu được gọi là *UpdateXML*. Giống như bất kỳ hàm SQL nào, *UpdateXML* không thực hiện cập nhật cơ sở dữ liệu, nhưng nó rất tiện dụng khi chúng ta sử dụng nó trong báo cáo cập nhật.

UpdateXML có ba tham số:

- Một chuỗi chứa một tài liệu XML
- Một biểu thức XPath trả về một phần tử
- Một đoạn XML

UpdateXML lấy tài liệu XML, tìm kiếm nội dung được chỉ ra bởi biểu thức XPath được chuyển như là tham số thứ hai và thay thế nó bằng đoạn XML được chuyển như là tham số thứ ba. Sau đó trả về XML mới được tạo bởi hàm như một chuỗi.

Ví dụ, để thay đổi *title* của mục blog thứ ba, chúng ta sử dụng lệnh sau:

```
mysql> UPDATE BlogPost Set Post =
```

```

UpdateXml(Post,'/post/title','<title>Size of the Solar
System</title>') Where PostId=3;

Query OK, 0 rows affected (0.07 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> SELECT PostId, ExtractValue(Post,'/post/title')
Title From BlogPost;
+-----+-----+
| PostId | Title          |
+-----+-----+
|      1 | A New Book    |
|      2 | A New Book    |
|      3 | Size of the Solar System |
|      4 | Go, Neutrino, Go! |
+-----+
4 rows in set (0.00 sec)

```

Hàm này rõ ràng là tiện dụng trong tình huống này, nhưng lưu ý rằng biểu thức XPath phải trả đến một phần tử. Điều này có nghĩa là mức độ chi tiết cập nhật ở cấp phần tử, do đó, nếu chúng ta muốn cập nhật giá trị thuộc tính, chúng ta sẽ không thực hiện được.

9.3. Sử dụng SQL Server với XML

SQL Server của Microsoft đã có chức năng XML kể từ phiên bản 2000. Phiên bản 2005 được bổ sung thêm nhiều hơn, nhưng kể từ đó không có nhiều thay đổi. Các ví dụ trong phần này có thể làm việc được với bất kỳ phiên bản nào từ năm 2005 trở lên.

9.3.1. Cài đặt SQL Server

Đối với những ví dụ này, chúng ta sẽ sử dụng phiên bản miễn phí của SQL Server. Chúng ta có thể tải xuống từ <https://www.microsoft.com/en-us/sql-server/default.aspx>. Chúng ta sẽ cần phải chọn tùy chọn thích hợp tùy thuộc vào việc chúng ta cần phiên bản 32 hoặc 64 bit. Đảm bảo chúng ta chọn cài đặt đi kèm với các công cụ dành cho nhà phát triển để chúng ta có thể sử dụng SQL Server Management Studio để chạy các ví dụ.

Chúng ta cũng sẽ cần phải tải xuống một CSDL mẫu để làm việc với. Cơ sở dữ liệu OLTP AdventureWorks có tại <http://msftdbprodsamples.codeplex.com/releases/view/125550> mô tả về một công ty sản xuất xe đạp chuyên nghiệp. Khi tải xuống SQL Server Management Studio, CSDL được gọi là *AdventureWorks*. Chúng ta thực hiện theo các bước như sau:

1. Sao chép *AdventureWorks_Data.mdf* và *AdventureWorks_Log.ldf* vào một thư mục phù hợp và sau đó mở SQL Server Management Studio (SSMS).
2. Kết nối cục bộ và click chuột phải lên nút *Databases* trong cửa sổ *object explorer*, sau đó chọn *Attach*.
3. Sử dụng nút *Add* để duyệt tệp *AdventureWorks_Data.mdf* và nhấp vào OK và sau đó nhấp OK lần nữa

4. Chúng ta có thể làm mới nút *Databases* bằng cách nhấn F5 và CSDL mới sẽ xuất hiện. Sau đó chúng ta có thể nhấp chuột phải vào nó và chọn *Rename* và gọi nó là *AdventureWorks2014R2* và nhấn F5 để hoàn thành nhiệm vụ.

Phản tiếp theo sẽ trình bày cách biểu diễn dữ liệu quan hệ dưới dạng XML.

9.3.2. Trình bày dữ liệu quan hệ dưới dạng XML

9.3.2.1. Sử dụng FOR XML RAW

Việc chuyển đổi dữ liệu dạng bảng sang định dạng XML là một yêu cầu khá phổ biến. SQL Server cung cấp một số tùy chọn để đạt được điều này và tất cả chúng liên quan đến việc thêm cụm từ *FOR XML <mode>* vào cuối một truy vấn *SELECT* thông thường. Chúng ta có thể sử dụng bốn chế độ khác nhau, mỗi chế độ cho phép chúng ta điều chỉnh các kết quả ở mức độ thấp hơn hoặc cao hơn. Phương thức cơ bản nhất của *FOR XML* là *RAW*.

Ví dụ 9.1: Giả sử chúng ta có truy vấn là chọn các chi tiết cơ bản của các đơn đặt hàng có giá trị lớn hơn 300,000\$.

```
SELECT [PurchaseOrderID], [RevisionNumber],
       [Status], [EmployeeID],
       [VendorID], [ShipMethodID],
       [OrderDate], [ShipDate],
       [SubTotal], [TaxAmt],
       [Freight], [TotalDue], [ModifiedDate]
  FROM [Purchasing].[PurchaseOrderHeader]
 WHERE [TotalDue] > 300000;
```

Kết quả có 3 hàng, được thể hiện trong Hình 9.1.

```
ForXmlQueries.sql - DESKTOP-V26PM4I.ADVENTUREWORKS2014R2 (DESKTOP-V26PM4I\THUTHUY (52))*
SELECT [PurchaseOrderID]
      ,[RevisionNumber]
      ,[Status]
      ,[EmployeeID]
      ,[VendorID]
      ,[ShipMethodID]
      ,[OrderDate]
      ,[ShipDate]
      ,[SubTotal]
      ,[TaxAmt]
      ,[Freight]
      ,[TotalDue]
      ,[ModifiedDate]
  FROM [Purchasing].[PurchaseOrderHeader]
 WHERE [TotalDue] > 300000;
```

PurchaseOrderID	RevisionNumber	Status	EmployeeID	VendorID	ShipMethodID	OrderDate	ShipDate	TotalDue
4007	10	2	164	102	3	2004-04-01 00:00:00.000	2004-04-26 00:00:00.000	5
4008	16	2	244	95	3	2004-05-23 00:00:00.000	2004-06-17 00:00:00.000	3
4012	5	2	231	29	3	2004-07-25 00:00:00.000	2004-08-19 00:00:00.000	9

Hình 9.1: Kết quả thực hiện truy vấn của Ví dụ 9.1

Để trả về XML, thêm *FOR XML RAW* vào truy vấn.

Ví dụ 9.2:

```
SELECT [PurchaseOrderID], [RevisionNumber],
       [Status], [EmployeeID],
       [VendorID], [ShipMethodID],
       [OrderDate], [ShipDate],
       [SubTotal], [TaxAmt],
       [Freight], [TotalDue], [ModifiedDate]
  FROM [Purchasing].[PurchaseOrderHeader]
 WHERE [TotalDue] > 300000;
FOR XML RAW;
```

Bây giờ chúng ta sẽ có được một cái nhìn dữ liệu dưới dạng XML, với mỗi hàng được bao bọc trong một phần tử. Tuy nhiên, không có phần tử tài liệu được thêm vào vì vậy nó thực sự là một đoạn XML. Khi double click vào đoạn kết quả XML thì chúng ta sẽ nhận được một hiển thị đầy đủ của dữ liệu XML. Kết quả thể hiện ở Hình 9.2.

Hình 9.2: Kết quả thực hiện truy vấn của Ví dụ 9.2

Nếu chúng ta muốn xem phần tử trung tâm, hãy thêm chỉ thị ELEMENTS vào truy vấn.

Ví dụ 9.3:

```
SELECT [PurchaseOrderID], [RevisionNumber],  
       [Status], [EmployeeID],  
       [VendorID], [ShipMethodID],  
       [OrderDate], [ShipDate],  
       [SubTotal], [TaxAmt],  
       [Freight], [TotalDue], [ModifiedDate]  
  FROM [Purchasing].[PurchaseOrderHeader]  
 WHERE [TotalDue] > 300000;  
FOR XML RAW, ELEMENTS ;
```

Kết quả thể hiện ở Hình 9.3, trong đó câu lệnh truy vấn ở bên phải và kết quả dữ liệu XML ở bên trái của hình.

```

XML_F52E2B61-18A...00805F49916B9.xml
<row>
    <PurchaseOrderID>4007</PurchaseOrderID>
    <RevisionNumber>10</RevisionNumber>
    <Status>2</Status>
    <EmployeeID>164</EmployeeID>
    <VendorID>102</VendorID>
    <ShipMethodID>3</ShipMethodID>
    <OrderDate>2004-04-01T00:00:00</OrderDate>
    <ShipDate>2004-04-26T00:00:00</ShipDate>
    <SubTotal>554020.0000</SubTotal>
    <TaxAmt>44321.6000</TaxAmt>
    <Freight>11080.4000</Freight>
    <TotalDue>609422.0000</TotalDue>
    <ModifiedDate>2005-09-12T12:25:46.407</ModifiedDate>
</row>
<row>
    <PurchaseOrderID>4008</PurchaseOrderID>
    <RevisionNumber>16</RevisionNumber>
    <Status>2</Status>
    <EmployeeID>244</EmployeeID>
    <VendorID>95</VendorID>
    <ShipMethodID>3</ShipMethodID>

```

```

ForXmlQueries.sql - DESKTOP-V26PM4I.ADVENTUREWORKS2014R2
SELECT [PurchaseOrderID]
      ,[RevisionNumber]
      ,[Status]
      ,[EmployeeID]
      ,[VendorID]
      ,[ShipMethodID]
      ,[OrderDate]
      ,[ShipDate]
      ,[SubTotal]
      ,[TaxAmt]
      ,[Freight]
      ,[TotalDue]
      ,[ModifiedDate]
  FROM [Purchasing].[PurchaseOrderHeader]
 WHERE [TotalDue] > 300000
 FOR XML RAW, ELEMENTS;

```

Hình 9.3: Kết quả thực hiện truy vấn của Ví dụ 9.3

Như đã đề cập ở trên, truy vấn trả về một đoạn XML chứ không phải là một tài liệu đầy đủ. Để thêm một phần tử gốc, chúng ta sử dụng chỉ thị FOOT kết hợp với tên của phần tử gốc mà chúng ta muốn.

Ví dụ 9.4:

```

SELECT [PurchaseOrderID], [RevisionNumber],
       [Status], [EmployeeID],
       [VendorID], [ShipMethodID],
       [OrderDate], [ShipDate],
       [SubTotal], [TaxAmt],
       [Freight], [TotalDue], [ModifiedDate]
  FROM [Purchasing].[PurchaseOrderHeader]
 WHERE [TotalDue] > 300000;
FOR XML RAW, ELEMENTS, ROOT('orders') ;

```

Kết quả thể hiện ở Hình 9.4.

```

XML_F52E2B61-18A...00805F49916B11.xml
<orders>
    <row>
        <PurchaseOrderID>4007</PurchaseOrderID>
        <RevisionNumber>10</RevisionNumber>
        <Status>2</Status>
        <EmployeeID>164</EmployeeID>
        <VendorID>102</VendorID>
        <ShipMethodID>3</ShipMethodID>
        <OrderDate>2004-04-01T00:00:00</OrderDate>
        <ShipDate>2004-04-26T00:00:00</ShipDate>
        <SubTotal>554020.0000</SubTotal>
        <TaxAmt>44321.6000</TaxAmt>
        <Freight>11080.4000</Freight>
        <TotalDue>609422.0000</TotalDue>
        <ModifiedDate>2005-09-12T12:25:46.407</ModifiedDate>
    </row>
    <row>
        <PurchaseOrderID>4008</PurchaseOrderID>
        <RevisionNumber>16</RevisionNumber>
        <Status>2</Status>
        <EmployeeID>244</EmployeeID>
        <VendorID>95</VendorID>

```

```

ForXmlQueries.sql - DESKTOP-V26PM4I.ADVENTUREWORKS2014R2
SELECT [PurchaseOrderID]
      ,[RevisionNumber]
      ,[Status]
      ,[EmployeeID]
      ,[VendorID]
      ,[ShipMethodID]
      ,[OrderDate]
      ,[ShipDate]
      ,[SubTotal]
      ,[TaxAmt]
      ,[Freight]
      ,[TotalDue]
      ,[ModifiedDate]
  FROM [Purchasing].[PurchaseOrderHeader]
 WHERE [TotalDue] > 300000
 FOR XML RAW, ELEMENTS, ROOT('orders');

```

Hình 9.4: Kết quả thực hiện truy vấn của Ví dụ 9.4

Chúng ta cũng có thể thay đổi tên của vùng chứa hàng mặc định, đó là <row>. Chỉ cần thêm tên trong dấu ngoặc đơn sau từ khóa RAW, thể hiện ở Ví dụ 9.5.

Ví dụ 9.5:

```
SELECT [PurchaseOrderID], [RevisionNumber],
       [Status], [EmployeeID],
       [VendorID], [ShipMethodID],
       [OrderDate], [ShipDate],
       [SubTotal], [TaxAmt],
       [Freight], [TotalDue], [ModifiedDate]
  FROM [Purchasing].[PurchaseOrderHeader]
 WHERE [TotalDue] > 300000;
FOR XML RAW('order'), ELEMENTS, ROOT('orders') ;
```

Kết quả truy vấn được thể hiện ở Hình 9.5.

The screenshot shows the SQL Server Management Studio interface. On the left, there is a 'Results' pane displaying XML output. The XML output shows two purchase orders, each with various details like PurchaseOrderID, RevisionNumber, Status, EmployeeID, VendorID, OrderDate, ShipDate, SubTotal, TaxAmt, Freight, TotalDue, and ModifiedDate. On the right, there is a 'Script' pane containing the SQL query used to generate the results. The query uses the FOR XML clause with RAW('order') and ELEMENTS options to produce XML output, and ROOT('orders') to define the root element.

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B.xml
<orders>
<order>
<PurchaseOrderID>4007</PurchaseOrderID>
<RevisionNumber>10</RevisionNumber>
<Status>2</Status>
<EmployeeID>164</EmployeeID>
<VendorID>102</VendorID>
<ShipMethodID>3</ShipMethodID>
<OrderDate>2004-04-01T00:00:00</OrderDate>
<ShipDate>2004-04-26T00:00:00</ShipDate>
<SubTotal>554020.0000</SubTotal>
<TaxAmt>44321.6000</TaxAmt>
<Freight>11080.4000</Freight>
<TotalDue>609422.0000</TotalDue>
<ModifiedDate>2005-09-12T12:25:46.407</ModifiedDate>
</order>
<order>
<PurchaseOrderID>4008</PurchaseOrderID>
<RevisionNumber>16</RevisionNumber>
<Status>2</Status>
<EmployeeID>244</EmployeeID>
<VendorID>95</VendorID>
</order>
</orders>
```

```
ForXmlQueries.sql - DESKTOP-V26PM4I.ADVENTUREWORKS2014R2 (DE)
SELECT [PurchaseOrderID]
      ,[RevisionNumber]
      ,[Status]
      ,[EmployeeID]
      ,[VendorID]
      ,[ShipMethodID]
      ,[OrderDate]
      ,[ShipDate]
      ,[SubTotal]
      ,[TaxAmt]
      ,[Freight]
      ,[TotalDue]
      ,[ModifiedDate]
  FROM [Purchasing].[PurchaseOrderHeader]
 WHERE [TotalDue] > 300000
FOR XML RAW('order'), ELEMENTS, ROOT('orders')
```

Hình 9.5: Kết quả thực hiện truy vấn của Ví dụ 9.5

Một vấn đề thường phát sinh là làm thế nào để xử lý các giá trị rỗng (null) trong kết quả. Mặc định là không đưa ra phần tử hoặc thuộc tính nào nếu giá trị của nó là null. Tuy nhiên, đôi khi phần tử hoặc thuộc tính null cần được đưa vào câu truy vấn để xử lý các kết quả.

Để phân biệt giữa một phần tử có chứa một giá trị null với một phần tử có một chuỗi rỗng thì cần phải có một điểm đánh dấu trên phần tử để biểu thị rằng giá trị của nó là null thay vì một chuỗi rỗng. Điểm đánh dấu được sử dụng là *xsi:nil = "true"*. Đây là một thuộc tính chuẩn từ không gian tên của giản đồ, vì vậy SQL Server cũng cần phải thêm ràng buộc không gian tên chính xác. Nếu chúng ta muốn thực hiện điều này, hãy sử dụng chỉ thị *XSNIL* sau từ khóa ELEMENTS. Đoạn mã sau đây chỉ ra cách đảm bảo rằng các phần tử có giá trị null là đầu ra thay vì bị bỏ qua.

Ví dụ 9.6:

```
SELECT [PurchaseOrderID], [RevisionNumber],
       [Status], [EmployeeID],
       [VendorID], [ShipMethodID],
```

```

[OrderDate], [ShipDate],
[SubTotal], [TaxAmt],
[Freight], [TotalDue], [ModifiedDate]
FROM [Purchasing].[PurchaseOrderHeader]
WHERE [TotalDue] > 300000;
FOR XML RAW('order'), ELEMENTS XSINIL, ROOT('orders') ;
Kết quả truy vấn được thể hiện ở Hình 9.6.

```

The screenshot shows the SQL Server Management Studio interface. On the left, the results of the XML query are displayed as an XML document titled 'orders'. It contains two 'order' elements. Each 'order' element has attributes like PurchaseOrderID, RevisionNumber, Status, EmployeeID, VendorID, ShipMethodID, OrderDate, ShipDate, SubTotal, TaxAmt, Freight, TotalDue, and ModifiedDate. The right pane shows the T-SQL code that was executed to produce this result.

```

<orders xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <order>
    <PurchaseOrderID>4007</PurchaseOrderID>
    <RevisionNumber>10</RevisionNumber>
    <Status>2</Status>
    <EmployeeID>164</EmployeeID>
    <VendorID>102</VendorID>
    <ShipMethodID>3</ShipMethodID>
    <OrderDate>2004-04-01T00:00:00</OrderDate>
    <ShipDate>2004-04-26T00:00:00</ShipDate>
    <SubTotal>554020.0000</SubTotal>
    <TaxAmt>44321.6000</TaxAmt>
    <Freight>11080.4000</Freight>
    <TotalDue>609422.0000</TotalDue>
    <ModifiedDate>2005-09-12T12:25:46.407</ModifiedDate>
  </order>
  <order>
    <PurchaseOrderID>4008</PurchaseOrderID>
    <RevisionNumber>16</RevisionNumber>
    <Status>2</Status>
    <EmployeeID>244</EmployeeID>
    <VendorID>95</VendorID>
  </order>
</orders>

```

```

SELECT [PurchaseOrderID]
      ,[RevisionNumber]
      ,[Status]
      ,[EmployeeID]
      ,[VendorID]
      ,[ShipMethodID]
      ,[OrderDate]
      ,[ShipDate]
      ,[SubTotal]
      ,[TaxAmt]
      ,[Freight]
      ,[TotalDue]
      ,[ModifiedDate]
  FROM [Purchasing].[PurchaseOrderHeader]
 WHERE [TotalDue] > 300000
 FOR XML RAW('order'), ELEMENTS XSINIL, ROOT('orders');

```

Hình 9.6: Kết quả thực hiện truy vấn của Ví dụ 9.6

Nếu bản sao của CSDL không có *ShipDate* là rỗng, chúng ta luôn có thể thay đổi một mã trong câu lệnh truy vấn để có sự thay đổi cần thiết và sau đó khôi phục nó ở cuối.

Một tính năng cuối cùng có thể hữu ích, đặc biệt là nếu dữ liệu của chúng ta được chuyển đến bên thứ ba, là khả năng thêm một giản đồ XML. Điều này được thực hiện bằng cách nối đoạn mã trước với chỉ thị XMLSCHEMA như sau:

Ví dụ 9.7:

```

SELECT [PurchaseOrderID], [RevisionNumber],
       [Status], [EmployeeID],
       [VendorID], [ShipMethodID],
       [OrderDate], [ShipDate],
       [SubTotal], [TaxAmt],
       [Freight], [TotalDue], [ModifiedDate]
  FROM [Purchasing].[PurchaseOrderHeader]
 WHERE [TotalDue] > 300000;
 FOR XML RAW('order'), ELEMENTS XSINIL, ROOT('orders'),
 XMLSCHEMA;

```

Kết quả của truy vấn trên là một giản đồ được chèn vào đầu dữ liệu XML, thể hiện ở Hình 9.7.

```

-- ForXmlQueries.sql - DESKTOP-V26PM4I.ADVENTUREWORKS2014R2 (DESKTOP-V26PM4I\THUTHU)
-- SELECT [PurchaseOrderID]
--     ,[RevisionNumber]
--     ,[Status]
--     ,[EmployeeID]
--     ,[VendorID]
--     ,[ShipMethodID]
--     ,[OrderDate]
--     ,[ShipDate]
--     ,[SubTotal]
--     ,[TaxAmt]
--     ,[Freight]
--     ,[TotalDue]
--     ,[ModifiedDate]
-- FROM [Purchasing].[PurchaseOrderHeader]
-- WHERE [TotalDue] > 300000
-- FOR XML RAW ('order'), ELEMENTS XSINIL, ROOT ('orders')) XMLSCHEMA

```

Hình 9.7: Kết quả thực hiện truy vấn của Ví dụ 9.7

Mặc dù chế độ RAW có một vài lựa chọn nhưng nó không thành công khi xử lý dữ liệu có tính phân cấp. Để kiểm soát nhiều hơn và để xử lý dữ liệu có tính phân cấp hiệu quả hơn chúng ta cần sử dụng FOR XML AUTO

9.3.2.2. Sử dụng FOR XML AUTO

Nếu chúng ta cố gắng sử dụng chế độ RAW với dữ liệu lồng nhau, chẳng hạn như đơn đặt hàng cùng với các mục hàng, chúng ta sẽ nhận được khối lặp đi lặp lại của XML, trong đó đơn đặt hàng được lặp lại cho mỗi mục hàng. Một trong những điểm mạnh của XML là khả năng hiển thị dữ liệu theo thứ bậc rõ ràng, do đó, cần tránh việc lặp lại dữ liệu trong XML. Trong phần này chúng ta sẽ biết được cách thức AUTO xử lý việc lặp lại dữ liệu.

Trong các ví dụ sau đây, chúng ta sẽ sử dụng chỉ thị *FOR XML AUTO* phirc tạp hơn. Chúng ta sẽ thấy cách kiểm soát này tốt hơn các truy vấn với *FOR XML RAW* mà chúng ta gặp trước đó. *FOR XML AUTO* chủ yếu tốt hơn ở xử lý XML trả về khi hai hoặc nhiều bảng được tham gia vào một truy vấn, ví dụ khi *PurchaseOrderHeader* tham gia với *PurchaseOrderDetail* để cung cấp cho một cái nhìn đầy đủ về một đơn đặt hàng.

Ví dụ 9.8: Thay thế từ khoá RAW bằng AUTO trong truy vấn cơ bản được giới thiệu trong Ví dụ 9.2:

```

SELECT [PurchaseOrderID], [RevisionNumber],
       [Status], [EmployeeID],
       [VendorID], [ShipMethodID],
       [OrderDate], [ShipDate],
       [SubTotal], [TaxAmt],
       [Freight], [TotalDue], [ModifiedDate]
  FROM [Purchasing].[PurchaseOrderHeader]
 WHERE [TotalDue] > 300000;
 FOR XML AUTO;

```

Chúng ta sẽ không thấy sự khác biệt nhiều trong các kết quả của truy vấn này so với phiên bản RAW, ngoại thực tế là tên của phần tử đang nắm giữ dữ liệu được lấy từ tên bảng chứ không phải là một phần tử hàng chung.

XML_F52E2B61-18A...00805F49916B1.xml

```
<Purchasing.PurchaseOrderHeader PurchaseOrderID="4007" RevisionNumber="10" Status="2"
<Purchasing.PurchaseOrderHeader PurchaseOrderID="4008" RevisionNumber="16" Status="2"
<Purchasing.PurchaseOrderHeader PurchaseOrderID="4012" RevisionNumber="5" Status="2"

ForXmlQueries.sql - DESKTOP-V26PM4I.ADVENTUREWORKS2014R2 (DESKTOP-V26PM4I\THUTHUY (52))

SELECT [PurchaseOrderID]
      ,[RevisionNumber]
      ,[Status]
      ,[EmployeeID]
      ,[VendorID]
      ,[ShipMethodID]
      ,[OrderDate]
      ,[ShipDate]
      ,[SubTotal]
      ,[TaxAmt]
      ,[Freight]
      ,[TotalDue]
      ,[ModifiedDate]
  FROM [Purchasing].[PurchaseOrderHeader]
 WHERE [TotalDue] > 300000
 FOR XML AUTO;
```

100 %

Results Messages

XML_F52E2B61-18A1-11d1-B105-00805F49916B

1 <Purchasing.PurchaseOrderHeader PurchaseOrderID=...

Hình 9.8: Kết quả thực hiện truy vấn của Ví dụ 9.8

Kết quả của Ví dụ 9.8 là một đoạn XML không có thẻ gốc. Để thêm thẻ gốc, chúng ta chỉ cần bổ sung *ROOT('orders')* vào sau lệnh FOR XML AUTO.

Sự khác biệt thực sự trở nên rõ ràng khi một truy vấn trích xuất dữ liệu từ hai bảng được liên kết được thực hiện. SQL sau đây hiển thị tất cả đơn đặt hàng trước cùng với các mục hàng riêng lẻ của chúng:

Ví dụ 9.9:

```
SELECT POH.[PurchaseOrderID], POH.[RevisionNumber],
       POH.[Status], POH.[EmployeeID],
       POH.[VendorID], POH.[ShipMethodID],
       POH.[OrderDate], POH.[ShipDate],
       POH.[SubTotal], POH.[TaxAmt],
       POH.[Freight], POH.[TotalDue],
       POH.[ModifiedDate], POD.[OrderQty],
       POD.[ProductID], POD.[UnitPrice]
  FROM [Purchasing].[PurchaseOrderHeader] POH
 INNER JOIN Purchasing.PurchaseOrderDetail POD
    ON POH.PurchaseOrderID = POD.PurchaseOrderID
 WHERE [TotalDue] > 300000;
```

Ở Ví dụ 9.9, các bảng đã được kết hợp trên trường *PurchaseOrderId* và các bảng đã được đặt tên mới để sử dụng các tên ngắn hơn, POH và POD. Các kết quả của truy vấn này được thể hiện trong Hình 9.9.

Tiếp theo, chúng ta sửa đổi truy vấn bằng cách thêm FOR XML AUTO vào bên dưới truy vấn của Ví dụ 9.9.

```

- SELECT POH.[PurchaseOrderID]
,POH.[RevisionNumber]
,POH.[Status]
,POH.[EmployeeID]
,POH.[VendorID]
,POH.[ShipMethodID]
,POH.[OrderDate]
,POH.[ShipDate]
,POH.[SubTotal]
,POH.[TaxAmt]
,POH.[Freight]
,POH.[TotalDue]
,POH.[ModifiedDate]
,POD.[OrderQty]
,POD.[ProductID]
,POD.[UnitPrice]
FROM [Purchasing].[PurchaseOrderHeader] POH
INNER JOIN Purchasing.PurchaseOrderDetail POD
ON POH.PurchaseOrderID = POD.PurchaseOrderID
WHERE [TotalDue] > 300000;

```

100 %

	PurchaseOrderID	RevisionNumber	Status	EmployeeID	VendorID	ShipMethodID	OrderDate	ShipDate	SubTotal	TaxAmt
1	4007	10	2	164	102	3	2004-04-01 00:00:00.000	2004-04-26 00:00:00.000	554020.00	44321
2	4007	10	2	164	102	3	2004-04-01 00:00:00.000	2004-04-26 00:00:00.000	554020.00	44321
3	4007	10	2	164	102	3	2004-04-01 00:00:00.000	2004-04-26 00:00:00.000	554020.00	44321

Hình 9.9: Kết quả thực hiện Ví dụ 9.9**Ví dụ 9.10:**

```

SELECT POH.[PurchaseOrderID], POH.[RevisionNumber],
       POH.[Status], POH.[EmployeeID],
       POH.[VendorID], POH.[ShipMethodID],
       POH.[OrderDate], POH.[ShipDate],
       POH.[SubTotal], POH.[TaxAmt],
       POH.[Freight], POH.[TotalDue],
       POH.[ModifiedDate], POD.[OrderQty],
       POD.[ProductID], POD.[UnitPrice]
FROM [Purchasing].[PurchaseOrderHeader] POH
INNER JOIN Purchasing.PurchaseOrderDetail POD
ON POH.PurchaseOrderID = POD.PurchaseOrderID
WHERE [TotalDue] > 300000
FOR XML AUTO, ROOT('orders');

```

Lưu ý rằng trong ví dụ 9.10, một phần tử gốc đã được đưa vào, giống như tùy chọn RAW. Các kết quả xuất hiện như Hình 9.10 với tính chất có tính phân cấp rõ ràng hơn nhiều.

XML_F52E2B61-18A...00805f49916B3.xml

```

<orders>
- <POH PurchaseOrderID="4007" RevisionNumber="10" Status="2" EmployeeID="164" VendorID="102" ShipMethodID="3" OrderDate="2004-04-01 00:00:00.000" ModifiedDate="2004-04-26 00:00:00.000">
  <POD OrderQty="5000" ProductID="849" UnitPrice="24.7500" />
  <POD OrderQty="5000" ProductID="850" UnitPrice="24.7500" />
  <POD OrderQty="5000" ProductID="851" UnitPrice="24.7500" />
  <POD OrderQty="750" ProductID="852" UnitPrice="30.9400" />
  <POD OrderQty="750" ProductID="853" UnitPrice="30.9400" />
  <POD OrderQty="750" ProductID="854" UnitPrice="30.9400" />
  <POD OrderQty="1050" ProductID="855" UnitPrice="37.1000" />
  <POD OrderQty="1000" ProductID="856" UnitPrice="37.1000" />
  <POD OrderQty="1000" ProductID="857" UnitPrice="37.1000" />
</POH>
- <POH PurchaseOrderID="4008" RevisionNumber="16" Status="2" EmployeeID="164" VendorID="102" ShipMethodID="3" OrderDate="2004-04-01 00:00:00.000" ModifiedDate="2004-04-26 00:00:00.000">
  <POD OrderQty="700" ProductID="858" UnitPrice="9.1500" />
  <POD OrderQty="700" ProductID="859" UnitPrice="9.1500" />
  <POD OrderQty="700" ProductID="860" UnitPrice="9.1500" />
  <POD OrderQty="900" ProductID="861" UnitPrice="15.6700" />
  <POD OrderQty="900" ProductID="862" UnitPrice="15.6700" />
  <POD OrderQty="900" ProductID="863" UnitPrice="15.6700" />
  <POD OrderQty="1250" ProductID="864" UnitPrice="23.7500" />
  <POD OrderQty="1250" ProductID="865" UnitPrice="23.7500" />
  <POD OrderQty="1250" ProductID="866" UnitPrice="23.7500" />
  <POD OrderQty="1250" ProductID="867" UnitPrice="23.7500" />
  <POD OrderQty="2000" ProductID="868" UnitPrice="26.1800" />
  <POD OrderQty="2000" ProductID="869" UnitPrice="26.1800" />
</POH>

```

100 %

Hình 9.10: Kết quả thực hiện truy vấn của Ví dụ 9.10

Điễn giải

Trong truy vấn ở Ví dụ 9.9, không có chỉ thị *FOR XML AUTO*, dẫn đến một kết quả lặp đi lặp lại với mỗi dòng đơn đặt hàng đều chứa đầy đủ chi tiết từ tiêu đề. Việc thêm *FOR XML AUTO*, *ROOT ('orders')* vào truy vấn để tạo ra một tập hợp các bản ghi lồng nhau, làm cho mỗi tiêu đề đơn đặt hàng là một phần tử với các chi tiết như ngày đặt hàng và ID được hiển thị dưới dạng các thuộc tính. Bên dưới mỗi thành phần *<POH>* là một phần tử *<POD>* đại diện cho một dòng từ thứ tự. Một lần nữa, mỗi yếu tố này sử dụng các thuộc tính để hiển thị các giá trị như số lượng đặt hàng và ID sản phẩm.

Các tùy chọn khác cho *FOR XML RAW*, chẳng hạn như *ELEMENTS*, *XSNIL*, và *XMLSCHEMA*, cũng có sẵn cho *FOR XML AUTO*.

9.3.2.3. Sử dụng *FOR XML PATH*

Tùy chọn *PATH*, dựa trên việc sử dụng XPath để chỉ định định dạng của đầu ra, làm cho việc xây dựng XML lồng ghép với sự kết hợp của các phần tử và các thuộc tính tương đối đơn giản. Lấy ví dụ kết quả truy vấn trước đó, trong đó các đơn đặt hàng trên 300.000 đô la đã được truy xuất và trả về dưới dạng XML dựa vào thuộc tính bằng cách sử dụng tùy chọn *AUTO*.

```
<orders>
    <PurchaseOrderHeader PurchaseOrderID="4007"
        RevisionNumber="16" Status="2" EmployeeID="251"
        VendorID="1594" ShipMethodID="3" OrderDate=
        "2008-04-01T00:00:00" SubTotal="554020.0000"
        TaxAmt="44321.6000" Freight="11080.4000"
        TotalDue="609422.0000"
        ModifiedDate="2009-09-12T12:25:46.407" />
</orders>
```

Điều gì xảy ra nếu có một bộ cục tài liệu khác xuất hiện? Ví dụ, nếu *PurchaseOrderID* và *EmployeeID* có trạng thái là thuộc tính nhưng ở phần tử cũng có *PurchaseOrderID* và *EmployeeID*? Tùy chọn *PATH* sử dụng bí danh của các column để chỉ định cách cấu trúc XML. Cú pháp tương tự như XPath, do đó có từ khoá *PATH*.

Truy vấn *PATH* cho dữ liệu đơn đặt hàng là một sự kết hợp của các thuộc tính và các phần tử sẽ như sau:

Ví dụ 9.11:

```
SELECT [PurchaseOrderID] [@PurchaseOrderID]
    ,[Status] [@Status]
    ,[EmployeeID] [@EmployeeID]
    ,[VendorID]
    ,[ShipMethodID]
    ,[OrderDate]
    ,[ShipDate]
    ,[SubTotal]
    ,[TaxAmt]
    ,[Freight],[TotalDue]
FROM [Purchasing].[PurchaseOrderHeader] POH
WHERE [TotalDue] > 300000
FOR XML PATH('order'), ROOT('orders');
```

Lưu ý cách dữ liệu cần được trả về dưới dạng các thuộc tính được đặt theo tên cột bắt đầu bằng @. Các cột không được phân loại sẽ được trả về dưới dạng các phần tử. Kết quả của truy vấn này sẽ giống với XML trong Hình 9.11:

```

XML_F52E2B61-18A..00805F49916B1.xml
<orders>
  <order PurchaseOrderID="4007" Status="2" EmployeeID="164">
    <VendorID>102</VendorID>
    <ShipMethodID>3</ShipMethodID>
    <OrderDate>2004-04-01T00:00:00</OrderDate>
    <ShipDate>2004-04-26T00:00:00</ShipDate>
    <SubTotal>554020.0000</SubTotal>
    <TaxAmt>44321.6000</TaxAmt>
    <Freight>11080.4000</Freight>
    <TotalDue>609422.0000</TotalDue>
  </order>
  <order PurchaseOrderID="4008" Status="2" EmployeeID="95">
    <VendorID>95</VendorID>
    <ShipMethodID>3</ShipMethodID>
    <OrderDate>2004-05-23T00:00:00</OrderDate>
    <ShipDate>2004-06-17T00:00:00</ShipDate>
    <SubTotal>396729.0000</SubTotal>
    <TaxAmt>31738.3200</TaxAmt>
    <Freight>7934.5800</Freight>
    <TotalDue>436401.9000</TotalDue>
  </order>
  <order PurchaseOrderID="4012" Status="2" EmployeeID="29">
    <VendorID>29</VendorID>
    <ShipMethodID>3</ShipMethodID>
    <OrderDate>2004-07-25T00:00:00</OrderDate>
    <ShipDate>2004-08-19T00:00:00</ShipDate>
    <SubTotal>997680.0000</SubTotal>
    <TaxAmt>79814.4000</TaxAmt>
    <Freight>19953.6000</Freight>
    <TotalDue>1097448.0000</TotalDue>
  </order>
</orders>

```

```

ForXmlQueries.sql - DESKTOP-V26PM4I.ADVENTUREWORKS2014R2 (DES)
FOR XML AUTO, ROOT('orders');

SELECT [PurchaseOrderID] AS '@PurchaseOrderID',
       [Status] AS '@Status',
       [EmployeeID] AS '@EmployeeID',
       [VendorID],
       [ShipMethodID],
       [OrderDate],
       [ShipDate],
       [SubTotal],
       [TaxAmt],
       [Freight],
       [TotalDue]
  FROM [Purchasing].[PurchaseOrderHeader] POH
 WHERE [TotalDue] > 300000
 FOR XML PATH('order'), ROOT('orders');

```

Hình 9.11: Kết quả thực hiện truy vấn của Ví dụ 9.11

Tùy chọn PATH cũng cung cấp quyền kiểm soát cấu trúc lồng nhau. Cách thông thường để làm điều này, thay vì sử dụng một JOIN SQL là sử dụng một subquery. Đoạn mã sau đây cho thấy tiêu đề đơn hàng là thuộc tính, với chi tiết đơn hàng là các phần tử lồng nhau:

Ví dụ 9.12:

```

SELECT [POH].[PurchaseOrderID] AS '@PurchaseOrderID',
       [POH].[Status] AS '@Status',
       [POH].[EmployeeID] AS '@EmployeeID',
       [POH].[VendorID] AS '@VendorID',
       [POH].[ShipMethodID] AS '@ShipMethodID',
       [POH].[SubTotal] AS '@SubTotal',
       [POH].[TaxAmt] AS '@TaxAmt',
       [POH].[Freight] AS '@Freight',
       [POH].[TotalDue] AS '@TotalDue',
       [POH].[OrderDate] AS 'Dates/Order',
       [POH].[ShipDate] AS 'Dates/Ship',
       (SELECT [POD].[OrderQty],
              [POD].[ProductID],
              [POD].[UnitPrice]
         FROM [Purchasing].[PurchaseOrderDetail] POD
        WHERE POD.OrderID = POH.OrderID)
      FOR XML PATH('order'), ROOT('orders');

```

```

        WHERE POH.[PurchaseOrderID] =
POD.[PurchaseOrderID]
            ORDER BY POD.[PurchaseOrderID]
            FOR XML PATH('orderDetail'), TYPE)
FROM [Purchasing].[PurchaseOrderHeader] POH
WHERE [POH].[TotalDue] > 300000
FOR XML PATH('order'), ROOT('orders');

```

Kết quả được trình bày ở Hình 9.12.

```

XML_F52E2B61-18A...00805F49916B2.xml x

    <order PurchaseOrderID="4007" Status="2" EmployeeID="164" VendorID="102" ShipMethodID="3" SubTotal="5540"
        <Dates>
            <Order>2004-04-01T00:00:00</Order>
            <Ship>2004-04-26T00:00:00</Ship>
        </Dates>
        <orderDetail>
            <OrderQty>5000</OrderQty>
            <ProductID>849</ProductID>
            <UnitPrice>24.7500</UnitPrice>
        </orderDetail>
        <orderDetail>
            <OrderQty>5000</OrderQty>
            <ProductID>850</ProductID>
            <UnitPrice>24.7500</UnitPrice>
        </orderDetail>
        <orderDetail>
            <OrderQty>5000</OrderQty>
            <ProductID>851</ProductID>
            <UnitPrice>24.7500</UnitPrice>
        </orderDetail>
        <orderDetail>
            <OrderQty>750</OrderQty>
            <ProductID>852</ProductID>
            <UnitPrice>30.9400</UnitPrice>
        </orderDetail>
        <orderDetail>
            <OrderQty>750</OrderQty>
            <ProductID>853</ProductID>
            <UnitPrice>30.9400</UnitPrice>
        </orderDetail>
        <orderDetail>
            <OrderQty>750</OrderQty>
            <ProductID>854</ProductID>
        </orderDetail>
    </order>

```

```

ForXmlQueries.sql - DESKTOP-V26PM4I.ADVENTUREWORKS2014R2 (DESKTOP-V26PM4I)
FOR XML PATH('order'), ROOT('orders');

SELECT [POH].[PurchaseOrderID] [@PurchaseOrderID],
       [POH].[Status] [@Status],
       [POH].[EmployeeID] [@EmployeeID],
       [POH].[VendorID] [@VendorID],
       [POH].[ShipMethodID] [@ShipMethodID],
       [POH].[SubTotal] [@SubTotal],
       [POH].[TaxAmt] [@TaxAmt],
       [POH].[Freight] [@Freight],
       [POH].[TotalDue] [@TotalDue],
       [POH].[OrderDate] [Dates/Order],
       [POH].[ShipDate] [Dates/Ship]

    ,
    (
        SELECT [POD].[OrderQty]
              ,[POD].[ProductID]
              ,[POD].[UnitPrice]
        FROM [Purchasing].[PurchaseOrderDetail] POD
        WHERE POH.[PurchaseOrderID] = POD.[PurchaseOrderID]
        ORDER BY POD.[PurchaseOrderID]
        FOR XML PATH('orderDetail'), TYPE
    )
    ,
    FROM [Purchasing].[PurchaseOrderHeader] POH
    WHERE [POH].[TotalDue] > 300000
    FOR XML PATH('order'), ROOT('orders');

```

Hình 9.12: Kết quả thực hiện truy vấn của Ví dụ 9.12

Trong đoạn mã trên, sự thay đổi quan trọng là *OrderDate* và *ShipDate* trong SELECT bên ngoài. Các cột được đặt bí danh là *Date/Order* và *Dates/Ship*, do đó, SQL Server tạo ra một phần tử mới, *Dates*, để giữ hai giá trị này. Ngoài ra còn có một bí danh trên toàn bộ truy vấn phụ, *OrderDetails*, làm cho tất cả các kết quả của nó được nhóm lại dưới một phần tử.

9.3.3. Tạo tài liệu XML trong SQL Server

Ví dụ sau đây cho thấy làm thế nào để tạo ra một bảng được thiết kế đặc biệt để giữ tài liệu XML. Khi bảng đã được tạo, chúng ta sẽ thấy cách chèn một vài tài liệu XML mẫu và sau đó lấy chúng bằng SQL.

- Mở SQL Server Management Studio (SSMS) và kết nối cục bộ với SQL Server trên máy tính (hoặc bất cứ máy chủ nào chúng ta muốn tạo cơ sở dữ liệu thử nghiệm trên đó).
- Trong Object Explorer, mở rộng các nút để các CSDL của người dùng được hiển thị. Nhấp chuột phải và chọn tùy chọn *New Database*. Khi

hộp thoại mở ra, nhập vào tên cho cơ sở dữ liệu, ví dụ, *XMLDocTest*. Trước khi nhấp OK, hãy đảm bảo rằng tùy chọn *Full Text Indexing* (Đánh chỉ mục văn bản đầy đủ) được chọn.

3. Tạo một bảng được gọi là Docs sử dụng SQL sau:

CREATE TABLE Docs

(DOCID INTEGER IDENTITY PRIMARY KEY, XMLDOC XML)

Cột XMLDoc có kiểu XML. Bây giờ chúng ta có một bảng trống.

4. Với mục đích của ví dụ này, thêm các tài liệu XML đơn giản với cấu trúc sau:

<Person>

```
<FirstName></FirstName>
<LastName></LastName>
</Person>
```

5. Chèn các tài liệu XML bằng cách sử dụng câu lệnh SQL INSERT, như sau:

INSERT Docs

```
VALUES ('<Person><FirstName>Joe</FirstName>
<LastName>Fawcett</LastName></Person>')
```

6. Sau khi sửa đổi các giá trị của các phần tử *FirstName* và *LastName* và thêm một vài dữ liệu vào cột XMLDoc, để kiểm tra dữ liệu đã nhập, chúng ta sử dụng câu lệnh SQL sau:

Ví dụ 9.13:

```
SELECT * FROM Docs;
```

Kết quả của truy vấn SQL này được hiển thị trong Hình 9.13.

```
SELECT * FROM Docs;
|  
|  
| DECLARE @myDoc xml;  
|  
| SET @myDoc = '<Person><FirstName>Joe</FirstName>  
<LastName>Fawcett</LastName></Person>';  
  
|  
| SELECT @myDoc;  
| SET @myDoc.modify(' delete /Person/*[2]');  
| SELECT @myDoc;  
100 % -  
  
Results Messages  


|   | DocID | XMLDoc                                                         |
|---|-------|----------------------------------------------------------------|
| 1 | 1     | <Person><FirstName>Joe</FirstName><LastName>Fawcett</LastName> |
| 2 | 2     | <Person><FirstName>Danny</FirstName><LastName>Ayman</LastName> |
| 3 | 3     | <Person><FirstName>Liam</FirstName><LastName>Quinn</LastName>  |


```

Hình 9.13: Kết quả truy vấn bảng Docs

Các giá trị chứa trong cột XMLDoc được hiển thị trong khung bên dưới của Hình 9.13. Tiếp theo, chúng ta sẽ tạo ra một số truy vấn XQuery đơn giản.

Tạo chỉ mục với kiểu dữ liệu XML

Các tài liệu XML trong SQL Server cũng có thể được lập chỉ mục để thu được hiệu quả hơn và có thể tạo một chỉ mục toàn văn. Để tạo một chỉ mục toàn văn trên một tài liệu, sử dụng một lệnh như sau:

```
--If no catalog exists so far
CREATE FULLTEXT CATALOG ft ON DEFAULT
CREATE FULLTEXT INDEX ON dbo.Docs(XmlDoc) KEY INDEX
<primary key name>
```

Kiểu dữ liệu xml cho phép chúng ta sử dụng các phương pháp sau để thao tác dữ liệu và trích xuất nó dưới nhiều dạng khác nhau: modify(), query(), value(), existing() và nodes(). Các phần sau sẽ xem xét từng phương pháp và mô tả cách chúng được sử dụng.

9.3.3.1. Sử dụng phương thức modify()

Kiểu dữ liệu xml có thể được truy vấn bằng ngôn ngữ XQuery. Trong SQL Server, các biểu thức XQuery được nhúng trong Transact-SQL. Transact-SQL là ngôn ngữ đánh máy của ngôn ngữ SQL được sử dụng trong SQL Server.

Microsoft đã giới thiệu phương thức modify() trước khi XQuery được sử dụng rộng rãi. Vào thời điểm đó, đã có cuộc nói chuyện về việc cập nhật lên tiêu chuẩn cơ bản khi nó xuất hiện, nhưng cho đến thời điểm đó vẫn chưa xảy ra.

XQuery 1.0 bị hạn chế vì nó chỉ có thể truy vấn nguồn dữ liệu XML. Không có thành phần nào trong XQuery 1.0 để thực hiện việc xóa, để chèn dữ liệu mới, hoặc để sửa đổi dữ liệu. Trong SQL Server, ngôn ngữ điều biến dữ liệu XML (DML) cho biết thêm ba từ khóa cho chức năng có sẵn trong XQuery 1.0. Chúng ta có thể thấy các từ khóa này hoạt động trong các ví dụ của các phần:

- delete
- insert
- replace value of

Xóa bằng XML DML

Trong phần này chúng ta sẽ xem xét làm thế nào để xóa một phần của một tài liệu XML bằng cách sử dụng phương thức *modify()* kết hợp với từ khóa *delete*. Chúng ta sẽ sử dụng một tài liệu XML đơn giản được lưu trữ trong một biến địa phương và sau đó nhắm vào một phần cụ thể để xóa. Đoạn mã sau cho thấy một ví dụ về cách nó có thể được sử dụng.

Ví dụ 9.14:

```
DECLARE @myDoc xml;
SET @myDoc = '<Person><FirstName>Joe</FirstName>
<LastName>Fawcett</LastName></Person>';
SELECT @myDoc;
SET @myDoc.modify(' delete /Person/*[2]');
SELECT @myDoc;
```

Dòng đầu tiên của Ví dụ 9.14 khai báo một biến, *myDoc*, và xác định kiểu dữ liệu là *xml*. Thông số *SET* chỉ định một giá trị cho biến *myDoc*. Đây là một thành viên quen thuộc của *Person* với các phần tử con *FirstName* và *LastName* và nội dung văn bản tương ứng.

Câu lệnh *SELECT* sau câu lệnh *SET* làm cho giá trị *myDoc* được hiển thị. Tiếp theo, chức năng sửa đổi được sử dụng để sửa đổi giá trị của kiểu dữ liệu XML. Câu lệnh bên trong hàm *modify()*, giống như XQuery, phân biệt chữ hoa chữ thường. Từ khóa *delete* được sử dụng để chỉ định phần nào của tài liệu XML sẽ bị xóa. Trong trường hợp này, biểu thức *XPath/Person/*[2]* xác định rằng phần tử con thứ hai của *Person*, phần tử sẽ bị xóa, đó là phần tử *LastNarie*.

Câu lệnh *SELECT* cuối cho thấy giá trị của *myDoc* sau khi xóa. Hình 9.14 cho thấy kết quả của cả hai câu lệnh *SELECT*.

```

DECLARE @myDoc xml;
SET @myDoc = '<Person><FirstName>Joe</FirstName>
<LastName>Fawcett</LastName></Person>';

SELECT @myDoc;
SET @myDoc.modify(' delete /Person/*[2]');
SELECT @myDoc;

```

Results

(No column name)
1 <Person><FirstName>Joe</FirstName><LastName>Fawc...

(No column name)
1 <Person><FirstName>Joe</FirstName></Person>

Hình 9.14: Kết quả thực hiện câu lệnh Ví dụ 9.14

Chèn bằng XML DML

Trong phần này cho thấy làm thế nào để thêm dữ liệu vào XML hiện có. Nó sử dụng phương thức *modify()* cùng với từ khóa *insert*. Mã Transact-SQL được hiển thị ở đây:

Ví dụ 9.15:

```

DECLARE @myDoc XML;
SET @myDoc =
'<Person><LastName>Fawcett</LastName></Person>';
SELECT @myDoc;
SET @myDoc.modify(' insert <FirstName>Joe</FirstName>
as first into /Person[1]');
SELECT @myDoc;

```

Trong dòng đầu tiên chúng ta khai báo một biến, *myDoc*, và chỉ định rằng nó có kiểu dữ liệu *xml*. Chúng ta đặt giá trị của biến *myDoc*. Sau đó chúng ta chỉ định một phần tử *Person* chỉ chứa một thành phần *LastName*, chứa tên *Fawcett*.

Hàm *modify()* được sử dụng để chứa phần mở rộng XQuery mà chúng ta muốn sử dụng. XML sẽ được chèn sau từ khóa *insert*. Lưu ý rằng nó không

được kèm theo bởi dấu nháy hoặc dấu ngoặc kép. Mệnh đề *SET* lần đầu tiên quy định rằng XML sẽ được chèn vào đầu tiên. Mệnh đề *into* sử dụng một biểu thức XPath, */Person*, để xác định rằng phần tử *FirstName* và nội dung của nó được thêm vào như là một phần tử con cho phần tử Person. Với mệnh đề đầu tiên, chúng ta biết rằng phần tử *FirstName* là con đầu tiên của phần tử *Person*.

Một cách khác, chúng ta cũng có thể sử dụng *after* hoặc *before*. Truy vấn của Ví dụ 9.15 có thể được viết lại như sau:

```
DECLARE @myDoc XML;
SET @myDoc =
    '<Person><LastName>Fawcett</LastName></Person>';
SELECT @myDoc;
SET @myDoc.modify(' insert <FirstName>Joe</FirstName>
before (/Person/LastName)[1]');
SELECT @myDoc;
```

Khi chúng ta chạy lệnh Transact-SQL, câu lệnh SELECT đầu tiên làm cho XML gốc được hiển thị và câu lệnh SELECT thứ hai làm cho XML được hiển thị sau khi hoạt động chèn đã hoàn tất.

Kết quả của Ví dụ 9.15 được thể hiện ở Hình 9.15.

The screenshot shows the SQL query window titled 'XmlDataType.sql - ...M4\THUTHUY (53)'. The code is identical to the one shown in the previous text block. The 'Results' tab is selected, displaying two rows of output. Row 1 shows the original XML state: '<Person><LastName>Fawcett</LastName></Person>'. Row 2 shows the state after modification: '<Person><FirstName>Joe</FirstName><LastName>Fawc...</Person>'. The ellipsis indicates the rest of the XML document.

Hình 9.15: Kết quả thực hiện truy vấn của Ví dụ 9.15

Cập nhật bằng XML DML

Ví dụ tiếp theo sẽ cập nhật nội dung của một biến XML để biến đổi giá trị của phần tử FirstName từ *Joe* sang *Gillian*. Mã được hiển thị ở đây:

Ví dụ 9.16:

```
DECLARE @myDoc XML;
SET @myDoc =
    '<Person><FirstName>Joe</FirstName><LastName>Fawcett</LastName></Person>';
SELECT @myDoc;
SET @myDoc.modify(' replace value of
    (/Person/FirstName/text())[1] with "Gillian" ');
SELECT @myDoc;
```

Giá trị thay thế của từ khoá cho thấy một sự cập nhật và biểu thức XPath chỉ ra phần nào của XML sẽ được áp dụng cập nhật. Trong trường hợp này nó

là nút văn bản con của phần tử *FirstName* - được xác định bởi biểu thức XPath- */Person/FirstName/text()*.

Kết quả của hai câu lệnh SELECT được thể hiện trong Hình 9.16.

The screenshot shows a query window titled "XmlDataType.sql - not connected". The code uses XML variables and the modify method to change the first name from "Joe" to "Gillian". The results pane shows two rows of XML output, both identical to the input, indicating no changes were made.

```
DECLARE @myDoc XML;
SET @myDoc = '<Person><FirstName>Joe</FirstName><LastName>Fawcett</LastName></Person>';
SELECT @myDoc;
SET @myDoc.modify(' replace value of (/Person/FirstName/text())[1] with "Gillian" ');
SELECT @myDoc;
```

Results

	(No column name)
1	<Person><FirstName>Joe</FirstName><LastName>Fawc...
1	<Person><FirstName>Gillian</FirstName><LastName>...

Hình 9.16: Kết quả truy vấn Ví dụ 9.16

Một trong những vấn đề chính khi sử dụng phương thức *modify()* là nó chứa một chuỗi lệnh cố định trong đối số của nó. Do đó khó thực hiện các truy vấn động trong thế giới thực - ví dụ như cần truy vấn dữ liệu XML mới được đưa vào từ bảng khác. Chúng ta có hai cách để giải quyết vấn đề này. Trước tiên, chúng ta có thể xây dựng truy vấn như một chuỗi và thực hiện nó một cách linh động bằng cách sử dụng EXEC. Ngoài ra, chúng ta có thể sử dụng chức năng tích hợp *sql:column* và *sql:function*.

Ví dụ của mỗi kỹ thuật sau đây.

Đối với những ví dụ này, chúng ta có thể sử dụng bảng *Docs* được tạo ra trước đó. Trước tiên, xét đoạn lệnh cập nhật tĩnh:

```
UPDATE Docs
SET XmlDocument.modify ('replace value of
    (/Person/FirstName/text())[1] with "Joseph" ')
WHERE DocId = 1;
```

Bây giờ giả sử chúng ta muốn thay thế giá trị cố định *Joseph* với một biến. Chúng ta có thể thử điều này như sau:

```
-- This will error
DECLARE @NewName NVARCHAR(100);
SET @NewName = N'Joseph';
UPDATE Docs
SET XmlDocument.modify(' replace value of
    (/Person/FirstName/text())[1] with "' + @NewName + '" ')
WHERE DocId = 1;
```

Đoạn lệnh trên sẽ không thực hiện. Phương thức *modify()* báo lỗi rằng nó cần một chuỗi chữ. Một cách để giải quyết vấn đề này là xây dựng toàn bộ câu lệnh SQL như sau:

```
DECLARE @NewName NVARCHAR(100);
SET @NewName = N'Joseph';
DECLARE @SQL NVARCHAR(MAX);
```

```

SET @SQL = 'UPDATE Docs SET XmlDocument.modify('' replace
value of (/Person/FirstName/text())[1] with ''' + @NewName
+ ''') WHERE DocId = 1';
PRINT(@SQL);
EXEC(@SQL);

```

Cách để xuất để cập nhật dựa trên dữ liệu sẽ chỉ được biết đến trong thời gian chạy là sử dụng các chức năng tích hợp *sql:column* hoặc *sql:variable*. Chức năng *sql:column* được sử dụng khi dữ liệu mới được lấy ra từ một bảng, do đó ở đây *sql:variable* là cần thiết:

```

DECLARE @NewName NVARCHAR(100);
SET @NewName = N'Joseph';
UPDATE Docs
SET XmlDocument.modify
(' replace value of (/Person/FirstName/text())[1] with
sql:variable("@NewName") ')
WHERE DocId = 1;

```

9.3.3.2. Sử dụng phương thức *query()*

Phương thức *query()* cho phép chúng ta xây dựng câu lệnh XQuery trong SQL Server. Cú pháp tuân theo cú pháp XQuery được thảo luận trong Chương 8.

Truy vấn sau đây sử dụng phương thức *query()* để xuất ra các tên của mỗi người trong một phần tử *Name* mới được tạo với giá trị của phần tử *Lastname* và dấu phẩy sau đó là giá trị của phần tử *FirstName*. Câu lệnh được hiển thị ở Ví dụ 9.17.

Ví dụ 9.17:

```

SELECT XMLDoc.query
('for $p in /Person return
<Name>{$p/LastName/text()} ,
{$p/FirstName/text()}</Name>')
FROM Docs;

```

Dòng đầu tiên chỉ ra rằng một sự lựa chọn đang được thực hiện bằng cách sử dụng phương thức *query()* được áp dụng cho cột *XMLDoc*.

Mệnh đề *for* xác định rằng biến *\$p* được ràng buộc với nút phần tử *Person*.

Mệnh đề *return* chỉ rõ rằng một phần tử *Name* được xây dựng bằng cách rút trích giá trị của *\$p/LastName/text()*, là nội dung văn bản của phần tử *Lastname*.

Hình 9.17 hiển thị đầu ra khi câu lệnh SELECT chứa truy vấn XQuery được chạy.

```

- SELECT XMLDoc.query
  ('for $p in /Person return
   <Name>{$p/LastName/text()}, {$p/FirstName/text()}</Name>')
  FROM Docs;

```

00 % ▾ <

Results Messages

(No column name)

1	<Name>Fawcett, Joseph</Name>
2	<Name>Ayers, Danny</Name>
3	<Name>Quinn, Liam</Name>

Hình 9.17: Kết quả thực hiện truy vấn của Ví dụ 9.17

Ngoài ra còn có một số phương thức khác như: value(), exist(), nodes()

Phương thức value()

Phương thức *value()* sử dụng XPath để xác định các dữ liệu cụ thể trong một tài liệu XML và sau đó chuyển nó thành một kiểu dữ liệu SQL Server tiêu chuẩn. Nó thường được sử dụng trong phần WHERE của truy vấn SQL.

Giả sử chúng ta muốn truy xuất tất cả những người trong bảng *Docs* có *FirstName* là *Joe*. Đây là một cách để thực hiện nó:

```

SELECT * FROM Docs
WHERE XMLDoc.value('(/*/FirstName)[1]',
                     'nvarchar(100)') = 'Joe';

```

Câu lệnh trên chỉ trả về một hàng cho tài liệu đầu tiên chúng ta đã thêm vào. Lưu ý về kiểu dữ liệu mà chúng ta đang chuyển đổi cần phải được trích dẫn ra, nhiều sai lầm phổ biến là quên đi điều này. Rõ ràng, chúng ta cũng có thể sử dụng phương thức *value()* trong danh sách SELECT. Nếu chúng ta chỉ muốn hiển thị *LastName* của mỗi người, chúng ta sẽ sử dụng lệnh sau:

```

SELECT DocId, XMLDoc.value('(/*/LastName)[1]',
                            'nvarchar(100)') LastName FROM Docs;

```

Lệnh trên trả về một bộ kết quả chuẩn có hai cột.

Phương thức exist()

Phương thức *exist()* dùng để kiểm tra một giá trị tồn tại hay không. Nó trả về giá trị 0 nếu nó không tồn tại, giá trị 1 nếu có và là giá trị *null* nếu cột XML chứa *null*. Vì vậy, chúng ta có thể viết lại truy vấn trên để hiển thị những người có *FirstName* là *Joe* theo cách này:

```

SELECT * FROM Docs
WHERE XMLDoc.exist('/*/FirstName[. = "Joe"]') = 1;

```

Lệnh này trả về kết quả giống như truy vấn trước với cách sử dụng phương thức *value()*.

Phương thức nodes()

Phương thức *nodes()* được sử dụng để trình bày một tài liệu XML dưới dạng một bảng SQL thông thường. Chúng ta thường cần điều này khi truy vấn của chúng ta cần một hàng dữ liệu từ một bảng được kết hợp với một phần tử con của XML. Xét ví dụ đơn giản với đoạn mã trong Ví dụ 9.18:

Ví dụ 9.18:

```
DECLARE @People xml;
SET @People = '<people><person>Joe</person>
<person>Danny</person>
<person>Liam </person></people>';
SELECT FirstName.value('text()[1]', 'nvarchar(100)')
FirstName FROM
@People.nodes('//*/person') Person(FirstName);
```

Phương thức `nodes()` lấy một XPath trả về một số phần tử con lặp đi lặp lại của tài liệu chính. Sau đó chúng ta cung cấp một tên bảng và một tên cột để sử dụng sau này trong các hình thức TableName (ColumnName). Ở đây, bảng là `Person` và cột là `FirstName`. Cột `FirstName` sau đó được truy vấn bằng cách sử dụng `value()` để nhận được văn bản. Kết quả được thể hiện trong Hình 9.18.

FirstName
1 Joe
2 Danny
3 Liam

Hình 9.18: Kết quả thực hiện truy vấn của Ví dụ 9.18

Tóm lại, trong chương này, chúng ta đã học được những nội dung sau:

- Phần lớn dữ liệu ngày nay đi kèm dưới dạng dữ liệu dạng bảng và XML kết hợp.
- Chúng ta cần một loại lưu trữ dành riêng cho XML thay vì chỉ sử dụng một trường văn bản. Chúng ta cũng cần các phương pháp để trích xuất các giá trị xác định và các đoạn XML cũng như các phương pháp để tạo các định dạng XML mới kết hợp các dữ liệu quan hệ và XML. Có thể chúng ta sẽ muốn cơ sở cập nhật các tài liệu XML mặc dù điều này không phải lúc nào cũng là điều cần thiết.
- CSDL quan hệ xử lý cả dữ liệu dạng bảng và tài liệu XML, cơ sở dữ liệu XML được thiết kế để đối phó với các tài liệu XML.
- Các hệ thống cao cấp như Oracle và SQL Server XML có loại dữ liệu riêng của họ và có các phương pháp thích hợp có sẵn trên các loại này để phục hồi và thao tác của XML.

BÀI TẬP CHƯƠNG 9

1. Liệt kê các lý do chính để chọn một cơ sở dữ liệu quan hệ với các tính năng XML thay vì chọn một CSDL XML nguyên bản.
2. MySQL chỉ có hai chức năng liên quan đến XML. Nếu chúng ta có thể yêu cầu thêm một tính năng hoặc chức năng, nó sẽ là gì?
3. Tìm hiểu một số phương pháp để chuyển dữ liệu từ một CSDL quan hệ thành dữ liệu XML,

Chương 10

LINQ VÀ MỘT SỐ ỨNG DỤNG XML

Như chúng ta đã biết, có một số cách để đọc, xử lý và tạo ra XML. Có thể sử dụng mô hình đối tượng tài liệu (DOM), tải toàn bộ tài liệu vào bộ nhớ, hoặc một trong các phương thức streaming được đề cập trong chương trước, chẳng hạn như XmlReader của Microsoft hoặc giao diện SAX. Chương này giới thiệu một tùy chọn khác, có nhiệm vụ tương tác với XML với một trong những công nghệ lập trình chính của Microsoft là LINQ.

10.1. LINQ là gì?

LINQ là ngôn ngữ truy vấn tích hợp, viết tắt của *Language Integrated Query*. LINQ cho XML cung cấp một giao diện lập trình XML. LINQ cho XML sử dụng những ngôn ngữ mới nhất của .NET Language Framework và được nâng cấp, thiết kế lại với giao diện lập trình XML Document Object Model (DOM).

Cú pháp của LINQ như sau:

```
from <range variable> in <collection>
where <predicate>
select <something using the range variable>
```

Ở đây, *range variable* là một tiêu chuẩn xác định được sử dụng để chỉ các mục đã chọn, *collection* là một tập hợp các đối tượng được truy vấn và *predicate* là một biểu thức để xác định có nên bao gồm các đối tượng trong kết quả cuối cùng hay không?

Ví dụ, thực hiện công việc đơn giản để chiết xuất các số chẵn từ mảng (các ví dụ này trong C#):

```
int [] numbers = new int[10] {0, 1, 2, 3, 4, 5, 6, 7,
8, 9};
var evenNumbers =
    from num in numbers
    where (num% 2) == 0
    select num;
```

Ở đây, *range variable* là *num*, *collection* là một mảng các con số được đặt tên số, và *predicate* là *(num%2) == 0*.

Với LINQ, truy vấn không được thực thi ngay lập tức. Cho đến nay, *evenNumbers* giữ chi tiết của truy vấn, chứ không phải là kết quả. Truy vấn sẽ thực sự chạy khi các kết quả được sử dụng như được hiển thị trong đoạn mã sau đây:

```
int [] numbers = new int[10] {0, 1, 2, 3, 4, 5, 6, 7,
8, 9};
var evenNumbers =
    from num in numbers
    where (num% 2) == 0
    select num;
// Output the even numbers to the console
// This will actually execute the LINQ operation
foreach(int number in evenNumbers)
```

```
{Console.WriteLine(number);}
```

Nếu chúng ta thực thi mã này trong trình gõ lỗi (debugger) và duyệt từng dòng, chúng ta sẽ thấy rằng các hoạt động của LINQ không thực hiện cho đến khi vòng lặp *foreach* xuất kết quả.

10.2. Sử dụng LINQ

LINQ cho phép chúng ta sử dụng một kỹ thuật tương tự để truy vấn nhiều nguồn dữ liệu XML. Nó cung cấp cách để trích xuất dữ liệu từ một tài liệu XML dễ dàng hơn nhiều so với cả DOM cũng như dễ dàng hơn cách lập trình theo sự kiện của .NET's XmlReader và SAX.

LINQ cung cấp một cách mới để tạo các tài liệu XML dễ sử dụng hơn DOM hoặc XmlWriter, bao gồm một cách đơn giản để xử lý các khía cạnh tên bắt chước cách chúng được khai báo trong XML. Nếu chúng ta đang phát triển .NET và phải trích xuất thông tin từ một tài liệu XML, chúng ta nên chọn LINQ to XML.

Thông thường trong hướng dẫn của LINQ cho XML, chúng ta sẽ được trình bày với một tài liệu XML mẫu và hiển thị cách truy vấn nó. Chúng ta sẽ làm ngược lại ở đây: chúng ta sẽ thấy làm thế nào để tạo ra một tài liệu XML sử dụng cái được gọi là xây dựng chức năng. Cách chuẩn để tạo XML sử dụng mô hình đối tượng tài liệu là tạo ra phần tử gốc và sau đó thêm vào bất cứ phần tử con và các thuộc tính nào là cần thiết. Một đoạn chương trình trong C# sau đây sẽ tạo một tệp XML mô tả một bộ sưu tập nhạc được hiển thị ở Ví dụ 10.1.

Ví dụ 10.1:

```
static XElement CreateCDElement(XmlDocument doc, string id, string title, int year, string artist, string genre)
{
    XElement cd = doc.CreateElement("cd");
    cd.SetAttribute("id", "1");
    XElement cdTitle = doc.CreateElement("title");
    cdTitle.InnerText = title;
    cd.AppendChild(cdTitle);
    XElement cdYear = doc.CreateElement("year");
    cdYear.InnerText = year.ToString();
    cd.AppendChild(cdYear);
    XElement cdArtist = doc.CreateElement("artist");
    cdArtist.InnerText = artist;
    cd.AppendChild(cdArtist);
    XElement cdGenre = doc.CreateElement("genre");
    cdGenre.InnerText = "New Wave";
    cd.AppendChild(cdGenre);
    return cd;
}
```

Đoạn mã ở Ví dụ 10.1 thêm một phần tử `<cd>` với thuộc tính và con của nó vào bộ sưu tập. Bằng cách lặp lại đoạn mã, các phần tử `<cd>` khác có thể được thêm vào để tạo thành bộ sưu tập nhạc hoàn chỉnh.

Mặc dù đoạn mã này có thể thực hiện công việc, nhưng nó khá dài dòng, chúng ta phải tạo, đặt, và nối các giá trị cho mỗi phần tử. Phương pháp tiếp

cận chức năng của LINQ cho XML ngắn hơn và rõ ràng hơn, như thể hiện ở Ví dụ 10.2.

Ví dụ 10.2:

```
static XElement CreateCDElement(string id, string title, int year, string artist, string genre)
{
    return new XElement("cd",
        new XAttribute("id", id),
        new XElement("title", title),
        new XElement("year", year),
        new XElement("artist", artist),
        new XElement("genre", genre));
}
```

Đoạn mã ở Ví dụ 10.2 sử dụng các lớp tạo ra không gian tên System.Linq.Xml. Các khái niệm xây dựng cơ bản trong thư viện này là *XElement* và *XAttribute*.

Tập tin XML hoàn chỉnh sau khi được tạo ra được thể hiện ở Ví dụ 10.3.

Ví dụ 10.3: MusicLibrary.xml

```
<?xml version="1.0" encoding="utf-8"?>
<musicLibrary>
    <cd id="1"> <title>Parallel Lines</title>
        <year>2001</year>
        <artist>Blondie</artist>
        <genre>New Wave</genre> </cd>
    <cd id="2"> <title>Bat Out of Hell</title>
        <year>2001</year>
        <artist>Meatloaf</artist>
        <genre>Rock</genre> </cd>
    <cd id="3"> <title>Abbey Road</title>
        <year>1987</year>
        <artist>The Beatles</artist>
        <genre>Rock</genre> </cd>
    <cd id="4"> <title>The Dark Side of the Moon</title>
        <year>1994</year>
        <artist>Pink Floyd</artist>
        <genre>Rock</genre> </cd>
    <cd id="5"> <title>Thriller</title>
        <year>2001</year>
        <artist>Michael Jackson</artist>
        <genre>Pop</genre> </cd>
</musicLibrary>
```

10.2.1. Tạo tài liệu XML với LINQ

Trong Ví dụ 10.2, chúng ta đã nhìn thấy lớp *XElement* và *XAttribute*. Chúng ta có thể tự hỏi tại sao chúng ta không sử dụng một lớp *XDocument* riêng biệt. Nếu chúng ta tạo một tài liệu XML bằng DOM, chúng ta cần phải sử dụng rất nhiều *DomDocument*. Đây là nơi LINQ cho XML và DOM khác biệt nhất. LINQ cho XML không có một lớp *XDocument*, nhưng chúng ta không phải sử dụng nó. Trong hầu hết các câu lệnh, chúng ta chỉ cần sử dụng lớp *XElement* để tải XML hoặc xây dựng các phần tử. Tuy nhiên, trong một số trường hợp lớp *XDocument* là vô cùng quan trọng.

Lớp *XDocument* rất hữu ích khi chúng ta cần bổ sung thêm một số siêu dữ liệu vào tài liệu XML, ví dụ như một khai báo XML hoặc khi chúng ta muốn có một chú thích hay lệnh xử lý xuất hiện trước phần tử tài liệu. Giả sử chúng ta muốn khai báo XML tiêu chuẩn tuyên bố rằng phiên bản là 1.0, mã hóa là UTF-8 và tài liệu là độc lập, giống như câu lệnh sau:

```
<? xml      version="1.0"    encoding="utf-8"
                           standalone="yes"?>
```

Chúng ta có thể đạt được điều này bằng cách sử dụng lớp *XDocument* ở cấp cao nhất và sau đó bằng cách sử dụng lớp *XDeclaration*, trong đó có ba tham số để mô tả phiên bản, mã hóa và giá trị cho thuộc tính độc lập. Xem Ví dụ 10.4.

Ví dụ 10.4:

```
static void CreateMusicLibrary()
{
    XDocument musicLibrary =
        new XDocument(
            new XDeclaration("1.0", "utf-8", "yes"),
            new XComment("This document holds details of my
                         music collection"),
            new XElement("musicLibrary",
                CreateCDElement("1", "Parallel Lines", 2001,
                               "Blondie", "New Wave"),
                CreateCDElement("2", "Bat Out of Hell", 2001,
                               "Meatloaf", "Rock"),
                CreateCDElement("3", "Abbey Road", 1987, "The
                               Beatles", "Rock"),
                CreateCDElement("4", "The Dark Side of the Moon",
                               1994, "Pink Floyd", "Rock"),
                CreateCDElement("5", "Thriller", 2001, "Michael
                               Jackson", "Pop")));
    Console.WriteLine(musicLibrary.ToString());
    Console.ReadLine();
}
```

Kết quả của Ví dụ 10.4 là đoạn XML sau:

```
<?xml      version="1.0"    encoding="utf-8"
standalone="yes"?>
<!-- This document holds details of my music
collection --&gt;
&lt;musicLibrary&gt;
    &lt;cd id="1"&gt;
        &lt;title&gt;Parallel Lines&lt;/title&gt;
        &lt;year&gt;2001&lt;/year&gt;
        &lt;artist&gt;Blondie&lt;/artist&gt;
        &lt;genre&gt;New Wave&lt;/genre&gt;
    &lt;/cd&gt;
&lt;/musicLibrary&gt;</pre>
```

Chúng ta cũng có thể sử dụng *XProcessingInstruction* theo cách tương tự. Ví dụ, nếu chúng ta muốn kết hợp chuyển đổi XSL với tài liệu thì chúng ta sử dụng câu lệnh **new XProcessingInstruction("xmlstylesheet", "href='music.xslt'")** trong Ví dụ 10.4.

Cho đến lúc này, các tài liệu chúng ta đã tạo ra đều không có không gian tên. Chuyện gì xảy ra khi chúng ta cần phải tạo ra các phần tử hoặc thuộc tính thuộc một không gian tên cụ thể? Phần tiếp theo giải quyết tình huống này.

10.2.2. Tạo tài liệu XML với không gian tên

Việc tạo các phần tử trong các không gian tên luôn có một chút phức tạp hơn. LINQ có gắng làm cho nó trở nên dễ dàng bằng cách tạo một lớp riêng biệt, *XNamespace*, có thể được sử dụng để khai báo và áp dụng không gian tên cho một phần tử hoặc một thuộc tính.

Để tạo một tài liệu với không gian tên, thực hiện các bước sau:

1. Tạo một phiên bản mới của thư viện nhạc, một trong đó các phần tử đều nằm trong không gian tên

<http://www.wrox.com/namespaces/apps/musicLibrary>.

2. Đặt tên cho không gian tên mặc định (không dùng tiền tố; tất cả các phần tử trong tài liệu sẽ tự động thuộc vùng tên này). Tài liệu mà chúng ta muốn tạo ra sẽ như sau:

```
<musicLibrary  
xmlns="http://www.wrox.com/namespaces/apps/musicLibrary">  
    <cd id="1">  
        <title>Parallel Lines</title>  
        <year>2001</year>  
        <artist>Blondie</artist>  
        <genre>New Wave</genre>  
    </cd>  
    <!-- more cd elements -->  
</musicLibrary>
```

3. Để thực hiện việc này, sử dụng lớp *XNamespace* để khai báo và áp dụng không gian tên như trong đoạn mã dưới đây:

```
static void CreateMusicLibrary()  
{  
    XNamespace ns =  
        "http://www.wrox.com/namespaces/apps/musicLibrary";  
    XElement musicLibrary =  
        new XElement(ns + "musicLibrary",  
            CreateCDElement(ns, "1", "Parallel Lines", 2001,  
                "Blondie", "New Wave"),  
            CreateCDElement(ns, "2", "Bat Out of Hell", 2001,  
                "Meatloaf", "Rock"),  
            CreateCDElement(ns, "3", "Abbey Road", 1987, "The  
                Beatles", "Rock"),  
            CreateCDElement(ns, "4", "The Dark Side of the Moon",  
                1994, "Pink Floyd", "Rock"),  
            CreateCDElement(ns, "5", "Thriller", 2001, "Michael  
                Jackson", "Pop"));  
    Console.WriteLine(musicLibrary.ToString());  
    Console.ReadLine(); }  
    static XElement CreateCDElement(XNamespace ns, string  
id, string title, int year, string artist, string genre)  
    {  
        return new XElement(ns + "cd",  
            new XAttribute("id", id),
```

```

        new XElement(ns + "title", title),
        new XElement(ns + "year", year),
        new XElement(ns + "artist", artist),
        new XElement(ns + "genre", genre));
    }

```

Việc đặt tên cho không gian tên với tiền tố được cho trước tương tự với các lệnh dùng cho không gian tên mặc định. Sự khác biệt chính là chúng ta cần phải sử dụng lớp *XAttribute* để xác định URI không gian tên của chúng ta trước như sau:

```

XNamespace ns = 
    "http://www.wrox.com/namespaces/apps/musicLibrary";
 XElement musicLibrary =
    new XElement(ns + "musicLibrary",
        new XAttribute(XNamespace.Xmlns + "ns",
            ns.NamespaceName),
        new XElement(ns + "cd", new XAttribute("id", 1),
        new XElement(ns + "title", "Parallel Lines"),
        new XElement(ns + "year", 2001),
        new XElement(ns + "artist", "Blondie"),
        new XElement(ns + "genre", "New Wave")));

```

Dòng được in đậm sử dụng lớp *XAttribute* và một thành viên tĩnh của lớp *XNamespace*, *Xmlns*, để tạo mã *xmlns:ns* = "http://www.wrox.com/namespaces/apps/musicLibrary" trên phần tử gốc. Bây giờ thì LINQ đã biết URI không gian tên bị ràng buộc với tiền tố *ns*, do đó tất cả các phần tử trong không gian tên này sẽ tự động được cung cấp tiền tố này. Tài liệu XML sẽ tương tự như sau:

```

<musicLibrary
xmlns:ns="http://www.wrox.com/namespaces/apps/musicLibrary">
    <ns:cd id="1">
        <ns:title>Parallel Lines</ns:title>
        <ns:year>2001</ns:year>
        <ns:artist>Blondie</ns:artist>
        <ns:genre>New Wave</ns:genre>
    </ns:cd>
    <!-- more cd elements -->
</ns:musicLibrary>

```

Đến lúc này thì chúng ta đã nắm được cách tạo XML với LINQ, tiếp theo chúng ta sẽ tìm hiểu cách trích xuất dữ liệu từ tài liệu XML với LINQ.

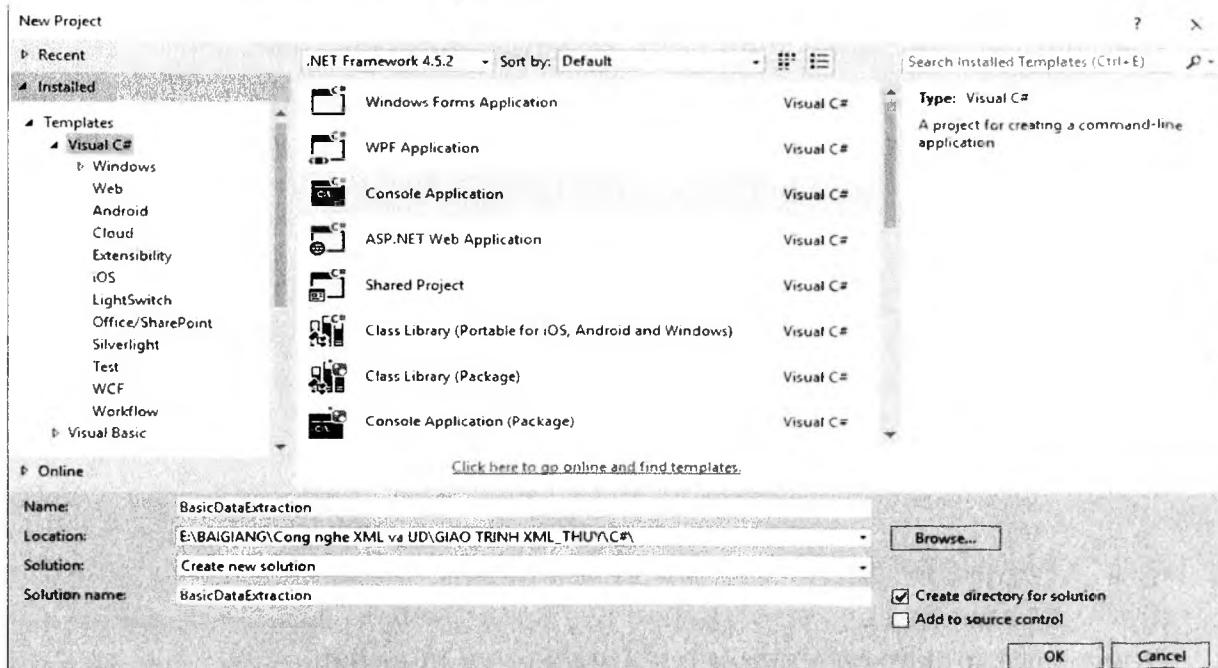
10.3. Xử lý dữ liệu XML với LINQ

Phần này xem xét một số kịch bản phổ biến liên quan đến tải một tệp tin XML hiện có và lấy ra các phần cụ thể của nó. Ví dụ, chúng ta cần tải *MusicLibrary.xml* và hiển thị danh sách tất cả các tiêu đề đĩa CD. Để thực hiện yêu cầu này, chúng ta sẽ sử dụng phương thức *Elements()*.

10.3.1. Trích xuất dữ liệu bằng phương thức *Elements()*

Phần này giới thiệu cách sử dụng phương thức *Elements()* để lấy các phần tử và nội dung của chúng. Chúng ta bắt đầu bằng cách tải một tập tin hiện có và sau đó xem cách điều hướng đến nội dung cụ thể mà chúng ta cần, trong trường hợp này là các phần tử *<title>* của đĩa CD của chúng ta.

- Để bắt đầu, tạo một dự án C # Console Application mới trong Visual Studio và đặt tên cho nó là *BasicDataExtraction*. Bước này được thể hiện trong Hình 10.1.



Hình 10.1: Tạo dự án BasicDataExtraction

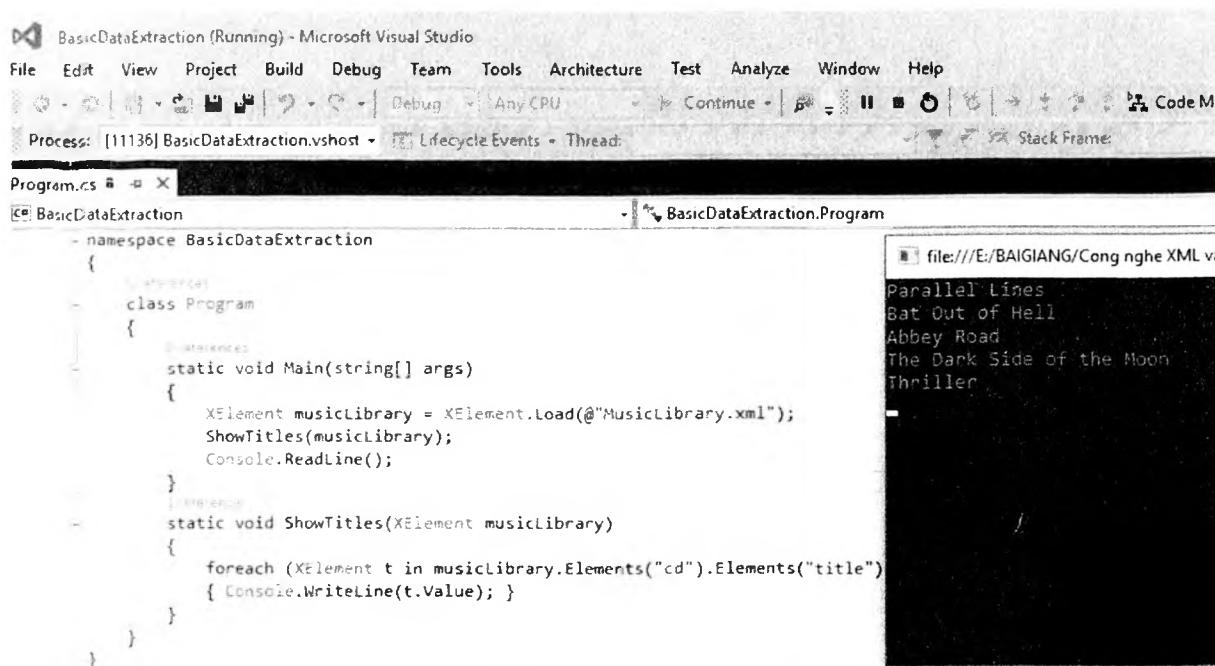
- Tiếp theo, thêm tệp tin *MusicLibrary.xml* vào dự án dưới dạng một mục hiện có. Thực hiện việc này bằng cách nhấp chuột phải vào dự án trong Solution Explorer và chọn Add \Rightarrow Existing Item... và duyệt qua tập tin như ở Ví dụ 10.3.

- Sau khi chúng ta thêm tệp này, hãy nhấp chuột phải vào tệp tin này trong Solution Explorer và chọn Properties. Trong cửa sổ Properties, hãy tìm cấu hình Copy to Output Directory và thay đổi thành Copy if Newer. Điều này đảm bảo rằng tệp tin kết thúc trong cùng thư mục với tệp thực thi, và có nghĩa là chúng ta có thể tham khảo nó trong mã bằng tên của nó chứ không phải là đường dẫn đầy đủ.

- Bây giờ chúng ta mở *Program.cs*, xóa tất cả mã mà nó hiện có và thay thế bằng mã sau đây:

```
using System;
using System.Linq;
using System.Xml.Linq;
namespace BasicDataExtraction
{
    class Program
    {
        static void Main(string[] args)
        {
            XElement musicLibrary =
                XElement.Load(@"MusicLibrary.xml");
            ShowTitles(musicLibrary);
            Console.ReadLine();
        }
        static void ShowTitles(XElement musicLibrary)
        {
            foreach (XElement t in
                musicLibrary.Elements("cd").Elements("title"))
            {
                Console.WriteLine(t.Value);
            }
        }
    }
}
```

5. Lưu tập tin và nhấn F5 để chạy ứng dụng. Chúng ta sẽ thấy một cửa sổ bảng điều khiển bật lên, như thể hiện trong Hình 10.2.



The screenshot shows the Microsoft Visual Studio interface with the title bar "BasicDataExtraction (Running) - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Architecture, Test, Analyze, Window, Help. The toolbar has icons for Save, Undo, Redo, Cut, Copy, Paste, Find, Replace, and others. The status bar at the bottom shows "Process: [11136] BasicDataExtraction.vshost - Lifecycle Events - Thread: Stack Frame:". The main code editor window displays the following C# code:

```
BasicDataExtraction.cs
BasicDataExtraction
    namespace BasicDataExtraction
    {
        class Program
        {
            static void Main(string[] args)
            {
                XElement musicLibrary = XElement.Load(@"MusicLibrary.xml");
                ShowTitles(musicLibrary);
                Console.ReadLine();
            }

            static void ShowTitles(XElement musicLibrary)
            {
                foreach (XElement t in musicLibrary.Elements("cd").Elements("title"))
                {
                    Console.WriteLine(t.Value);
                }
            }
        }
    }
```

To the right of the code editor is a black terminal window titled "file:///E/BAIGIANG/Cong nghe XML vi...". It contains the following output:

```
Parallel Lines
Bat Out of Hell
Abbey Road
The Dark Side of the Moon
Thriller
```

Hình 10.2: Kết quả khi chạy ứng dụng

Giai thích đoạn mã trên:

Để sử dụng các tính năng LINQ, chúng ta cần một tham chiếu đến *System.Xml.Linq.dll*. Điều này được thêm tự động vào dự án khi nó được tạo, nhưng chúng ta vẫn cần thêm câu lệnh sử dụng thứ hai trong đoạn mã dưới đây để có thể sử dụng mẫu ngắn của tên lớp (tức là *XElement* thay vì *System.Xml.Linq.XElement* và quan trọng hơn, để có thể truy cập các phương pháp mở rộng LINQ):

```
using System;
using System.Xml.Linq;
```

Phương thức *Main()* là điểm nhập ban đầu cho ứng dụng. Chúng ta sử dụng phương thức *Load()* của *XElement* để tải thư viện nhạc:

```
static void Main(string[] args)
{ XElement musicLibrary =

```

```
    XElement.Load(@"MusicLibrary.xml");
}
```

Sau đó được chuyển đến phương thức *ShowTitles()*:

```
{ static void Main(string[] args)
{ XElement

```

```
    musicLibrary

```

```
=
```

```
    XElement.Load(@"MusicLibrary.xml");
}
```

```
 ShowTitles(musicLibrary);
}
```

Phương thức *ShowTitles()* sử dụng phương thức *Elements()* hai lần. Phương thức này có hai tùy chọn, một không có tham số và một với tên của một phần tử. Nếu chúng ta không vượt qua tham số, nó trả về tất cả các con của phần tử; Nếu chúng ta vượt qua tên của một phần tử, nó trả về tất cả các phần tử có tên đó.

Trong đoạn code dưới đây chúng ta đã xác định các con có tên là *cd*, sau đó sử dụng *Elements()* để trích xuất các phần tử *<title>*:

```

static void ShowTitles(XElement musicLibrary) {
    foreach ( XElement t in
        musicLibrary.Elements("cd").Elements("title"))
        { Console.WriteLine(t.Value); }
    }
}

```

Khi các phần tử `<title>` được tìm thấy, chúng ta lặp lại tất cả chúng bằng cách sử dụng một `foreach` chuẩn và đưa ra giá trị.

Bởi vì có một lệnh `Console.ReadLine()` ở cuối của phương thức `Main()` nên chúng ta sẽ cần nhấn một phím, chẳng hạn như phím cách hoặc Enter để bỏ qua cửa sổ bảng điều khiển.

Phương thức `Elements()` điều hướng xuống trực con. Thay vì sử dụng phương thức `Elements()`, chúng ta có thể sử dụng `Descendants()`, nó sẽ lấy tất cả các con cháu chứ không chỉ là con.

Đoạn mã ở trên sẽ được viết lại giống như sau nếu chúng ta sử dụng `Descendants()` thay vì `Elements()`:

```

static void ShowTitles(XElement musicLibrary)
{
    foreach ( XElement t in
        musicLibrary.Descendants("title"))
        { Console.WriteLine(t.Value); } }
}

```

Một số phương thức thông dụng được thể hiện trong Bảng 10.1.

Bảng 10.1: Một số phương thức thông dụng trong LINQ

Tên phương thức	Mô tả
Ancestors	Trả về tất cả các phần tử tổ tiên
AncestorsAndSelf	Trả về tất cả tổ tiên và bao gồm cả phần tử hiện tại
Attributes	Trả về các thuộc tính của phần tử hiện tại
Descendants	Trả về các phần tử là hậu duệ của phần tử hiện tại
DescendantsAndSelf	Trả về tất cả con cháu và bao gồm phần tử hiện tại
DescendantNodes	Trả về tất cả các con cháu và bao gồm các loại nút khác như comments (không có thuộc tính)
Elements	Trả về các phần tử con của phần tử hiện tại
ElementsAfterSelf	Trả về một tập hợp các phần tử anh chị em đi sau phần tử này theo thứ tự tài liệu
ElementsBeforeSelf	Trả về một tập hợp các phần tử anh chị em đi trước phần tử này theo thứ tự tài liệu
Nodes	Trả về bất kỳ nút con nào của phần tử này
NodesAfterSelf	Trả về bất kỳ nút anh chị em nào đi sau phần tử này theo thứ tự tài liệu
NodesBeforeSelf	Trả về bất kỳ nút anh chị em nào đi trước phần tử này theo thứ tự tài liệu

Các phương thức bao gồm `Before` hoặc `After` được sử dụng khi chúng ta cần lấy các phần tử dựa trên thứ tự tài liệu của chúng. Ví dụ, giả sử chúng ta có một tham chiếu đến phần tử `<cd>` có id là 3 và chúng ta muốn hiển thị tiêu đề của tất cả các phần tử `<cd>` trước đó theo thứ tự tài liệu. Đoạn mã sau lấy ra phần tử `<cd>` thứ ba:

```

static void ShowTitlesBefore(XElement musicLibrary)
{ XElement cd3 = (from cd in
    musicLibrary.Elements("cd") where
    cd.Attribute("id").Value == "3"
    select cd).FirstOrDefault();
// code continued }

```

Ví dụ này sử dụng cách tích hợp các từ khóa LINQ chứ không phải sử dụng hàm. Đầu tiên, chúng ta chọn tất cả các phần tử <cd>, sau đó chúng ta kiểm tra thuộc tính *id* để xem nó bằng 3.

Một khi chúng ta đã tham chiếu tới phần tử <cd> chúng ta muốn, sử dụng phương thức *ElementsBeforeSelf()* để lấy các phần tử <cd> trước và phần tử <title> của chúng như thể hiện trong đoạn mã dưới đây:

```
static void ShowTitlesBefore(XElement musicLibrary)
```

```

{ XElement cd3 = (from cd in
    musicLibrary.Elements("cd")
    where cd.Attribute("id").Value == "3"
    select cd).FirstOrDefault();
foreach (XElement t in
    cd3.ElementsBeforeSelf("cd").Elements("title"))
    {Console.WriteLine(t.Value); } }
```

Sau đó chúng ta lặp lại bộ sưu tập và hiển thị giá trị của mỗi <title> như trước. Đoạn mã hiển thị tiêu đề cho phần tử <cd> có id 1 và 2.

Ví dụ tiếp theo sử dụng hàm để hiển thị tất cả các tiêu đề sau <cd> thứ ba. Nó cũng sử dụng *ElementsAfterSelf()* để tìm các anh chị em sau đĩa CD thứ ba trong tài liệu:

```
static void ShowTitlesAfter(XElement musicLibrary)
{ XElement cd3 = musicLibrary.Elements("cd").Where(cd
=> cd.Attribute("id").Value == "3").FirstOrDefault();
foreach (XElement t in
    cd3.ElementsAfterSelf("cd").Elements("title"))
    { Console.WriteLine(t.Value); } }
```

Việc lựa chọn các phần tử dựa trên một thuộc tính có thể là một điều đơn giản nhưng có nhiều tính năng nâng cao của LINQ, đặc biệt khi chúng áp dụng cho XML. Một trong những tính năng này là gom nhóm. Một yêu cầu chung khi xử lý bất kỳ dữ liệu nào là gom nhóm các mục dựa trên một đặc tính cụ thể.

Ví dụ, chúng ta muốn nhóm các đĩa CD của mình dựa trên thể loại của họ. Chúng ta có thể sử dụng các toán tử LINQ để hoàn thành nhiệm vụ này. Trước tiên, chúng ta nhóm các phần tử <cd> dựa trên phần tử <genre> như thể hiện trong đoạn mã sau:

```
static void GroupOnGenre(XElement musicLibrary)
{ var groupQuery = from cd in
    musicLibrary.Elements("cd")
    group cd by cd.Element("genre").Value into genreGroup
    orderby genreGroup.Key
    select new { Genre = genreGroup.Key,
    Titles = from title in genreGroup.Elements("title")
    select title.Value };
```

Ở đây chúng ta chọn các phần tử `<cd>` như trước, nhưng thêm một toán tử `group` nhóm bằng cách sử dụng giá trị của phần tử `<genre>` làm thuộc tính để nhóm vào.

Kết quả được lưu trữ trong `genreGroup`. Sau đó chúng được sắp xếp sử dụng thuộc tính `Key` của biến nhóm được tạo ra bằng cách sử dụng LINQ.

Trong trường hợp này `Key` giữ giá trị của `genre`. Sử dụng `genreGroup` chúng ta tạo một kiểu ẩn danh có hai thành viên.

Thứ nhất, `Genre`, có cùng thuộc tính `Key` đã được sử dụng để phân loại. Thành viên thứ hai, `Titles`, sử dụng một truy vấn LINQ thứ hai để trích xuất tất cả các phần tử `<title>`.

Phần thứ hai của hàm được sử dụng để xuất các kết quả như thể hiện trong đoạn mã sau đây:

```
static void GroupOnGenre(XElement musicLibrary)
{
    var groupQuery = from cd in
                      musicLibrary.Elements("cd")
                    group cd by cd.Element("genre").Value into genreGroup
                    orderby genreGroup.Key
                    select new
                    {
                        Genre = genreGroup.Key,
                        Titles = from title in
                                  genreGroup.Elements("title")
                                select title.Value
                    };
    foreach (var entry in groupQuery)
    {
        Console.WriteLine("Genre: {0}", entry.Genre);
        Console.WriteLine("-----");
        foreach (var title in entry.Titles)
        {
            Console.WriteLine("\t{0}", title);
        }
        Console.WriteLine();
    }
}
```

Vòng lặp `foreach` bên ngoài duyệt qua tất cả các mục trong `groupQuery`. Câu lệnh sau đó xuất ra thuộc tính `Genre` và sử dụng một vòng lặp `foreach` thứ hai để lặp qua bộ sưu tập `Titles` để hiển thị mỗi tiêu đề trong nhóm.

Nếu chúng ta thêm các phương thức `ShowTitlesBefore()`, `ShowTitlesAfter()` và `GroupOnGenre()` vào tệp `Program.cs` gốc bên dưới phương thức `ShowTitles()` và nhấn F5 để chạy mã, chúng ta sẽ thấy các kết quả thể hiện trong Hình 10.3.

Chúng ta đã thấy làm thế nào để trích xuất các nút và các giá trị của chúng từ một tài liệu. Tính năng tiếp theo của LINQ là làm thế nào để sửa đổi một tài liệu XML.

```

using System;
using System.Linq;
using System.Xml.Linq;

namespace BasicDataExtraction
{
    class Program
    {
        static void Main(string[] args)
        {
            XElement musicLibrary = XElement.Load(@"MusicLibrary.xml");
            Console.WriteLine("All Titles\n=====");
            ShowTitles(musicLibrary);
            Console.WriteLine("\nTitles before CD 3\n=====");
            ShowTitlesBefore(musicLibrary);
            Console.WriteLine("\nTitles after CD 3\n=====");
            ShowTitlesAfter(musicLibrary);
            Console.WriteLine("\nTitles by Genre\n=====");
            GroupOnGenre(musicLibrary);
            Console.ReadLine();
        }

        static void ShowTitles(XElement musicLibrary)
        {
            // Output of the application window
            // The Dark Side of the Moon
            // Thriller
            //
            // Titles before CD 3
            // =====
            // Parallel Lines
            // Bat Out of Hell
            //
            // Titles after CD 3
            // =====
            // The Dark Side of the Moon
            // Thriller
            //
            // Titles by Genre
            // =====
            // Genre: New Wave
            // -----
            // Parallel Lines
            //
            // Genre: Pop
            // -----
            // Thriller
            //
            // Genre: Rock
            // -----
            // Bat Out of Hell
            // Abbey Road
            // The Dark Side of the Moon.
        }
    }
}

```

Hình 10.3: Kết quả chạy ứng dụng hiển thị các Title

10.3.2. Sửa đổi tài liệu XML với LINQ

LINQ có rất nhiều phương pháp cho phép chúng ta sửa đổi một tài liệu XML hiện có. Điều này có nghĩa là chúng ta có thể thêm các nút mới, xóa các nút hiện có và cập nhật giá trị của thuộc tính và nội dung văn bản.

10.3.2.1. Thêm nội dung vào tài liệu XML

Một trong những hoạt động phổ biến nhất là thêm một nút mới. Chúng ta có thể thử điều này bằng cách thêm một `<cd>` cho thư viện nhạc của chúng ta. Để làm như vậy, thực hiện các bước sau:

- Trước tiên, sử dụng đoạn mã sau để tạo ra một phương thức để trả về một `<cd>` `XElement` mới với các giá trị có liên quan như `id`, `title` và `year`:

```

static XElement CreateCDElement(string id, string title, int year, string artist, string genre)
{
    return new XElement("cd",
        new XAttribute("id", id),
        new XElement("title", title),
        new XElement("year", year),
        new XElement("artist", artist),
        new XElement("genre", genre));
}

```

Tiếp theo, sử dụng phương thức `Add()` của `XElement` để thêm phần tử mới vào phần tử `<cd>` hiện có như sau:

```

static void AddNewCD(XElement musicLibrary)
{
    XElement cd = CreateCDElement("6", "Back in
                                    Black", 2003, "AC/DC", "Rock");
    musicLibrary.Add(cd);
}

```

Kết quả của đoạn mã này là làm cho một phần của tập tin `MusicLibrary.xml` trông như sau:

```

<musicLibrary>
    <cd id="1"> <title>Parallel Lines</title>
                <year>2001</year>

```

```

<artist>Blondie</artist>
<genre>New Wave</genre>  </cd>
<!-- cd elements 2, 3 and 4 -->
<cd id="5">  <title>Thriller</title>
  <year>2001</year>
  <artist>Michael Jackson</artist>
  <genre>Pop</genre>  </cd>
<cd id="6">
  <title>Back in Black</title>
  <year>2003</year>
  <artist>AC/DC</artist>
  <genre>Rock</genre>  </cd>
</musicLibrary>

```

10.3.2.2. Xóa nội dung từ tài liệu XML

Chúng ta đã thấy cách thêm nội dung, hãy thử ngược lại: xóa nội dung. Cách dễ nhất để thực hiện việc này là điều hướng đến nút mà chúng ta muốn xóa và gọi phương thức *Remove()*. Xem ví dụ sau:

```

static void RemoveCD(XElement musicLibrary)
{ XElement cd = (from entry in
    musicLibrary.Elements("cd") where
    entry.Attribute("id").Value == "6" select
    entry).FirstOrDefault();
if (null != cd) {cd.Remove();} }

```

Đoạn mã ở trên nhắm mục tiêu `<cd>` có `id` là 6, đó là `<cd>` chúng ta vừa mới thêm vào với phương thức *AddNewCD()*. Mã sau đó gọi phương thức *Remove()*, kết quả là chúng ta chỉ còn 5 phần tử `<cd>` trong `MusicLibrary.xml`.

Phương thức *Remove()* cũng hoạt động trên một tập các phần tử. Câu lệnh dưới đây loại bỏ tất cả các phần tử `<cd>` trong tài liệu:

```
musicLibrary.Elements("cd").Remove();
```

10.3.2.3. Cập nhật và thay thế nội dung trong tài liệu XML

Kỹ thuật cuối cùng là làm thế nào để cập nhật một tài liệu hiện có. Hai hoạt động cần được thực hiện thường xuyên: một là cập nhật dữ liệu trong tài liệu (giá trị của một thuộc tính hoặc nội dung văn bản của một phần tử); Và thứ hai là thay thế toàn bộ phần tử hoặc cây của các phần tử.

Chúng ta có một số cách để cập nhật nội dung văn bản của một phần tử. Một cách là sử dụng phương thức *ReplaceNodes()*, thay thế các nút của `XElement` mà nó được gọi. Giả sử chúng ta muốn cập nhật phần tử `<year>` của `Abbey Road CD`, trong đó có `id` là 3. Đoạn mã sau xác định phần tử này và thay đổi giá trị của `<year>` thành 1986:

```

static void UpdateYearWithReplaceNodes(Xelement
    musicLibrary)
{ XElement cd = (from entry in
    musicLibrary.Elements("cd") where
    entry.Attribute("id").Value == "3" select
    entry).FirstOrDefault();
cd.Element("year").ReplaceNodes("1986"); }

```

Phương thức *ReplaceNodes()* cũng có thể làm việc với các nút của cây chứ không chỉ đơn giản là nội dung văn bản.

Cách thứ hai để cập nhật văn bản là sử dụng phương thức *SetElementValue()* như sau:

```
static void UpdateYearWithSetElementValue(XElement musicLibrary)
{
    XElement cd = (from entry in
        musicLibrary.Elements("cd")
        where entry.Attribute("id").Value == "3" select
            entry).FirstOrDefault();
    cd.SetElementValue("year", "1987");}
```

Một lần nữa, chúng ta chỉ ra phần tử đích sử dụng truy vấn LINQ và sau đó sử dụng *SetElementValue()* trên phần tử cha của phần tử chúng ta muốn thay đổi. Phương pháp này cũng có những cách sử dụng khác. Chúng ta có thể loại bỏ một phần tử hoàn toàn bằng cách thiết lập các đối số thứ hai là null. Chúng ta cũng có thể tạo ra các phần tử mới. Nếu phần tử *<year>* của phần tử *<cd>* chúng ta đã chọn chưa tồn tại, nó sẽ được tự động tạo ra.

Có một kỹ thuật tương tự để cập nhật, tạo, hoặc xóa tên giá trị của thuộc tính: *SetAttributeValue()*. Nếu chúng ta muốn cập nhật *id* của phần tử Abbey Road *<cd>*, đoạn mã sau sẽ thực hiện được điều đó:

```
static void UpdateAttributeValue(XElement musicLibrary)
{
    XElement cd = (from entry in
        musicLibrary.Elements("cd") where
        entry.Attribute("id").Value == "3" select
            entry).FirstOrDefault();
    cd.SetAttributeValue("id", "7");}
```

Phương pháp cuối cùng để cập nhật nội dung tài liệu XML là sử dụng phương thức *ReplaceContent()*. Phương thức này sẽ thay thế nút hiện đang được chọn bằng một nội dung XML cụ thể. Ví dụ, nếu chúng ta muốn thay thế phần tử *<cd>* đầu tiên trong bộ sưu tập với một phần tử khác, chúng ta sẽ sử dụng *ReplaceContent()* như sau:

```
static void ReplaceCD(XElement musicLibrary)
{
    XElement cd = (from entry in
        musicLibrary.Elements("cd") where
        entry.Attribute("id").Value == "1"
        select entry).FirstOrDefault();
    cd.ReplaceWith( new XElement("cd",
        new XAttribute("id", 1),
        new XElement("title", "Back in Black"),
        new XElement("year", 2003),
        new XElement("artist", "AC/DC"),
        new XElement("genre", "Rock"))); }
```

Đoạn lệnh trên nhắm mục tiêu là phần tử *<cd>* đầu tiên, sau đó gọi *ReplaceContent()* và chuyển qua một cây mới.

10.4. Chuyển đổi tài liệu XML

Sử dụng kết hợp các kỹ thuật chúng ta đã thấy cho đến nay, chúng ta có thể chuyển đổi một tài liệu XML sang một định dạng khác bằng cách sử dụng LINQ. Nói chung, sử dụng LINQ không mạnh bằng cách sử dụng XSLT, nhưng có lợi thế là đơn giản hơn cho rất nhiều biến đổi và ngăn cản sự cần thiết phải học một mô hình lập trình hoàn toàn khác.

Trong phần này sẽ đưa chúng ta qua các bước chuyển đổi MusicLibrary.xml sang một định dạng khác.

10.4.1. Chuyển đổi XML với LINQ

Hiện tại đa số các nút trong tài liệu *MusicLibrary.xml* là phần tử, ngoại trừ thuộc tính *id* trên phần tử *<cd>*. Ví dụ sau sẽ cho chúng ta thấy làm thế nào để sử dụng LINQ để chuyển phần tử thành thuộc tính, trong đó mỗi phần tử *<cd>* sẽ như sau:

```
<newMusicLibrary>
  <cd id="1" year="2001" artist="Blondie"
      genre="New Wave">Parallel Lines</cd>
  <!-- more cd elements -->
</newMusicLibrary>
```

Về cơ bản, tất cả các thuộc tính của phần tử *<cd>*, ngoại trừ tiêu đề, được xác định bằng các thuộc tính. Tiêu đề chính nó chỉ là nội dung văn bản.

1. Bước đầu tiên là tạo một ứng dụng giao diện điều khiển mới trong Visual Studio như chúng ta đã làm trước đó. Tên dự án TransformingXml.

2. Một khi dự án này đã được tạo ra, hãy thêm tập tin *MusicLibrary.xml* như trước, và một lần nữa tìm thấy thiết lập *Copy to Output Directory* và thay đổi này thành *Copy if Newer*.

3. Mở *program.cs* từ *Solution Explorer* và thay thế các câu lệnh sử dụng hiện tại bằng ba lệnh sau:

```
using System;
using System.Linq;
using System.Xml.Linq;
```

Các bước này là tất cả những gì chúng ta cần để đảm bảo chúng ta có thể sử dụng cả các từ khoá LINQ chuẩn và các lớp từ LINQ, chẳng hạn như *XElement*.

4. Tiếp theo, thay thế phương thức *Main()* bằng phương thức *main* hiển thị ở đây:

```
Static void Main(string[] args)
{
    XElement musicLibrary =
    XElement.Load(@"MusicLibrary.xml");
    XElement newMusicLibrary =
        TransformToAttributes(musicLibrary);
    Console.WriteLine(newMusicLibrary);
    newMusicLibrary.Save(@"newMusicLibrary.xml");
    Console.ReadLine();
}
```

Đoạn mã trên tải *MusicLibrary.xml* và chuyển nó tới phương thức *TransformToAttributes()*. Phương thức này trả về *XElement* mới có chứa định dạng mới mong muốn. XML mới sẽ được ghi vào giao diện điều khiển và

cũng được lưu vào một tệp mới có tên newMusicLibrary.xml khi lệnh được chạy. Phương pháp thực hiện tất cả các công việc thực tế như sau:

```
static XElement TransformToAttributes(XElement musicLibrary)
{
    XElement newMusicLibrary =
        new XElement("newMusicLibrary",
            from cd in musicLibrary.Elements("cd")
            select new XElement("cd",
                new XAttribute("id", cd.Attribute("id").Value),
                new XAttribute("year", cd.Element("year").Value),
                new XAttribute("artist", cd.Element("artist").Value),
                new XAttribute("genre", cd.Element("genre").Value),
                cd.Element("title")));
    return newMusicLibrary; }
```

Bây giờ chúng ta có thể chạy dự án bằng cách nhấn F5. Cửa sổ giao diện điều khiển sẽ hiển thị kiểu XML mới và, nếu chúng ta tìm trong thư mục bin\debug bên dưới nơi chứa *program.cs*, chúng ta cần tìm thấy một tệp tin có tên *newMusicLibrary.xml*, ở định dạng mới.

10.4.2. Chuyển đổi XML bằng cách sử dụng các tính năng XML Literals của VB.NET

VB.NET có hai tính năng mà cho đến nay trong C# hoặc bất kỳ ngôn ngữ .NET khác không có, đó là các *XML Literals* và *Axis Properties*. *XML Literals* chứa các cách thức mới để tạo các tài liệu XML và các cách quản lý không gian tên dễ dàng hơn. *Axis Properties* có nghĩa là chúng ta có thể điều hướng qua một tài liệu và lấy các phần tử, thuộc tính và các giá trị của chúng bằng một cú pháp ngắn gọn.

XML Literals cho phép chúng ta nhúng XML trực tiếp vào mã. XML Literals cũng tạo điều kiện bao gồm các khai báo không gian tên.

Bắt đầu bằng một ví dụ đơn giản, *MusicLibrary*, như sau:

```
Dim musicLibrary As XElement =
<musicLibrary>
    <cd id="1"> <title>Parallel Lines</title>
        <year>2001</year> <artist>Blondie</artist>
        <genre>New Wave</genre> </cd>
    <cd id="2"> <title>Bat Out of Hell</title>
        <year>2001</year> <artist>Meatloaf</artist>
        <genre>Rock</genre> </cd>
    <cd id="3"> <title>Abbey Road</title>
        <year>1987</year> <artist>The Beatles</artist>
        <genre>Rock</genre> </cd>
    <cd id="4"> <title>The Dark Side of the Moon</title>
        <year>1994</year> <artist>Pink Floyd</artist>
        <genre>Rock</genre> </cd>
    <cd id="5"> <title>Thriller</title>
        <year>2001</year> <artist>Michael Jackson</artist>
        <genre>Pop</genre> </cd>
</musicLibrary>
```

Trong các đoạn mã trước, biến *musicLibrary* giống như *musicLibrary.xml* đã được nạp bằng cách sử dụng phương thức *Load()* được hiển thị trước đó. Trong các phần trước, biến đã được đặt là *System.Xml.Linq.XElement*, nhưng chúng ta có thể đã sử dụng một khai báo tiềm ẩn thay vào đó, giống như đoạn sau đây:

```
Dim musicLibrary =  
    <musicLibrary>  
        <cd id="1">  
            <title>Parallel Lines</title>  
            <year>2001</year>  
            <artist>Blondie</artist>  
            <genre>New Wave</genre>    </cd>  
            <!-- rest of cd elements -->  
    </musicLibrary>
```

Nếu chúng ta thử mã này và sau đó di chuột qua biến *musicLibrary*, chúng ta sẽ thấy rằng nó vẫn là *XElement*. Nếu chúng ta đã bao gồm một khai báo XML, hoặc bất kỳ dạng prolog nào, chẳng hạn như trong đoạn mã sau, *musicLibrary* sẽ được đánh máy như *System.Xml.Linq.XDocument*:

```
Dim musicLibrary =  
    <?xml version="1.0" encoding="utf-8"?>  
    <musicLibrary> <cd id="1">  
        <title>Parallel Lines</title>  
        <year>2001</year> <artist>Blondie</artist>  
        <genre>New Wave</genre> </cd>  
        <!-- rest of cd elements -->  
    </musicLibrary>
```

Tuy nhiên, nhung một tập tin hoàn chỉnh như thế này là bất thường. Có nhiều lúc chúng ta cần kết nối với dữ liệu từ một nguồn bên ngoài. *XML Literals* sẽ giúp chúng ta dễ dàng làm điều này.

Trong ví dụ sau, chúng ta tạo thư viện nhạc bằng cách sử dụng một nguồn dữ liệu bên ngoài và kết hợp nó với một *XML Literal*. Kịch bản này thường được nhìn thấy khi chúng ta cần trình bày dữ liệu nằm trong CSDL quan hệ trong một định dạng XML.

1. Nếu chúng ta đang sử dụng phiên bản đầy đủ của Visual Studio thì tạo một dự án mới bằng cách sử dụng phần Visual Basic. Nếu không, tạo một dự án mới bằng Visual Basic Express. Dự án sẽ là một ứng dụng giao diện điều khiển mà chúng ta nên đặt tên *VbXmlFeatures*.

2. Chúng ta cần một lớp đại diện cho dữ liệu CD của chúng ta, vì vậy hãy mở *Module1.vb* và thêm mã sau trong *Module1/End*:

```
Private Class CD  
    Public Property ID As String  
    Public Property Title As String  
    Public Property Year As Integer  
    Public Property Artist As String  
    Public Property Genre As String  
End Class
```

Lớp CD, được đánh dấu riêng vì nó chỉ được sử dụng trong mô đun, đơn giản xác định các thuộc tính cần thiết cho XML của mỗi phần tử *<cd>*.

3. Bây giờ chúng ta cần một hàm mô phỏng thu thập dữ liệu từ một nguồn bên ngoài như là một cơ sở dữ liệu. Đối với ví dụ này, chúng ta sẽ chỉ cần mã hóa dữ liệu như thể hiện ở đây:

```
Private Function GetCDs() As List(Of CD)
    Dim cdList As New List(Of CD) From {
        New CD() With {.ID = "1", .Title = "Parallel Lines",
        .Year = 2001, .Artist = "Blondie", .Genre = "New Wave"}, 
        New CD() With {.ID = "2", .Title = "Bat Out of Hell",
        .Year = 2001, .Artist = "Meatloaf", .Genre = "Rock"}, 
        New CD() With {.ID = "3", .Title = "Abbey Road",
        .Year = 1987, .Artist = "The Beatles", .Genre = "Rock"}, 
        New CD() With {.ID = "4", .Title = "The Dark Side of
        the Moon", 
        .Year = 1994, .Artist = "Pink Floyd", .Genre = "Rock"}, 
        New CD() With {.ID = "5", .Title = "Thriller",
        .Year = 2001, .Artist = "Michael Jackson", .Genre =
        "Pop"} }
    Return cdList
End Function
```

4. Bây giờ cho các chức năng chính kết hợp dữ liệu với một XML Literal, thêm mã sau đây vào Module1:

```
Private Function CreateMusicLibrary() As XElement
    Dim cdData = GetCDs()
    Dim musicLibrary = <musicLibrary>
        <%= From item In cdData
Select <cd id=<%= item.ID %>> <title><%= item.Title %>
        </title>
        <year><%= item.Year %></year>
        <artist><%= item.Artist %></artist>
        <genre><%= item.Genre %></genre>
    </cd> %
</musicLibrary>
Return musicLibrary
End Function
```

5. Cuối cùng, mã được khởi tạo từ điểm nhập vào module, Sub Main():

```
Sub Main()
    Dim musicLibrary As XElement = CreateMusicLibrary()
    Console.WriteLine("XML Literals with Placeholders" &
        vbCrLf & "=====")
    Console.WriteLine(musicLibrary)
    Console.WriteLine(vbCrLf & "Shortcuts Demo" &
    vbCrLf & "=====")
    ShortcutsDemo()
    Console.WriteLine(vbCrLf & "Managing Namespaces" &
        vbCrLf & "=====")
    NamespaceDemo()
    Console.ReadLine()
End Sub
```

6. Để chạy mã nhấn F5 và xem kết quả trong cửa sổ bảng điều khiển xuất hiện

10.4.3. Hiểu thuộc tính Axis trong VB.NET

Axis Properties là một tính năng XML khác chỉ được tìm thấy trong VB.NET. Chúng được sử dụng để làm chuyển hướng thông qua một tài liệu XML dễ dàng hơn cũng như để tạo điều kiện thu thập các giá trị từ XML.

Bốn trục thuộc tính trong các tính năng XML của VB.NET được dùng để trích xuất dữ liệu từ một nguồn XML. Ba trong số chúng có dạng các phím tắt có thể được sử dụng thay cho các phương thức khác nhau Elements(), Attributes() và Descendants() và thứ tư là cách tiện lợi để lấy ra phần tử hay giá trị của thuộc tính. Cách thứ tự được biết như sau:

- Child Axis
- Attribute Axis
- Descendants Axis
- Value Property

Các phần sau sẽ giải thích từng phím tắt chi tiết hơn và cung cấp một ví dụ về cách sử dụng mỗi phím tắt.

Sử dụng Child Axis

Phím tắt thuộc tính đầu tiên được sử dụng khi chúng ta muốn truy cập các phần tử nằm trên trực con.

Nếu chúng ta đã nạp thư viện nhạc vào bộ nhớ và muốn truy cập vào tất cả các phần tử <cd>, chúng ta vẫn sử dụng mã sau đây:

```
musicLibrary.Elements("cd")
```

Sử dụng Child Axis, chúng ta có thể viết:

```
musicLibrary.<cd>
```

Điều này ngắn hơn và dễ đọc hơn, nhưng thực hiện cùng chức năng.

Sử dụng Attribute Axis

Phím tắt tiếp theo được sử dụng để lấy các thuộc tính. Trước đây, để xác định thuộc tính chúng ta đã sử dụng phương thức *Attributes()* hoặc *Attribute()*. Để hiển thị thuộc tính *id* của một phần tử <cd>, chúng ta đã sử dụng như sau:

```
cd3.Attribute("id")
```

Sử dụng Attribute Axis, chúng ta có thể viết như sau:

```
cd3.@id
```

Lệnh trên sử dụng ký hiệu quen thuộc @ được sử dụng trong XPath để biểu thị chúng ta đang tìm kiếm các bộ sưu tập thuộc tính.

Sử dụng Descendants Axis

Không có gì ngạc nhiên, Descendants Axis được sử dụng để tìm ra con cháu.

Với Children thì nút con được giới hạn ở mức thấp hơn một phần tử, còn với Descendant thì con cháu có thể ở bất cứ đâu bên dưới. Trong mã trước đó, chúng ta phải viết đoạn sau để tìm tất cả các phần tử <title> bắt đầu nơi nào bên dưới <musicLibrary>:

```
MusicLibrary.Descendants ("title")
```

Bây giờ với Descendants Axis, chúng ta có thể sử dụng ba dấu chấm (...) làm phím tắt:

```
MusicLibrary ... <title>
```

Sử dụng Value Property

Phím tắt cuối cùng, được gọi là *Value Property* của Microsoft nhưng thực sự chỉ hoạt động trên các giá trị, cho phép xác định giá trị của một mục nhanh hơn.

Nếu chúng ta lấy ra một bộ sưu tập các phần tử, chúng ta thường cần phải sử dụng *FirstOrDefault()* hoặc một bộ chỉ mục để tìm mục đầu tiên và sau đó sử dụng thuộc tính *Value* để lấy nội dung của nó. Ví dụ: để lấy giá trị phần tử `<title>` đầu tiên chúng ta sử dụng:

```
MusicLibrary ... <title> (0) .Value
```

Value Property loại bỏ sự cần thiết của trình chỉ mục và lấy giá trị của phần tử hoặc thuộc tính đầu tiên trong bộ sưu tập. Đoạn mã sau cho kết quả tương tự như đoạn mã trước đó:

```
MusicLibrary ... <title> .Value
```

10.5. Một số ứng dụng liên quan XML

10.5.1. Đọc tin nhanh RSS

RSS là một định dạng tập tin thuộc họ XML dùng trong việc chia sẻ tin tức Web (Web syndication) được dùng bởi nhiều website tin tức và weblog. Chữ viết tắt (theo tiếng Anh) dùng để chỉ các chuẩn sau:

Rich Site Summary (RSS 0.91)

RDF Site Summary (RSS 0.9 and 1.0)

Really Simple Syndication (RSS 2.0.0)

RSS là một chủ đề rất nhiều người quan tâm, dễ tham khảo. Trong phần này chúng ta chỉ xét những vấn đề chính:

- Khuôn mẫu một tệp RSS đơn giản.
- Xây dựng một trang ASP để đọc RSS, vì là một tệp có cấu trúc như tệp XML nên hoàn toàn có thể sử dụng mô hình DOM để lập trình.

Cấu trúc tệp RSS (extension: .rss hay .xml, các thẻ là quy định) như sau:

```
<?xml version="1.0" encoding="utf-8" ?>
<rss version="2.0">
<channel>
    <title>Danang College of Technology</title>
    <link>http://www.dct.udn.vn</link>
    <description>Technology RSS</description>
    <item> <title>Science-Technology</title>
            <link>http://vietnamnet.vn/khoa hoc/index.rss
        </link>
        <description>New RSS Science-Technology on
            VNN</description> </item>
    <item>
        <title>Information Technology</title>
        <link>http://www.pcworld.com.vn</link>
        <description>PCWorld Viet Nam</description>
    </item>
</channel>
<! - - Item khác - ->
</rss>
```

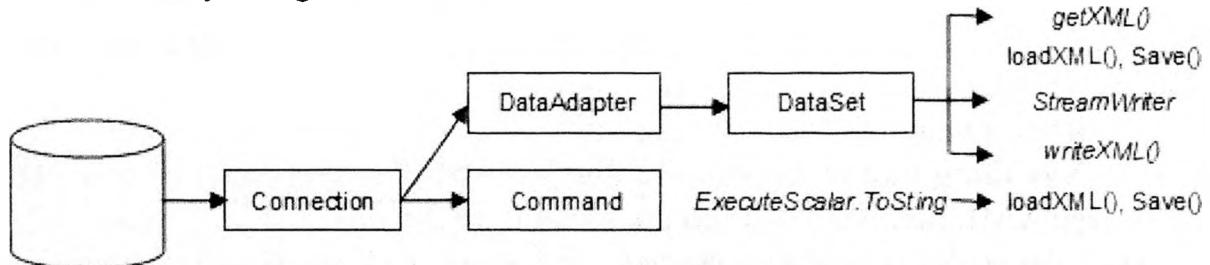
Tệp Read_RSS.asp (chỉ nêu phần chính)

```
<% theFeed = "http://localhost/ASP/RSS/ndtfit.xml"
' tên tệp RSS
' có thể dùng phần mở rộng .xml thay cho .rss
Set objXML = Server.CreateObject("Microsoft.XMLDOM")
objXML.Load(theFeed)
set objRoot = objXML.documentElement
set objItems = objRoot.getElementsByTagName("item")
NewsItem=""
For each objItem in objItems theTitle =
    objItem.selectSingleNode("title").Text
    theLink = objItem.selectSingleNode("link").Text
    theDesc = objItem.selectSingleNode("description").Text
    NewsItem= NewsItem & "<a href=" & theLink & ">" &
    & theTitle & "</a>" & "<br />" & theDesc & "<br /><hr />"
    Response.Write(NewsItem) ' In một News
Next
set objItems = Nothing : set objXML = Nothing %>
```

10.5.2. Chuyển đổi CSDL quan hệ sang XML

10.5.2.1. Tổng quan các phương pháp chuyển đổi CSDL quan hệ sang XML

Tổng quan về các phương pháp chuyển đổi CSDL quan hệ sang XML được trình bày trong Hình 10.4.



Hình 10.4: Các phương pháp chuyển đổi CSDL quan hệ sang XML

Cốt lõi của phương pháp là chuyển đổi dữ liệu lưu trong *DataSet* hay dữ liệu trả về của phương thức *ExecuteScalar* thành một Stream XML.

- Với đối tượng *StreamWriter* lấy dữ liệu trong *DataSet*. *Tables(0).Rows(0).Item(0)* và lưu thành tệp XML. Sớ dĩ chọn chỉ số 0 của các đối tượng vì câu lệnh Select ... for XML trả về một bảng chỉ có một dòng, một cột.
- Với đối tượng *DataSet* thì bản thân nó chứa dữ liệu dưới dạng XML, phương thức *getXML()* lấy thông tin trong *DataSet* và cho lại một Stream XML. Lúc này sử dụng các phương thức của lớp *XmlDocument* để nạp vào cây XML và lưu thành tệp XML.

Sau đây là các ví dụ:

10.5.2.2. Sử dụng đối tượng *Command*

Thể mạnh của đối tượng *Command* là nó cho phép thực thi câu lệnh SQL, trong đó câu lệnh *Select ... for XML* cho phép kết xuất kết quả ra một XML Stream. Tùy theo nhu cầu cụ thể, chúng ta sử dụng câu lệnh này với các nhiệm ý phù hợp.

Cú pháp : Select ... from ... for XML auto | raw | explicit [,xmldata | , elements]

Giả sử chúng ta có cơ sở dữ liệu SQL Server QLSV trong đó có bảng SV(Maso, Hoten, Lop).

- Select * from SV for XML auto cho kết quả:

```
<SV Maso="01" Hoten="Võ Văn Quân" Lop="08SPT"/>
<SV Maso="02" Hoten="Nguyễn Tiệp" Lop="08CNTT1"/>
```

- Select * from SV for XML auto, elements cho kết quả:

```
<SV><Maso>01</Maso> <Hoten>Võ Văn Quân</Hoten>
<Lop>08SPT</Lop></SV>
<SV><Maso>02</Maso> <Hoten>Nguyễn Tiệp</Hoten>
<Lop>08CNTT1</Lop></SV>
```

- Select * from SV for XML raw cho kết quả:

```
<row Maso="01" Hoten="Võ Văn Quân" Lop="08SPT"/>
<row Maso="02" Hoten="Nguyễn Tiệp" Lop="08CNTT01"/>
```

- Select * from SV for XML raw, elements cho kết quả:

```
<row><Maso>01</Maso>
<Hoten>Võ Văn Quân</Hoten>
<Lop>08SPT</Lop></raw>
<row><Maso>02</Maso>
<Hoten>Nguyễn Tiệp</Hoten>
<Lop>08CNTT1</Lop></raw>
```

Các phương pháp trên cho lại một XML Stream nhưng thiếu *chỉ thị xử lý* và *nút gốc của cây XML* và chúng ta phải bổ sung trước khi lưu tệp XML. Ví dụ:

```
Imports System.Xml
Imports System.Data.SqlClient
Module Module1
    Dim xmlDoc As New XmlDocument
    Sub Main()
        Dim objConn As New SqlConnection("Data _ 
Source=NDT\SqlExpress;Database=QLSV;Integrated 
Security=SSPI;")
        objConn.Open()
        Dim objCmd As New SqlCommand("select * from SV for 
xml auto, _ elements", _ objConn)
        Dim strXML As String = objCmd.ExecuteScalar.ToString
        strXML = "<?xml version='1.0'?>" 
        strXML &= "<UED>" & strXML & "</UED>" 
        xmlDoc.LoadXml(strXML)
        xmlDoc.Save("D:\UED.XML")
    End Sub
End Module
```

Trong môi trường VB 6.0 hay VbScript chúng ta cũng có thể thực hiện việc chuyển đổi như trên, nhưng mã nguồn dài hơn khá nhiều.

10.5.2.3. Sử dụng đối tượng DataSet

Giả sử rằng cơ sở dữ liệu QLSV được đặt trên máy NDT\SQLExpress và được phân quyền Select cho user có tên “TH” và mật khẩu ”1234”.

a- Dùng phương thức WriteXML của đối tượng DataSet

```
Imports System.Xml
Imports System.Data.SqlClient
...
Dim strCon As String = "Data Source=NDT\SQLExpress;Initial Catalog=SV;_
    User ID=TH;Password=1234; Trusted_Connection=Yes"
Dim connection As New SqlConnection(strCon)
Dim adapter As New SqlDataAdapter
Dim ds As New DataSet
Dim strSql As String
strSql = "Select * from SV"
Try
    connection.Open()
    adapter = New SqlDataAdapter(strSql, strCon)
    adapter.Fill(ds)
    connection.Close()
    ds.WriteXml("D:\UED.XML")
    MsgBox("Done!")
Catch ex As Exception
    MsgBox(ex.ToString())
End Try
```

b- Dùng đối tượng StreamWriter

```
Imports System.Xml
Imports System.Data.SqlClient
Imports System.IO
...
Dim connection As SqlConnection
Dim adapter As SqlDataAdapter
Dim ds As New DataSet
Dim sql As String
Dim strXML As String
Dim SqlCon As String = "Data Source=NDT\SQLExpress;Initial Catalog=SV;_
    User ID=TH;Password=1234; Trusted_Connection=Yes"
connection = New SqlConnection(SqlCon)
sql = "select * from SV for xml auto"
Try connection.Open()
    adapter = New SqlDataAdapter(sql, SqlCon)
    adapter.Fill(ds)
    connection.Close()
    strXML = "<?xml version='1.0' ?>" & _
        "<UED>" & ds.Tables(0).Rows(0).Item(0) &
        "</UED>"
    Using writer As StreamWriter = New
        StreamWriter("D:\UED.xml")
        writer.Write(strXML)
    End Using
    MsgBox("Done!")
Catch ex As Exception
    MsgBox(ex.ToString())
End Try
```

c- Dùng phương thức getXML của đối tượng DataSet

Ví dụ sau đây dùng cơ sở dữ liệu MS Access 2003 để minh họa cho cách sử dụng OLEDB

```
Imports System.Xml
Imports System.Data.SqlClient
...
Dim strSQL As String = "Select * from SV"
Dim path As String = "D\QLSV.MDB"
Dim cbjDataSet As New DataSet
Dim cbjCon As OleDbConnection
Dim strCon As String
Dim xmlDoc As New XmlDocument
strCn = "Provider=Microsoft.Jet.OleDb.4.0;Data Source=" &
path
objCn = New OleDbConnection(strCon)
Dim cbjAdapter As New OleDbDataAdapter(strSQL, objCon)
objAcapter.Fill(objDataSet)
objCn.Close()
xmlDcc.LoadXml(objDataSet.GetXml())
xmlDcc.Save("D:\UED.XML")
MsgBox("Export completed!")
```

10.5.2.4. Sử dụng CSDL MySQL

Việc chuyển đổi dữ liệu MySQL thành dữ liệu XML với PHP đã nêu trên, tuy nhiên trong môi trường Net Framework cần tiến hành một số cài đặt bổ sung: Tài phần mềm *mysql-connector-net.exe* tại địa chỉ: (chọn phiên bản phù hợp): <http://dev.mysql.com/downloads/connector/net/1.0.html>

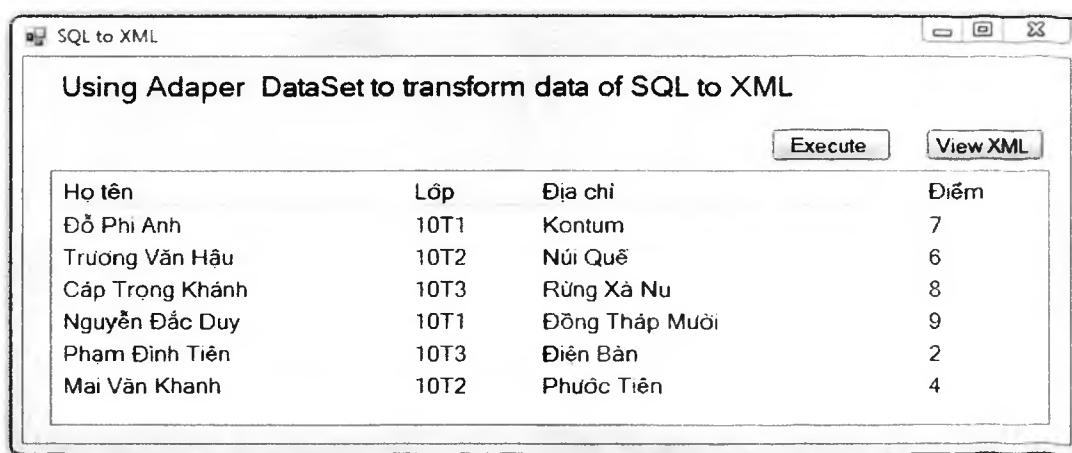
Khi tham chiếu name space, thực thi các phương thức, sử dụng các thuộc tính chú ý đến tiền tố MySQL thay vì SQL. Sau đây là một ví dụ:

```
Imports MySql.Data.MySqlClient
Imports System.IO
Dim myCommand As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter
Dim myData As New DataSet
Dim table As String = "SV"
Dim strSQL As String = "Select * from SV"
Dim conn = New MySqlConnection()
conn.ConnectionString = "server=localhost; user
id=root; password='';_
database=qlsv"
Try conn.Open()
Try
myCommand.Connection = conn
myCommand.CommandText = strSQL
myAdapter.SelectCommand = myCommand
myAdapter.Fill(myData, table)
myData.WriteXml("C:\ued.xml")
MsgBox("Done!")
Catch myerror As MySqlException
```

```
MsgBox("There was an error reading from the  
database" & myerror.message )  
End Try  
Catch myerror As MySqlException  
    MessageBox.Show("Error connecting to the database")  
Finally  
    If conn.State <> ConnectionState.Closed Then  
        conn.Close()  
    End Try
```

Giả sử ta xây dựng một ứng dụng Window với VB.NET, Form có dạng như Hình 10.5. Ở đây có 2 điều khiển:

- *Button*: Execute (chuyển dữ liệu SQL Server->XML) và View XML (Xem dữ liệu của tệp XML trong ListView)
 - *ListView* : để trình bày dữ liệu XML.
 - Cơ sở dữ liệu *OLSV.MDF* có table *SV(HoTen, Lop, QueQuan, Diem)*



Hình 10.5: Form chuyển đổi CSDL sang XML

- Khởi tạo ListView khi load Form

- Đọc dữ liệu trong CSDL SQL Server và chuyển sang tên XML.

Đọc và học

- *Kết nối CSDL*: Khai báo Connection String phù hợp với loại CSDL

Open Connection

Open Connectio
Tạo đối tượng Adapter

Chú ý hai tham số: Chuỗi kết nối và câu lệnh SQL

- *Điền đầy (fill) dữ liệu từ Adapter vào đối tượng Dataset*
- *Chuyển dữ liệu trong Dataset sang tệp XML*

```

Private Sub btnExecute_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim path As String = "../../SV.XML"
    Dim strCon As String =
        "Data Source=NDT\SQLExpress;
        Initial Catalog=SV;
        User ID=TH;Password=;
        Trusted_Connection=True"
        ' User name là TH, đăng nhập với quyền của Windows
    Dim connection As New SqlConnection(strCon)
    Dim adapter As New SqlDataAdapter
    Dim ds As New DataSet
    Dim strSql As String
        strSql = "Select * from SV"
    Try
        connection.Open()
        adapter = New SqlDataAdapter(strSql, strCon)
        adapter.Fill(ds)
        connection.Close()
        ds.WriteXml(path)
        MsgBox("Done!")
    Catch ex As Exception
        MsgBox(ex.ToString)
    End Try
End Sub

```

10.5.3. Xây dựng ứng dụng quản lý với XML

10.5.3.1. Ý tưởng và công cụ

Trong phần này, chúng ta sẽ tìm hiểu cách xây dựng ứng dụng quản lý sách với công cụ VB.Net

- Chuẩn bị tệp BOOKS.XML

```

<?xml version="1.0"?>
<PN tên="Nhà sách Phương Nam">
    <book ID="01">
        <title>Trăm năm cô đơn </title>
        <author>Gabriel Market</author>
        <price>767</price>
    </book>
    <!-- Other book -->
</PN>

```

- Công cụ : Lable, Button và đối tượng ListView



- Form của ứng dụng có khuôn dạng như hình dưới:

Mã số	Tên sách	Tác giả	Đơn giá
01	Trăm năm cô đơn	Gabriel Market	767
02	Mùa lá rụng trong vườn	Ma Văn Kháng	125
03	Giáo trình Java toàn tập	Nguyễn Văn Lành	900

Hình 10.6: Form của ứng dụng quản lý sách với XML

Kết chương

Trong chương này, chúng ta đã học được những nội dung sau:

- LINQ được thiết kế để thống nhất truy cập và thao tác các bộ sưu tập dữ liệu từ nhiều nguồn khác nhau.
- LINQ là cần thiết để tạo ra các tài liệu XML đơn giản hơn và để làm cho chuyển hướng và truy xuất dữ liệu từ XML một quy trình tương tự với bất kỳ dữ liệu lấy từ bất kỳ bộ sưu tập nào khác.
- Hàm tạo ra các tài liệu XML với lớp XElement giúp đơn giản hóa để xác định một tài liệu XML.
- Sử dụng LINQ to XML để trích xuất dữ liệu được thực hiện chủ yếu thông qua các *Elements()* và *Attributes()*.
- Sử dụng LINQ để sửa đổi dữ liệu được thực hiện bằng cách sử dụng các phương pháp như *ReplaceNodes()* và *SetElementValue()*.
- Các tính năng XML bổ sung của VB.NET là XML Literals, để khai báo các tài liệu XML và các thuộc tính Axis để đơn giản hóa việc điều hướng tới một mục tiêu và lấy giá trị của nó.

BÀI TẬP CHƯƠNG 10

1. Tạo các tệp dữ liệu sv.xml, hp.xml và kq.xml được lưu trong thư mục Debug, dữ liệu như sau:

sv.xml

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

<data>
```

```
<student scode="S01" sname="Nguyễn Thái Minh Anh"
class="11SPT"/>
<student scode="S02" sname="Ngô Thị Thu Hiền" class="11SPT"/>
<student scode="S03" sname="Nguyễn Đức Tiên" class="11SPT"/>
<student scode="S04" sname="Phạm Mạnh Linh" class="11T2"/>
<student scode="S05" sname="Phan Văn Thai" class="11T2"/>
<student scode="S06" sname="Huỳnh Bá Minh Thiện" class="11T2"/>
<student scode="S05" sname="Nguyễn Thị Kim Anh" class="11T2"/>
</data>
```

hp.xml

```
<?xml version="1.0" encoding="utf-8"?>
<data>

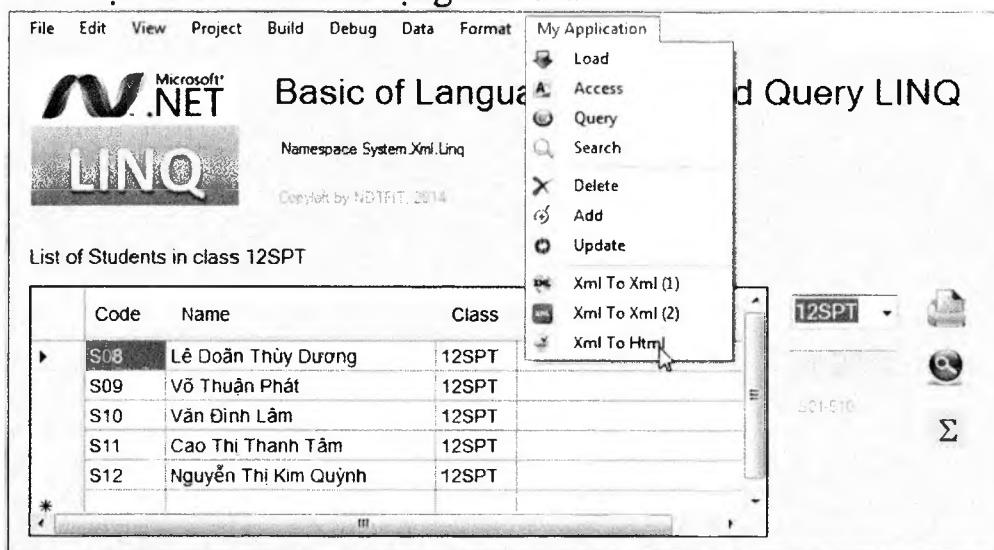
<course ccode="H01" cname="Ngôn ngữ lập trình Java" credit="3"/>
<course ccode="H03" cname="Phương pháp giảng dạy" credit="2"/>
<course ccode="H02" cname="Cơ sở dữ liệu" credit="3"/>
<course ccode="H04" cname="Công nghệ XML và Ứng dụng"
credit="3"/>
<course ccode="H05" cname="Hệ phân tán" credit="3"/>
</data>
```

kq.xml

```
<?xml version="1.0" encoding="utf-8"?>
<data>

<result scode="S01" ccode="H010214">9</result>
<result scode="S01" ccode="H020114">3</result>
<result scode="S01" ccode="H030214">7</result>
<result scode="S01" ccode="H020214">2</result>
<result scode="S01" ccode="H020314">7</result>
<result scode="S02" ccode="H010214">6</result>
<result scode="S02" ccode="H020214">7</result>
<result scode="S02" ccode="H030214">8</result>
<result scode="S03" ccode="H010214">9</result>
<result scode="S03" ccode="H020114">6</result>
<result scode="S03" ccode="H030214">7</result>
<result scode="S03" ccode="H040214">8</result>
<result scode="S03" ccode="H050314">5</result>
</data>
```

2. Thiết kế một form có khuôn dạng như sau:



Các điều khiển

frmLinq: tên Form

MenuStrip1 : đối tượng tạo menu (không cần đổi tên, sẽ hướng dẫn trên lớp)

dgvStudent : đối tượng DataGridView

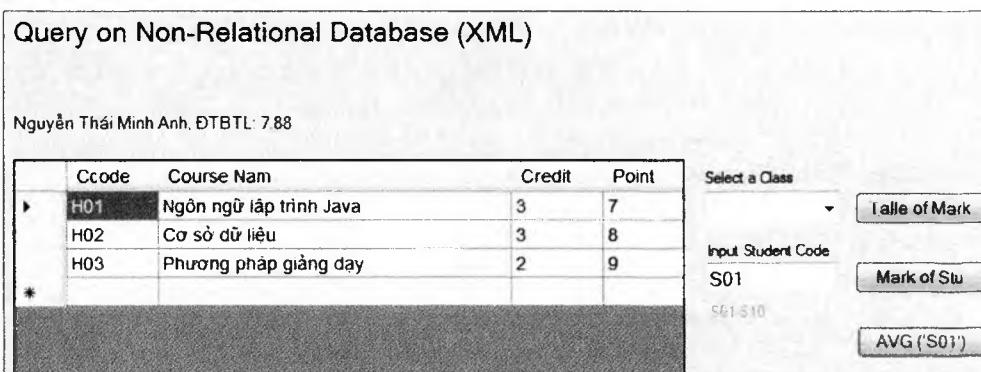
lblInfo : đối tượng Label

cboClass : đối tượng ComboBox

txtCode : đối tượng Text Box

ptbList, ptbSearch, ptbAggregate: các đối tượng Picture Box (các ảnh có trong thư mục)

3. Xây dựng một form có khuôn dạng như minh họa.



Gồm các đối tượng:

frmLinqPlus: Form

dgvStudent: DataGridView

lblInfo : Label

cboCourse: ComboBox, cho phép chọn lớp

btnTable : Button, ra lệnh đưa bảng điểm của một học phần vào dgvStudent

txtScode : Text Box, cho phép nhập mã số sinh viên

btnMark : Button, ra lệnh đưa kết quả học tập của sinh viên vào dgvStudent

btnTestAvg : Button tính điểm trung bình tích lũy của sinh viên có mã số trong text box, hiển thị những môn học đã tích lũy.

Cơ sở dữ liệu: gồm các tệp *ht.xml*, *sv.xml*, *kq.xml*

Chương 11

XML VÀ WEB NGỮ NGHĨA

XML là lớp nền tảng cú pháp của Web ngữ nghĩa. Tất cả các công nghệ khác cung cấp các tính năng cho Web ngữ nghĩa sẽ được xây dựng trên đầu trang của XML.

Yêu cầu các công nghệ Web ngữ nghĩa khác được xếp lớp trên đầu trang của XML đảm bảo mức cơ sở của khả năng tương tác.

11.1. Web ngữ nghĩa là gì?

Theo Tim Berners-Lee: Web ngữ nghĩa là sự mở rộng của Web hiện tại mà trong đó thông tin được định nghĩa rõ ràng sao cho con người và máy tính có thể cùng làm việc với nhau một cách hiệu quả hơn.

Theo định nghĩa của Tổ chức World Wide Web (W3C):

"Web ngữ nghĩa là một cách nhìn về cách thức tổ chức dữ liệu: đó là ý tưởng về việc dữ liệu trên Web được định nghĩa và liên kết theo một cách mà nó có thể được sử dụng bởi máy tính với mục đích không chỉ cho việc hiển thị mà còn tự động hóa, tích hợp và sử dụng lại dữ liệu qua các ứng dụng khác nhau".

Vì sao phải mở rộng Web hiện tại Web ngữ nghĩa hay còn được gọi là Web 3.0 là mở rộng của Web hiện tại (Web 2.0).

Web hiện tại có các đặc điểm và hạn chế như sau:

- Lượng tài nguyên trên web vô cùng lớn.
- HTML chỉ trình bày thông tin chứ không mô tả thông tin.
- Thiếu cơ cấu hiệu quả để chia sẻ dữ liệu vì các ứng dụng được phát triển một cách độc lập.

Do đó cần phải mở rộng thế hệ web hiện tại để máy tính có thể hiểu, tích hợp dữ liệu, cũng như tái sử dụng dữ liệu thông qua các ứng dụng khác nhau. Các công nghệ Web ngữ nghĩa giúp con người có thể tạo ra các kho dữ liệu trên Web, xây dựng bộ từ vựng và viết các luật để xử lý dữ liệu.

Tóm lại, Web ngữ nghĩa đơn thuần chỉ là một sự mở rộng của Web hiện hành mà không phải là một sự đột phá thay thế công nghệ Web cũ. Ngược lại, Web ngữ nghĩa kế thừa từ Web hiện hành và cho phép khai thác Web hiện tại trong một con đường mới, con đường mà máy và người có thể làm việc cộng tác trong khai thác tài nguyên Web.

11.2. Kiến trúc của Web ngữ nghĩa

Kiến trúc của Web ngữ nghĩa được cho bởi sơ đồ trong Hình 11.1.

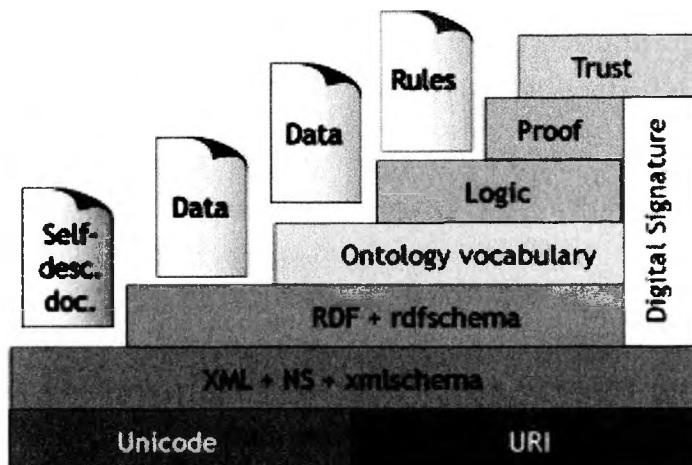
Theo kiến trúc này, Web ngữ nghĩa được phân thành các tầng. Cụ thể mỗi tầng được miêu tả như sau:

11.2.1. Tầng Unicode và URI

Tầng Unicode và URI là tầng đầu tiên của kiến trúc Web ngữ nghĩa. Đây là hạ tầng đầu tiên cho xây dựng Web ngữ nghĩa, là nền tảng để mã hóa, định vị và vận chuyển thông tin. Trong đó: Unicode là chuẩn mã hóa dữ liệu để vận chuyển thông tin. Unicode là chuẩn mã hóa quốc tế, nó cho phép mã hóa mọi

ngôn ngữ; URI-Uniform Resource Identifier là nền tảng để xác định vị trí cho các tài nguyên Web, cũng chính là việc xác định tài nguyên Web.

Thực tế tầng này đã được hoàn thiện và sử dụng trong nền Web hiện tại. Các URI được miêu tả với các giao thức khác nhau như: HTTP, FTP, SMTP ...hiện đang được sử dụng rộng rãi trên Internet. Sự xuất hiện của tầng này cho thấy được sự kế thừa thực sự của Web ngữ nghĩa. Web ngữ nghĩa thực sự chỉ là một sự mở rộng của Web hiện tại, nó giữ lại hầu hết những đặc điểm thiết kế bên dưới của Web hiện tại và chỉ mở rộng thêm phần ngữ nghĩa ở những mức bên trên nhằm tạo thêm một khung nhìn mới cho Web hiện tại, đó là khung nhìn của các ứng dụng (hay của máy tính).



Hình 11.1: Kiến trúc của Web ngữ nghĩa

11.2.2. Tầng XML, NS và XML Schema

Đây là tầng thứ hai của kiến trúc phân tầng Web ngữ nghĩa. XML và các chuẩn liên quan tới nó, cung cấp cho ta một con đường để diễn đạt cấu trúc thông tin bất kỳ và đây cũng là một chuẩn thực tế để truyền dữ liệu giữa các ứng dụng. Do vậy mà XML được hỗ trợ rộng rãi về cả các công cụ và người dùng. Đó cũng là lý do mà XML tồn tại ở tầng thứ hai này với vai trò làm một điểm trung gian giữa những dạng biểu diễn giàu ngữ nghĩa hơn và các dạng dữ liệu thô giàu cấu trúc.

XML cung cấp một cú pháp chung cho biểu diễn dữ liệu trong môi trường Internet. XML Schema cung cấp các định nghĩa kiểu dữ liệu và các cấu trúc cho tài liệu XML. Các không gian tên (namespace-NS) cũng được sử dụng như một giải pháp đã được áp dụng cho các tài liệu XML.

Thực sự thì ở tầng thứ hai này mọi cái vẫn được kế thừa từ những gì mà Web hiện tại đã làm được. Các chuẩn cú pháp XML, không gian tên và XML Schema vẫn được chấp nhận trong Web ngữ nghĩa. Sự kế thừa này chính là một cơ sở để tích hợp các định nghĩa Web ngữ nghĩa với các chuẩn XML khác.

11.2.3. Tầng RDF và RDF Schema

Bắt đầu từ tầng này, các đặc điểm mới của Web thực sự được bộc lộ làm cơ sở để khẳng định rằng đó là Web ngữ nghĩa. Sự thực thì tầng này có vai trò như một mô hình, một ngôn ngữ để biểu diễn ngữ nghĩa hay tạo ra các khung nhìn đơn giản tới máy tính.

RDF là viết tắt của Resource Description Framework (Khung mô tả tài nguyên) là một ngôn ngữ, một mô hình dữ liệu cho phép biểu diễn các siêu dữ liệu hay các phát biểu ngữ nghĩa về dữ liệu. Trong ngữ cảnh của Web thì các dữ liệu này còn được hiểu là các tài nguyên (resource).

RDFS là một ngôn ngữ để miêu tả các từ vựng được sử dụng trong tài liệu RDF. Sự xuất hiện của RDFS giúp tăng cường ngữ nghĩa cho mô hình dữ liệu RDF.

11.2.4. Tầng từ vựng Ontology

Thông qua việc miêu tả các từ vựng dưới dạng cây hay dạng phân lớp, RDFS đã góp phần mở rộng ngữ nghĩa cho dữ liệu RDF. Tuy nhiên nếu dùng lại ở đây, ngữ nghĩa mà dữ liệu RDF cung cấp thực sự chưa đủ để đạt được những gì mà Web ngữ nghĩa mong đợi. Từ đó mà tầng Ontology cần thiết được xây dựng.

Ontology cho phép mở rộng các từ vựng để miêu tả những thuộc tính và những lớp, chẳng hạn như các mối quan hệ khác giữa các lớp mà không chỉ đơn thuần là quan hệ cha con.

Có thể nói rằng RDFS giống như các kiểu tài nguyên sử dụng trong các tài liệu RDF. Nhờ định nghĩa về kiểu một cách thống nhất này mà phần nào đó ngữ nghĩa được thêm vào trong RDF. Đến lượt Ontology thì ngữ nghĩa được thêm vào đúng với mức từ vựng và được chia sẻ trên Internet.

11.2.5. Tầng Logic

Tầng Logic hiện tại vẫn đang được phát triển và chưa có một công bố nào có tính chất chuẩn và hoàn thiện về tầng này. Nhóm phát triển Web ngữ nghĩa của W3C thực sự vẫn đang phát triển tầng này và cũng vẫn chưa hé đưa ra những miêu tả hoàn chỉnh nào về tầng này.

Bản chất của tầng Logic là cung cấp những cơ sở để siêu dữ liệu RDF có thể trở thành tri thức, cái được áp dụng để thực hiện các suy luận logic nhằm chứng minh hoặc đưa ra các thông tin mới từ các thông tin đã có.

11.2.6. Tầng Proof và tầng Trust

Proof và Trust là những tầng cuối cùng trong kiến trúc của Web ngữ nghĩa. Hiện nay chưa có nhiều miêu tả về các tầng này cũng như giải pháp thực sự cho chúng.

Một điều đơn giản để hiểu sự khắc nghiệt thật sự của vấn đề này, đó là sự mâu thuẫn của thông tin. Chẳng hạn như có người nói rằng: x có màu xanh, lại có người nói rằng x có màu khác xanh, phải chăng Web ngữ nghĩa sẽ sụp đổ với những trường hợp như thế này? Câu trả lời dĩ nhiên là không, bởi vì hai lý do cơ bản sau:

- Một là ứng dụng trên Web ngữ nghĩa ở hiện tại thường dựa trên nền tảng một ngữ cảnh cụ thể.
- Hai là các ứng dụng trong tương lai sẽ thường chứa đựng các kỹ thuật kiểm tra các chứng cứ và xác thực điện tử (digital signatures).

Các ứng dụng trên Web ngữ nghĩa dựa trên một ngữ cảnh thường để mọi người xác thực sự đúng đắn của dữ liệu.

Ngữ cảnh là một cơ sở tốt bởi vì chúng ta có thể tin tưởng được mà không cần phải nhò vào những sự thẩm định phức tạp và các hệ thống kiểm tra. Tuy nhiên vấn đề xuất hiện đối tượng thứ ba, kẻ giả mạo là không tránh khỏi và đó là đòi hỏi sự ra đời của digital signature (“chữ ký số” hay còn gọi là “chữ ký điện tử”).

Việc áp dụng công nghệ mã hóa và chữ ký điện tử trong RDF sẽ đảm bảo rằng nguồn tài liệu mà ta đang sử dụng là do chính xác một nhà cung cấp nào đó mà ta tin tưởng. Điều này giống như sự xác thực điện tử mà trong an toàn thông tin đã đề cập đến.

Trong kiến trúc của Web ngữ nghĩa, chữ ký điện tử đóng một vai trò rất quan trọng. Nó gắn liền với các tầng của kiến trúc Web ngữ nghĩa kể từ tầng thứ ba RDF, với vai trò là mở rộng cho các tầng này để đảm bảo rằng những thông tin trong các tài liệu này là xác thực do một nhà cung cấp nào đó. Điều này giúp ngăn chặn sự sụp đổ của Web ngữ nghĩa do chính đặc tính đơn giản và phổ cập của nó mang đến. Với chữ ký điện tử, các ứng dụng sẽ có căn cứ để sử dụng các thông tin chính xác do những nhà cung cấp mà ứng dụng đó tin tưởng.

Phần tiếp theo trong chương này, chúng ta sẽ tìm hiểu cụ thể hơn các ngôn ngữ chính được sử dụng trong Web ngữ nghĩa, cụ thể là RDF và OWL.

11.3. RDF – Khung mô tả tài nguyên

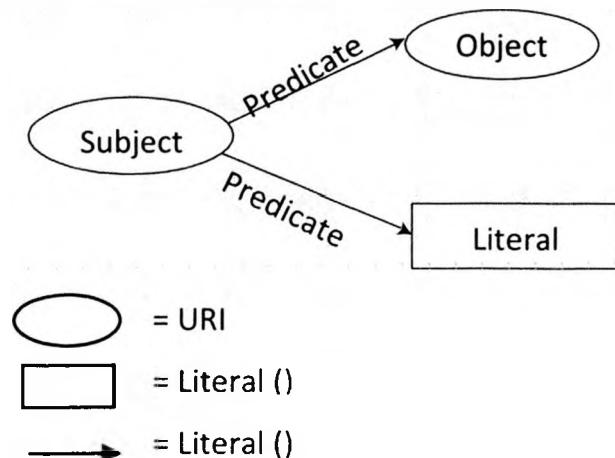
Khung mô tả tài nguyên (RDF) là một ngôn ngữ siêu dữ liệu để biểu diễn dữ liệu trên Web và cung cấp một mô hình để mô tả và tạo các mối quan hệ giữa các tài nguyên.

RDF định nghĩa một nguồn tài nguyên (resource) như một đối tượng bất kỳ có khả năng xác định duy nhất bởi một URI. Các nguồn tài nguyên có các thuộc tính đi kèm.

Các thuộc tính (predicate/property) được xác định bởi các kiểu thuộc tính và các kiểu thuộc tính có các giá trị tương ứng.

Kiểu thuộc tính biểu diễn các mối quan hệ của các giá trị được kết hợp với các tài nguyên.

Mô hình dữ liệu của RDF là các bộ ba (triple) gồm: <Subject, Predicate, Object>, được biểu diễn ở Hình 11.2.



Hình 11.2: Bộ ba RDF

- Subject (Chủ thể): được xác định bởi URI cụ thể.
- Predicate (Thuộc tính): thuộc tính của siêu dữ liệu, cũng được xác định bởi một URI.
- Object (Đối tượng): giá trị của thuộc tính, có thể là một giá trị (literal) hoặc một URI.
- Literal được sử dụng để biểu diễn các giá trị như con số, ngày tháng, chuỗi... Bất cứ cái gì có thể biểu diễn bởi một giá trị Literal cũng có thể được biểu diễn dưới dạng một URI. Một Literal có thể là object của một phát biểu nhưng không thể là subject hay là predicate.

RDF mô tả siêu dữ liệu về các tài nguyên trên Web. RDF dựa trên cú pháp XML. Tuy nhiên XML chỉ mô tả dữ liệu, RDF còn có khả năng biểu diễn ngữ nghĩa giữa chúng thông qua các tài nguyên được định danh bằng URI.

RDF chính là nền tảng trung tâm của Web ngữ nghĩa. Trong khi XML cung cấp cú pháp để mã hóa dữ liệu thì RDF mô tả siêu dữ liệu về các tài nguyên trên Web. RDF dựa trên cú pháp XML, tuy nhiên XML chỉ mô tả dữ liệu, RDF còn có khả năng biểu diễn ngữ nghĩa giữa chúng thông qua các tài nguyên định danh bằng URI.

11.3.1. Cấu trúc RDF/XML

Mô hình RDF thể hiện một mô hình ở mức trừu tượng để định nghĩa metadata. Cú pháp RDF được dùng để tạo ra và trao đổi metadata. RDF dựa trên cú pháp XML.

Cú pháp cơ bản của RDF có dạng như sau:

- [1] RDF ::= [<rdf:RDF>] description* [</rdf:RDF>]
- [2] description ::= '<rdf:Description' idAboutAttr? '>' propertyElt* '</rdf:Description>'
- [3] idAboutAttr ::= idAttr | aboutAttr
- [4] aboutAttr ::= 'about='"" URI-reference ""
- [5] idAttr ::= 'ID='"" IDsymbol ""
- [6] propertyElt ::= '<' propName '>' value '</' propName '>'| '<' propName resourceAttr '/>'
- [7] propName ::= QName
- [8] value ::= description | string
- [9] resourceAttr ::= 'resource='"" tham chiếu URI """
- [10] QName ::= [NSprefix ':'] name
- [11] URI-reference ::= string, interpreted per [URI]
- [12] IDsymbol ::= (bất kỳ ID nào hợp lệ nào của XML)
- [13] name ::= (bất kỳ tên hợp lệ nào của XML)
- [14] NSprefix ::= (bất kỳ tiếp đầu ngữ namespace hợp lệ nào)
- [15] string ::= (bất kỳ chuỗi nào)

Ví dụ : Xét phát biểu

ex: index.html exterms:creation-date "August 16, 1999" .

Cú pháp RDF/XML để biểu diễn cho phát biểu trên như sau:

1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:exterms="http://www.example.org/terms/">

4. <rdf:Description rdf:about="http://www.example.org/index.html">
5. <exterm:creation-date>August 16, 1999</exterm:creation-date>
6. </rdf:Description>
7. </rdf:RDF>

Trong đó:

Dòng 1: là khai báo XML, cho biết nội dung sau dựa trên cú pháp XML và phiên bản XML được dùng.

Dòng 2 và 3: bắt đầu với thẻ rdf:RDF, cho biết rằng nội dung XML tiếp theo mô tả RDF. Từ khóa này xác định tài liệu này được biểu diễn dưới dạng RDF. Tiếp theo là phần khai báo XML namespace được sử dụng trong tài liệu, tùy vào nhu cầu và mục đích sử dụng mà ta có thể dùng các namespace khác nhau cho từng tài liệu.

Dòng 4, 5, 6: mô tả những mệnh đề RDF. Để mô tả bất kỳ phát biểu nào dạng RDF/XML có thể dùng rdf:Description và rdf:about, đây chính là subject của phát biểu. Thẻ bắt đầu rdf:Description trong dòng 4 cho biết bắt đầu mô tả về một resource và tiếp tục định danh resource này dùng thuộc tính rdf:about để chỉ ra URI của subject resource. Dòng 5 cung cấp 1 phần tử thuộc tính, với QName exterm:creation-date như là thẻ của nó. Nội dung của phần tử thuộc tính này là object của statement, có giá trị là kiểu plain literal “August 19, 1999”.

Dòng 7: cho biết kết thúc của thẻ rdf:RDF bắt đầu ở dòng 2 và cũng là thẻ kết thúc của tài liệu RDF

11.3.2. RDF Container

Để mô tả tập hợp của nhiều đối tượng như một quyển sách được viết bởi nhiều tác giả, danh sách các học viên trong một khóa học... RDF cung cấp nhiều kiểu và nhiều thuộc tính tích hợp sẵn giúp mô tả được những tập như vậy, trong đó có kiểu khai báo container, dùng để lưu danh sách các tài nguyên hoặc các kiểu giá trị.

Các phần tử của một container có thể là các tài nguyên URI (có thể là blank nodes) hay là các giá trị literals.

RDF định nghĩa 3 loại đối tượng container: Bag, Sequence, và Alternative.

- Bag là danh sách không có thứ tự của các tài nguyên hoặc các giá trị. Bag cho phép những giá trị có thể trùng lặp nhau.

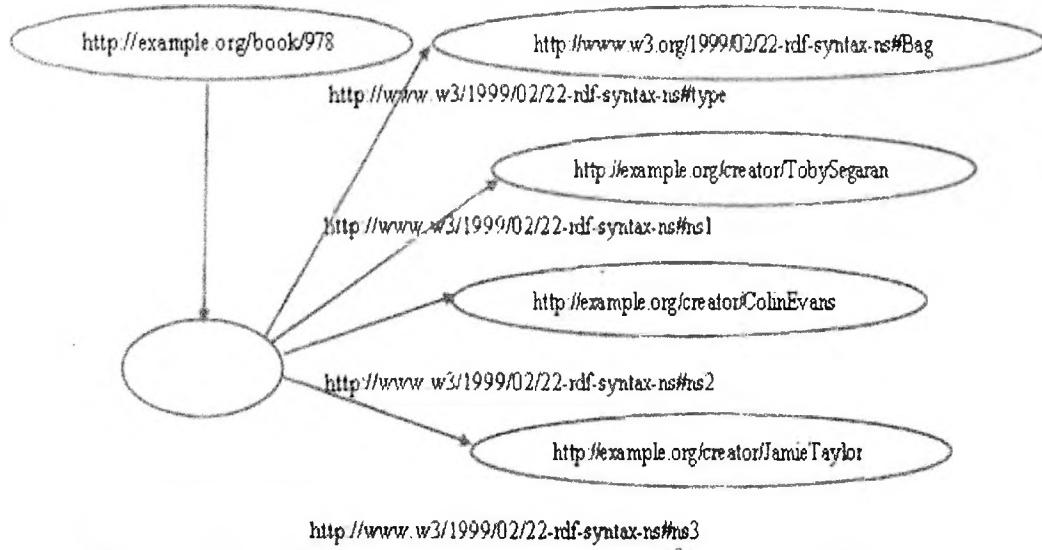
- Sequence là danh sách có thứ tự của các tài nguyên hoặc các giá trị. Chẳng hạn dùng Sequence để lưu trữ các giá trị theo thứ tự bảng chữ cái. Sequence cho phép những giá trị có thể trùng lặp nhau.

- Alternative là một danh sách các tài nguyên hoặc các giá trị, được dùng để biểu diễn các giá trị lựa chọn của một thuộc tính.

Ví dụ 11.1: Xét phát biểu sau

“Book with ISBN 978-0-596-15381-6 wrote by Toby Segaran, Colin Evans, Jamie Taylor”

Có mô hình RDF là:



Hình 11.3: Mô hình RDF của phát biểu trong Ví dụ 11.1

Ta biểu diễn dưới dạng cú pháp RDF/XML như sau:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:c="http://example.org/creator/vocab#">
  <rdf:Description rdf:about="http://example.org/Book/978-0-596-15381-6">
    <c:creator> <rdf:Bag>
      <rdf:li rdf:resource="http://example.org/creator/TobySegaran"/>
      <rdf:li rdf:resource="http://example.org/creator/ColinEvans"/>
      <rdf:li rdf:resource="http://example.org/creator/JamieTaylor"/>
    </rdf:Bag> </c:creator> </rdf:Description>
</rdf:RDF>
```

11.3.3. RDF Collection

RDF Collection cho phép khai báo một tập hợp đóng, tương tự một danh sách, có phần tử đầu (rdf:first), phần tử kế (rdf:rest) và phần tử cuối (rdf:nil).

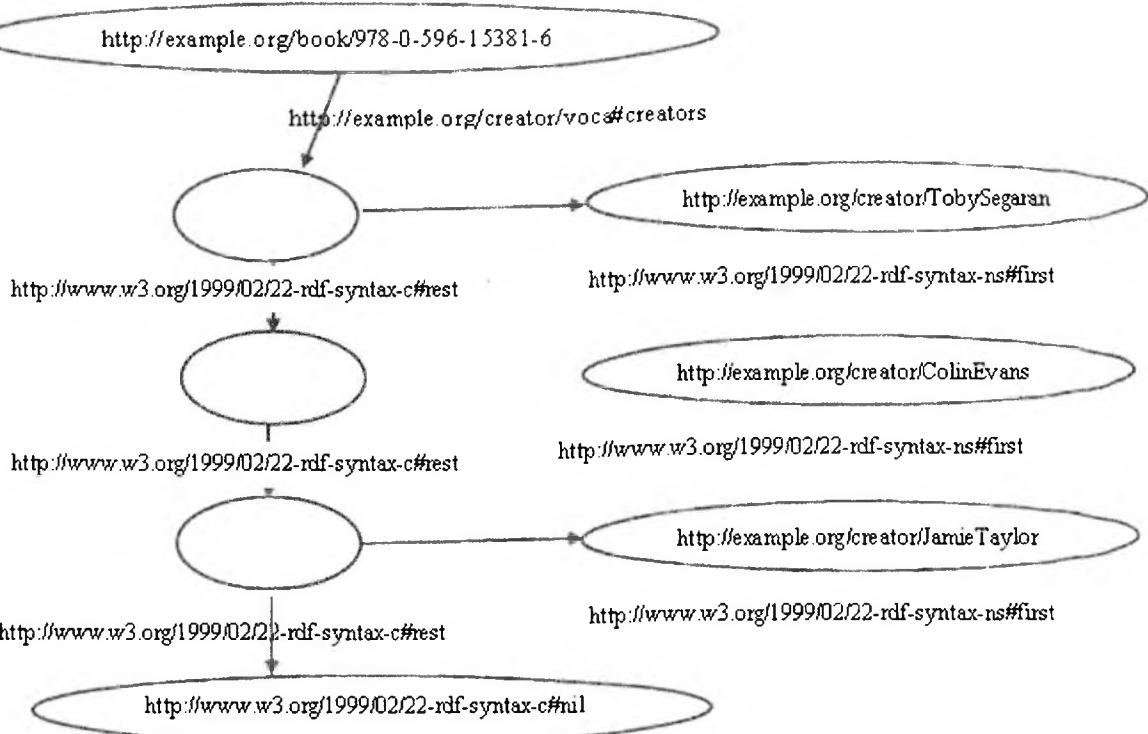
Ví dụ 11.2: Xét phát biểu sau

"Book with ISBN 978-0-596-15381-6 wrote by Toby Segaran, Colin Evans, Jamie Taylor"

Có mô hình RDF như Hình 11.4:

RDF/XML cung cấp một cách định nghĩa để mô tả một tập hợp, bằng cách sử dụng một thuộc tính có attribute là rdf:parseType="Collection". Ta có thể viết lại ví dụ trên một cách ngắn gọn hơn như sau:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:c="http://example.org/creator/vocab#">
  <rdf:Description rdf:about="http://example.org/Book/978-0-596-15381-6">
    <c:creator rdf:parseType="Collection">
      <rdf:Description rdf:about="http://example.org/creator/TobySegaran"/>
      <rdf:Description rdf:about="http://example.org/creator/ColinEvans"/>
      <rdf:Description rdf:about="http://example.org/creator/Jamie Taylor"/>
    </c:creator> </rdf:Description>
</rdf:RDF>
```



Hình 11.4: Mô hình RDF của phát biểu trong Ví dụ 11.2

11.4. Ngôn ngữ Ontology

Thuật ngữ Ontology bắt nguồn từ triết học, nó được sử dụng như tên của một lĩnh vực nghiên cứu về sự tồn tại của tự nhiên, xác định các vật thể trong tự nhiên và làm thế nào để mô tả chúng. Chẳng hạn như quan sát thế giới thực, xác định các đối tượng và sau đó nhóm chúng lại thành các lớp trừu tượng dựa trên thuộc tính chung.

Tuy nhiên trong những năm gần đây, Ontology đã trở thành một thuật ngữ được biết đến nhiều trong lĩnh vực khoa học máy tính và có ý nghĩa khác xa so với nghĩa ban đầu của nó. Ontology được xem như là “linh hồn” của Web ngữ nghĩa. Các Ontology được phát triển trong lĩnh vực trí tuệ nhân tạo để sử dụng lại và chia sẻ tri thức được thuận tiện hơn. Đầu những năm 1990, Ontology trở thành một chủ đề nghiên cứu phổ biến. Ontology được nghiên cứu bởi một số cộng đồng nghiên cứu trí tuệ nhân tạo, bao gồm kỹ sư tri thức, xử lý ngôn ngữ tự nhiên và biểu diễn tri thức.

Một định nghĩa chung cho ontology là: Ontology là một đặc tả hình thức của sự khái niệm hóa về một lĩnh vực ứng dụng cụ thể. Định nghĩa này nhấn mạnh hai điểm chính: sự khái niệm hóa (conceptualisation) là hình thức và do đó cho phép suy diễn bởi máy tính và một ontology trên thực tế được thiết kế cho một miền ứng dụng cụ thể nào đó. Các ontology bao gồm các khái niệm (các lớp - classes), các quan hệ (các thuộc tính - properties), các thể hiện (instances) và các tiên đề (axioms).

11.4.1. RDF Schema

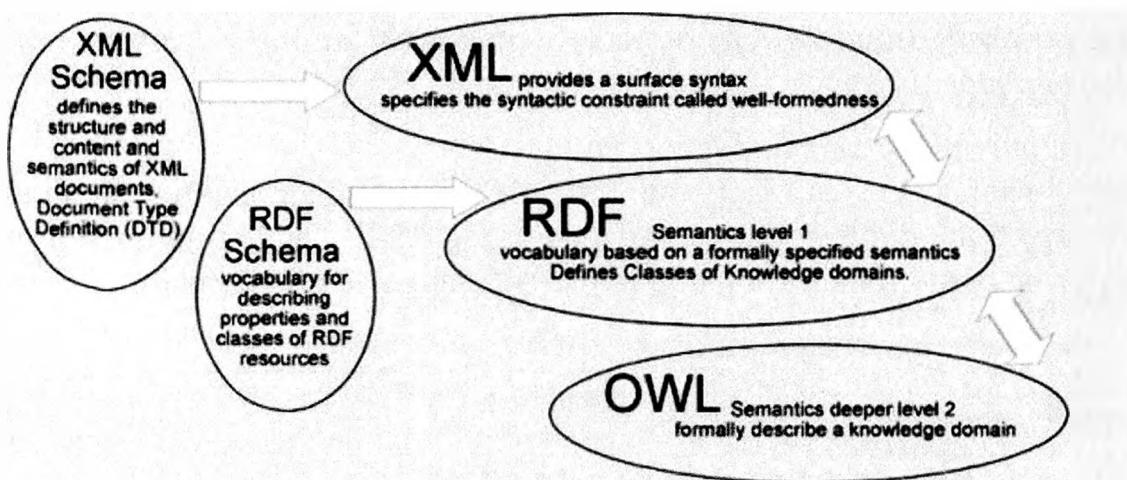
Như đã mô tả ở trên, RDF được dùng để mô tả các nguồn tài nguyên bằng các lớp, các thuộc tính và các giá trị, cung cấp tính liên thông giữa các ứng dụng.

Tuy nhiên, chúng ta cần phải định nghĩa các lớp và các thuộc tính trong các ứng dụng chuyên biệt. Để làm được điều này, chúng ta sẽ phải sử dụng một phiên bản mở rộng của RDF, gọi là RDFS hay RDF Schema.

Ngôn ngữ RDF chỉ giúp cho thông tin được thể hiện ở dạng bộ ba theo đúng mô hình RDF chứ thông tin vẫn chưa thể hiện gì về mặt ngữ nghĩa. Bởi vậy, xây dựng RDFS là điều cần thiết để hình thành nên ngữ nghĩa cho thông tin, là cơ sở để xây dựng các công cụ tìm kiếm ngữ nghĩa. RDFS và RDF có mối liên hệ tương đối gần nhau nên đôi lúc ta gọi ngôn ngữ này là RDF/RDFS.

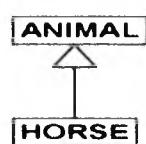
RDFs:

- Cho biết DL được mô tả trên RDF được diễn dịch như thế nào?
- Định nghĩa từ vựng cho RDF
- Tổ chức từ vựng dưới dạng cây phân cấp
- Khả năng khai báo tường minh các quan hệ ngữ nghĩa giữa các vocabulary terms



Hình 11.5: Mối quan hệ giữa RDFS và RDF:

Ví dụ 11.3: Xét lược đồ RDF sau Horse là subclass của class Animal



Hình 11.6: Horse là subclass của class Animal

Ta sử dụng RDF Schema để định nghĩa, chú thích các mối quan hệ:

```

<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
           xml:base="http://www.animals.fake/animals#">
    <rdf:Description rdf:ID="animal">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    </rdf:Description>
    <rdf:Description rdf:ID="horse">
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:subClassOf rdf:resource="#animal"/>
    </rdf:Description> </rdf:RDF>
  
```

Khi lớp RDFS là tài nguyên RDF, ta có thể dùng rdfs:Class thay vì dùng rdf:Description và bỏ qua thông tin trong rdf:type như sau:

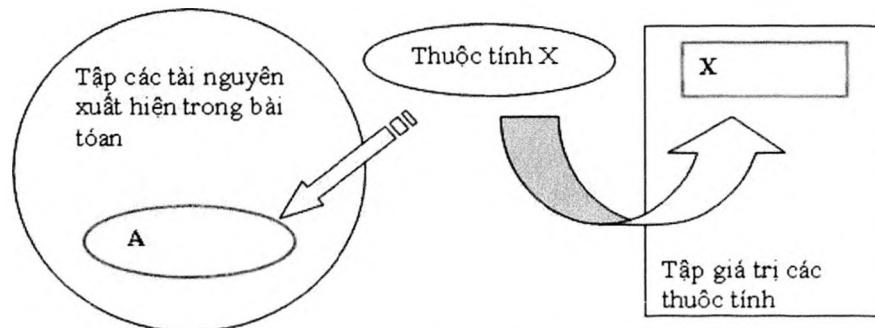
```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.animals.fake/animals#">
  <rdfs:Class rdf:ID="animal"/>
  <rdfs:Class rdf:ID="horse">
    <rdfs:subClassOf rdf:resource="#animal"/>
  </rdfs:Class>
</rdf:RDF>
```

11.4.1.1 Định nghĩa lớp (class)

Các tài nguyên trên Web có thể chia thành các nhóm gọi là class. Các thành viên của nhóm được xem như là thể hiện của lớp đó. Thông qua các định danh URI, các tài nguyên được truy xuất và có thể được mô tả bằng các RDF properties. Thuộc tính **rdf: type** được sử dụng để chỉ ra một resource là một thể hiện của một class.

11.4.1.2 Định nghĩa thuộc tính (property)

RDF Schema cung cấp một bộ từ vựng để mô tả làm thế nào mà các thuộc tính và lớp có thể được sử dụng cùng nhau trong RDF. Thuộc tính quan trọng nhất được sử dụng trong trường hợp này là rdfs:range và rdfs:domain



Hình Không gian miền và giới hạn của thuộc tính RDFS

Hình 11.7: Thuộc tính rdfs:range và rdfs:domain

Cách sử dụng rdfs:range

Thuộc tính rdfs:range dùng để chỉ giá trị của một thuộc tính là thể hiện của một lớp. Ví dụ nếu công ty *example.org* muốn chỉ rằng thuộc tính *ex:author* có giá trị là thể hiện của lớp *ex:Person*, ta sẽ viết phát biểu RDF như sau:

ex:Person rdf:type rdfs:Class.

ex:author rdf:type rdf:Property.

ex:author rdfs:range ex:Person.

Phát biểu này chỉ rằng *ex:Person* là một lớp, *ex:author* là một thuộc tính và thuộc tính *ex:author* có object là thực thể của lớp *ex:Person*. Tuy nhiên, một thuộc tính có thể có nhiều *rdfs:range*, như ví dụ sau:

ex:hasMother rdfs:range ex:Female.

ex:hasMother rdfs:range ex:Person.

Thuộc tính `rdfs:range` có thể được sử dụng để chỉ ra giá trị của kiểu typed literal

```
ex:age rdf:type rdf:Property.  
ex:age rdfs:range xsd:integer.
```

Cách sử dụng `rdfs:domain`

Thuộc tính `rdfs:domain` được sử dụng để chỉ rằng một thuộc tính là thuộc tính của một lớp nào đó. Ví dụ, công ty `example.org` muốn thuộc tính `ex:author` là thuộc tính của lớp `ex:Book`, ta có phát biểu như sau:

```
ex:Book rdf:type rdfs:Class.  
ex:author rdf:type rdf:Property.  
ex:author rdfs:domain ex:Book.
```

Và một thuộc tính cũng có nhiều thuộc tính `rdfs:domain` khác

```
exterms:weight rdfs:domain ex:Book.  
exterms:weight rdfs:domain ex:MotorVehicle.
```

11.4.1.3 Luật suy diễn ngữ nghĩa trong RDFS

Có 6 trường hợp suy diễn theo các cấp loại lớp, thuộc tính, lớp con, thuộc tính con, miền (domain), dài (range)

Trường hợp 1: Suy diễn về loại của lớp (type)

```
IF (c2, subClassOf, c1)  
AND (x, type, c2)  
THEN (x, type, c1)
```

Ví dụ: IF (Man, subClassOf, Person)

```
AND (Tom, type, Man)  
THEN (Tom, type, Person)
```

Trường hợp 2: Suy diễn về thuộc tính của lớp (Property)

```
IF (p2, subPropertyOf, p1)  
AND (x, p2, y)  
THEN (x, p1, y)
```

Ví dụ: IF (author, subPropertyOf, creator)

```
AND (Tom, author, Report2010)  
THEN (Tom, creator, Report2010)
```

Trường hợp 3: Suy diễn về lớp con của lớp (subClassOf)

```
IF (c2, subClassOf, c1)  
AND (c3, subClassOf, c2)  
THEN (c3, subClassOf, c1)
```

Ví dụ:

```
IF (Person, subClassOf, Animal)  
AND (Man, subClassOf, Person)  
THEN (Man, subClassOf, Animal)
```

Trường hợp 4: Suy diễn về thuộc tính con của thuộc tính (subPropertyOf)

```
IF (p2, subPropertyOf, p1)  
AND (p3, subPropertyOf, p2)  
THEN (p3, subPropertyOf, p1)
```

Ví dụ:

```
IF (parent, subPropertyOf, ancestor)  
AND (father, subPropertyOf, parent)  
THEN (father, subPropertyOf, ancestor)
```

Trường hợp 5: Suy diễn trong miền (domain)

```
IF (p1, domain, c1)
AND (x, p1, y)
THEN (x, type, c1)
```

Ví dụ: IF (author, domain, Human)
AND (Tom, author, Report2010)
THEN (Tom, type, Human)

Trường hợp 6: Suy diễn trong dãy giới hạn (range)

```
IF (p1, range, c1)
AND (x, p1, y)
THEN (y, type, c1)
```

Ví dụ: IF (author, range, Document)
AND (Tom, author, Report2010)
THEN (Report, type, Document)

11.4.1.4. Ví dụ về xây dựng ontology với RDFS

Chúng ta sẽ dùng cú pháp cơ bản sau để biểu diễn cho một phát biểu (statement): {subject,predicate,object}

Nếu có nhiều hơn một thuộc tính cho một chủ thể thì:

```
{subject,
  {predicate1, object1}
  {predicate2, object2}
}
```

Nếu có nút trống (blank node) thì biểu diễn như sau:

```
{subject, predicate,
  {predicate-of-bnode, object-of-bnode}
}
```

Một tài nguyên bắt đầu bởi dấu # chẳng hạn #resource_1 thì có nghĩa là tài nguyên đó đã được khai báo trong cùng một tài liệu mà chúng ta đang xây dựng ontology. Sau này muốn sử dụng tài nguyên đó thì ta dùng #resource_1 chứ không cần phải lấy URI của nó cho phức tạp, khó nhìn.

Bây giờ chúng ta sẽ đi vào xây dựng một Ontology đơn giản, Ontology này sẽ xây dựng tóm gọn như sau:

```
{rdfs:Class
{rdf:ID,"StaffMe {rdfs:Class}//Khai báo không tường minh, không nên dùng
{rdf:resource,"http://www.semantic.vn/2009/01/rdf-schema#StaffMember"}
}

{rdfs:Class
  {rdf:about,"AcademicStaff"}
  {rdfs:subClassOf, "# StaffMember"} }

{rdfs:Class
  {rdf:about,"Lecturer"} //Khai báo tường minh
  {rdfs:subClassOf, #AcademicStaff} }

{rdfs:Class
  {rdf:ID,"Course"}
  {rdf:resource, "http://www.semantic.vn/2009/01/rdf-schema#Course"} }

{rdf:Property
{ rdf:ID,"phone"}
  {rdfs:domain, #staffMember}}
```

```

{rdfs:range , "http://www.semantic.vn/2009/01/rdf-schema#Literal"}
}
{rdf:Property
{ rdf:ID,"ID"}
{rdfs:domain, #staffMember}
{rdfs:range , "http://www.semantic.vn/2009/01/rdf-schema#Literal"}
{rdf:Property
{ rdf:ID,"involve"}
{rdfs:domain, #Course}
{rdfs:range ,#AcademicStaff} }
{rdf:Property
...

```

Từ Ontology được xây dựng như trên, ta có thể vẽ ra sơ đồ phân cấp các tài nguyên như sau :

```

StaffMember
Academic Staff
Lecturer
Course

```

Nhìn sơ đồ ta thấy có hai tài nguyên là *StaffMember* và *Course*. Các lớp *AcademicStaff* và *Lecturer* chỉ là thừa kế từ *StaffMember*.

11.4.2. OWL (Ontology Web Language)

OWL là ngôn ngữ ontology khá mạnh, nó ra đời sau RDFS nên biết kế thừa những lợi thế ngôn ngữ này đồng thời bổ sung thêm nhiều yếu tố giúp khắc phục được những hạn chế RDFS.

OWL giúp tăng thêm yếu tố logic cho thông tin và khả năng phân loại, ràng buộc cũng như lượng số tương đối mạnh.

Chúng ta hãy tìm hiểu sơ qua các lớp và thuộc tính của OWL để thấy được những ưu điểm OWL so với RDFS.

Classes	Properties	
owl:AllDifferent	owl:allValuesFrom	owl:TransitiveProperty
owl:Class	owl:backwardCompatibleWith	owl:inverseOf
owl:DataRange	owl:cardinality	owl:maxCardinality
owl:DataTypeProperty	owl:complementOf	owl:minCardinality
owl:DeprecatedProperty	owl:distinctMembers	owl:oneOf
owl:DeprecatedClass	owl:differentFrom	owl:onProperty
owl:FunctionalProperty	owl:disjointWith	owl:priorVersion
owl:InverseFunctionalProperty	owl:equivalentClass	owl:sameAs
owl:Nothing	owl:equivalentProperty	owl:sameIndividualAs
owl:ObjectProperty	owl:hasValue	owl:someValuesFrom
owl:Ontology	owl:imports	owl:subClassOf
owl:Restriction	owl:incompatiblewith	owl:unionOf
owl:SymmetricProperty	owl:intersectionOf	owl:versionInfo

Trong OWL có thêm một số thuộc tính hỗ trợ suy luận và ràng buộc.

*Hỗ trợ suy luận

- **Tính chất bắc cầu:** Nếu như chúng ta có một lớp thuộc tính “contain” và gán cho nó thuộc tính *owl:transitiveProperty* thì thuộc tính “contain” này sẽ có tính chất bắc cầu. Giả sử ta có thông tin A contain B và B contain C, thì hệ thống sẽ tự suy luận ra một thông tin khác là A contain C. Và đây là biểu diễn thuộc tính contain trong OWL:

```
{owl:TransitiveProperty  
{rdf:ID, "contain"} }
```

- **Tính chất đảo ngược:** Nếu A có thuộc tính hasParent là B, thì suy ra B có thuộc tính hasChild là A.

```
{owl:ObjectProperty  
{Rdf:ID, 'hasChild'}  
{owl:inverseOf, #hasParent} }
```

*Hỗ trợ ràng buộc

- **Ràng buộc kiểu:** Giả sử ta đã có lớp Human, thừa kế từ hai lớp này là Man và Woman. Ta muốn một đối tượng thuộc lớp Man thì không thể thuộc lớp Woman và ngược lại.

```
{owl :Class  
{rdf :about, #Man}  
{owl :disjointWith, #Woman} }
```

Ta muốn một người thì chỉ có một cha và một mẹ. Tức thuộc tính hasParent thì luôn gán số lượng là 2.

```
{owl :Restriction  
{owl :onProperty, #hasParent}  
{owl :cardinality, 2} }
```

Trên đây là một vài ví dụ mà trong RDFS không thể làm được. Rõ ràng OWL có nhiều ưu điểm hơn trong việc xây dựng hệ thống ontology thông minh và có phân loại tốt. Với những đặc điểm đó, OWL ngày nay đã trở thành ngôn ngữ ontology chính thức cho việc xây dựng và phát triển các hệ thống Web ngữ nghĩa. Có 3 loại OWL:

- OWL đầy đủ (OWL Full): không giới hạn việc các từ vựng được xây dựng như thế nào.
- OWL DL (Description Logic): sử dụng logic mô tả để dẫn đến các suy luận
- OWL Lite: dạng đơn giản nhất để cài đặt OWL

DAML + OIL

DAML + OIL là kết quả từ hai dự án nghiên cứu độc lập với nhau là DAML (DARPA Agent Markup Language) và OIL (Ontology Inference Layer) nhằm khắc phục những hạn chế về kiểu dữ liệu trong các ngôn ngữ Ontology trước đó là RDF, RDFS. DAML + OIL (gọi tắt là DAML) là ngôn ngữ đánh dấu cho các tài nguyên trên Web, có hỗ trợ suy luận. Ngôn ngữ này được xây dựng có kế thừa từ các chuẩn của W3C như XML, RDF, RDFS... Một số điểm đáng chú ý của ngôn ngữ này là:

- Cho phép giới hạn các kiểu dữ liệu được định nghĩa trong XML Schema hay bởi người dùng. Trong DAML, một thuộc tính có thể

nhận giá trị trên nhiều khoảng khác nhau, tạo nên tính uyển chuyển trong việc mô tả dữ liệu.

- Cho phép định nghĩa thuộc tính unique để xác định các đối tượng.
- Cho phép mô tả các quan hệ như hoán đổi và bắc cầu.

DAML sau đó tiếp tục trở thành nền tảng cho một ngôn ngữ Ontology khác là OWL. DAML về cơ bản rất giống với OWL (ngoại trừ tên một số ít thuật ngữ, cú pháp được sửa đổi), tuy nhiên khả năng mô tả các ràng buộc kém hơn.

11.5. Công cụ xây dựng và quản trị ontology

11.5.1 Giới thiệu

Về mặt lý thuyết, người xây dựng và quản trị Ontology có thể không cần các công cụ hỗ trợ, thay vào đó có thể thực hiện trực tiếp bằng các ngôn ngữ. Tuy nhiên, cách thứ hai sẽ không khả thi khi Ontology có kích thước lớn và cấu trúc phức tạp. Thêm vào đó, việc xây dựng và quản trị Ontology không chỉ đòi hỏi việc tạo cấu trúc lớp phân cấp, định nghĩa các thuộc tính, ràng buộc... mà còn bao hàm việc giải quyết các bài toán liên quan trên nó. Có rất nhiều bài toán liên quan đến một hệ thống Ontology như:

- Trộn 2 hay nhiều Ontology.
- Chuẩn đoán và phát hiện lỗi.
- Kiểm tra tính đúng đắn và đầy đủ.
- Ánh xạ qua lại giữa các Ontology.
- Suy luận trên Ontology.
- Sao lưu và phục hồi một Ontology.
- Xóa, sửa và tinh chỉnh các thành phần trong Ontology.
- Tách biệt Ontology với ngôn ngữ sử dụng (DAML, OWL...).

Những khó khăn trên đã khiến các công cụ trở thành một thành phần không thể thiếu, quyết định đến chất lượng của một hệ thống Ontology. Hiện có rất nhiều công cụ có khả năng hỗ trợ người thiết kế giải quyết những bài toán liên quan. Có thể kể ra một số như: *Sesame*, *Protégé*, *Ontolingua*, *Chimaera*, *OntoEdit*, *OidEd*...

Nội dung phần này chỉ đề cập đến công cụ Protégé. Các công cụ còn lại người đọc tự tìm hiểu thêm.

11.5.2. Protégé

Protégé là bộ phần mềm mã nguồn mở Java nổi tiếng. Protégé được nghiên cứu và phát triển từ năm 1998 bởi nhóm nghiên cứu của Mark Musen, ĐH. Stanford nhằm quản lý các thông tin trong lĩnh vực sinh y học. Đây là dự án được nhận được sự quan tâm và tài trợ từ rất nhiều tổ chức, trong đó có Bộ Quốc Phòng Mỹ. Mã nguồn Protégé có thể được tìm thấy tại website: <http://smi-protege.stanford.edu/repos/protege/owl/trunk>.

Hiện tại, Protégé đã có phiên bản 4.3. Các ưu điểm của Protégé là:

- Hỗ trợ đầy đủ ba phiên bản của ngôn ngữ OWL là OWL-Full, OWL-Lite và OWL-DL.
- Nhờ sử dụng mô hình hướng đối tượng của ngôn ngữ Java, Protégé tỏ ra rất hiệu quả trong việc mô hình các lớp, thực thể, quan hệ...

- Giao diện thiết kế trực quan có tính tương tác cao. Người sử dụng có thể định nghĩa các thành phần của Ontology trực tiếp từ các form.
- Cho phép biểu diễn trực quan Ontology dưới dạng các sơ đồ.
- Cho phép xây dựng Ontology từ nhiều nguồn khác nhau.
- Protégé tự động lưu một bản tạm của Ontology. Nếu có lỗi phát sinh trong quá trình thao tác thì Ontology cũ sẽ tự động được phục hồi. Người thiết kế cũng có thể chuyển qua lại giữa hai bản Ontology này bằng chức năng Revert to a Previous Version và Active Current Version.
- Cung cấp chức năng tìm kiếm lỗi, kiểm tra tính nhất quán và đầy đủ của Ontology. Để sử dụng, người thiết kế chọn chức năng Run Ontology Test và Check Consistency.
- Cho phép các lớp và thuộc tính của Ontology này có thể được sử dụng trong một Namespace khác mà chỉ cần sử dụng các URL để tham khảo. Để sử dụng, chọn chức năng Move Resource to Namespace.
- Hỗ trợ suy luận trực tiếp trên Ontology dựa trên Interface chuẩn DL Implementation Group (DIG).
- Hỗ trợ sinh mã tự động. Protégé cho phép chuyển Ontology thành mã nguồn RDF/XML, OWL, DIG, Java, EMF Java Interfaces, Java Schema Classes... Các mã này có thể được nhúng trực tiếp vào ứng dụng và là đầu vào cho các thao tác trên Ontology khi cần.
- Cung cấp đầy đủ chuẩn giao tiếp cho các Plug-in.

Tuy nhiên, Protégé cũng thể hiện một số hạn chế như không cho phép truy vấn từng phần một cơ sở tri thức dẫn tới việc không quản lý hiệu quả các cơ sở tri thức có kích thước lớn, hoặc chưa hỗ trợ kết nối trực tiếp với một số hệ quản trị cơ sở tri thức phổ biến như Sesame

11.5.3. Các ứng dụng của Web ngữ nghĩa

11.5.3.1. Xây dựng các bộ máy tìm tin

Vấn đề hiện nay là đa số các bộ máy tìm tin đều thực hiện cho phép người sử dụng có thể tạo các câu truy vấn gồm các từ khóa tìm kiếm để nhận về kết quả mong muốn. Tuy nhiên, phương pháp này gặp hai vấn đề chính sau đây:

- Mỗi từ khóa có thể có một hay nhiều ý nghĩa tùy theo từng ngữ cảnh và bộ máy tìm kiếm không thể hiện mối quan hệ giữa các từ khóa với nhau.
- Có thể các thông tin cùng ý nghĩa với thuật ngữ trong biểu thức tìm của người sử dụng sẽ không tồn tại trong kết quả tìm.

Ví dụ: Ta cần tìm thông tin về người trưởng bộ môn công nghệ thông tin của MIT, ta gõ: “*MIT information technology chair*” vào Google, nhưng kết quả thu được là không chính xác. Nguyên nhân của việc tìm kiếm thất bại là do: Từ khoá “*MIT*” có nhiều ý nghĩa. Ngoài ra, máy tìm không thể hiểu mối liên hệ giữa các từ khoá: *MIT*, *information technology* và *chair*. Nếu bộ máy tìm kiếm được tích hợp tri thức để hiểu được ý nghĩa của các từ, thì rất có thể nó cho ta kết quả chính xác hơn, lúc đó việc tìm kiếm sẽ dựa trên khái niệm (concept) chứ không phải theo từ khóa (keyword).

11.5.3.2. Ứng dụng công nghệ ngữ nghĩa trong thư viện số

Thư viện số phải thường xuyên xử lý một lượng lớn thông tin từ các dạng tài liệu số. Phần lớn chúng được rút ra từ thư viện truyền thống, được tập trung biên tập lại thành nguồn thông tin sẵn dùng cho một nhóm người liên quan bằng cách quét bài báo, sách, tài liệu... Bằng cách này đã làm hạn chế lợi thế của các hệ thống máy tính hiện đại và gây khó khăn cho quá trình xử lý sau này. Áp dụng công nghệ Web ngữ nghĩa chúng ta có thể nghiên cứu và phát triển hệ thống thư viện số có thể thực hiện xử lý, lưu trữ, tìm kiếm và phân tích tất cả các kiểu thông tin số. Công nghệ ngữ nghĩa cho phép miêu tả đối tượng, thiết lập các lược đồ cần thiết trong các dạng của ontologies cho các định danh của các đối tượng số. Mục tiêu chính là làm cho thao tác giữa các phần có thể xử lý thông minh, nhất quán, mạch lạc tương tự các lớp của đối tượng số và các dịch vụ.

Ứng dụng ontologies trong việc mô tả hệ thống thư mục: Thông thường một thư viện số sử dụng dữ liệu mô tả có cấu trúc để mô tả hệ thống thư mục tuy nhiên các trường trong dữ liệu mô tả lại không được định nghĩa ngữ nghĩa một cách đầy đủ, việc ứng dụng ontologies trong thư viện số không những thực hiện lưu trữ dữ liệu mô tả để mô tả hệ thống thư mục mà còn mô tả được nội dung của nó. Thay vì trong trường hợp một quyển sách được lưu trữ trong thư viện số chúng ta có thể tách riêng cấu trúc từng chương của nó, cung cấp mô tả cho mỗi chương và thực hiện lưu trữ mối quan hệ của các chương khác nhau. Bằng việc sử dụng tư tưởng cấu trúc của ontologies và sử dụng tư tưởng này trong việc mô tả dữ liệu, chúng ta cung cấp một tầng tổng quát dữ liệu mô tả và nội dung.

Một trong những ứng dụng quan trọng nữa chúng ta có thể thấy hệ thống dữ liệu của thư viện số rất lớn và đa dạng nó thường phục vụ cho nhiều tổ chức, cá nhân vào nhiều mục đích khác nhau, trong khi đó dữ liệu chủ yếu thuộc vào hai dạng là dữ liệu có cấu trúc (trong database) và dữ liệu phi cấu trúc (các nguồn lấy từ web). Một vấn đề đặt ra là làm thế nào để các ứng dụng sử dụng được đồng thời cả hai loại dữ liệu này, bởi vì trên thực tế mỗi ứng dụng chỉ sử dụng một loại dữ liệu có cấu trúc hoặc phi cấu trúc. Chúng ta có chuẩn chung phục vụ cho hầu hết các loại ứng dụng đó là sử dụng XML, nó được xem là nền tảng công nghệ của Web ngữ nghĩa. Nó sẽ là cầu nối thực hiện chuẩn hóa các nguồn dữ liệu, từ đó có thể phục vụ cho mọi loại ứng dụng.

11.5.3.3. Khung làm việc để quản lý tri thức (Framework for Knowledge Management)

Web ngữ nghĩa là một hệ nền nhiều hứa hẹn cho việc phát triển các hệ thống quản lý tri thức. Tuy nhiên, vấn đề ở đây là làm thế nào biểu diễn tri thức ở dạng thức máy có thể hiểu được, để tri thức cần thiết có thể được tìm thấy bởi các máy tìm kiếm (search engine). Chúng ta sử dụng giải pháp quản lý tri thức dựa trên định dạng tương thích RDF để biểu diễn các luật và dựa trên một kỹ thuật mới để chú giải các nguồn tri thức bằng cách sử dụng các câu điều kiện.

Giải pháp là dựa trên các công cụ Web ngữ nghĩa đang tồn tại. Điểm thuận lợi chính là sự thúc đẩy khả năng tìm kiếm tri thức với độ chính xác cao, cũng

như khả năng truy cập cấu tạo các nguồn tri thức cần thiết cho việc giải quyết một vấn đề nào đó. Dạng thức này có thể được biểu diễn bằng cách dùng các câu lệnh If–Then (statement If-Then), được thiết lập theo cách suy diễn (inference) và ủy quyền (trust) trên Web ngữ nghĩa. Các statement (*câu lệnh*) điều kiện có thể được dùng để lập chỉ mục nội dung các tài nguyên Web một cách nhiều ý nghĩa hơn so với liên kết các từ khóa, khái niệm hay metadata (*siêu dữ liệu*). Điều này có thể sẽ hình thành các truy vấn dựa trên ngữ cảnh hơn, tăng cường độ chính xác trong tìm kiếm tri thức. Ví dụ: Trong vấn đề định chỉ mục tài liệu, dù có hay không có tài liệu được định chỉ mục bằng từ khóa aspirin (*thuốc aspirin*) và headache (*bệnh đau đầu*), cách aspirin trị headache hay aspirin gây ra headache đều có thể được giải quyết dễ dàng bằng cách sử dụng các câu điều kiện định nghĩa trước. Việc xây dựng và quản lý tri thức trên Web ngữ nghĩa một cách khoa học cho phép sự chuyển đổi đa dạng trong môi trường phân tán.

Kết chương

Internet ra đời đã mang lại nhiều hữu ích cho con người, đặc biệt là trong tìm kiếm thông tin. Tuy nhiên việc tìm tin trên mạng thường bị nhiều và nhiều khi rất khó lựa chọn được thông tin cần thiết. Web ngữ nghĩa ra đời hy vọng sẽ sớm khắc phục được những nhược điểm này, góp phần nâng cao hiệu quả của mạng toàn cầu trong việc tìm và khai thác thông tin của người dùng

TÀI LIỆU THAM KHẢO

- [1]. Lê Quang Anh Hưng, “GIỚI THIỆU XML”, www.viet-ebook.co.cc
- [2]. Viet Jack, “CƠ BẢN VỀ XML”,
http://vietjack.com/xml/xml_syntax.jsp
- [3]. Tutorialspoint, “LEARN XML”, <https://www.tutorialspoint.com/>
- [4]. Nguyễn Thiên Bằng, “GIÁO TRÌNH NHẬP MÔN XML”, NXB Lao động - Xã hội, 2002.
- [5]. Nguyễn Phương Lan, Hoàng Đức Hải, “XML NỀN TẢNG VÀ ỨNG DỤNG”, NXB Lao động – Xã hội, 2006.
- [6]. Nguyễn Tiến Huy, “BÀI GIẢNG CÔNG NGHỆ XML VÀ ỨNG DỤNG”, Trường ĐH KHTN, 2010.
- [7]. Hoàng Hữu Hạnh, “WEB NGỮ NGHĨA: NHỮNG THÁCH THỨC VÀ HƯỚNG TIẾP CẬN MỚI”, Tạp chí khoa học, ĐH Huế, Số 48, 2008.
- [8]. Lương Đỗ Long, “ỨNG DỤNG WEB NGỮ NGHĨA TRONG LUỒN TRỮ VÀ QUẢN LÝ CÁC TÀI LIỆU SỐ”, Luận văn thạc sĩ khoa học, Trường ĐH Công nghệ, ĐH Quốc gia Hà Nội, 2011.
- [9]. Joe Fawcett, Liam R.E. Quin, Danny Ayers, “BEGINNING XML”, Wiley Publishing, 2012.
- [10]. Erik T. Ray, “LEARNING XML”, ISBN: 0-59600-046-4, 2001
- [11]. Michael C. Daconta, Leo J. Obrst, Kevin T. Smith, “THE SEMANTIC WEB - A GUIDE TO THE FUTURE OF XML, WEB SERVICES, AND KNOWLEDGE MANAGEMENT”, Wiley Publishing, 2003.
- [12]. Hà Quang Thụy (Chủ biên), “GIÁO TRÌNH KHAI PHÁ DỮ LIỆU WEB”, NXB Giáo dục, Hà Nội, 2009.
- [13]. Hiroshi Maruyama và cộng sự, “XML AND JAVA, DEVELOPING WEB APPLICATION”, Addison Wesley, 2002.

DANH MỤC CÁC TỪ VIẾT TẮT

ADO: ActiveX Data Object

CSDL: Cơ sở dữ liệu

CDF: Channel Definition Format

DTD: Document Type Definition

DOM: Document Object Model

IE: Internet Explorer

FLWOR: For, Let, Where, Order by, Return

HTML: HyperText Markup Language

LINQ: Language Integrated Query

OWL: Ontology Web Language

SGML: Standard Generalized Markup Language

XML: eXtensible Markup Language (ngôn ngữ đánh dấu mở rộng)

XSD: XML Schema Document

XSL: eXtensible Style Language

XPath: XML Path Language

XSLT: eXtensible Stylesheet Language

RDF: Resource Description Framework

URL: Uniform Resource Locator

URI: Uniform Resource Identifier

W3C: World Wide Web Consortium

MỤC LỤC

LỜI MỞ ĐẦU	5
------------------	---

Chương 1

TỔNG QUAN VỀ XML

1.1. Giới thiệu	7
1.1.1. Đôi nét về lịch sử XML	7
1.1.2. XML là gì?	7
1.1.3. XML được dùng như thế nào?	9
1.2. Trình soạn thảo XML (XML Editors)	9
1.3. Trình duyệt XML	11
1.4. Bộ phân tích XML (XML Parser).....	11
1.5. Quy tắc cú pháp XML.....	12
1.5.1. Khai báo định dạng XML (XML Declaration)	12
1.5.2. Thẻ và phần tử (Tags and Elements).....	13
1.5.3. Quy tắc cú pháp cho thuộc tính trong XML	16
1.5.4. Tham chiếu trong XML	16
1.5.5. Text trong XML	16
1.6. Các bộ kiểm tra XML (XML Validators)	17
Bài tập chương 1	19

Chương 2

TẠO TÀI LIỆU XML HỢP KHÔN DẠNG

2.1. XML hợp khuôn dạng	20
2.1.1. Điều gì tạo nên một tài liệu XML hợp khuôn dạng?	20
2.1.2. Tạo một tài liệu XML mẫu	20
2.2. Các ràng buộc hợp khuôn dạng	22
2.3. Sử dụng không gian tên trong XML	23
2.3.1. Xử lý xung đột tên phần tử	24
2.3.2. Sử dụng không gian tên	24
2.3.3. Thuộc tính không gian tên	25

2.3.4. Định nghĩa không gian tên với URI	25
2.3.5. Tạo không gian tên cục bộ	27
2.3.6. Không gian tên mặc định	27
Bài tập chương 2	29

Chương 3

ĐỊNH NGHĨA KIỂU TÀI LIỆU - DTD

3.1. Định nghĩa kiểu tài liệu (DTD)	30
3.1.1. Giới thiệu về DTD	30
3.1.2. DTD nội tại trong XML (Internal DTD).....	31
3.1.3. DTD bên ngoài XML (External DTD).....	32
3.1.4. Quản lý không gian tên trong DTD.....	34
3.2. Các thành phần của DTD	35
3.2.1. Phần tử (Element)	35
3.2.2. Thuộc tính (Attribute)	38
3.2.3. Thực thể (Entity).....	43
3.3. Dùng DTD kiểm tra tính hợp lệ của XML	47
Bài tập chương 3	48

Chương 4

LƯỢC ĐỒ XML

4.1. Vì sao dùng lược đồ XML?	51
4.2. Khai báo các phần tử và kiểu dữ liệu.....	54
4.3. Tạo các kiểu đơn giản.....	58
4.4. Tạo các phần tử.....	60
4.4.1. Tạo các phần tử rỗng.....	60
4.4.2. Tạo các phần tử có nội dung hỗn hợp	60
4.4.3. Lược đồ chú giải (annotation schema).....	61
4.4.4. Tạo các lựa chọn (choices).....	62
4.4.5. Tạo các khai báo tuần tự (sequence).....	63
4.4.6. Tạo nhóm và thuộc tính nhóm (attribute group)	63
4.4.7. Tạo nhóm all	64
Bài tập chương 4	65

Chương 5

TRUY XUẤT TÀI LIỆU XML

5.1. Mô hình đối tượng tài liệu (DOM – Document Object Model)	68
5.2. Nạp dữ liệu XML.....	70
5.2.1. Truy xuất giá trị của một nút.....	71
5.2.2. Nạp một chuỗi XML	71
5.3. Giao diện lập trình.....	72
5.3.1. Thuộc tính XML DOM	72
5.3.2. Phương pháp XML DOM (XML DOM Methods)	72
5.3.3. Các nút của DOM (DOM nodes)	72
5.4. Truy cập vào các nút XML DOM	73
5.5. Truy cập vào các nút	75
5.5.1. Phương thức getElementsByTagName ()	75
5.5.2. Duyệt qua các nút của cây.....	75
5.5.3. Điều hướng cây nút.....	76
5.6. Các thuộc tính của một nút trong XML DOM	77
5.6.1. Thuộc tính của nodeName	77
5.6.2. Thuộc tính của nodeValue	77
5.6.3. Thuộc tính của.nodeType	78
5.7. Danh sách nút của XML DOM	78
5.7.1. Chiều dài của danh sách nút.....	79
5.7.2. Danh sách thuộc tính DOM	80
5.8. Duyệt cây nút XML DOM	81
5.8.1. Khoảng trắng và dòng mới trong DOM (White Spaces and New Lines)...	81
5.8.2. Dữ liệu ký tự được phân tích (Parsed Character Data - PCDATA).....	82
5.8.3. Dữ liệu ký tự không được phân tích (Unparsed Character Data - CDATA) ...	82
5.9. Duyệt qua các nút trong DOM	83
5.9.1. Nút cha (ParentNode).....	84
5.9.2. Tránh các nút rỗng	84
5.9.3. Lấy phần tử con đầu tiên.....	84
5.9.4. Lấy phần tử con cuối cùng	85
5.9.5. Lấy nút anh em tiếp theo.....	85
5.9.6. Lấy nút anh em trước đó	86
5.10. Lấy giá trị của nút	86
5.11. Thao tác trên giá trị của các nút.....	88
5.11.1. Thay đổi giá trị của nút	88
5.11.2. Gỡ bỏ các nút (Remove nodes)	89

5.11.3. Thay thế các nút (Replace nodes)	90
5.11.4. Tạo các nút (Create nodes).....	91
5.11.5. Sao chép các nút (Copy nodes)	95
Bài tập chương 5	97

Chương 6

XPATH

6.1. Giới thiệu XPath	100
6.2. Các nút của XPath	100
6.3. Cú pháp của XPath.....	101
6.4. Các ví dụ minh họa XPath	105
6.4.1. XPath_Cú pháp cơ bản.....	105
6.4.2. XPath_Axis	108
Bài tập chương 6	112

Chương 7

ĐỊNH DẠNG TÀI LIỆU XML VỚI XSLT

7.1. Giới thiệu XSLT.....	113
7.1.1. XSLT như là một ngôn ngữ khai báo.....	113
7.1.2. XSLT là một ngôn ngữ chức năng.....	115
7.1.3. Quá trình thực hiện chương trình XSLT	115
7.2. Các phần tử cơ bản của XSLT	117
7.2.1. Phần tử <xsl:stylesheet>	118
7.2.2. Phần tử <xsl:template>	118
7.2.3. Phần tử <xsl: apply-templates >.....	119
7.2.4. Phần tử <xsl: value-of >.....	120
7.2.5. Phần tử <xsl: for-each>.....	121
7.2.6. Phần tử <xsl: call-template>	124
7.2.7. Phần tử logic có điều kiện (conditional logic)	129
7.2.8. Phần tử <xsl:param>	133
7.2.9. Phần tử <xsl:sort>	133

7.2.10. Phần tử <xsl:copy> và <xsl:copy-of>.....	134
7.2.11. Phần tử <xsl:include> và <xsl:import>.....	136
Bài tập chương 7	139

Chương 8

XQUERY

8.1. Cài đặt BaseX và thực hiện truy vấn	142
8.2. Câu lệnh FLWOR.....	143
8.2.1. Mệnh đề for	144
8.2.2. Mệnh đề let	144
8.2.3. Mệnh đề where.....	144
8.2.4. Mệnh đề return	144
8.2.5. Mệnh đề order by	144
8.3. Một số ví dụ XQuery	145
8.4. Một số khái niệm XQuery nâng cao.....	157
Bài tập chương 8	160

Chương 9

XML VÀ CƠ SỞ DỮ LIỆU QUAN HỆ

9.1. Cơ sở dữ liệu quan hệ và XML.....	161
9.2. Sử dụng MySQL với XML	161
9.2.1. Cài đặt MySQL	161
9.2.2. Tạo và nạp một CSDL MySQL	162
9.2.3 Truy vấn MySQL	164
9.2.4. Cập nhật XML trong MySQL	166
9.3. Sử dụng SQL Server với XML	167
9.3.1. Cài đặt SQL Server	167
9.3.2. Trình bày dữ liệu quan hệ dưới dạng XML	168
9.3.3. Tạo tài liệu XML trong SQL Server	178
Bài tập chương 9	187

Chương 10

LINQ VÀ MỘT SỐ ỨNG DỤNG XML

10.1. LINQ là gì?	188
10.2. Sử dụng LINQ.....	189
10.2.1. Tạo tài liệu XML với LINQ.....	190
10.2.2. Tạo tài liệu XML với không gian tên.....	192
10.3. Xử lý dữ liệu XML với LINQ	193
10.3.1. Trích xuất dữ liệu bằng phương thức Elements()	193
10.3.2. Sửa đổi tài liệu XML với LINQ.....	199
10.4. Chuyển đổi tài liệu XML.....	202
10.4.1. Chuyển đổi XML với LINQ	202
10.4.2. Chuyển đổi XML bằng cách sử dụng các tính năng XML Literals của VB.NET	203
10.4.3. Hiểu thuộc tính Axis trong VB.NET.....	206
10.5. Một số ứng dụng liên quan XML	207
10.5.1. Đọc tin nhanh RSS	207
10.5.2. Chuyển đổi CSDL quan hệ sang XML	208
10.5.3. Xây dựng ứng dụng quản lý với XML.....	213
Bài tập chương 10	215

Chương 11

XML VÀ WEB NGỮ NGHĨA

11.1. Web ngữ nghĩa là gì?	217
11.2. Kiến trúc của Web ngữ nghĩa.....	217
11.2.1. Tầng Unicode và URI	217
11.2.2. Tầng XML, NS và XML Schema	218
11.2.3. Tầng RDF và RDF Schema.....	218
11.2.4. Tầng từ vựng Ontology	219
11.2.5. Tầng Logic	219
11.2.6. Tầng Proof và tầng Trust	219
11.3. RDF – Khung mô tả tài nguyên.....	220
11.3.1. Cấu trúc RDF/XML	221
11.3.2. RDF Container	222
11.3.3. RDF Collection	223

11.4. Ngôn ngữ Ontology	224
11.4.1. RDF Schema	224
11.4.2. OWL (Ontology Web Language).....	229
11.5. Công cụ xây dựng và quản trị ontology	231
11.5.1 Giới thiệu	231
11.5.2. Protégé	231
11.5.3. Các ứng dụng của Web ngữ nghĩa	232
TÀI LIỆU THAM KHẢO	235
DANH MỤC CÁC TỪ VIẾT TẮT	236

PHẠM THỊ THU THỦY

CÔNG NGHỆ XML VÀ ỨNG DỤNG

Chịu trách nhiệm xuất bản:
GIÁM ĐỐC - TỔNG BIÊN TẬP
PHẠM NGỌC KHÔI

Biên tập : Trương Thanh Sơn
Trình bày : Nguyễn Thị Loan
Thiết kế bìa : Hoàng Việt

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

70 Trần Hưng Đạo – Quận Hoàn Kiếm – Hà Nội

ĐT: (024) 3942 2443 Fax: (024) 3822 0658

Website: <http://www.nxbkhkt.com.vn>. Email: nxbkhkt@hn.vnn.vn

CHI NHÁNH NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

28 Đồng Khởi – Quận 1 – TP Hồ Chí Minh

ĐT: (028) 3822 5062 Fax: (028) 3829 6628

Email: chinhhanhnxbkhkt@yahoo.com.vn

(Sách thật có đóng dấu và dán tem ở bìa 3)

In 300 bản, khổ 16^{cm} × 24^{cm} tại Công ty CPTM In Nhật Nam

Địa chỉ: 007 Lô I – KCN Tân Bình – Quận Tân Phú – TP Hồ Chí Minh

Số đăng ký xuất bản: 4364 – 2017 / CXBIPH / 2 – 151 / KHKT

Quyết định xuất bản số: 210 / QĐ - NXBKHK, ngày 12 / 12 / 2017

Mã ISBN: 978 – 604 – 67 – 1027 – 1

In xong và nộp lưu chiểu quý I năm 2018