การวิเคราะห์และเปรียบเทียบภาษาสืบค้นสำหรับ XML ระหว่าง XQuery และ XSLT

เอกพล จีรังสุวรรณ และ สมนึก คีรีโต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์

ABSTRACT -- XML is a great technology that changes the way of collecting and using data. It can also merge the usage of database and document together. However, using XML with most efficiency it is necessary to deal with an appropriate query language too. At present, XQuery is the lastest query language for XML that is proposed by W3C, the organization that has the authority to settling many Internet standards. XQuery is being developed continuously and is expected to become the complete standard of query language for XML soon. However, in opinions of authors, XQuery still lacks several features to manage data collecting in document forms. This article is an analysis and comparison between 2 famous XML proposals related to query fields, XQuery and XSLT. XSLT is the standard for transforming XML document that can work with document very well. This article identifies differences, advantages and disadvantages between both languages and hopefully this article can bring some ideas to develop a complete standard query language for XML.

KEY WORDS -- XML, Query Language, XQuery, XSLT

บทคัดย่อ -- XML เป็นเทคโนโลยีซึ่งสร้างรูปแบบใหม่ในการจัดเก็บและใช้งานข้อมูล ทำให้การใช้งานร่วมกันระหว่างฐานข้อมูลและเอกสารเป็นไปอย่างราบรื่น อย่างไรก็ตามการใช้งาน XML ให้มีประสิทธิภาพมากที่สุดนั้นจำเป็นต้องทำงานร่วมกับภาษาสืบค้นสำหรับ XML ที่เหมาะสมด้วย, ปัจจุบัน XQuery นับเป็นภาษาสืบค้นสำหรับ XML ล่าสุดที่เสนอโดยหน่วยงาน W3C ซึ่งมีหน้าที่กำกับดูแลเกี่ยวกับมาตรฐานต่างๆ ของอินเตอร์เน็ต, XQuery ยังคงได้รับการพัฒนาอยู่อย่างต่อเนื่องและเป็นที่คาดกันว่า XQuery จะกลายเป็นมาตรฐานของภาษาสืบค้นสำหรับ XML ที่สมบูรณ์ในอนาคตอันใกล้ อย่างไรก็ตาม ในมุมมองของผู้เขียนมีความเห็นว่า XQuery นั้นยังขาดคุณสมบัติที่ดีในการจัดการกับข้อมูล XML ที่อยู่ในรูปแบบของเอกสารหลายๆ ประการ, บทความนี้นำเสนอการวิเคราะห์และเปรียบเทียบระหว่าง XQuery และ XSLT, สำหรับ XSLT นั้นเป็นมาตรฐานในการแปลงข้อมูลที่ถูกออกแบบมาเพื่อการจัดการกับเอกสารโดยเฉพาะ โดยพยายามชี้ให้เห็นถึงความเหมือนและความแตกต่าง รวมถึงข้อดีและข้อเสียระหว่างภาษาทั้ง 2 แบบ เพื่อเป็นแนวทางในการพัฒนามาตรฐานของภาษาสืบค้นสำหรับ XML ที่สมบูรณ์ยิ่งขึ้นต่อไป

คำสำคัญ -- ภาษาสืบค้น, XML, XQuery, XSLT

1. บทนำ

Extensible Markup Language (XML) [7] เป็นภาษา markup เช่นเดียวกับ Hypertext Markup Language (HTML) [8] แต่จะเหนือกว่าตรงที่ XML จะยอมให้ผู้ใช้นิยาม markup หรือ tag ไว้ใช้งานได้เอง ทำให้ไม่ติดกับ รูปแบบและ tag ที่มีจำนวนจำกัดเหมือนกับที่มีอยู่ใน HTML นอกจากนี้ tag ของ XML จะทำหน้าที่ในการขยายความหมายของข้อมูลให้สมบูรณ์ยิ่งขึ้น อันจะทำให้การประมวลผลข้อมูลอัตโนมัติเป็นไปอย่างมีประสิทธิภาพมากขึ้น ดังตัวอย่างเปรียบเทียบข้อมูลที่อยู่ในรูปแบบของ HTML และ XML ต่อไปนี้

ข้อมูลในรูปแบบของ HTML

```
<b>book</b>
<table>
 <tr><td>year</td><td>1992</td></tr>
 <tr><td>title</td><td>XML Bible</td></tr>
 <tr><td>publisher</td><td>Addison-
Wesley</td></tr>
 <tr><td>author</td><td>John Smith</td></tr>
 <tr><td>price</td><td>US$60</td></tr>
</table>
```

ข้อมูลในรูปแบบของ XML

```
<book year="1992">
  <title>XML Bible</title>
  <publisher>Addison-Wesley</publisher>
```

```
  <author>
    <last>John</last>
    <first>Smith</first>
  </author>
  <price>US$60</price>
</book>
```

จะเห็นได้ว่าข้อมูลเดียวกัน เมื่อนำเสนอด้วย HTML จะใช้เพื่อการแสดงผลเป็นหลัก และการนำข้อมูลที่อยู่ในรูปของ HTML กลับมาใช้ใหม่หรือนำมาประมวลผล จะเป็นไปได้ยากเพราะ tag <b>, <table>, <tr>, <td> ไม่ได้ช่วยสื่อให้เห็นถึงความหมายของข้อมูลที่อยู่ภายในแต่อย่างไร แต่ XML นั้นอนุญาตให้มี markup ที่สร้างขึ้นเองเพื่อใช้ในการอธิบายความหมายของข้อมูลด้วย เช่น tag <book>, <title>, <publisher> ทำให้เห็นได้ว่า XML นั้นทำให้ข้อมูลมีความหมายสมบูรณ์ยิ่งขึ้นและทำให้ง่ายต่อการนำมาประมวลผล

XML ยังสามารถนำมาใช้ร่วมกับงานทางด้านเอกสาร ทำให้ข้อจำกัดในการใช้งานฐานข้อมูลและเอกสารร่วมกันหมดไป อันจะทำให้สามารถประยุกต์ใช้งานข้อมูลทั้ง 2 รูปแบบได้หลากหลายมากยิ่งขึ้น อย่างไรก็ตามข้อมูลที่เก็บด้วย XML นั้นจะไม่มีประโยชน์เมื่อไม่สามารถนำออกมาใช้ หรือเรียกค้นได้ตามความต้องการ จึงเกิดความจำเป็นในการใช้งานภาษาสืบค้นสำหรับ XML ขึ้น เช่นเดียวกับภาษา Structured Query Language (SQL) [4] ที่ใช้สำหรับฐานข้อมูลแบบสัมพันธ์ (Relational Database) แตกต่างแต่ XML นั้นเป็นฐานข้อมูลที่มีความซับซ้อนมากกว่า และยังสามารถใช้งานร่วมกับข้อมูลที่เป็นเอกสารได้ด้วย

การพัฒนาภาษาสืบค้นสำหรับ XML นั้นเป็นไปอย่างต่อเนื่อง ภาษาสืบค้นหลายๆ แบบถูกนำเสนอออกมา ซึ่งแต่ละแบบก็ล้วนมีจุดเด่นและจุดด้อยแตกต่างกันไป ภาษาสืบค้นสำหรับ XML ในรุ่นหลังๆ ใช้ภาษาสืบค้นในรุ่นแรกๆ เป็นพื้นฐาน คุณสมบัติหลายๆ อย่างได้รับการพัฒนาให้ดีขึ้น ในขณะที่พยายามลดข้อจำกัดและข้อเสียต่างๆลง แต่ปัญหาที่สำคัญที่สุดในการออกแบบก็คือ การทำให้ภาษาสืบค้นสำหรับ XML นี้สามารถใช้งานได้ทั้งฐานข้อมูลและเอกสารร่วมกันอย่างสมบูรณ์ โดยทั่วไป ภาษาสืบค้นที่พัฒนามาในแนวทางของฐานข้อมูลก็จะทำงานกับฐานข้อมูลได้ดีแต่ไม่สามารถทำงานกับเอกสารได้ดีเท่าที่ควร ในขณะที่ภาษาสืบค้นที่พัฒนามาเพื่อเน้นการจัดการกับเอกสาร ก็จะขาดคุณสมบัติในการจัดการฐานข้อมูลที่ดีไป ทำให้ภาษาสืบค้นสำหรับ XML ส่วนใหญ่จะใช้งานได้ดีกับข้อมูลเฉพาะแบบเท่านั้น

ในปัจจุบัน W3C ซึ่งเป็นหน่วยงานที่กำกับดูแลเกี่ยวกับมาตรฐานต่างๆ ของอินเตอร์เน็ตได้เสนอภาษาสืบค้นสำหรับ XML แบบล่าสุดออกมา คือ XQuery [14] ซึ่งได้รับการพัฒนามาอย่างต่อเนื่องโดยกลุ่มนักวิจัยของ W3C, แม้ขณะนี้ XQuery ยังมีสถานะเป็นแค่ Working Draft ซึ่งจะต้องได้รับการพิจารณา และปรับปรุงอีกมาก แต่ก็เป็นที่คาดกันว่า XQuery จะได้รับการพัฒนาต่อจนกลายเป็นมาตรฐานของภาษาสืบค้นสำหรับ XML ที่สมบูรณ์ในอนาคตอันใกล้นี้, XQuery นั้นถูกพัฒนาโดยมีพื้นฐานมาจากภาษา SQL ซึ่งใช้งานได้ง่ายและเป็นที่ยอมรับอย่างแพร่หลาย จึงทำให้ XQuery สามารถทำความเข้าใจและใช้งานได้ง่ายเช่นเดียวกัน อย่างไรก็ตามด้วยเหตุผลนี้จึงทำให้ XQuery ต้องใช้รูปแบบการทำงานที่ไม่ใช่ XML นอกจากนี้ XQuery ยังถูกพัฒนามาในแนวทางของ ฐานข้อมูล จึงทำให้ขาดคุณสมบัติที่ดีในการจัดการข้อมูลที่อยู่ในรูปของเอกสารหลายๆ ประการไป, รายละเอียดและการทำงานของ XQuery จะกล่าวถึงต่อไปในหัวข้อที่ 2, การแนะนำ XQuery

ในบทความนี้ จะกล่าวถึงมาตรฐานอีกหนึ่งมาตรฐานซึ่งมีบทบาทสำคัญในการจัดการกับเอกสารที่อยู่ในรูปของ XML ก็คือ Extensible Stylesheet Language - Transformation (XSLT) [15] , XSLT ถูกออกแบบมาเพื่อการแสดงผลเอกสาร XML, อย่างไรก็ตาม XSLT กลับสามารถทำงานในการสืบค้นข้อมูล XML ได้เป็นอย่างดี [2] นอกจากนี้ยังมีบางบทความชี้ให้เห็นว่าคำสั่งและการทำงานต่างๆ ของ XQuery นั้น ซ้ำซ้อนกับที่มีอยู่แล้วใน XSLT [3] อย่างไรก็ตามในความคิดเห็นของ ผู้เขียน บทความข้างต้นเป็นมุมมองจากทางด้านของ XSLT มากจนเกินไป ผู้เขียนยอมรับว่าความสามารถหลายๆ อย่างของ XQuery คล้ายคลึงหรือซ้ำซ้อนกับ XSLT จริง อย่างไรก็ตามบทความข้างต้นเป็นเพียงการแสดงตัวอย่างเปรียบเทียบให้เห็นเท่านั้น ไม่ได้มีการวิเคราะห์ให้เห็นถึงรายละเอียดและแนวคิดพื้นฐานที่แตกต่างกัน รวมถึงไม่ได้นำเสนอใน มุมมองที่เป็นข้อได้เปรียบของ XQuery และข้อเสียที่ชัดเจนของ XSLT ออกมา, รายละเอียดและการทำงานของ XSLT จะกล่าวถึงต่อไปใน หัวข้อที่ 3, การแนะนำ XSLT

ในความคิดเห็นของผู้เขียน ทั้ง XQuery และ XSLT นั้นแม้จะมีความสามารถบางส่วนที่ซ้ำซ้อนกันบ้าง แต่ก็ยังมีความสามารถและรายละเอียดอีกหลายๆ ส่วนที่ไม่สามารถทำงานทดแทนกันได้ ซึ่งทำให้ภาษาทั้ง 2 แบบมีความสามารถเฉพาะที่แตกต่างกันไป, บทความนี้เป็นการวิเคราะห์ให้เห็นถึงจุดเหมือน และจุดที่แตกต่างที่สำคัญบางประการระหว่างภาษาทั้ง 2 แบบ โดยหวังว่าจะสามารถเป็นแนวทางในการรวมความสามารถและจุดเด่นที่มีอยู่ในทั้ง 2 ภาษานี้เข้าด้วยกันเพื่อจะนำไปสู่การพัฒนาภาษาสืบค้นสำหรับ XML ที่สมบูรณ์ สามารถเรียกค้นข้อมูลได้ตามความต้องการได้อย่างยืดหยุ่น และสามารถทำงานร่วมกันระหว่างข้อมูลและเอกสารได้อย่างเหมาะสมที่สุด

ในบทความนี้ จะใช้ตัวอย่างเพื่อช่วยให้สามารถเข้าใจการทำงานของภาษาทั้ง 2 แบบได้ดียิ่งขึ้น โดยกำหนดให้เอกสาร XML ตัวอย่างคือ bib.xml โดยมี Document Type Definition (DTD) ดังนี้

```
<!ELEMENT bib (book*)>
<!ELEMENT book (title, (author+ |editor+),
                publisher, price)>

<!ATTLIST book year CDATA>
<!ELEMENT author (last, first)>
<!ELEMENT editor (last, first, affiliation)>
<!ELEMENT title #PCDATA>
<!ELEMENT last  #PCDATA>
<!ELEMENT first #PCDATA>
<!ELEMENT affiliation #PCDATA>
<!ELEMENT publisher #PCDATA>
<!ELEMENT price #PCDATA>
```

เอกสาร bib.xml มี root element เป็น bib (บรรณานุกรม) โดยใน bib ประกอบไปด้วยหลายๆ book ซึ่งหมายถึงหนังสือแต่ละเล่ม ในหนังสือแต่ละเล่มจะประกอบไปด้วย attribute year ซึ่งแสดงปีที่พิมพ์, ชื่อหนังสือ (title), รายชื่อผู้แต่ง (author) หรือรายชื่อบรรณาธิการ (editor) อย่างใดอย่างหนึ่ง ตามด้วยชื่อสำนักพิมพ์ (publisher) และราคาหนังสือ (price), ชื่อผู้แต่งและบรรณาธิการจะประกอบไปด้วย นามสกุล (last) และ ชื่อ (first) นอกจากนั้นชื่อบรรณาธิการยังประกอบไปด้วยชื่อหน่วยงาน (affiliation) ด้วย

2. แนะนำ XQuery

XQuery เป็นภาษาสืบค้นสำหรับ XML ในรุ่นหลัง ซึ่งได้รับอิทธิพลมาจากภาษาสืบค้นสำหรับ XML ในรุ่นแรกๆ และมาตรฐานอื่นๆ ที่เกี่ยวข้อง อันได้แก่ XPath [10], XQL [5], SQL, XML-QL [1], OQL [6] เป็นต้น ดังมีรายละเอียดต่อไปนี้

XQuery จะใช้แนวคิดของ path expression ในการระบุไปยังข้อมูลตำแหน่งต่างๆ ในเอกสารที่กำลังสนใจ, path expression ของ XQuery จะอ้างอิงมาจากมาตรฐานของ XPath เสริมด้วยคำสั่งหรือฟังก์ชันบางส่วนที่ขอยืมมาจาก XQL, path expression เป็นวิธีที่สะดวกและมีประสิทธิภาพมากในการระบุไปยังข้อมูลที่สนใจในโครงสร้างที่ซับซ้อนของเอกสาร ตัวอย่าง path expression เช่น document("bib.xml")/bib/book หมายถึง element ที่ชื่อ book ทุกๆ อันที่อยู่ภายใต้ root element ที่ชื่อ bib ของเอกสาร bib.xml

โครงสร้างการทำงานของ XQuery จะประกอบไปด้วยลำดับของประโยค FOR - LET - WHERE - RETURN ซึ่งได้รับอิทธิพลมาจากภาษา SQL ที่เป็นลำดับประโยค SELECT - FROM - WHERE, โดยประโยค FOR จะระบุไปยังข้อมูลที่กำลังสนใจโดยอาศัย path expression และนำข้อมูลนั้นมากำหนดให้กับตัวแปร ในแต่ละรอบของการทำงาน, ประโยค LET จะใช้ในการกำหนดค่าให้กับตัวแปรเพื่อช่วยเสริมการทำงานกับประโยค FOR, ประโยค WHERE จะตรวจสอบเงื่อนไขของข้อมูลที่เก็บอยู่ใน ตัวแปรที่ได้มาจากในส่วนของประโยค FOR, LET ที่อยู่ข้างต้นและส่งผลลัพธ์ที่ตรงตามเงื่อนไขให้กับส่วนประโยค RETURN เพื่อสร้างผลลัพธ์ที่ต้องการ

XQuery จะใช้งานตัวแปรเพื่อเชื่อมโยงความสัมพันธ์ระหว่างส่วนของประโยคต่างๆ เข้าด้วยกัน ซึ่งได้รับอิทธิพลมาจาก XML-QL, และ XQuery จะประกอบไปด้วยการใช้งานประโยคคำสั่งต่างๆ, การเรียกใช้งานฟังก์ชัน, การนิยามฟังก์ชัน เช่นเดียวกับที่มีในภาษา OQL

XQuery นั้นถือได้ว่าเป็นภาษาสืบค้นสำหรับ XML ที่มีประสิทธิภาพมากที่สุดในปัจจุบัน อย่างไรก็ตาม XQuery นั้นถูกพัฒนามาในแนวทางของฐานข้อมูล จึงทำให้สามารถจัดการกับข้อมูลที่อยู่ในรูปของฐานข้อมูลได้ดี แต่สำหรับข้อมูลที่อยู่ในรูปของเอกสารนั้นยังมีข้อจำกัดอยู่บางประการ ดังจะได้พิจารณาให้เห็นต่อไป

ตัวอย่างการใช้งาน XQuery เพื่อเรียกค้นข้อมูล เป็นดังนี้

```
<results>
  {
    FOR $b IN document("bib.xml")/bib/book
    WHERE $b/publisher = "Addison-Wesley" AND
          $b/@year > 1991
    RETURN
      <book year={ $b/@year }>
        { $b/title }
      </book>
  }
</results>
```

จากตัวอย่าง เป็นการหารายชื่อหนังสือและปีที่พิมพ์ โดยที่หนังสือเล่มนั้นๆ ต้องพิมพ์โดยสำนักพิมพ์ Addison-Wesley หลังจากปี 1991, ประโยค FOR จะกำหนดค่าของแต่ละ element ที่ชื่อ book ที่คืนมาจาก path expression, document("bib.xml")/bib/book มาเก็บไว้ในตัวแปร $b, ต่อจากนั้นประโยค WHERE จะตรวจสอบเงื่อนไขของค่าที่เก็บอยู่ใน ตัวแปร $b ว่าเป็นไปตามที่ต้องการ ในที่นี้คือมีค่าของ element publisher เป็น "Addison-Wesley" และค่าของ attribute year มากกว่า 1991 (เครื่องหมาย @ แสดงถึงการเป็น attribute), หลังจากนั้นประโยค RETURN ก็จะอาศัยค่าตัวแปร $b เพื่ออ้างอิงถึง ค่าของ attribute year และ element title ในการสร้างผลลัพธ์ที่ต้องการ

3. แนะนำ XSLT

ใน HTML นั้น tag ที่ใช้ได้จะมีจำนวนจำกัด เนื่องจาก tag แต่ละตัวจะมีความหมายในแง่ของการแสดงผลที่แตกต่างกันไป เช่น tag ที่ใช้ในการแสดงย่อหน้า, ตาราง หรือ รูปภาพ เป็นต้น แต่ XML จะยอมให้ผู้ใช้นิยาม tag เพื่อใช้งานเองได้ ปัญหาที่ตามมาก็คือ โปรแกรม browser จะไม่รู้จัก tag เหล่านี้ทำให้ไม่สามารถนำมาแสดงผลได้ ดังนั้นเพื่อเป็นการนำเสนอหรือแสดงผลข้อมูล XML จึงมีการพัฒนา XSLT ขึ้น เพื่อใช้ในการแปลงเอกสาร XML, โดยจะแปลง tag ที่ผู้ใช้นิยามขึ้น ให้กลายเป็น tag ที่โปรแกรม browser สามารถเข้าใจ และนำไปแสดงผลได้ ซึ่งโดยทั่วไปก็จะเป็นการแปลงจากเอกสาร XML ให้เป็นเอกสาร HTML เป็นต้น ดังตัวอย่าง

```
<xsl:template match="book">
  <font color="red" size="20">
    <xsl:value-of select="title">
  </font>
  <table border="1">
    <xsl:apply-templates select="author"/>
  </table>
</xsl:template>

<xsl:template match="author">
  <tr>
    <td><xsl:value-of select="last"/></td>
    <td><xsl:value-of select="first"/></td>
  </tr>
</xsl:template>
```

จากตัวอย่างจะเป็นการแปลงเอกสาร XML ที่มี element ที่ชื่อbook นำมาแสดงผลด้วย HTML โดย เมื่อพบ element ที่ชื่อ book ก็ให้พิมพ์ชื่อหนังสือ (title) ด้วยตัวอักษรสีแดงขนาด 20 และตามด้วยตารางซึ่งแสดงชื่อผู้แต่ง (author) ของหนังสือเล่มนั้นๆ

คำสั่งของ XSLT นั้นจะเป็น element ของ XML ตามปกติ เพียงแต่จะถูกนำหน้าด้วย xsl: (มี prefix ของ namespace [9] เป็น xsl), การทำงานของ XSLT จะประกอบไปด้วยหลาย template, แต่ละ template จะอยู่ภายใน element xsl:template โดยจะมี attribute ที่ชื่อ match ซึ่งจะมีค่าเป็น path expression (XPath) ที่ระบุถึงข้อมูลใดๆ ในเอกสาร XML, เมื่ออ่านข้อมูลภายในเอกสาร XML มา match กับ path expression ของ template ใด template นั้นก็จะถูกเรียกขึ้นมาทำงาน เช่น จากตัวอย่าง เมื่ออ่านข้อมูลภายในเอกสาร XML มาพบ element ที่ชื่อ book, template แรกก็จะถูกเรียกขึ้นมาทำงาน โดยจะพิมพ์ tag <font> และค่าของชื่อหนังสือ ด้วย คำสั่ง xsl:value-of จากนั้นจึงพิมพ์ตาราง <table> โดยเนื้อหาภายใน ตารางนั้นจะส่งการทำงานไปให้ template ที่สองด้วยการส่งค่า element author ที่พบไปให้

นอกจากจะใช้ XSLT เพื่อแปลงเอกสาร XML ให้อยู่ในรูปของ HTML แล้ว XSLT นั้นยังสามารถแปลงเอกสาร XML ให้อยู่ในรูปแบบของเอกสารใดๆ ก็ได้ เช่น ไฟล์ XML เองหรือแม้แต่ไฟล์ข้อความ ทำให้ประโยชน์ของ XSLT ไม่ได้จำกัดอยู่แค่งานแปลงเอกสารเพื่อแสดงผลเท่านั้น แต่สามารถนำมาประยุกต์ใช้งานในการจัดการเอกสารได้อย่างกว้างขวาง

ในที่นี้ได้นำ XSLT มาใช้แทนภาษาสืบค้นในการเรียกดูข้อมูล XML ซึ่งสามารถใช้งานได้ดีในระดับหนึ่ง ดังตัวอย่าง

```
<xsl:template match="/">
  <results>
    <xsl:apply-templates select="bib/book"/>
  </results>
</xsl:template>

<xsl:template match="book[
  publisher = 'Addison-Wesley' and @year >
1991]">
  <book year="{@year}">
    <title>
      <xsl:value-of select="title"/>
    </title>
  </book>
</xsl:template>
```

จากตัวอย่าง จะเป็นการหาหนังสือที่พิมพ์โดยสำนักพิมพ์ "Addison-Wesley" หลังจากปี 1991 เช่นกัน, ในที่นี้ประกอบไปด้วย 2 template โดย template แรกจะมี path expression ที่ match กับ root ของเอกสาร, ภายใน template นี้จะสร้าง element results สำหรับคำตอบ และส่งการทำงานต่อไปให้ template ถัดไปด้วยคำสั่ง xsl:apply-templates เพื่อส่งค่า element bib/book ออกไป, template ต่อมาจะมี path expression ที่รับกันได้กับ element book ที่มีเงื่อนไขเพิ่มเติมอยู่ด้วย หากข้อมูล book ที่ถูกส่งมาจาก template แรก match ตามเงื่อนไขนี้ template นี้ก็จะถูกเรียกให้ทำงานโดยสร้าง element book และข้อมูลอื่นๆ ตามที่ต้องการ

XSLT นั้นถูกออกแบบมาเพื่อการใช้งานกับข้อมูล XML ที่อยู่ในรูปของเอกสารโดยเฉพาะ จึงประกอบไปด้วยคำสั่งและคุณสมบัติมากมายที่สามารถจัดการกับข้อมูลเอกสารและส่วนประกอบอื่นๆ ได้อย่างมีประสิทธิภาพ นอกจากนั้น XSLT ยังสามารถใช้งานในการสืบค้นข้อมูลได้อีก อย่างไรก็ตาม XSLT ก็ยังขาดคุณสมบัติในการเป็นภาษาสืบค้นสำหรับ XML ที่ดีในหลายๆ ประการดังจะได้แสดงให้เห็นต่อไป

4. การเปรียบเทียบระหว่าง XQuery และ XSLT

4.1 โครงสร้างการทำงานพื้นฐาน

การทำงานในการเรียกค้นข้อมูลของ XQuery นั้นจะใช้ประโยค FOR - LET - WHERE - RETURN ในขณะที่ XSLT จะใช้ template, การทำงานของภาษาทั้ง 2 แบบมีจะมีความคล้ายคลึงกันบางอย่าง ซึ่งสามารถแสดงให้เห็นจากตัวอย่างต่อไปนี้

การทำงานของ XQuery

```
FOR $b IN document("bib.xml")/bib/book
RETURN <book year={ $b/@year }>
         { $b/title }
       </book>
```

การทำงานของ XSLT

```
<xsl:template match="/bib/book">
  <book year="{@year}">
    <title>
      <xsl:value-of select="title"/>
    </title>
  </book>
</xsl:template>
```

ภายในประโยค FOR และคำสั่ง xsl:template จะมีส่วนที่ระบุไปยัง path expression เช่นเดียวกัน โดยหากพบข้อมูลตามที่ระบุไว้ ภายในประโยค FOR หรือ template นั้นๆ ก็จะทำงานเพื่อสร้างคำตอบ และจะวนไปจนกว่าจะครบทุกข้อมูลที่พบ จากตัวอย่างข้างต้นจะเห็นได้ว่าโครงสร้างของ XQuery และ XSLT นั้นไม่มีความแตกต่างกัน อย่างไรก็ตามหากการทำงานภายในแต่ละรอบมีรายละเอียดลึกลงไปอีก ก็จะสามารถมองเห็นถึงความแตกต่างระหว่างภาษาทั้ง 2 แบบได้ ดังตัวอย่าง

การทำงานของ XQuery

```
FOR $b IN document("bib.xml")/bib/book
RETURN <book year={ $b/@year }>
         { $b/title }
         {
           FOR $a IN $b/author
           RETURN $a
         }
       </book>
```

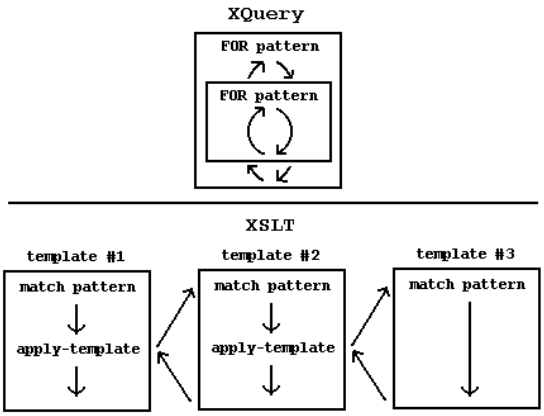
การทำงานของ XSLT

```
<xsl:template match="/bib/book">
  <book year="{@year}">
    <title>
      <xsl:value-of select="title"/>
    </title>
    <xsl:apply-templates select="author"/>
  </book>
</xsl:template>

<xsl:template match="author">
  <author>
    <last>
      <xsl:value-of select="last"/>
    </last>
    <first>
      <xsl:value-of select="first"/>
    </first>
  </author>
</xsl:template>
```

ตัวอย่างนี้จะคล้ายกับตัวอย่างก่อนหน้านี้ เพียงแต่จะเพิ่มการเรียกดูในส่วนของชื่อผู้แต่งขึ้นมา ในหนังสือแต่ละเล่มอาจจะมีผู้แต่งได้หลายคน ดังนั้น XQuery จะใช้ FOR ซ้อนอีกชั้นเพื่อ match ไปยังชื่อผู้แต่งภายใน FOR ครั้งแรกที่ match ไปยังหนังสือแต่ละเล่ม ในขณะที่ XSLT จะแยกการทำงานในส่วนนี้ออกเป็นอีก template แล้วให้ template แรกส่งการทำงานต่อมาให้

XQuery จะทำงานโดยมีประโยค FOR เป็นหลัก ภายในประโยค FOR จะประกอบไปด้วยประโยค อื่นๆ เช่น ประโยค WHERE, ประโยค RETURN และอาจประกอบไปด้วยประโยค FOR อื่นๆ ด้วย, XQuery จะทำงานโดยรวมทุกอย่างไว้ที่จุดเดียวกัน ความซับซ้อนของการทำงานจะอยู่ในทิศทางที่ลึกลงไปในประโยค FOR, ในขณะที่ XSLT จะแยกทำงานย่อยๆ ออกเป็นแต่ละ template และให้ template เหล่านี้ส่งการทำงานไปมาด้วยความสัมพันธ์ของคำสั่ง xsl:apply-templates และ พารามิเตอร์ match ของคำสั่ง xsl:template



รูปที่ 1. แสดงโครงสร้างการทำงานพื้นฐานของ XQuery และ XSLT

แม้การทำงานโดยทั่วไปของ XSLT จะแบ่งการทำงานย่อยๆ ออกเป็น template, อย่างไรก็ตาม XSLT นั้นสามารถมีโครงสร้างในการทำงานเหมือนประโยค FOR ของ XQuery ด้วยคำสั่ง xsl:for-each ดังตัวอย่าง

```
<xsl:template match="/bib/book">
  <book year="{@year}">
    <title>
      <xsl:value-of select="title"/>
    </title>
    <xsl:for-each select="author"/>
      <author>
        <last>
          <xsl:value-of select="last"/>
        </last>
        <first>
          <xsl:value-of select="first"/>
        </first>
      </author>
    </xsl:for-each>
  </book>
</xsl:template>
```

จากตัวอย่างนี้ แทนที่จะใช้คำสั่ง xsl:apply-templates เพื่อส่งการทำงานให้ template อื่น แต่จะใช้คำสั่ง xsl:for-each ทำให้การ match และการทำงานทั้งหมดจะอยู่ภายใน template เดียว ซึ่งมีโครงสร้างการทำงานเป็นเช่นเดียวกับ XQuery

โดยปกติ XQuery จะทำงานโดยประกอบไปด้วยประโยค FOR หลักเพียงส่วนเดียว อย่างไรก็ตาม XQuery สามารถกระจายการทำงานจากส่วนหลักส่วนเดียวไปยังส่วนย่อยอื่นๆ ได้อีก คล้ายแนวคิด template ของ XSLT ด้วยคุณสมบัติของฟังก์ชัน ดังตัวอย่าง

```
FUNCTION My_Function(ELEMENT $b) RETURN ELEMENT {
  <book year={ $b/@year }>
    { $b/title }
  </book>
}
```

```
<results>
  {
    FOR $b IN document("bib.xml")/bib/book
    WHERE $b/@year > 1991 AND
          $b/publisher = "Addison-Wesley"
    RETURN My_Function($b)
  }
</results>
```

ในที่นี้ ภายในประโยค FOR ที่เป็นส่วนหลักจะไม่ทำงานด้วยตัวเองแต่จะไปเรียกฟังก์ชันให้ทำงานแทน, ภายในแต่ละฟังก์ชันก็อาจเรียกไปยังฟังก์ชันอื่นๆ ได้อีก ผ่านการทำงานต่อไปเรื่อยๆ คล้ายแนวคิดของการส่งการทำงานระหว่าง template ของ XSLT นั่นเอง

สรุป -- โครงสร้างพื้นฐานของ XQuery จะทำงานด้วยประโยค FOR ซึ่งมีคำสั่งอื่นๆ อยู่ภายใน นั่นคือ XQuery จะทำงานโดยมีส่วนหลักเป็น ศูนย์กลางเพียงส่วนเดียว แต่อาจกระจายการทำงานออกเป็นส่วนย่อยๆ ได้ด้วยฟังก์ชัน ส่วน XSLT นั้นจะกระจายการทำงานด้วย template โดยแต่ละ template จะมีความสำคัญเท่าเทียมกันไม่มีส่วนใดเป็นส่วนหลัก อย่างไรก็ตาม XSLT นั้นอาจใช้คำสั่ง xsl:for-each เพื่อเลียนแบบการทำงานแบบรวมศูนย์ของ XQuery ได้เช่นกัน จะเห็นได้ว่าภาษาสืบค้นทั้งสองแบบสามารถจำลองโครงสร้างการทำงานของกันและกันได้ อย่างไรก็ตามด้วยคุณสมบัติพื้นฐานบางประการทำให้ภาษาสืบค้นทั้งสองแบบไม่สามารถทดแทนการทำงานกันได้สมบูรณ์ทุกประการดังจะได้กล่าวถึงต่อไป

4.2 การใช้งานตัวแปร

ในการเรียกค้นข้อมูลด้วย XQuery นั้น ข้อมูลในส่วนต่างๆ จะถูกสร้างความสัมพันธ์และอ้างอิงกันด้วยตัวแปร, ในขั้นตอนการทำงานของ FOR ค่า node ของข้อมูลที่สัมพันธ์กับ path expression จะถูกกำหนดให้กับตัวแปรในแต่ละรอบของการทำงานและสามารถอ้างอิงได้ทุกๆ จุดภายในรอบการทำงานนั้น จากตัวอย่าง

```
<results>
  {
   FOR $a IN distinct(document("bib.xml")//author)
   RETURN
     <result>
       { $a }
       {
       FOR $b IN
         document("bib.xml")/bib/book[author = $a]
         RETURN $b/title
       }
     </result>
  }
</results>
```

จากตัวอย่างข้างต้น เป็นการจัดรายชื่อหนังสือโดยแบ่งตามชื่อผู้แต่ง การทำงานจะเริ่มจาก FOR ครั้งแรกเพื่อหาชื่อผู้แต่งแต่ละคนที่ไม่ซ้ำกันออกมาด้วยฟังก์ชัน distinct() และ FOR ภายในเพื่อหารายชื่อหนังสือทุกเล่มที่แต่งโดยผู้แต่งคนนี้ การ FOR ครั้งแรกในแต่ละรอบของการทำงาน ชื่อของผู้แต่งแต่ละคนจะถูกกำหนดให้กับตัวแปร $a ซึ่งสามารถถูกใช้งานได้อย่างอิสระภายใน เช่น การสร้างคำตอบ $a เพื่อแสดงชื่อของผู้แต่งเองและจาก document("bib.xml")/bib/book[author = $a] เพื่อหาหนังสือที่มีชื่อผู้แต่งเหมือนกับค่าของตัวแปร $a, ภายใน FOR ครั้งที่สอง จะได้ตัวแปรใหม่อีกตัวคือ $b เพื่ออ้างอิงถึงข้อมูลหนังสือที่มีเงื่อนไขที่ต้องการ

การทำงานโดยทั่วไปของ XSLT จะไม่ได้ใช้ตัวแปรแต่จะอาศัยแนวคิดของ context คือภายใน template ใดๆ การใช้งาน element หรือส่วนประกอบใดๆ จะถูกอ้างอิงกับ node ที่ match ตรงกับเงื่อนไขของ path expression ในรอบนั้นๆ เช่น จากตัวอย่าง

```
<xsl:template match="book">
  <book year="{@year}">
    <title>
      <xsl:value-of select="title"/>
    </title>
    <xsl:apply-templates select="author"/>
  </book>
</xsl:template>

<xsl:template match="author">
  <author>
    <last>
      <xsl:value-of select="last"/>
    </last>
    <first>
      <xsl:value-of select="first"/>
    </first>
  </author>
</xsl:template>
```

จากตัวอย่างข้างต้น, template แรกจะ match กับหนังสือทุกเล่ม โดยผลลัพธ์ที่ต้องการในขั้นตอนนี้คือปีที่พิมพ์และชื่อหนังสือ นอกจากนั้นยังต้องการชื่อผู้แต่งทุกคนซึ่งจะสร้างจาก template ที่สอง โดยในผู้แต่งแต่ละคนประกอบไปด้วยนามสกุลและชื่อ จากตัวอย่างจะเห็นได้ว่าไม่มีการใช้งานตัวแปรใดๆ เลย, จาก template แรก การอ้างอิงถึงปีที่พิมพ์และชื่อหนังสือจะใช้เพียง @year และ title เท่านั้น เนื่องจากภายใน template นั้นกำลังอยู่ใน context ของ element book ซึ่งกำลังถูก match อยู่ ดังนั้น title จะหมายถึง book/title นั่นเอง และเช่นกันภายใน template ที่สอง last และ first จะหมายถึง author/last และ author/first ตามลำดับ โดย author ในที่นี้จะอ้างอิงมาจาก book ใน template แรก context จะเปลี่ยนไปเมื่อมีการทำงานข้าม template หรือเมื่อใช้คำสั่ง xsl:for-each, จะเห็นได้ว่าปัญหาจะเกิดขึ้นเมื่อมีการเปลี่ยน context ไปแล้วแต่ต้องการใช้งานข้อมูลที่ถูกอ้างอิงโดย context เดิม เช่น หากใน template ที่สองมีความจำเป็นจะต้องใช้ชื่อของหนังสือ ในที่นี้จะไม่

สามารถใช้งาน title ได้แล้วเนื่องจากในที่นี้จะหมายความถึง author/title แทน, อย่างไรก็ตาม XSLT นั้นมี คุณสมบัติของตัวแปรเช่นกัน จากตัวอย่าง

```
<xsl:template match="book">
  <xsl:variable name="printed_year">
    <xsl:value-of select="@year"/>
  </xsl:variable>
  <xsl:variable name="book_title">
    <xsl:value-of select="title"/>
  </xsl:variable>
  <book year="{$printed_year}">
    <title>
      <xsl:value-of select="$book_title"/>
    </title>
  </book>
</xsl:template>
```

จากตัวอย่างข้างต้นเป็นการกำหนดค่าของ attribute year ให้กับตัวแปร printed_year และค่าของ title ให้กับตัวแปร book_title หลังจากนั้นจึงนำตัวแปรทั้งสองมาใช้ในการสร้างผลลัพธ์, ในที่นี้จะเป็นการกำหนดตัวแปรเพื่อใช้ใน template เดียว ส่วนการส่งค่าของตัวแปรข้าม template ต้องใช้กลไกของการ call template ดังตัวอย่าง

```
<xsl:template match="book">
  <book>
    <xsl:call-template name="PRINT_TITLE">
      <xsl:with-param name="book_title">
        <xsl:value-of select="title"/>
      </xsl:with-param>
    </xsl:call-template>
  </book>
</xsl:template>


<xsl:template name="PRINT_TILTE">
  <xsl:param name="book_title">
  <title>
    <xsl:value-of select="$book_title"/>
  </title>
</xsl:template>
```

ในตัวอย่างนี้เป็นการส่งค่าของตัวแปรข้าม template, template แรกจะ match กับหนังสือทุกเล่มและส่งชื่อหนังสือในรูปของตัวแปรให้กับ template ที่สองเพื่อพิมพ์ค่าให้ ในที่นี้ template ที่สองจะแตกต่างจาก template ธรรมดา คือ template นี้จะต้องถูกระบุชื่อไว้ การเรียกใช้ก็จะใช้คำสั่ง xsl:call-template แทนไม่ใช่ xsl:apply-templates และผ่านค่าของตัวแปรด้วยคำสั่ง xsl:with-param, การใช้งาน template ในลักษณะนี้จะทำงานเหมือนกับฟังก์ชันของ XQuery นั่นเอง

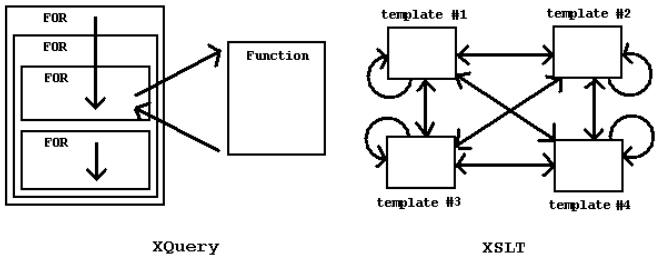
สรุป -- ตัวแปรนับเป็นคุณสมบัติหนึ่งซึ่งจำเป็นต่อการเรียกค้นข้อมูล ซึ่งจะทำให้เกิดการอ้างอิงระหว่างข้อมูลในส่วนต่างๆ ได้ แม้ว่าภาษาสืบค้นทั้งสองแบบจะสามารถใช้งานตัวแปรได้ แต่การใช้งาน XQuery นั้นอยู่บนพื้นฐานของตัวแปร การใช้งานจึงสะดวกและมีความยืดหยุ่นมาก ในขณะที่งานแปลงเอกสารของ XSLT นั้นสะดวกต่อการใช้งาน context มากกว่า ตัวแปรจึงเป็นเพียงคุณสมบัติเสริมซึ่งค่อนข้างยุ่งยากในการใช้งาน และมีข้อจำกัดมากกว่า

4.3 การเรียกค้นข้อมูลที่เป็นเอกสาร

ฐานข้อมูลที่อยู่ในรูปของ XML นั้นแม้จะมีความซับซ้อนได้มากกว่าฐานข้อมูลแบบสัมพันธ์ที่มีเพียง 2 มิติ โครงสร้างของข้อมูลจะสามารถซ้อนกันได้หลายชั้น แต่ละชั้นอาจประกอบด้วยโครงสร้างย่อยอีกมากมาย แต่อย่างไรก็ตามข้อมูลยังคงมีรูปแบบที่แน่นอนตายตัวและจำกัด เช่นจาก ตัวอย่างของข้อมูลหนังสือดังนี้

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  ...
</bib>
```

เนื่องจากโครงสร้างของข้อมูลแบบนี้ ค่อนข้างแน่นอนและจำกัด จึงเหมาะกับโครงสร้างการทำงานของ XQuery, อย่างไรก็ตาม ข้อมูล XML ที่อยู่ในรูปของเอกสาร ความสัมพันธ์ของแต่ละ element ภายในเอกสารจะมีความซับซ้อนมากกว่า, element ต่างๆ สามารถประกอบกันได้อย่างอิสระ ไม่มีรูปแบบที่ตายตัว และสามารถวนซ้ำได้ไม่จำกัดเท่าที่ต้องการ เช่นไฟล์เอกสาร HTML เป็นต้น ทำให้ความสัมพันธ์ระหว่างส่วนต่างๆ ในการเรียกค้นข้อมูลมีความซับซ้อนตามไปด้วย ดังรูปที่ 2



รูปที่ 2. แสดงความสัมพันธ์ในการทำงานของ XQuery และ XSLT

จากรูป การทำงานของ XQuery นั้นจะใช้ประโยค FOR เป็นหลักซึ่งภายในก็อาจจะประกอบไปด้วยประโยค FOR อื่นๆ ได้อีก ยิ่งข้อมูลที่ต้องการเรียกดูมีโครงสร้างที่ลึกเพียงใด ก็จำเป็นต้องมี FOR หลายชั้น

ตามไปด้วย ความสัมพันธ์ของประโยค FOR แต่ละอันจะอยู่ในทิศทางเดียวคือ จากข้างนอกลึกเข้าไปข้างในเรื่อยๆ แม้ว่า XQuery จะสามารถเรียกใช้ฟังก์ชันอื่นๆ ได้ แต่ก็ไม่ทำให้ทิศทางในการประมวลผลเปลี่ยนแปลงไป แต่ใน XSLT นั้น แต่ละ template จะสามารถเรียกใช้ template อื่นๆ ได้ตามอิสระ ความสัมพันธ์ของข้อมูลในแต่ละส่วนจะเป็นไปได้อย่างหลายหลาก ไม่มีรูปแบบตายตัวและข้อจำกัดใดๆ ซึ่งเหมาะกับการใช้งานกับข้อมูลในรูปแบบเอกสารที่มีโครงสร้างยืดหยุ่น

ตัวอย่างข้อมูลที่อยู่ในรูปของเอกสารที่ไม่มีรูปแบบตายตัวหรือข้อจำกัด เช่น ข้อมูลบทความที่มี DTD ดังนี้

```
<!ELEMENT section (title,
                   (paragraph  |  figure  |  section)*
)>
<!ELEMENT title #PCDATA>
<!ELEMENT paragraph #PCDATA>
<!ELEMENT figure EMPTY>
<!ATTLIST figure source CDATA>
```

จาก DTD ข้างต้น สามารถอธิบายได้ดังนี้, ภายใน section จะประกอบไปด้วย ชื่อ (title) ตามด้วย ย่อหน้า (paragraph), รูปภาพ (figure) และ section ย่อย ซึ่งสามารถมีจำนวนได้ไม่จำกัด และสามารถเรียงลำดับกันได้อย่างอิสระ ดังนั้นในที่นี้ทั้งจำนวนลำดับชั้นของ section รวมถึงการเรียงตัวของแต่ละ element จะไม่สามารถกำหนดได้ล่วงหน้า

ถ้าต้องการเรียกดูข้อมูล title ของทุกๆ section ที่มี จะสามารถเรียกค้นข้อมูลด้วย XQuery ได้ดังนี้

```
<results>
  {
    FOR $t IN document("book.xml")//title
    RETURN $t
  }
</results>
```

ในที่นี้จะเป็นการเรียกดูข้อมูลโดยจะยุบโครงสร้างของ section ออกทั้งหมดให้เหลือแต่เฉพาะข้อมูล title ที่ต้องการ ทำให้โครงสร้างของผลลัพธ์ที่ต้องการสามารถถูกกำหนดได้ด้วยประโยค FOR ของ XQuery, อย่างไรก็ตาม หากเปลี่ยนการเรียกดูเป็นชื่อ title ของทุกๆ section โดยที่ยังคงโครงสร้างของแต่ละ section ไว้ด้วย, ในกรณีนี้ประโยค FOR จะไม่สามารถทำงานตามที่ต้องการได้ การเรียกดูข้อมูลในลักษณะนี้จะ เหมาะสมกับโครงสร้าง template ของ XSLT มากกว่าดังตัวอย่าง

```
<xsl:template match="section">
  <section>
    <title>
      <xsl:value-of select="title"/>
    </title>
    <xsl:apply-templates select="section"/>
  </section>
</xsl:template>
```

ในที่นี้ template ที่ match กับ element section จะสร้างคำตอบ title ที่ต้องการจากนั้นจึงส่งการทำงานไปให้ยัง template ถัดไป ซึ่งก็คือ template เดิม โดยจะวนทำงานไปเรื่อยๆ จนกว่าจะหมดโครงสร้างของ section ก็จะได้คำตอบที่ยังคงโครงสร้างของ section ไว้เหมือนในเอกสารต้นฉบับทุกประการ, จะเห็นได้ว่าประโยค FOR ของ XQuery นั้นไม่สามารถทำงานในลักษณะนี้ได้ อย่างไรก็ตามเนื่องจาก XQuery สามารถใช้งานฟังก์ชันได้ และการเรียกดูในตัวอย่างนี้ก็ไม่ซับซ้อนจนเกินไป ดังนั้นจึงสามารถเรียกดูข้อมูลนี้ด้วย XQuery ได้ดังตัวอย่าง

```
FUNCTION print_section(ELEMENT $s) RETURN ELEMENT
{
  <section>
    { $s/title }
    {
      FOR $ss IN $s/section
      RETURN print_section($ss)
    }
  </section>
}

<result>
  {
    FOR $s IN document("book.xml")/section
    RETURN print_section($s)
  }
</result>
```

สรุป -- โครงสร้างข้อมูลของฐานข้อมูลจะมีรูปแบบเป็นลำดับชั้นที่แน่นอนและจำกัด เหมาะกับการประมวลผลโดยใช้ FOR ซ้อนกันหลายๆ ชั้นของ XQuery แต่ XSLT นั้นถูกออกแบบมาเพื่อจัดการกับข้อมูลแบบเอกสารซึ่งข้อมูลในแต่ละส่วนจะไม่มีรูปแบบที่ตายตัว การประมวลผลอาจต้องอาศัยการวนกลับไปกลับมา โดยขึ้นอยู่กับลักษณะของข้อมูลที่พบ โดยอาจไม่สามารถคาดการณ์ไว้ล่วงหน้าได้ อย่างไรก็ตามหากข้อมูลมีความซับซ้อนไม่มากเกินไป XQuery ก็สามารถทำได้ด้วยการใช้ความสามารถของฟังก์ชัน โดยเปรียบแต่ละฟังก์ชันเป็น template แต่การเรียกใช้งานก็จะมีความเฉพาะเจาะจงมากกว่า

4. 4 การสร้างส่วนประกอบของผลลัพธ์

ข้อมูลภายในเอกสาร XML จะกระจายไปตามโครงสร้างต่างๆ ของเอกสาร ซึ่งประกอบไปด้วยส่วนประกอบย่อยหลายๆ ส่วนได้แก่ element, attribute, namespace, comment และ processing instruction, การสืบค้นข้อมูลภายในเอกสาร XML ในบางครั้งนอกจากต้องการดึงข้อมูลที่สนใจออกมาแล้วยังต้องมีการจัดโครงสร้างของข้อมูลเหล่านั้นขึ้นใหม่ ดังนั้นภาษาสืบค้นสำหรับ XML จึงต้องสามารถสร้างส่วนประกอบเหล่านี้ได้อย่างอิสระ

4.4.1 การสร้าง element

XQuery จะสร้าง element ได้ 2 วิธี ได้แก่ การใช้ element constructor คือการสร้าง tag เปิดและ tag ปิดเอง และเนื้อหาภายใน element ก็จะประกอบไปด้วยประโยคคำสั่งอื่นๆ เช่น

```
<bib>
  {
    FOR $b IN //book
    RETURN
      <book>
        <title>{ $b/title/text() }</title>
      </book>
  }
</bib>
```

จากตัวอย่างข้างต้น เป็นการเรียกดูชื่อของหนังสือทุกๆ เล่ม ในที่นี้จะใช้ element constructor เพื่อสร้าง element ที่ชื่อ bib, book และ title, วิธีการใช้ element constructor ของ XQuery นี้จะทำให้ผู้ใช้สามารถสร้าง element ได้อย่างอิสระ แต่ข้อเสียของมันก็คือ ผู้ใช้จำเป็นต้องสร้างเนื้อหาภายในของ element นั้นๆ ขึ้นมาเองด้วย จึงทำให้เกิดการสร้าง element ด้วย XQuery อีกวิธีหนึ่งคือ การอ้างอิงมาจาก element ในเอกสาร ต้นฉบับ เช่น

```
<bib>
  {
    FOR $b IN //book
    RETURN
      <book>
        { $b/title }
        { $b/price }
      </book>
  }
</bib>
```

จากตัวอย่างข้างต้น เป็นการสร้าง element title และ price โดยดึงมาจากข้อมูลในเอกสารต้นฉบับโดยตรง วิธีนี้จะทำให้เนื้อหาต่างๆ ภายใน element ที่ถูกดึงมานี้เหมือนกับต้นฉบับทุกประการ เช่น attribute, namespace, sub element เป็นต้น วิธีการนี้จะทำให้ผู้ใช้สามารถสร้าง element ที่มีความซับซ้อนของโครงสร้างในผลลัพธ์อย่างง่ายดาย

การสร้าง element ของ XSLT นั้นจะทำได้คล้ายวิธีแรกของ XQuery เท่านั้นคือไม่สามารถอ้างอิงแล้วดึง element รวมทั้งโครงสร้างภายในของมันทั้งหมดมาได้โดยตรง ดังตัวอย่าง

ตัวอย่างที่ 1

```
<xsl:template match="book">
  <book>
    <title>
      <xsl:value-of select="title"/>
    </title>
  </book>
</xsl:template>
```

ตัวอย่างที่ 2

```
<xsl:template match="book">
  <xsl:element name="book">
    <xsl:element name="title">
      <xsl:value-of select="title"/>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

ตัวอย่างแรกจะเป็นการสร้าง element book และ title ตามปกติซึ่งจะเป็นการระบุอย่างตายตัว ตัวอย่างที่สองจะเป็นการสร้าง element โดยใช้คำสั่งของ xsl คือ xsl:element ซึ่งจะทำให้การกำหนดชื่อของ element มีความยืดหยุ่นมากกว่า

ทั้ง XQuery และ XSLT ต่างก็มีความสามารถในการสร้าง element ที่มีความยืดหยุ่นสูง อย่างไรก็ตาม XQuery นั้นจะได้เปรียบในแง่ที่ใช้ไวยากรณ์ที่กระชับมากกว่า นอกจากนี้ยังสามารถดึง element รวมทั้งเนื้อหาทั้งหมดของมันมาได้โดยตรง ในขณะที่ XSLT จะต้องสร้างเนื้อหาทุกอย่างขึ้นมาเองทั้งหมด ซึ่งทำให้การใช้งานยุ่งยากซับซ้อนจนเกินไป

4.4.2 การสร้าง attribute

XQuery จะสามารถสร้าง attribute ได้ 2 วิธีเช่นกันดังตัวอย่าง

ตัวอย่างที่ 1

```
<bib>
  {
    FOR $b IN //book
    RETURN
      <book year={ $b/@year }>
        <title>{ $b/title/text() }</title>
      </book>
  }
</bib>
```

ตัวอย่างที่ 2

```
<bib>
  {
    FOR $b IN //book
    RETURN
      <book>
        { $b/@year }
        <title>{ $b/title/text() }</title>
      </book>
  }
</bib>
```

ทั้งสองตัวอย่างจะแสดงให้เห็นการสร้าง attribute year ภายใน element book, จากตัวอย่างแรกจะเป็นการสร้าง attribute เข้าไปในส่วน element constructor และวิธีที่สองคือสร้างในส่วนเนื้อหาของ element ด้วยการดึงค่า attribute จากเอกสารต้นฉบับมาโดยตรง

การสร้าง attribute ด้วย XSLT สามารถทำได้ 2 วิธี ดังตัวอย่าง

ตัวอย่างที่ 1

```
<xsl:template match="book">
  <book year="{@year}">
    <title>
      <xsl:value-of select="title"/>
    </title>
  </book>
</xsl:template>
```

ตัวอย่างที่ 2

```
<xsl:template match="book">
  <book>
    <xsl:attribute name="year">
      <xsl:value-of select="@year"/>
    </xsl:attribute>
    <title>
      <xsl:value-of select="title"/>
    </title>
  </book>
</xsl:template>
```

ตัวอย่างแรกจะเป็นการสร้าง attribute year ลงใน element book โดยตรงซึ่งเหมือนกับ XQuery และตัวอย่างที่สองคือ ใช้คำสั่ง xsl:attribute ซึ่งสามารถระบุทั้งชื่อของ attribute และเนื้อหาได้ตามที่ต้องการ

4.4.3 การสร้าง namespace

ทั้ง XQuery และ XSLT ต่างก็สามารถระบุ namespace เพื่อเป็นเงื่อนไขในการเรียกค้นข้อมูล และสร้าง namespace ของผลลัพธ์ได้ตามที่ต้องการ ตัวอย่างต่อไปนี้จะเป็นการค้นหาข้อมูลหนังสือที่อยู่ใน namespace ของ http://www.bn.com/book เพื่อเปลี่ยนให้เป็นข้อมูลที่อยู่ใน namespace ของ http://www.amazon.com/book แทน

การเรียกค้นข้อมูลโดยใช้ XQuery

```
NAMESPACE bn = "http://www.bn.com/book"
NAMESPACE amazon = "http://www.amazon.com/book"

FOR $b IN //bn:book
RETURN
  <amazon:book>
    { $b/@* }
    { $b/* }
  </amazon:book>
```

การเรียกค้นข้อมูลโดยใช้ XSLT

```
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:bn="http://www.bn.com/book"
  xmlns:amazon="http://www.amazon.com/book">

    <xsl:template match="bn:book">
      <amazon:book>
        <xsl:apply-templates select="*"/>
      </amazon:book>
    </xsl:template>

  ...
</xsl:stylesheet>
```

จะเห็นได้ว่าทั้ง XQuery และ XSLT ก็ต่างทำงานร่วมกับ namespace ได้ดี แต่ก็ยังมีข้อจำกัดบางประการอยู่ทำให้ไม่สามารถสร้างและแก้ไข namespcae ได้อิสระตามใจผู้เรียกค้นข้อมูลได้อย่างสมบูรณ์นัก เช่น ค่าของ namespace ใดๆ ต้องถูกระบุไว้ล่วงหน้าโดยผู้เรียกค้น ยังไม่สามารถระบุค่าของ namespace ที่ถูกอ้างอิงมาจากที่อื่นได้ เป็นต้น

4.4.4 การสร้าง comment และ processing instruction

comment และ processing instruction เป็นส่วนประกอบที่ไม่ใช่ข้อมูลหลักที่กำลังสนใจ อย่างไรก็ตาม ทั้งสองส่วนเป็นสิ่งที่จะทำให้คนอื่นๆ หรือโปรแกรมอื่นๆ สามารถเข้าใจข้อมูลภายในเอกสารนั้นได้ดีมากยิ่งขึ้น จึงถือว่าเป็นส่วนประกอบสำคัญที่จะขาดไม่ได้ในการใช้งานเอกสาร XML, XQuery จะสร้าง comment และ procession instruction ด้วยฟังก์ชัน comment() และ pi() ตามลำดับ ส่วน XSLT จะใช้คำสั่ง xsl:comment และ xsl:processing-instruction เช่นเดียวกับการสร้าง element และ attribute

ทั้ง XQuery และ XSLT สามารถสร้าง comment และ processing instruction ได้ อย่างไรก็ตามการใช้ฟังก์ชันของ XQuery นั้นยังมีข้อจำกัดอยู่บ้าง เช่น ไม่สามารถสร้างเนื้อหาของ comment และ processing instruction จากโครงสร้างที่มีความซับซ้อนได้เทียบเท่ากับการใช้งาน element ในการสร้างเนื้อหาของ XSLT

สรุป -- ทั้ง XQuery และ XSLT ต่างก็มีความสามารถในการสร้าง element, attribute, namespace, comment, processing instruction ได้ค่อนข้างยืดหยุ่น และมีความสามารถในการทำงานที่ใกล้เคียงกัน อย่างไรก็ตาม ฟังก์ชันการทำงานของ XQuery จะกระชับและมีความสามารถมากกว่า ทำให้ง่ายต่อการใช้งาน ต่างจากคำสั่งของ XSLT ที่จะทำงานในระดับพื้นฐานซึ่งทำให้ต้องใช้ความพยายามในการสืบค้นมากกว่า อย่างไรก็ตามสำหรับการสร้าง comment และ processing instruction ซึ่ง XQuery นั้นจะใช้ฟังก์ชันเพียงอย่างเดียว จึงทำให้ไม่สามารถสร้างเนื้อหา จากโครงสร้างของข้อมูลที่มีความซับซ้อนได้ดีเทียบเท่ากับ XSLT

4.5 การใช้งานร่วมกับเอกสาร XML

เนื่องจากความสามารถในการประยุกต์ใช้งานที่หลากหลายของ XML เกินกว่าที่มาตรฐานดั้งเดิมของ XML จะรองรับได้ทั้งหมด จึงเกิดมาตรฐานต่างๆ ขึ้นมาอีกมากมายเพื่อเสริมความสามารถของ XML เช่น Namespace, XSLT, XML Schema [12, 13] เป็นต้น มาตรฐานเหล่านี้นั้นนอกจากจะสามารถใช้งานกับ XML ได้อย่างสมบูรณ์แล้ว ยังใช้ XML เป็นเครื่องมือในการอธิบายด้วย เช่น XSLT และ XML Schema เป็นต้น ซึ่งการใช้งานสามารถเขียนอยู่ในรูปของเอกสาร XML นั่นเอง เหตุผลหนึ่งก็เพื่อความเป็นอันหนึ่งอันเดียวกันอย่างสมบูรณ์ อย่างไรก็ตาม รูปแบบของ XML นั้นค่อนข้างไม่สะดวกต่อการใช้งาน ซึ่ง XQuery นั้นต้องการเน้นคุณสมบัติในแง่ของความง่ายต่อการใช้งาน จึงทำให้ไวยากรณ์สำหรับ XQuery นั้นเป็นแบบที่สามารถเข้าใจได้ง่าย คือ ไม่ได้อยู่ในรูปแบบของเอกสาร XML นั่นเอง

จุดเด่นซึ่งคือความง่ายของ XQuery ซึ่งได้มาด้วยการไม่ใช้ XML ในการอธิบายนั้นต้องแลกกับข้อด้อยที่สำคัญประการหนึ่งก็คือ การใช้งาน XQuery ร่วมกับเอกสาร XML โดยตรงนั่นเอง เช่น ในกรณีที่ต้องการแทรกผลลัพธ์จากการสืบค้นข้อมูลด้วย XQuery ลงในเอกสาร XML ซึ่งสามารถแสดงให้เห็นได้จากตัวอย่าง

```
<?XML version="1.0"?>
<bib>
  <xql:query>
    <![CDATA[
      FOR $b IN document("bib.xml")//book
      RETURN
        <book year={ $b/@year }>
          { $b/title }
          { $b/price }
        </book>
    ]]>
  </xql:query>
</bib>
```

รายละเอียดของ XQuery ในปัจจุบันยังไม่ได้กล่าวถึงวิธีการในการรวม XQuery เข้ากับเอกสาร XML, ตัวอย่างข้างต้นเป็นหนึ่งในหลายๆ วิธีที่สามารถใช้งานได้ โดยเอาส่วนของการเรียกค้นข้อมูลทั้งหมดของ XQuery ไปไว้ภายใต้ element เฉพาะอันหนึ่งซึ่ง XQuery processor สามารถเข้าใจได้ (ในที่นี้กำหนดให้เป็น element ที่ชื่อ query ภายใต้ prefix ของ namespace เป็น xql) แต่ประโยคคำสั่งของ XQuery นั้นจะประกอบไปด้วยอักขระที่ไม่สามารถใช้งานทั่วไปได้ ส่วนเรียกค้นข้อมูลทั้งหมดจึงต้องอยู่ภายใต้ markup ของ <![CDATA[และ]]> อีกทีหนึ่ง, เมื่อต้องการอ่านข้อมูลจากเอกสารนี้ XQuery processor จะถูกเรียกขึ้นมาทำงานก่อนเพื่อเรียกค้นข้อมูลเหล่านั้นออกมาจริงๆ จากนั้นจึงค่อยส่งการทำงานไปให้โปรแกรมในส่วนอื่นๆ ต่อไป

XSLT นั้นจะไม่มีปัญหาในการใช้งานร่วมกับเอกสาร XML หรือมาตรฐานอื่นๆ เนื่องจากตัว XSLT ก็เป็นเอกสาร XML นั่นเอง

สรุป -- ไวยากรณ์ของ XQuery ไม่ได้อยู่ในรูปแบบ XML จึงทำให้มีข้อได้เปรียบที่สามารถทำความเข้าใจได้ง่าย เขียนได้สะดวก แต่ด้วยความที่ XQuery เองไม่ได้เป็นเอกสาร XML จึงทำให้การใช้งานร่วมกันระหว่าง XQuery และเอกสาร XML หรือมาตรฐาน XML อื่นๆ มีความไม่สะดวกอยู่บ้าง แตกต่างจากไวยากรณ์ของ XSLT ซึ่งเป็นเอกสาร XML และสามารถเข้ากันได้กับเอกสาร XML และมาตรฐานอื่นๆ ได้อย่างสอดคล้อง

5. บทสรุป

XQuery เป็นผลมาจากการพัฒนาภาษาสืบค้นสำหรับ XML มาอย่างต่อเนื่อง จึงทำให้ XQuery นั้นนับได้ว่าเป็นภาษาสืบค้นสำหรับ XML ที่มีคุณสมบัติและความสามารถที่ค่อนข้างเพียบพร้อม อย่างไรก็ตาม XQuery นั้นถูกพัฒนามาตามแนวทางของฐานข้อมูลซึ่งมีภาษา SQL เป็นพื้นฐาน คุณสมบัติหลายๆ อย่างที่สืบทอดมาจะอ้างอิงกับรูปแบบการทำงานบนฐานข้อมูล ซึ่งบางครั้งไม่สามารถทำงานกับข้อมูลที่อยู่ใน รูปแบบเอกสารที่มีโครงสร้างยืดหยุ่นมากได้ดีเท่าที่ควร นอกจากนั้น XQuery ยังใช้ไวยากรณ์ของภาษาที่ใกล้เคียงกับ SQL เพื่อต้องการให้สามารถใช้งานและเข้าใจได้ง่าย ความกะทัดรัดของประโยคคำสั่งของ XQuery นี้บางครั้งทำให้ไม่สามารถทำงานบางอย่างที่ซับซ้อนเกี่ยวกับ ข้อมูลเอกสารได้เพียงพอ และจุดอ่อนประการสำคัญอีกข้อหนึ่งของ XQuery ก็คือการใช้รูปแบบที่ไม่ใช่เอกสาร XML นั่นเอง ทำให้ XQuery ไม่สามารถใช้งานร่วมกับเอกสาร XML และมาตรฐานอื่นๆ ของ XML ที่มีอยู่แล้วได้อย่างสมบูรณ์

XSLT เป็นมาตรฐานในการแปลงข้อมูลของ XML ซึ่งถูกออกแบบมาเพื่อจัดการกับความซับซ้อนของข้อมูลแบบเอกสารโดยเฉพาะ และยังประกอบด้วยคำสั่ง และการทำงานที่สามารถเป็นประโยชน์ในการสืบค้นข้อมูลได้ อย่างไรก็ตามเนื่องจาก XSLT นั้นไม่ได้ถูกออกแบบมาเพื่อการสืบค้นข้อมูลโดยตรงจึงยังขาดคุณสมบัติที่จำเป็นบางอย่างในการสืบค้นข้อมูลไป นอกจากนั้นรูปแบบการทำงานของ XSLT จะเข้าใจและใช้งานได้ยาก อย่างไรก็ตามจุดเด่นประการสำคัญของ XSLT ก็คือการอ้างอิงอยู่บนมาตรฐานของ XML นั่นเอง ทำให้ XSLT สามารถใช้งานร่วมกับเอกสาร XML และมาตรฐานอื่นๆ ที่เกี่ยวข้องได้อย่างสมบูรณ์

ทั้ง XQuery และ XSLT แม้จะถูกออกแบบมาเพื่อวัตถุประสงค์ในการใช้งานที่แตกต่างกัน แต่กลับมีแนวคิดในการทำงานและความสามารถในการสืบค้นข้อมูลที่คล้ายคลึงกัน อย่างไรก็ตามเนื่องจากการมองข้อ

มูลในมุมที่แตกต่างกัน จึงทำให้ภาษาทั้ง 2 แบบนี้มีจุดเด่นและจุดด้อยในการสืบค้นข้อมูลที่แตกต่างกัน ซึ่งผู้เขียนเชื่อว่าหากสามารถประยุกต์รวมจุดเด่นของภาษาทั้ง 2 แบบนี้เข้าด้วยกันและกำจัดข้อด้อยที่สำคัญออกไปได้ จะสามารถนำไปสู่การพัฒนาภาษาสืบค้นสำหรับ XML ที่สมบูรณ์และมีประสิทธิภาพมากยิ่งขึ้นได้

ตารางที่ 1. แสดงการเปรียบเทียบคุณสมบัติระหว่าง XQuery และ XSLT โดยอ้างอิงจากเอกสาร XML Query Requirements ของ W3C [11]

คุณสมบัติที่เปรียบเทียบ	XQuery	XSLT
สามารถเข้าใจและใช้งานได้ง่าย	X	-
มีรูปแบบการทำงานที่ใช้ XML	-	X
เป็นภาษาแบบ declarative	X	X
เป็นอิสระจากมาตรฐานอื่นๆ ที่เกี่ยวข้อง	X	X
มีการนิยามข้อผิดพลาด (exception) พื้นฐาน	-	-
รองรับการขยายความสามารถของภาษาในอนาคต	X	X
ใช้งานกับแบบจำลองข้อมูล (data model) ที่จำกัด	X	X
ทำงานร่วมกับ XML Schema ได้	X	-
รองรับการทำงานบนชุดของเอกสาร XML ได้	X	-
สามารถสืบค้นข้อมูลที่อ้างอิงกันระหว่างเอกสาร	X	-
การทำงานร่วมกับ XML Namespace	X	X
มี operation สำหรับข้อมูลพื้นฐานประเภทต่างๆ	-	-
มี Universal และ Existential Quantifiers	X	-
รองรับข้อมูลทั้งแบบโครงสร้างและเรียงลำดับ	X	X
มีฟังก์ชันในการสรุปผลข้อมูล (aggregation)	X	-
สามารถเรียงลำดับข้อมูลจากการสืบค้นได้	X	X
สามารถเปลี่ยนหรือสร้างโครงสร้างข้อมูลได้	X	X
ทำงานร่วมกับมาตรฐาน XML อื่นๆที่มีอยู่แล้วได้	X	X

เอกสารอ้างอิง

[1] A. Deutsch, M. Fernadez, D. Florescu, A. Levy, and D. Suciu, A Query Language for XML. Available http://www.research.att.com/~mff/files/final.html

[2] A. Sahuguet, XSLT as a query language for XML, April 2000. Available http://db.cis.upenn.edu/XSLT/

[3] E. Lenz, XQuery: Reinventing the Wheel?, February 2001. Available http://www.xmlportfolio.com/xquery.html

[4] International Organization for Standardization (ISO), Information Technology-Database Language SQL. Standard No. ISO/IEC 9075:1999.

[5] J. Robie, XQL (XML Query Language), August 1999. Available http://metalab.unc.edu/xql/xql-proposal.html

[6] R. Cattell, The Object Database Standard: ODMG-93, Release 1.2. Morgan Kaufmann Publisher, San Francisco, 1996.

[7] World Wide Web Consortium, Extensible Markup Language (XML) Version 1.0 (Second Edittion). W3C Recommendation, October 6, 2000. Available http://www.w3.org/TR/2000/REC-xml-20001006

[8] World Wide Web Consortium, HTML 4.01 Specification. W3C Recommendation, December 24, 1999. Available http://www.w3.org/TR/html4/

[9] World Wide Web Consortium, Namespaces in XML. W3C Recommendation, January 14, 1999. Available http://www.w3.org/TR/REC-xml-names

[10] World Wide Web Consortium, XML Path Language (XPath) Version 1.0. W3C Recommendation, November 16, 1999. Available http://www.w3.org/TR/xpath.html

[11] World Wide Web Consortium, XML Query Requirements. W3C Working Draft, February 15, 2001. Available http://www.w3.org/TR/xmlquery-req/

[12] World Wide Web Consortium, XML Schema Part 1: Structures. W3C Recommendation, May 2, 2001. Available http://www.w3.org/TR/xmlschema-1/

[13] World Wide Web Consortium, XML Schema Part 2: Datatypes. W3C Recommendation, May 2, 2001. Available http://www.w3.org/TR/xmlschema-2/

[14] World Wide Web Consortium, XQuery: A Query Language for XML. W3C Working Draft, June 7, 2001. Available http://www.w3.org/TR/xquery/

[15] World Wide Web Consortium, XSL Transformations (XSLT) Version 1.0. W3C Recommendation, November 16, 1999. Available http://www.w3.org/TR/xslt.html

เกี่ยวกับผู้เขียน

Somnuk Keretho, Ph.D.

Assistant Professor

Dr. Somnuk Keretho received his B.Eng. and M.Eng. in Electrical Engineering from Kasetsart University in 1981 and 1986, respectively. With the support from Carl Duisberg Gesellschaft Scholarship, he completed his M.Eng. in Computer Applications from Asian Institute of Technology in 1985. As a Fulbright scholar, he got his Ph.D. degree in Computer Science from University of Southwestern Louisiana, U.S.A. in 1992. His current research interest is related to Software Process Improvement with Capability Maturity Model, Object-Oriented Methodology and Unified Modeling Language, Software Components and Multi-tier Web-based Information Systems Development with Java, and eXtensible Markup Language (XML). At present, he is an assistant professor in Computer Engineering and serves as the chairman of Master of Science Program in Information Technology, Department of Computer Engineering, Kasetsart University.

Ekapol Jeerangsuwan

Mr. Ekapol Jeerangsuwan received his B.Eng. in Computer Engineering from Kasetsart University in 1997. He's studying M.Eng in Computer Engineering at Kasetsart University. His current research is about query language of eXtensible Markup Language (XML).