ภาษาสืบค้นสำหรับ XML, Another XML Query Language (AXQL)

เอกพล จีรังสุวรรณ และ สมนึก คีรีโต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์

ABSTRACT -- The standard of XML improves the way of collecting and using data, both in database and document formats. However, it still lacks mechanism to manage and query data with efficiency. This article proposes another query language for XML that the authors call, Another XML Query Language or AXQL. AXQL is based on previous prominent query languages, XQuery and XSLT. Both languages can work with XML very well. XQuery is easy to understand but it can't deal with complex querying well. XSLT concentrates on the way of processing data that can work with complex querying very well but also makes it hard to use too. AXQL is an attempt to combine their advantages together and get rid of major disadvantages. Hopefully, AXQL can be used to query XML data in both database and complex document forms with efficiency, simplicity, compatibility with other standards of XML and can be easily extended in the future.

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

บทคัดย่อ -- มาตรฐาน XML ทำให้การจัดเก็บและใช้งานข้อมูลที่อยู่ในรูปทั้งฐานข้อมูลและเอกสารมีประสิทธิภาพมากขึ้น อย่างไรก็ตาม XML ยังขาดกลไกในการจัดการและเรียกค้นข้อมูลที่มีประสิทธิภาพ, บทความนี้เป็นการนำเสนอภาษาสืบค้นสำหรับ XML อีกรูปแบบหนึ่งที่ผู้เขียนให้ชื่อว่า Another XML Query Language หรือ AXQL, AXQL ถูกพัฒนาขึ้นโดยมีภาษาต้นแบบ คือ XQuery และ XSLT, ภาษาทั้งสองนี้มีประสิทธิภาพในการจัดการข้อมูล XML มาก อย่างไรก็ตาม XQuery จะเน้นความง่ายในการใช้งานจึงทำให้ไม่สามารถเรียกค้นข้อมูลที่มีความซับซ้อนมากๆ ได้ ในขณะที่ XSLT ซึ่งสามารถทำงานที่มีความซับซ้อนได้ดี แต่จะมีรูปแบบที่ยากต่อการใช้งาน, AXQL เป็นความพยายามในการรวมข้อดีของทั้งสองภาษานี้เข้าด้วยกันและกำจัดข้อเสียที่สำคัญบางประการออกไป โดยหวังว่า AXQL จะสามารถทำงานร่วมกับฐานข้อมูลและเอกสารที่มีความซับซ้อนได้อย่างมีประสิทธิภาพ, ใช้งานได้ง่าย, ทำงานร่วมกับมาตรฐาน XML อื่นๆ ได้อย่างราบรื่น และสามารถขยายความสามารถเพิ่มเติมได้ในอนาคต

คำสำคัญ -- ภาษาสืบค้น, XML, AXQL, XQuery, XSLT

1. บทนำ

มาตรฐาน Extensible Markup Language (XML) [3] ทำให้เกิดรูปแบบใหม่ในการจัดเก็บ และจัดการข้อมูลทั้งที่เป็นฐานข้อมูลและเอกสารที่มีประสิทธิภาพมากยิ่งขึ้น อย่างไรก็ตามมาตรฐาน XML นั้นยังขาดกลไกในการสืบค้นข้อมูล, ภาษาสืบค้นสำหรับ XML รูปแบบต่างๆ ถูกนำเสนอออกมา ซึ่งแต่ละรูปแบบก็ล้วนมีข้อดีและข้อเสียที่แตกต่างกันไป และยังไม่มีภาษาสืบค้นรูปแบบใดได้รับการยอมรับว่าเป็นมาตรฐาน ดังนั้นการพัฒนาภาษาสืบค้นสำหรับ XML จึงยังคงเป็นไปอย่างต่อเนื่องเพื่อค้นหาภาษาสืบค้นสำหรับ XML ที่มีประสิทธิภาพและเหมาะกับการใช้งานมากที่สุด

XQuery [9] เป็นภาษาสืบค้นสำหรับ XML แบบล่าสุดที่เสนอโดยหน่วยงาน W3C ซึ่งมีหน้าที่ดูแลเกี่ยวกับมาตรฐานต่างๆ ของอินเตอร์เน็ต XQuery ถูกพัฒนามาในแนวทางของภาษาสืบค้นสำหรับฐานข้อมูลซึ่งมีภาษา Structured Query Language (SQL) [2] เป็นพื้นฐาน จึงทำให้รูปแบบไวยากรณ์และการใช้งานมีความใกล้เคียงกับ SQL ซึ่งทำให้สามารถเข้าใจและใช้งานได้ง่าย อย่างไรก็ตาม จุดด้อยที่สำคัญประการหนึ่งของ XQuery ก็คือการไม่ใช้รูปแบบ XML ในการอธิบาย ทำให้ไม่สามารถทำงานร่วมกับเอกสาร XML ได้ดีเท่าที่ควร นอกจากนี้ไวยากรณ์ของ XQuery ซึ่งเน้นที่ความง่ายนั้น บางครั้งมีรูปแบบที่ไม่เหมาะสมในการใช้งานร่วมกับข้อมูลที่อยู่ในรูปของเอกสารที่มีความซับซ้อนสูง [1]

Extensible Stylesheet Language (XSLT) [10] เป็นมาตรฐานที่ถูกพัฒนาขึ้นเพื่อใช้ในการแปลงเอกสาร XML เพื่อการแสดงผล อย่างไรก็ตาม ความสามารถในการแปลงข้อมูลของ XSLT นั้นสามารถนำมาใช้ในการสืบค้นข้อมูลได้เป็นอย่างดี, XSLT มีจุดเด่นประการสำคัญคือการใช้งานไวยากรณ์ที่เป็น XML ทำให้สามารถใช้งานร่วมกับเอกสาร XML ได้อย่างสอดคล้อง นอกจากนี้ยังรองรับในการสืบค้นข้อมูลที่มีความซับซ้อนได้เป็นอย่างดี อย่างไรก็ตามเนื่องจาก XSLT ไม่ได้ถูกพัฒนาขึ้นตั้งแต่ต้นเพื่อจุดประสงค์ในการสืบค้นข้อมูล จึงทำให้รูปแบบการใช้งานมีความซับซ้อน, ยากต่อการทำความเข้าใจและใช้งาน และยังขาดคุณสมบัติที่จำเป็นในการสืบค้นข้อมูลอีกหลายๆ ประการ [1]

จะเห็นได้ว่าทั้ง XQuery และ XSLT ต่างก็มีจุดเด่นและจุดด้อยที่สำคัญแตกต่างกันไป ดังนั้นบทความนี้จึงขอเสนอภาษาสืบค้นสำหรับ XML อีกรูปแบบหนึ่งที่เรียกว่า Another XML Query Language (AXQL) ซึ่งได้รวบรวมคุณสมบัติเด่น ของภาษาทั้งสองแบบข้างต้นเข้าด้วยกัน กำจัดจุดด้อยที่เป็นปัญหาในภาษาทั้งสองแบบออกไป, AXQL ถูกออกแบบขึ้นมาโดยมีวัตถุประสงค์หลักคือ จะต้องสามารถเรียกค้นข้อมูลทั้งที่อยู่ในรูปของฐานข้อมูล และเอกสารที่มีความซับซ้อนได้อย่างมีประสิทธิภาพ โดยยังคงสามารถทำความเข้าใจและใช้งานได้ง่าย, สามารถใช้งานร่วมกับเอกสาร XML และมาตรฐานสำหรับ XML แบบอื่นๆ ได้อย่างสอดคล้อง, มีกลไกในการแก้ไขและจัดการข้อมูล นอกจากนี้ยังต้องมีความยืดหยุ่นที่จะขยายความสามารถของภาษาเพิ่มเติมได้ในอนาคต

ในบทความนี้จะแสดงตัวอย่างในการสืบค้นข้อมูลด้วย AXQL โดยจะเรียกค้นข้อมูลจากเอกสาร XML ตัวอย่างคือ bib.xml ซึ่งมี Document Type Definition (DTD) ดังนี้

```
<!ELEMENT bib (book*)>
<!ELEMENT book (title, author+, publisher, price)>
<!ATTLIST book year CDATA>
<!ELEMENT author (last, first)>
<!ELEMENT title #PCDATA>
<!ELEMENT last  #PCDATA>
<!ELEMENT first #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), ชื่อสำนักพิมพ์ (publisher) และราคาหนังสือ (price), ภายในชื่อผู้แต่งจะประกอบไปด้วย นามสกุล (last) และชื่อ (first)

บทความนี้จะแบ่งเนื้อหาของ AXQL เป็นหัวข้อย่อยๆ ดังนี้

หัวข้อที่ 2 แนะนำ path expression
หัวข้อที่ 3 คำสั่งและการทำงานพื้นฐานของ AXQL
หัวข้อที่ 4 การใช้งานด้วยรูปแบบย่อของ AXQL
หัวข้อที่ 5 การสร้างผลลัพธ์จากการสืบค้น
หัวข้อที่ 6 คำสั่งในการตรวจสอบเงื่อนไข
หัวข้อที่ 7 การนิยามและเรียกใช้งานฟังก์ชัน
หัวข้อที่ 8 การเรียงลำดับข้อมูลจากการสืบค้น
หัวข้อที่ 9 การทดสอบชนิดของข้อมูล
หัวข้อที่ 10 การเพิ่มเติม, ลบ และแก้ไขข้อมูล
หัวข้อที่ 11 การพัฒนาระบบการสืบค้นด้วยภาษา AXQL
หัวข้อที่ 12 บทสรุป และตารางเปรียบเทียบคุณสมบัติ

2. แนะนำ Path Expression

path expression เป็นกลไกที่ใช้ในการอ้างอิงไปยังข้อมูลต่างๆ ที่อยู่ภายในเอกสาร XML ได้อย่างกะทัดรัดและมีประสิทธิภาพ, ในปัจจุบันมาตรฐานสำหรับ path expression ที่ประกาศอย่างเป็นทางการนั่นก็คือมาตรฐาน XPath [5] , XPath ถูกใช้อย่างแพร่หลายรวมทั้งใน XQuery และ XSLT, AXQL เองก็อาศัยกลไกของ XPath เช่นเดียวกัน

เอกสาร XML จะถูกมองเป็นแผนภูมิต้นไม้ (tree) ที่ประกอบขึ้นมาจาก node ประเภทต่างๆ 7 ประเภท ได้แก่

1. root node คือ node ที่เป็นจุดอ้างอิงของแต่ละเอกสาร
2. element node คือ node ที่แทน tag หรือ element
3. attribute node คือ node ที่แทน attribute ของ element
4. namespace node คือ node ที่ใช้ในการนิยาม namespace [4]
5. data node คือ node ที่แทนข้อมูลที่อยู่ภายใน element
6. comment node คือ node ที่แสดงความเห็นของผู้เขียน
7. processing instruction node คือ node ที่เป็นคำสั่งพิเศษ

การทำงานของ path expression จะแบ่งเป็น step ซึ่งก็คือ node ในแต่ละระดับ, แต่ละ step จะถูกอ้างอิงมาจาก step ก่อนหน้า โดยจะคั่นระหว่าง step ด้วยเครื่องหมาย "/" ซึ่งหมายถึง child หรือเป็น node ลูกของ node นั้นๆ หรือเครื่องหมาย "//" ซึ่งหมายถึง node ใดๆ ที่อยู่ภายใต้ node นั้นๆ โดยไม่สนใจลำดับขั้น, ตำแหน่งของ node หรือ set ของ node ที่ใช้ในการอ้างอิงขณะใดๆ จะถูกเรียกว่า context ซึ่งหมายถึงสภาวะแวดล้อมในขณะนั้นๆ, path expression ยังสามารถมีเงื่อนไขในการเรียกค้นข้อมูล (predicate) ซึ่งจะทำให้ได้ข้อมูลที่มีความเฉพาะเจาะจงมากยิ่งขึ้นด้วย, ตัวอย่างการใช้งาน path expression เช่น

document('bib.xml') หมายถึง root node ของเอกสาร bib.xml
/ หมายถึง root node ของเอกสารปัจจุบัน
/bib หมายถึง element ที่ชื่อ bib ที่เป็น root element

. หมายถึง context หรือ node ปัจจุบัน

bib หมายถึง element ที่ชื่อ bib ที่อ้างอิงกับ context ขณะนั้น

bib/book หมายถึง element book ที่อยู่ภายใต้ element bib

//book หมายถึง element book ใดๆ ที่อยู่ภายในเอกสาร

book/@year หมายถึง attribute year ของ element book

title/data() หมายถึง data node ของ element title

comment() หมายถึง comment node

book[publisher = 'Addison-Wesley' and @year > 1991] หมายถึง หนังสือทุกๆ เล่มที่พิมพ์โดยสำนักพิมพ์ Addison-Wesley หลังจากปี 1991

3. การสืบค้นข้อมูลด้วย AXQL

ในหัวข้อนี้จะเป็นการแนะนำคำสั่ง และรูปแบบการใช้งานพื้นฐานที่สำคัญบางอย่างของ AXQL, AXQL ถูกออกแบบให้มีรูปแบบในการนำเสนอที่เป็น XML คำสั่งต่างๆ ของ AXQL จะอยู่ในรูปของ element, AXQL สามารถใช้งานร่วมกับข้อมูล XML ตามปกติได้ ดังนั้นเพื่อเป็นการแบ่งแยกระหว่าง element ที่เป็นคำสั่งของ AXQL และ element ที่เป็นข้อมูล XML ชื่อของ element ที่เป็นคำสั่งของ AXQL จะมี prefix หรือ namespace เป็น "axql" ซึ่งจะอ้างอิงไปยัง URI คือ "http://www.cpe.ku.ac.th/AXQL"

3.1. คำสั่ง axql:query

เนื่องจาก AXQL นั้นเป็นเอกสาร XML จึงต้องมี root element ดังเช่นเอกสาร XML ปกติ, root element ของ AXQL นั้นถูกกำหนดให้เป็น axql:query ดังตัวอย่าง

```
<axql:query
  xmlns:axql="http://www.cpe.ku.ac.th/AXQL">
  <!-- AXQL top level element -->
</axql:query>
```

ภายใน tag เปิดของ axql:query จะมีการนิยาม namespace ของ AXQL ด้วยเสมอเพื่อบอกให้รู้ว่าเอกสารนี้เป็นการเรียกค้นข้อมูลโดยใช้ AXQL, ภายใน axql:query จะต้องประกอบไปด้วยคำสั่งของ AXQL ที่เรียกว่า top level element เท่านั้น ซึ่งได้แก่คำสั่ง axql:include, axql:function และ axql:main ซึ่งจะได้กล่าวถึงต่อไป

3.2. คำสั่ง axql:main

ในการเรียกค้นข้อมูลตามปกติ จะประกอบไปด้วย axql:query และ axql:main ซึ่งเป็น top level element ที่เป็นส่วนในการเรียกค้นหลัก ดังตัวอย่าง

```
<axql:query
  xmlns:axql="http://www.cpe.ku.ac.th/AXQL">
  <axql:main>
    <!-- AXQL instructions -->
  </axql:main>
</axql:query>
```

3.3. คำสั่ง axql:for

ภายใน axql:main จะประกอบไปด้วยคำสั่งต่างๆ ของ AXQL, คำสั่งที่ใช้เป็นพื้นฐานในการเรียกค้นข้อมูลของ AXQL คือ axql:for ซึ่งจะมี attribute select ที่มีค่าเป็น path expression ที่จะอ้างอิงไปยังเซตของ node ของข้อมูลที่กำลังสนใจ คำสั่ง axql:for จะวนรอบทำงานคำสั่งที่อยู่ภายในจนกว่าจะครบทุก node ในเซตของ node ที่อ้างอิงถึง ดังตัวอย่าง

```
<axql:query
  xmlns:axql="http://www.cpe.ku.ac.th/AXQL">
  <axql:main>
    <axql:for select="document('bib.xml')//book">
      <book>
        <title><axql:value select="title"/></title>
      </book>
    </axql:for>
  </axql:main>
</axql:query>
```

จากตัวอย่างข้างต้น ภายในคำสั่ง axql:for, path expression จะอ้างอิงไปยังเซตของ element book ทุกๆ อันที่อยู่ภายในเอกสาร bib.xml, ในแต่ละรอบการทำงานของ axql:for จะสร้างผลลัพธ์ของการเรียกค้นคือ element book โดยภายในประกอบไปด้วย element title ที่มีค่าเป็นชื่อของหนังสือแต่ละเล่ม ค่าของชื่อหนังสือนี้จะถูกสร้างขึ้นมาด้วยคำสั่งของ AXQL คือ axql:value ซึ่งจะสร้าง data node ที่มีค่าเป็นข้อมูลที่ถูกอ้างอิงโดย path expression ที่อยู่ภายใน attribute select , คำสั่ง axql:for จะมีผลทำให้ context เปลี่ยนไป จะเห็นได้จาก path expression ภายในคำสั่ง axql:value ระบุเพียง title ซึ่งในที่นี้จะหมายถึง element title ที่อ้างอิงกับ context คือ element book ที่กำลังทำงานอยู่ในรอบนั้นๆ

3.4. คำสั่ง axql:where

การใช้ path expression สามารถระบุเงื่อนไขเพิ่มเติมใน node ที่ต้องการเป็นพิเศษได้ ด้วยการใช้คุณสมบัติ predicate ที่มีอยู่แล้วภายใน path expression เอง หรืออาจใช้คำสั่ง axql:where เพื่อช่วยในการตรวจสอบเงื่อนไขของแต่ละ node, ตัวอย่างต่อไปนี้เป็นการเปรียบเทียบการตรวจสอบเงื่อนไขโดยใช้ predicate และคำสั่ง axql:where ซึ่งจะให้ผลลัพธ์ที่เหมือนกันทุกประการ

การตรวจสอบเงื่อนไขโดยใช้ predicate

```
<axql:for select="//book[@year > 1991]">
  <book>
    <title><axql:value select="title"/></title>
  </book>
</axql:for>
```

การตรวจสอบเงื่อนไขโดยใช้คำสั่ง axql:where

```
<axql:for select="//book">
  <axql:where test="@year > 1991">
    <book>
      <title><axql:value select="title"/></title>
    </book>
  </axql:where>
</axql:for>
```

จากตัวอย่างเป็นการเพิ่มเงื่อนไขภายใน element book ที่อ้างอิงมาว่าจะต้องพิมพ์หลังจากปี 1991, คำสั่ง axql:where โดยปกติจะใช้ตรวจสอบเงื่อนไขของ node หลังจากคำสั่ง axql:for โดยหาก node ที่ได้เป็นไปตามเงื่อนไขที่อยู่ใน attribute test ก็จะทำคำสั่งอื่นๆ ต่อไป

3.5. คำสั่ง axql:var

จะเห็นได้ว่า AXQL นั้นใช้แนวคิดของ context จึงทำให้การอ้างอิงไปยัง element ใดๆ ที่เกี่ยวข้องกับ context มีความสะดวกมาก อย่างไรก็ตามมีบางกรณีที่ context ไม่อาจใช้งานได้ เช่น มีการใช้คำสั่งของ AXQL บางคำสั่งที่ทำให้ context เปลี่ยนไป เช่น คำสั่ง axql:for แต่คำสั่งภายในกลับต้องการอ้างอิงถึงข้อมูลที่อยู่ภายใต้ context เดิม ในกรณีเช่นนี้จำเป็นต้องใช้ตัวแปรมาช่วยเก็บค่าของข้อมูลที่ต้องการไว้ก่อน หลังจากนั้นจึงค่อยนำมาใช้ในภายหลัง ดังตัวอย่าง

```
<axql:for select="distinct(//author)">
  <result>
    <author><axql:value select="."/></author>
    <axql:var name="my_var" select="."/>
    <axql:for select="//book[author = $my_var]">
      <title><axql:value select="title"/></title>
    </axql:for>
  </result>
</axql:for>
```

จากตัวอย่างข้างต้น เป็นการเรียกดูข้อมูลผู้แต่งแต่ละคน โดยภายในประกอบด้วยข้อมูลผู้แต่ง และชื่อหนังสือที่แต่งโดยผู้แต่งคนนั้นๆ, ใน path expression ที่ระบุไปยังชื่อผู้แต่ง จะใช้ฟังก์ชัน distinct() เพื่อจัดกลุ่มตามชื่อผู้แต่งโดยกำจัด node ที่มีค่าซ้ำซ้อนกันออกไป โดยภายในจะสร้างผลลัพธ์อยู่ภายใต้ element result ซึ่งประกอบไปด้วย element author นอกจากนั้นคำตอบยังต้องการชื่อหนังสือที่แต่งโดยผู้แต่งคนนั้นๆ อีก ในที่นี้จึงใช้คำสั่ง axql:for อีกครั้งเพื่อวนหาชื่อหนังสือที่ต้องการ แต่ชื่อของผู้แต่งใน element author นั้นจะถูกอ้างอิงไม่ได้เมื่อ context ถูกเปลี่ยนไป จึงต้องมีการเก็บค่า element author นี้ไว้ในตัวแปร $my_var ก่อนด้วยคำสั่ง axql:var

4. รูปแบบย่อของ AXQL

เนื่องจากรูปแบบการใช้งานของ AXQL จะอยู่ในรูปของเอกสาร XML, คำสั่งต่างๆ จะอยู่ในรูปของ element ซึ่งบางครั้งยากต่อการใช้งานและทำความเข้าใจ อย่างไรก็ตามรูปแบบคำสั่งบางประเภทจะถูกใช้งานบ่อยและในสภาวะการทำงานที่เจาะจงบางรูปแบบสามารถจะเขียนย่อได้ ซึ่งจะทำให้สามารถใช้งานได้สะดวกมากยิ่งขึ้น, รูปแบบคำสั่งของ AXQL ที่สามารถย่อได้ มีรูปแบบการใช้งานดังนี้

4.1. รูปแบบย่อของ axql:query และ axql:main

หากภายใน axql:query ประกอบไปด้วย top level element คือ axql:main เพียงอย่างเดียว และภายใน axql:main มีการสร้าง root element ไว้ การทำงานจะสามารถย่อได้ ดังตัวอย่างแสดงรูปแบบเต็มและรูปแบบย่อ ดังนี้

รูปแบบเต็ม

```
<axql:query
  xmlns:axql:="http://www.cpe.ku.ac.th/AXQL">
  <axql:main>
    <results>
      <!-- AXQL insturctions -->
    </results>
  </axql:main>
</axql:query>
```

รูปแบบย่อของ axql:query และ axql:main

```
<results
  xmlns:axql:="http://www.cpe.ku.ac.th/AXQL">
  <!-- AXQL insturctions -->
</results>
```

รูปแบบการใช้งานข้างต้นจะเป็นการละคำสั่ง axql:query และ axql:main ไว้ซึ่งจะทำให้การเขียนมีความกะทัดรัดมากขึ้น อย่างไรก็ตามจำเป็นต้องมีการนิยาม namespace axql ไว้เสมอเพื่อบอกให้รู้ว่าเอกสารนี้เป็นการเรียกค้นข้อมูลด้วย AXQL ไม่ใช่ข้อมูล XML ตามปกติ

4.2. รูปแบบย่อของ axql:for

หากภายใน axql:for มีการสร้าง element หลักเพียง element เดียว การทำงานจะสามารถย่อได้ ดังตัวอย่าง

รูปแบบเต็ม

```
<axql:for select="//book">
  <book>
    <!-- AXQL insturctions -->
  </book>
</axql:for>
```

รูปแบบย่อของ axql:for

```
<book axql:for="//book">
  <!-- AXQL insturctions -->
</book>
```

จากรูปแบบการใช้งานย่อของคำสั่ง axql:for, จะเปลี่ยนสถานะจาก element ไปเป็น attribute axql:for แทน แต่การทำงานยังคงเหมือนเดิม คือ เมื่อพบ node ตรงตามเงื่อนไขที่อ้างอิงโดย attribute axql:for ก็จะวน

รอบการทำงานโดยสร้าง element book และเรียกใช้คำสั่งภายในอื่นๆ เพื่อสร้างเนื้อหาของ element book

4.3. รูปแบบย่อของ axql:where

หากภายใน axql:where มีการสร้าง element หลักเพียง element เดียว การทำงานจะสามารถย่อได้ ดังตัวอย่าง

รูปแบบเต็ม

```
<axql:for select="//book">
  <axql:where test="@year > 1991">
    <book>
      <!-- AXQL insturctions -->
    </book>
  </axql:where>
</axql:for>
```

รูปแบบย่อของ axql:where

```
<axql:for select="//book">
  <book axql:where="@year > 1991">
    <!-- AXQL insturctions -->
  </book>
</axql:for>
```

รูปแบบย่อของ axql:for และ axql:where

```
<book axql:for="//book" axql:where="@year > 1991">
  <!-- AXQL insturctions -->
</book>
```

จากตัวอย่างข้างต้น เป็นการใช้รูปแบบการทำงานย่อของ axql:where นอกจากนี้ ทั้งคำสั่ง axql:for และ axql:where นั้นสามารถย่อนำมารวมกันได้ ทำให้มีความกะทัดรัดมาก สามารถเข้าใจได้ง่าย โดยที่ยังคงสามารถทำงานได้เหมือนเดิมทุกประการ

4.4. รูปแบบย่อของ axql:value

คำสั่ง axql:value นั้นใช้เพื่อสร้าง data node เพื่อใช้เก็บข้อมูลภายใน element ซึ่งเป็นรูปแบบการทำงานที่พบค่อนข้างบ่อย, หากภายใน element ที่มีการสร้าง data node นั้นมีส่วนประกอบคือคำสั่ง axql:value เพียงหนึ่งเดียว จะสามารถเขียนในรูปแบบย่อได้ดังนี้

รูปแบบเต็ม

```
<book axql:for="//book">
  <title><axql:value select="title"/></title>
  <price><axql:value select="price"/></price>
</book>
```

รูปแบบย่อของ axql:value

```
<book axql:for="//book">
  <title axql:value="title"/>
  <price axql:value="price"/>
</book>
```

5. การสร้างผลลัพธ์จากการสืบค้นด้วย AXQL

การสืบค้นข้อมูล XML นอกจากเพื่อเรียกค้นข้อมูลที่สนใจแล้ว ยังต้องการข้อมูลเหล่านั้นในรูปแบบการนำเสนอที่แตกต่างกัน, ในหัวข้อนี้จะพูดถึงวิธีต่างๆ ที่ AXQL ใช้ในการสร้างผลลัพธ์ นั่นคือการสร้างโครงสร้างและส่วนประกอบต่างๆ ของข้อมูลตามต้องการ

5.1. การสร้างผลลัพธ์แบบ literal

การสร้างผลลัพธ์แบบ literal นั้นก็คือการสร้าง tag ที่ต้องการลงไปใน AXQL โดยตรง ดังที่ได้พบในตัวอย่างก่อนหน้านี้ เช่น

```
<axql:for select="//book">
  <book year="{@year}">
    <title><axql:value select="title"/></title>
  </book>
</axql:for>
```

จากตัวอย่างเป็นการสร้าง element book, attribute year และ element title แบบ literal คือระบุเข้าไปโดยตรง, ค่าของ attribute year ในที่นี้จะใช้รูปแบบพิเศษในการให้ค่าข้อมูลที่เรียกว่า attribute value template ที่ขอยืมมาจาก XSLT โดยค่าที่อยู่ภายในวงเล็บ { } คือ path expression และค่าที่ได้ออกมาจะเหมือนกับการใช้งานคำสั่ง axql:value นั่นเอง, ในที่นี้จะใช้การสร้างผลลัพธ์แบบ literal ผสมกับคำสั่งของ AXQL เพื่อสร้างข้อมูลบางส่วนที่ไม่สามารถสร้างขึ้นมาได้เองโดยตรง

5.2. การใช้คำสั่ง axql:copy

การสร้างผลลัพธ์แบบ literal นั้น ผู้เรียกค้นจะต้องเป็นผู้สร้าง element ของคำตอบเอง รวมถึงข้อมูลภายใน element นั้นๆ ด้วย ซึ่งบางครั้งคำตอบที่ต้องการอาจจะเป็นเพียงการยกทั้ง sub tree ของเอกสารต้นฉบับมาโดยตรงเลยก็ได้ การสร้าง element รวมถึงเนื้อหาภายในเองจึงเป็นเรื่องที่เกินจำเป็นและเพิ่มความยากในการเรียกค้นจนเกินไป, AXQL มีคำสั่งที่ใช้ในการทำงานนี้โดยเฉพาะคือ axql:copy ดังตัวอย่าง

การสร้าง element และเนื้อหาเอง

```
<axql:for select="//book">
  <book year="{@year}">
    <title><axql:value select="title"/></title>
    <author>
      <last>
        <axql:value select="author/last"/>
      </last>
      <first>
        <axql:value select="author/first"/>
      </first>
    </author>
    <price><axql:value select="price"/></price>
  </book>
</axql:for>
```

การใช้คำสั่ง axql:copy ดึงข้อมูลมาจากเอกสารต้นฉบับ

```
<axql:for select="//book">
  <book>
    <axql:copy select="@year"/>
    <axql:copy select="title"/>
    <axql:copy select="author"/>
    <axql:copy select="price"/>
  </book>
</axql:for>
```

คำสั่ง axql:copy นั้นไม่จำเป็นว่าจะต้องทำงานกับ element เท่านั้นแต่สามารถใช้งานร่วมกับ node ทุกๆ ประเภทภายในเอกสาร ดังจะเห็นได้จากตัวอย่างว่าสามารถใช้คำสั่ง axql:copy กับ attribute year ได้ด้วย, จะเห็นได้ว่าคำสั่ง axql:copy นั้นเพิ่มความสะดวกในการเรียกค้นข้อมูลได้อย่างมาก

5.3. การสร้าง element และ attribute ด้วยคำสั่งของ AXQL

การใช้งาน literal เพื่อสร้าง element และ attribute นั้นจะมีข้อจำกัดที่จะต้องระบุชื่อ, namespace prefix ของ element และ attribute อย่างตายตัว, การใช้คำสั่ง axql:copy แม้จะสะดวกแต่ก็ไม่เปิดโอกาสให้ผู้เรียกค้นสามารถแก้ไขรายละเอียดของข้อมูลได้ตามที่ต้องการ แต่ด้วยคำสั่ง axql:element และ axql:attribute จะทำให้การสร้าง element และ attribute มีความยืดหยุ่นมากยิ่งขึ้นดังตัวอย่าง

```
<axql:for select="//book">
  <axql:element name="book">
    <axql:attribute name="year" value="@year"/>
    <axql:element name="title">
      <axql:value select="title"/>
    </axql:element>
  </axql:element>
</axql:for>
```

จากตัวอย่างเป็นการสร้าง element book, attribute year และ element title ด้วยคำสั่ง axql:element และ axql:attribute ซึ่งแม้จะมีรูปแบบการใช้งานที่ยุ่งยากกว่า แต่ก็ให้อิสระแก่ผู้เรียกค้นในการปรับแต่งข้อมูลได้มากขึ้นเช่นกัน

5.4. การสร้าง namespace

การสร้าง namespace สามารถทำได้ 2 วิธีคือ สร้างโดยตรงภายใน element แบบ literal และสร้างโดยใช้คำสั่ง axql:namespace ดังตัวอย่าง

การสร้าง namespace แบบ literal

```
<axql:for select="//book">
  <bn:book xmlns:bn="http://www.bn.com">
    <bn:title>
      <axql:value select="title"/>
    </bn:title>
  </bn:book>
</axql:for>
```

การสร้าง namespace ด้วยคำสั่ง axql:namespace

```
<axql:for select="//book">
  <bn:book>
    <axql:namespace prefix="bn"
      uri="http://www.bn.com"/>
    <bn:title>
      <axql:value select="title"/>
    </bn:title>
  </bn:book>
</axql:for>
```

จากตัวอย่างจะเป็นการสร้างข้อมูลที่อยู่ภายใต้ namespace เป็น "bn" ซึ่งอ้างอิงกับ URI คือ "http://www.bn.com", โดยปกติการใช้งาน namespace นั้นมักจะถูกนิยามแบบ literal มากกว่า อย่างไรก็ในกรณีที่ต้องการนิยาม namespace ที่มีความยืดหยุ่นมากๆ เช่น สามารถเปลี่ยนแปลงตามสถานการณ์ได้ การใช้คำสั่ง axql:namespace จะมีความเหมาะสมมากกว่า

5.5. การสร้าง node ของข้อมูล

เป็นการสร้างข้อมูลที่แท้จริงที่ถูกเก็บอยู่ภายใน element, การสร้างข้อมูลจะทำได้ 2 วิธีเช่นกัน คือ แบบ literal แต่วิธีนี้ค่อนข้างจะเป็นวิธีที่ไม่มีประโยชน์ในการเรียกค้นข้อมูลเนื่องจากไม่สามารถปรับเปลี่ยนค่าได้ และวิธีที่ใช้คำสั่งของ AXQL ซึ่งจะเห็นได้จากหลายๆ ตัวอย่างที่ผ่านมาคือคำสั่ง axql:value, attribute ที่สำคัญของคำสั่ง axql:value นี้ นอกจาก select ซึ่งเป็น path expression ที่อ้างอิงไปยังข้อมูลที่ต้องการแล้ว ยังประกอบไปด้วย attribute type ซึ่งสามารถระบุประเภทของข้อมูลได้ โดยประเภทของข้อมูลนี้จะสัมพันธ์กับประเภทข้อมูลที่ถูกนิยามโดย XML Schema [7, 8], และ attribute format ซึ่งใช้ในการจัดรูปแบบของข้อมูลที่จะแสดงไปยังผลลัพธ์ตามที่ต้องการ ดังตัวอย่าง

```
<axql:value select="title" type="xsd:string"/>
<axql:value select="price" type="xsd:decimal"
  format="0.00"/>
```

จากตัวอย่างเป็นการสร้าง data node ที่มีข้อมูลเป็นข้อความของชื่อหนังสือ และข้อมูลราคาหนังสือที่มีทศนิยม 2 ตำแหน่ง

5.6. การสร้าง comment และ processing instruction

ทั้ง comment และ processing instruction แม้จะไม่ใช่ข้อมูลที่ผู้ใช้กำลังสนใจโดยตรง แต่ก็เป็นส่วนประกอบสำคัญที่ช่วยอธิบายให้ผู้อื่น หรือโปรแกรมอื่นๆ สามารถทำความเข้าใจและทำงานร่วมกันได้ดียิ่งขึ้น, การสร้าง comment และ processing instruction จะทำได้โดยใช้คำสั่ง axql:comment และ axql:pi ดังตัวอย่างต่อไปนี้

ตัวอย่างที่ 1

```
<axql:comment value="This is comment."/>
<axql:pi name="user-parameter" value="none">
```

ตัวอย่างที่ 2

```
<axql:comment>
  This is comment. The book's title is
  <axql:value select="title"/>.
  Printed by <axql:value select="publisher">
  in <axql:value select="@year + 543"> BC.
</axql:comment>
<axql:pi name="user-parameter">
  <axql:value select="string('none')"/>
</axql:pi>
```

จากตัวอย่างข้างต้นเป็นการสร้าง comment และ processing instruction ด้วยค่าของ path expression จาก attribute value และ ด้วยคำสั่งของ AXQL ตามลำดับ, การสร้าง comment และ processing instruction ของคำตอบ ไม่สามารถใส่ลงไปในการเรียกค้นของ AXQL ได้โดยตรงเหมือนการสร้าง element แบบ literal เนื่องจากข้อมูลเหล่านี้จะไม่ถูกสนใจโดย query processor

6. คำสั่งตรวจสอบเงื่อนไข

ในบางครั้งการสร้างผลลัพธ์จากการเรียกค้นจะขึ้นอยู่กับเงื่อนไขหลายๆ ประการ, คำสั่งตรวจสอบเงื่อนไขที่ได้พบมาก่อนหน้านี้ คือ axql:where นั้นใช้ในการตรวจสอบเงื่อนไขว่าจะทำหรือไม่ทำเท่านั้น แต่ในความเป็นจริงเงื่อนไขในการเลือกทำอาจมีมากกว่านี้ ดังจะอธิบายได้จากคำสั่งต่อไปนี้ คือ axql:if และ axql:choose มีรายละเอียดต่อไปนี้

6.1. คำสั่ง axql:if

คำสั่ง axql:if จะมี attribute test ซึ่งค่า expression ภายในจะถูกตีความออกมาเป็น boolean, หากค่าที่ได้เป็น true คำสั่งที่อยู่ภายใน element axql:then จะถูกเรียกให้ทำงาน ไม่เช่นนั้นคำสั่งภายใน element axql:else ก็จะถูกเรียกให้ทำงานแทน ดังตัวอย่าง

```
<axql:for select="//book">
  <axql:if test="price > 100">
    <axql:then>
      <expensive-book title="{title}"/>
    </axql:then>
    <axql:else>
      <cheap-book title="{title}"/>
    </axql:else>
  </axql:if>
</axql:for>
```

จากตัวอย่าง เป็นการทดสอบเงื่อนไขราคาของหนังสือแต่ละเล่ม หากหนังสือมีราคาสูงกว่า 100 ก็จะสร้าง element คำตอบชื่อ expensive-book ไม่เช่นนั้นก็สร้าง cheap-book แทน

6.2. คำสั่ง axql:choose

คำสั่ง axql:if จะมีทางเลือกในการทำงานได้เพียง 2 ทางเลือกเท่านั้น แต่ในบางสถานการณ์ เงื่อนไขและทางเลือกอาจมีมากกว่านั้น ซึ่ง AXQL จะทำงานโดยใช้คำสั่ง axql:choose โดยภายในจะประกอบด้วยหลายๆ ทางเลือกคือคำสั่ง axql:when ที่มีเงื่อนไขในแต่ละกรณีระบุอยู่ด้วย attribute test ไม่เช่นนั้นก็ต้องทำในทางเลือกสุดท้ายในคำสั่ง axql:otherwise ดังตัวอย่าง

```
<axql:for select="//book">
  <axql:choose>
    <axql:when test="price > 100">
      <expensive-book title="{title}"/>
    </axql:when>
    <axql:when test="price > 20">
      <cheap-book title="{title}"/>
    </axql:when>
    <axql:otherwise>
      <sale-book title="{title}"/>
    </axql:otherwise>
  </axql:choose>
</axql:for>
```

จากตัวอย่าง หากราคาของหนังสือสูงกว่า 100 ก็จะสร้าง element ที่ชื่อ expensive-book, ถ้าราคาต่ำว่า 100 แต่ไม่ต่ำไปกว่า 20 ก็จะสร้าง element ที่ชื่อ cheap-book, หากไม่มีเงื่อนไขใดที่ตรงเลย ก็จะทำเงื่อนไขสุดท้าย คือ สร้าง element ที่ชื่อ sale-book

7. การเรียกใช้งานฟังก์ชัน

7.1. การนิยามและเรียกใช้ฟังก์ชัน

การเรียกใช้งานคำสั่งของ AXQL บางครั้งชุดของคำสั่งหนึ่งๆ มีความจำเป็นต้องถูกเรียกใช้มากกว่าหนึ่งครั้ง การเขียนคำสั่งที่ซ้ำซ้อนกันทุกๆ ครั้งไม่ใช่วิธีการที่ดี AXQL จึงมีคุณสมบัติของฟังก์ชันเพื่อช่วยลดความซ้ำซ้อนนี้ นอกจากนั้นฟังก์ชันยังช่วยให้สามารถใช้งานและทำความเข้าใจคำสั่งในการเรียกค้นข้อมูลได้ง่ายยิ่งขึ้น การนิยามฟังก์ชันจะใช้คำสั่ง axql:function ซึ่งเป็น top level element ของคำสั่ง axql:query และเรียกใช้ฟังก์ชันด้วยคำสั่ง axql:call-function ดังตัวอย่าง

```
<axql:query
  xmlns:axql="http://www.cpe.ku.ac.th/AXQL">

  <axql:function name="create-book">
    <book year="{@year}">
      <title><axql:value select="title"/></title>
    </book>
  </axql:function>

  <axql:main>
    <results>
      <axql:for select="//book">
        <axql:call-function name="create-book"/>
      </axql:for>
```

```
    </results>
  </axql:main>

</axql:query>
```

จากตัวอย่าง เป็นการนิยามฟังก์ชันชื่อ create-book เพื่อสร้าง element book, title และ attribute year, ภายใน axql:main ซึ่งเป็นส่วนหลักของการทำงาน หลังจากคำสั่ง axql:for เพื่อค้นหาหนังสือทุกๆ เล่มแล้ว จึงส่งการทำงานไปให้ยังฟังก์ชัน create-book แทนด้วยคำสั่ง axql:call-function และระบุชื่อของฟังก์ชันที่เรียกใน attribute name

7.2. การเรียกใช้ฟังก์ชันด้วย pattern

จากตัวอย่างการใช้งานฟังก์ชันที่ผ่านมาเป็นการเรียกใช้งานฟังก์ชันโดยการเรียกจากชื่อของฟังก์ชัน แต่ในบางครั้งข้อมูลที่อยู่ในรูปของเอกสาร XML นั้นจะประกอบด้วย element ที่เป็นไปได้จำนวนมากมาย จนบางครั้งไม่สามารถคาดเดาได้ว่าในกรณีเช่นใด ควรจะต้องเรียกใช้งานฟังก์ชันใด, AXQL ได้อาศัยคุณสมบัติของการเรียกใช้งาน template ใน XSLT มาประยุกต์ใช้งานร่วมกับฟังก์ชัน ทำให้สามารถเรียกใช้ฟังก์ชันโดยพิจารณาจาก pattern ของข้อมูลที่เหมาะสม ดังตัวอย่าง

```
<axql:query
  xmlns:axql="http://www.cpe.ku.ac.th/AXQL">

  <axql:function pattern="book[price > 100]">
    <expensive-book title="{title}"/>
  </axql:function>

  <axql:function pattern="book[price &lt;= 100]">
    <cheap-book title="{title}"/>
  </axql:function>

  <axql:main>
    <results>
      <axql:for select="//book">
        <axql:call-function select="."/>
      </axql:for>
    </results>
  </axql:main>

</axql:query>
```

จากตัวอย่างข้างต้น จะเรียกใช้งานฟังก์ชันจากความสัมพันธ์ระหว่างค่าของ attribute select ใน axql:call-function และ ค่าของ attribute pattern ใน axql:function โดย axql:call-function จะ select "." ซึ่งในที่นี้คือ element book แต่ละเล่มนั่นเอง ส่งไปให้ยังฟังก์ชันต่างๆ ที่เหมาะสม หาก book นั้นมีราคาสูงกว่า 100 ก็จะตรงกับ pattern ของฟังก์ชันแรก แต่หากราคาของ book ต่ำกว่า 100 ก็จะตรงกับ pattern ของฟังก์ชันที่สอง (ภายในค่าของ attribute เครื่องหมาย < จะไม่สามารถใช้งานได้โดยตรง เนื่องจากเป็นเครื่องหมายที่แสดงถึงการเปิด markup ดังนั้นจึงต้องใช้ markup < แทน[1])

ในกรณีที่ข้อมูลสามารถ match กับ pattern ของฟังก์ชันได้มากกว่าหนึ่งฟังก์ชัน ในที่นี้ AXQL จะเลือกใช้งานเพียงฟังก์ชันเดียว โดยดูจากเงื่อนไข ดังต่อไปนี้

- กรณีฟังก์ชันที่ถูกนิยามไว้ในไฟล์อื่นๆ ที่ถูกรวมเข้ามาด้วยคำสั่ง axql:include (จะได้กล่าวถึงต่อไปในหัวข้อ 7.5) ลำดับของไฟล์ที่ถูกรวมเข้ามานี้จะมีความสำคัญแตกต่างกัน โดยไฟล์ที่ถูกรวมเข้ามาในลำดับหลังจะมีความสำคัญมากกว่า แต่ก็ไม่เท่ากับฟังก์ชันที่ถูกนิยามไว้ในเอกสารเองซึ่งจะมีระดับความสำคัญสูงสุด

- ถ้าฟังก์ชันยังคงมีระดับความสำคัญเท่ากัน AXQL จะดูจาก attribute priority ของคำสั่ง axql:function ซึ่งจะมีค่าเป็นตัวเลข ค่าที่มากกว่าจะมีระดับความสำคัญมากกว่า

- ในกรณีที่ไม่สามารถเลือกใช้งานฟังก์ชันที่เหมาะสม จากเงื่อนไขข้างต้นได้ AXQL จะแจ้งความผิดพลาดให้ผู้ใช้ทราบ

7.3. การรับพารามิเตอร์และการส่งค่าคืนของฟังก์ชัน

โดยปกติในภาษาโปรแกรมใดๆ การใช้งานฟังก์ชันจะมีการระบุประเภทของข้อมูลที่จะคืนมาจากการทำงานของฟังก์ชัน, AXQL จะสามารถระบุประเภทของข้อมูล ที่คืนมาจากการเรียกใช้งานฟังก์ชันในส่วนของการนิยามด้วย attribute type ดังตัวอย่าง

```
<axql:query
  xmlns:axql="http://www.cpe.ku.ac.th/AXQL">

  <axql:function name="find-book" type="xsd:any">
    <axql:param name="author">
    <axql:for select="//book[author = $author]">
      <book title="{title}" year="{year}">
    </axql:for>
  </axql:function>

  <axql:main>
    <results>
      <axql:for select="distinct(//auhtor)">
        <result>
          <axql:copy select="."/>
          <axql:call-function name="find-book">
            <axql:with-param name="author"
              select="."/>
          </axql:call-function>
        </result>
      </axql:for>
    </results>
  </axql:main>

</axql:query>
```

[1] แม้เครื่องหมาย < ซึ่งแสดงถึงการเปิด markup จะถูกห้ามใช้ในค่าของ attribute แต่เครื่องหมาย > ซึ่งแสดงถึงการปิด markup สามารถใช้งานได้ตามปกติ โดยเป็นไปตามข้อกำหนดของมาตรฐานเอกสาร XML version 1.0, นอกจากนี้แม้เครื่องหมาย < จะเป็นเครื่องหมายที่ถูกต้องของมาตรฐาน XPath version 1.0 อย่างไรก็ตามเมื่อนำมาใช้ร่วมกับ XML จึงต้องเป็นไปตามข้อกำหนดของมาตรฐาน XML

จากตัวอย่าง เป็นการนิยามฟังก์ชันชื่อ find-book โดยค่าที่ฟังก์ชันคืนกลับมาจะเป็นประเภทข้อมูล xsd:any ที่ถูกระบุด้วย attribute type, xsd:any เป็นประเภทของข้อมูลชนิดหนึ่งที่นิยามโดย XML Schema หมายถึง sub tree ใดๆ ซึ่งจะมีค่าเท่ากับกรณีที่ไม่ได้ระบุไว้ใน attribute type ในความเป็นจริงการระบุชนิดของข้อมูลเป็น xsd:any ในที่นี้ไม่ได้ช่วยในการตรวจสอบใดๆ เลย โดยปกติมักจะใช้ประเภทของข้อมูลที่ผู้ใช้ได้นิยามไว้ก่อนแล้ว เพื่อตรวจสอบว่าข้อมูลที่คืนกลับมานั้นมีความถูกต้องตามประเภทที่ต้องการจริงๆ, ในตัวอย่างนี้ยังได้แสดงการใช้งานคำสั่ง axql:param และ axql:with-param เพื่อใช้ในการส่งค่าไปให้ยัง function ด้วย, ตัวอย่างข้างต้นเป็นการเรียกดูหนังสือโดยแบ่งตามชื่อของผู้เขียนแต่ละคน ฟังก์ชัน distinct() ใช้เพื่อจัดกลุ่มตามชื่อผู้แต่ง หลังจากนั้นจึงสร้าง element author แล้วจึงเรียกฟังก์ชันเพื่อสร้าง element book ของหนังสือทุกเล่มที่แต่งโดยผู้แต่งคนนี้

7.4. การเรียกใช้ฟังก์ชันภายใน path expression

ในบางครั้งการเรียกใช้งานฟังก์ชันโดยผ่าน element axql:call-function ไม่สะดวกและไม่สามารถทำงานในบางลักษณะได้ AXQL จึงกำหนดให้มีวิธีการเรียกใช้งานฟังก์ชันได้อีกวิธีหนึ่ง คือการเรียกใช้งานฟังก์ชันภายใน path expression ซึ่งมีความยืดหยุ่นในการใช้งานมากกว่า สามารถแสดงได้ดังตัวอย่าง

```
<axql:query
  xmlns:axql="http://www.cpe.ku.ac.th/AXQL">

<axql:function name="depth" type="xsd:integer">
  <axql:param name="e">
  <axql:if test="empty($e/*)">
    <axql:then>1</axql:then>
    <axql:else>
      <axql:value type="xsd:integer"
        select="max(depth($e/*)) + 1">
    </axql:else>
  </axql:if>
</axql:function>

<axql:main>
  <depth>
    <axql:value type="xsd:integer"
      select="depth(document('partlist.xml'))"/>
  </depth>
</axql:main>

</axql:query>
```

จากตัวอย่างข้างต้น เป็นการนิยามฟังก์ชัน depth เพื่อใช้หาระดับความลึกของ sub element โดยจะทำงานในลักษณะ recursive คือมีการวนเรียกเข้าตัวเอง, การใช้งานฟังก์ชัน depth จะถูกเรียกใช้ได้จากภายใน path expression เช่น depth(document('partlist.xml')) หรือ max(depth($e/*)) + 1 เป็นต้น

การรองรับการทำงานของฟังก์ชันแบบ recursive ทำให้ AXQL สามารถเรียกค้นไปในโครงสร้างที่ซับซ้อนของเอกสาร XML ได้อย่างมีประสิทธิภาพ ซึ่งการใช้งานฟังก์ชันแบบ recursive นี้จะสามารถใช้ได้กับทั้งการเรียกใช้งานฟังก์ชันจากชื่อฟังก์ชันโดยตรง และโดยอาศัย pattern ของข้อมูล

7.5. คำสั่ง axql:include

ในบางครั้งต้องการเรียกใช้งานฟังก์ชันที่ถูกนิยามไว้ในไฟล์ AXQL อื่นๆ จึงจำเป็นต้องมีกลไกในการรวมเอกสาร AXQL หลายๆ ไฟล์เข้าด้วยกัน ด้วยคำสั่ง axql:include ซึ่งเป็น top level element ของ axql:query, อย่างไรก็ตามเมื่อรวมเอกสาร AXQL จากหลายๆ แหล่งเข้าด้วยกันแล้วจะต้องมี element axql:main ได้เพียงหนึ่งเดียวเท่านั้น, การใช้งาน axql:include แสดงได้ดังตัวอย่าง

```
<axql:query
  xmlns:axql="http://www.cpe.ku.ac.th/AXQL">

  <axql:include href="library.axql"/>

  <axql:main>
    <!-- AXQL instructions-->
  </axql:main>

</axql:query>
```

7.6. ฟังก์ชันแบบ built-in

การใช้งานฟังก์ชันที่กล่าวมาข้างต้น เป็นการเรียกใช้ฟังก์ชันที่ผู้ใช้เป็นผู้นิยามขึ้นเอง อย่างไรก็ตาม AXQL ได้จัดเตรียมฟังก์ชันแบบ built-in ซึ่งเป็นฟังก์ชันพื้นฐานที่มีความจำเป็นในการใช้งานสำหรับการสืบค้นข้อมูล XML ให้ผู้ใช้สามารถเรียกใช้ได้, ตัวอย่างฟังก์ชันแบบ built-in ที่ได้แสดงให้เห็นก่อนหน้านี้ ยกตัวอย่างเช่น ฟังก์ชัน document() เพื่อคืนค่า root node ของเอกสาร XML, ฟังก์ชัน distinct() เพื่อใช้ในการจัดกลุ่มข้อมูล, ฟังก์ชัน empty() เพื่อใช้ตรวจสอบว่าเป็น node ว่างหรือไม่ เป็นต้น

เนื่องจาก AXQL มีการใช้งาน XPath เป็น path expression, ดังนั้น AXQL จะสามารถเรียกใช้งานฟังก์ชันต่างๆ ที่มีอยู่ภายใน XPath ได้ เช่น ฟังก์ชัน position() เพื่อระบุตำแหน่งของ node ภายใน node list, ฟังก์ชัน id() เพื่อการเชื่อมโยงกันระหว่างข้อมูลในส่วนต่างๆ ด้วยความสัมพันธ์ของ attribute แบบ ID และ IDREF หรือ IDREFS, ฟังก์ชันเกี่ยวกับ data type ประเภทต่างๆ เช่น string(), substring(), boolean(), number() เป็นต้น นอกจากนี้ AXQL ยังมีการนิยามฟังก์ชันแบบ build-in อื่นๆ ที่มีประโยชน์ในการสืบค้นข้อมูลเพิ่มเติมอีก เช่น ฟังก์ชันที่เกี่ยวกับการคำนวณค่าสรุปของข้อมูล (aggregation function) เช่นเดียวกับที่มีในภาษา SQL อันได้แก่ฟังก์ชัน sum(), count(), max(), min(), avg() เป็นต้น การใช้งานฟังก์ชันการคำนวณค่าสรุปของข้อมูล แสดงได้ดังตัวอย่าง

```
<axql:for select="distinct(//publisher)">
  <axql:var name="publisher" select="text()"/>
  <axql:var name="books_price"
    select="//book[publisher = $publisher]/price"/>
  <publisher axql:value="$publisher"/>
  <avg_books_price axql:value="avg($books_price)"/>
</axql:for>
```

จากตัวอย่างข้างต้น เป็นการใช้งานฟังก์ชัน distinct() เพื่อจัดกลุ่มข้อมูลตามชื่อสำนักพิมพ์ และใช้ฟังก์ชัน avg() เพื่อหาค่าเฉลี่ยของราคาหนังสือของแต่ละสำนักพิมพ์

8. การเรียงลำดับข้อมูล

จากตัวอย่างการเรียกค้นข้อมูลที่ผ่านมา ลำดับข้อมูลของคำตอบจะเป็นเช่นเดียวกับลำดับที่ในเอกสารต้นฉบับ ในกรณีที่ต้องการให้ผลลัพธ์เรียงลำดับตามข้อมูลที่ต้องการจะเรียกใช้คำสั่ง axql:sort, คำสั่ง axql:sort จะใช้งานร่วมกับคำสั่ง axql:for และ axql:call-function เมื่อเรียกใช้ฟังก์ชันโดยการ match pattern เท่านั้น ตัวอย่างการใช้งาน เช่น

ใช้คำสั่ง axql:sort ร่วมกับ axql:for

```
<axql:for select="//book">
  <axql:sort select="title" type="xsd:string"/>
  <book>
    <title><axql:value select="title"/></title>
  </book>
</axql:for>
```

ใช้คำสั่ง axql:sort ร่วมกับ axql:call-function

```
<axql:call-function select="//book">
  <axql:sort select="title" type="xsd:string"/>
</axql:call-function>
```

จากตัวอย่างข้างต้น ภายในคำสั่ง axql:for และ axql:call-function จะอ้างอิงไปยังเซตของ element book แต่ก่อนที่จะวนรอบแต่ละ node เพื่อสร้างผลลัพธ์หรือส่งไปให้ฟังก์ชันนั้นจะต้องจัดเรียงลำดับการทำงานของข้อมูลตามคำสั่ง axql:sort ซึ่งในที่นี้ ต้องการให้ข้อมูลหนังสือ ถูกจัดเรียงตามชื่อหนังสือ โดยจัดเรียงในลักษณะของข้อความธรรมดาเป็นต้น, การใช้งานคำสั่ง axql:sort กับ axql:for นั้น สามารถเขียนอยู่ในรูปแบบย่อได้ ดังตัวอย่าง

```
<book axql:for="//book" axql:sort="title">
  <title axql:value="title"/>
</book>
```

9. การทดสอบชนิดของข้อมูล

การใช้งาน XML ในยุคแรกๆ นั้น ข้อมูลที่เก็บอยู่ภายในจะอยู่ในรูปของข้อความเท่านั้น อย่างไรก็ตามความจำเป็นในการระบุประเภทของข้อมูลที่จัดเก็บในระดับของเอกสาร XML นั้นนับว่ามีความสำคัญมาก จึงเกิดมาตรฐาน XML Schema ขึ้นทำให้การระบุชนิดของข้อมูลภายในเอกสาร XML มีความชัดเจนยิ่งขึ้น ข้อมูลที่จัดเก็บสามารถมีประเภทข้อมูลที่หลากหลาย นอกจากนั้นยังสามารถตรวจสอบได้ในระดับของเอกสาร XML, AXQL สามารถทำงานร่วมกับ XML Schema เพื่อการตรวจสอบประเภทของข้อมูลที่เป็นคำตอบจากการเรียกค้นว่าเป็นไปตามที่ต้องการหรือไม่ ด้วยคำสั่ง axql:data-type ดังตัวอย่างต่อไปนี้

การนิยามประเภทข้อมูลด้วย XML Schema

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2000/08/XMLSchema">

  <xsd:complexType name="MyResultType">
    <xsd:element name="results">
      <xsd:complexType>
        <xsd:element name="book" type="BookType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:complexType>

  <xsd:complexType name="BookType">
    <xsd:attribute name="year" type="xsd:integer"/>
    <xsd:element name="title" type="xsd:string"/>
  </xsd:complexType>

</xsd:schema>
```

การทดสอบประเภทของข้อมูล

```
<axql:data-type name="MyResultType">
  <results>
    <axql:for select="//book">
      <book year="{@year}">
        <title><axql:value select="title"/></title>
      </book>
    </axql:for>
  </results>
</axql:data-type>
```

จากตัวอย่างข้างต้น เป็นการตรวจสอบประเภทข้อมูลของผลลัพธ์ที่ได้จากการเรียกค้นข้อมูลว่าเป็นประเภท MyResultType ที่ถูกนิยามไว้จาก XML Schema หรือไม่ ถ้าประเภทของข้อมูลไม่ถูกต้อง AXQL processor ก็จะแจ้งเตือนให้รู้ถึงข้อผิดพลาด, คำสั่ง axql:data-type สามารถใช้ร่วมกับการสร้าง element แบบ literal ได้เป็นรูปแบบย่อดังนี้

```
<results axql:data-type="MyResultType">
  <book year="{@year}" axql:for="//book">
    <title axql:value="title"/>
  </book>
</results>
```

10. การแก้ไขข้อมูล

จากคุณสมบัติของ AXQL ที่ได้กล่าวมาทั้งหมดข้างต้นล้วนเป็นคุณสมบัติที่ใช้ในการเรียกค้นข้อมูลทั้งสิ้น อย่างไรก็ตามนอกเหนือจากการใช้งานภาษาสืบค้นเพื่อเรียกค้นข้อมูลแล้ว จุดประสงค์ที่สำคัญอีก

ประการหนึ่งก็คือเพื่อการจัดการข้อมูลที่ถูกเก็บอยู่ภายในเอกสาร XML เช่น การเพิ่มข้อมูลใหม่, การลบข้อมูล และการแก้ไขข้อมูลเป็นต้น, ในหัวข้อนี้จะเป็นการแนะนำคุณสมบัติของ AXQL ที่ทำหน้าที่ในการจัดการกับข้อมูลในเอกสาร XML บางส่วน ซึ่งได้แก่คำสั่ง axql:insert, axql:delete และ axql:update ดังมีรายละเอียดต่อไปนี้

10.1. คำสั่ง axql:insert

คำสั่ง axql:insert จะใช้ในการแทรก node หรือ sub tree ของข้อมูลเข้าไปในเอกสาร XML, มีพารามิเตอร์ที่สำคัญคือ attribute select เพื่ออ้างอิงไปยัง node ที่จะนำข้อมูลใหม่เข้าไปแทรก, attribute prep เป็นตัวระบุตำแหน่งที่แน่นอนในการแทรกข้อมูลเมื่อเทียบกับ node ที่อ้างอิงมาจาก attribute select, attribute new-node เป็น path expression ที่ระบุถึงข้อมูลใหม่ที่จะนำมาแทรก หรือหากไม่ต้องการอ้างอิงถึงข้อมูลเก่าก็สามารถสร้างข้อมูลใหม่ขึ้นมาได้ภายในคำสั่ง axql:insert ดังตัวอย่าง

ตัวอย่างที่ 1

```
<axql:insert select="//book/price" prep="after">
  <new-price>
    <axql:value select="0.80 * ./data()">
  </new-price>
</axql:insert>
```

ตัวอย่างที่ 2

```
<axql:insert select="//book[position() = last()]"
  prep="after">
  <book year="2000">
    <title>XML Tutorial</title>
    <author>
      <last>Adam</last>
      <first>Smith</first>
    </author>
    <publisher>New-Wave</publisher>
    <price>49.99</price>
  </book>
</axql:insert>
```

คำสั่งในตัวอย่างที่ 1 จะดำเนินการวนรอบหนังสือทุกๆ เล่ม แล้วแทรก element new-price ที่มีค่าเป็น 80% ของราคาเดิมต่อท้ายจาก element price ที่มีอยู่, คำสั่งในตัวอย่างที่ 2 ทำการแทรก element book ใหม่ต่อท้าย element book ที่มีอยู่เดิมอันสุดท้าย

10.2. คำสั่ง axql:delete

คำสั่ง axql:delete จะใช้ลบ node ออกจากเอกสาร XML มีพารามิเตอร์ที่สำคัญคือ attribute select ที่จะชี้ไปยัง node ที่ต้องการจะลบออก ดังตัวอย่าง

```
<axql:insert select="//book/price" prep="after">
  <new-price axql:value="0.80 * ./data()"/>
</axql:insert>
<axql:delete select="//book/price"/>
```

จากตัวอย่าง ในหนังสือทุกๆ เล่ม จะการสร้าง element new-price ขึ้นมาใหม่ จากนั้นจึงลบ element price เดิมทิ้งไป

10.3. คำสั่ง axql:update

คำสั่ง axql:update ใช้ในการแทนที่ node ด้วย node ของข้อมูลใหม่ ดังตัวอย่าง

```
<axql:update select="//price">
  <new-price xql:value="0.80 * ./data()"/>
</axql:update>
```

จากตัวอย่างข้างต้นเป็นการแทนที่ element price ด้วย element new-price

11. การพัฒนาระบบการสืบค้นด้วยภาษา AXQL

เนื่องจาก AXQL เป็นภาษาที่ถูกพัฒนาขึ้นมาโดยมีพื้นฐานเป็นเอกสาร XML จึงทำให้การพัฒนาระบบการสืบค้นด้วยภาษา AXQL (AXQL Query Engine) เป็นไปด้วยความสะดวก เนื่องจากสามารถใช้ parser ของ XML ที่มีอยู่เป็นแล้วพื้นฐานได้ โดยไม่ต้องเขียน parser ขึ้นมาใหม่โดยเฉพาะ ทำให้ลดเวลาในการพัฒนาลง ผู้พัฒนาจะสามารถเน้นความสนใจไปยังขั้นตอนในการทำงานของ query engine ได้อย่างเต็มที่

ตัวอย่างการนำระบบการสืบค้นด้วยภาษา AXQL ไปใช้งาน เช่น การใช้งานร่วมกับโปรแกรมจัดการฐานข้อมูล (Database Management System), เนื่องจากในปัจจุบันข้อมูลเอกสาร XML มีจำนวนมากขึ้นเรื่อยๆ และโปรแกรมจัดการฐานข้อมูลในรุ่นใหม่จะสามารถรองรับการเก็บข้อมูลแบบ XML ได้ อย่างไรก็ตามยังไม่มีกลไกในการสืบค้นข้อมูล XML ที่เป็นมาตรฐาน ดังนั้นผู้ผลิตแต่ละรายก็จะกำหนดวิธีการของตัวเองทำให้ผู้ใช้เกิดความสับสน และต้องผูกติดกับผลิตภัณฑ์ตัวใดตัวหนึ่งจนเกินไป ในที่นี้ AXQL จะสามารถถูกใช้เป็น gateway เพื่อเชื่อมต่อระหว่างผู้ใช้ซึ่งจะสืบค้นข้อมูลด้วยภาษา AXQL และโปรแกรมจัดการฐานข้อมูล ระบบจะเป็นตัวรองรับในการติดต่อกับกลไกที่หลากหลายของโปรแกรมจัดการฐานข้อมูลแบบต่างๆ ให้ ทำให้โปรแกรมที่เขียนมีความยืดหยุ่นสูง และมีความซับซ้อนน้อยลง เป็นต้น ตัวอย่างอื่นที่เห็นได้ชัดในการใช้งาน AXQL เช่น การใช้งานร่วมกับ Web Server เป็นต้น เนื่องจากปัจจุบัน โปรแกรม Web Browser สามารถรองรับการทำงานกับ XML ได้ ดังนั้นข้อมูลที่ถูกส่งมาจาก Web Server จึงสามารถเป็นเอกสาร XML ได้ อย่างไรก็ตามข้อมูล XML แบบ static จะยากต่อการจัดการ ซึ่งสามารถแก้ไขได้โดยเก็บข้อมูล XML เหล่านี้ไว้ในฐานข้อมูล จากนั้นเมื่อผู้ใช้ต้องการข้อมูล Web Server ก็จะเรียกโปรแกรมหรือ script ที่ทำงานร่วมกับ AXQL เพื่อเรียกค้นข้อมูลที่ผู้ใช้ต้องการส่งกลับไป, ในกรณีนี้ทั้งข้อมูลและเงื่อนไขในการเข้าถึงข้อมูลของผู้ใช้จะถูกแยกออกจากกัน ทำให้ง่ายต่อการจัดการมากยิ่งขึ้น

12. บทสรุป

AXQL เป็นภาษาสืบค้นสำหรับ XML ที่ได้รับอิทธิพลมาจาก ทั้ง XQuery และ XSLT ทำให้ได้รับการสืบทอดข้อดีหลายๆ ประการจากภาษาทั้งสองมา, AXQL ใช้ไวยากรณ์ที่เป็นแบบ XML และการทำงานเลียนแบบ template ของ XSLT ทำให้สามารถทำงานกับเอกสารที่มีความซับซ้อนสูงได้ดี แต่อาศัยรูปแบบการทำงานที่กะทัดรัดและกระชับของ XQuery จึงทำให้เข้าใจได้ง่าย นอกจากนี้ยังมีรูปแบบการใช้งานแบบย่อ ซึ่งเป็นรูปแบบการใช้งานหลักๆ ที่มักจะใช้บ่อย จึงทำให้แม้จะอยู่ในรูปแบบของ XML แต่ก็มีความง่ายต่อการใช้งานเหมือน XQuery, สามารถทำงานร่วมกับข้อมูลในแบบเอกสาร และฐานข้อมูลได้ดีเหมือนทั้ง XQuery และ XSLT, สามารถทำงานร่วมกับมาตรฐานอื่นๆ ของ XML ที่มีอยู่แล้วได้อย่างสอดคล้อง, มีคุณสมบัติที่จำเป็นในการจัดการข้อมูลภายในเอกสาร XML เช่น คำสั่งในการเพิ่มข้อมูล, ลบข้อมูล และแก้ไขข้อมูล นอกจากนี้ยังสามารถขยายคุณสมบัติเพิ่มเติมในอนาคตได้โดยง่ายเนื่องจากใช้รูปแบบ XML ที่เป็นมาตรฐานและมีความยืดหยุ่นสูง

ตารางที่ 1 แสดงการเปรียบเทียบคุณสมบัติที่สำคัญๆ โดยสรุประหว่าง XQuery, XSLT และ AXQL ซึ่งจะทำให้เห็นได้ว่า AXQL นั้นมีความสามารถที่ไม่ด้อยไปกว่า XQuery ในขณะที่ได้รับคุณสมบัติที่มีประโยชน์หลายประการมาจาก XSLT, ซึ่งผู้เขียนหวังว่า AXQL จะเป็นทางเลือกหนึ่งที่น่าสนใจในการสืบค้นข้อมูล XML หรือเป็นแนวทางเพื่อนำไปสู่การพัฒนาภาษาสืบค้นสำหรับ XML ที่สมบูรณ์ยิ่งขึ้นต่อไป

ตารางที่ 1. แสดงการเปรียบเทียบระหว่าง XQuery, XSLT และ AXQL โดยอ้างอิงจากเอกสาร XML Query Requirements ของ W3C [6]

คุณสมบัติที่เปรียบเทียบ	XQuery	XSLT	AXQL
สามารถเข้าใจและใช้งานได้ง่าย	X	-	X
มีรูปแบบการใช้งานเป็น XML[2]	-	X	X
เป็นอิสระจากมาตรฐานอื่นๆ ที่เกี่ยวข้อง	X	X	X
มีการนิยามข้อผิดพลาดพื้นฐาน (exception)	-	-	-
รองรับการขยายความสามารถในอนาคต	X	X	X
ทำงานร่วมกับ XML Schema	X	-	X
รองรับการทำงานบนเซตของเอกสาร XML	X	-	X
สามารถสืบค้นข้อมูลที่อ้างอิงกันระหว่างเอกสารได้	X	-	X
ทำงานร่วมกับ XML Namespace	X	X	X
รองรับข้อมูลทั้งแบบโครงสร้างและเรียงลำดับ	X	X	X
มีฟังก์ชันในการคำนวณค่าสรุปของข้อมูล (aggregation function)	X	-	X
เรียงลำดับข้อมูลจากการสืบค้นได้	X	X	X
เปลี่ยนหรือสร้างโครงสร้างข้อมูลได้	X	X	X
ทำงานร่วมกับมาตรฐาน XML อื่นๆ ที่มีอยู่แล้วได้	X	X	X
การแก้ไขและเปลี่ยนแปลงข้อมูล[3]	-	-	X

[2] เอกสาร requirements ระบุว่าอาจเป็นรูปแบบที่มนุษย์สามารถใช้งานได้ง่าย หรือเป็นรูปแบบ XML ก็ได้

[3] เป็นคุณสมบัติที่นอกเหนือจากที่ระบุไว้ในเอกสาร requirements

เอกสารอ้างอิง

[1] เอกพล จีรังสุวรรณ และ สมนึก คีรีโต, การวิเคราะห์และเปรียบเทียบภาษาสืบค้นสำหรับ XML ระหว่าง XQuery และ XSLT, 2544 (กำลังอยู่ในระหว่างการพิจารณาเพื่อขอตีพิมพ์ลงในวารสารวิชาการเนคเทค)

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

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

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

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

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

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

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

[9] World Wide Web Consortium, XQuery 1.0: An XML Query Language. W3C Working Draft, June 07, 2001. Available http://www.w3.org/TR/xquery

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

ภาคผนวก ก. ไวยากรณ์ของ AXQL

```
<!-- Category: instruction -->
<axql:attribute
  name = { ncname }
  prefix = { prefix }
  value = string-expression>
  <!-- Content: instructions -->
</axql:attribute>

<!-- Category: instruction -->
<axql:call-function
  name = qname
  select = node-set-expression>
  <!-- Content: (axql:sort | axql:with-param)* -->
</axql:call-function>

<!-- Category: instruction -->
<axql:choose>
  <!-- Content: (axql:when+, axql:otherwise?) -->
</axql:choose>

<!-- Category: instruction -->
<axql:comment
  value = string-expression>
  <!-- Content: instructions -->
</axql:comment>

<!-- Category: instruction -->
<axql:copy
  select = expression />

<!-- Category: instruction -->
<axql:data-type
  name = schema-type>
  <!-- Content: instructions-->
</axql:data-type>

<!-- Category: instruction-->
<axql:delete
  select = node-set-expression />

<!-- Category: instruction -->
<axql:element
  name = { ncname }
  prefix = { prefix }>
  <!-- Content: instructions -->
</axql:element>

<!-- Category: instruction -->
<axql:else>
  <!-- Content: instructions -->
</axql:else>

<!-- Category: instruction -->
<axql:for
  select = node-set-expression>
  <!-- Content: (axql:sort*, instructions) -->
</axql:for>

<!-- Category: top-level-element -->
<axql:function
  name = qname
  pattern = pattern
  type = schema-type
  priority = number>
  <!-- Content: (axql:param*, instructions) -->
</axql:function>

<!-- Category: instruction -->
<axql:if
  test = boolean-expression>
  <!-- Content: (axql:then, axql:else?) -->
</axql:if>

<!-- Category: top-level-element -->
<axql:include
  href = uri-reference />

<!-- Category: instruction-->
<axql:insert
  select = node-set-expression
  prep = ("before" | "after" | "in")
  new-node = node-set-expression>
  <!-- Content: instructions -->
</axql:insert>

<!-- Category: top-level-element -->
<axql:main>
  <!-- Content: instructions -->
</axql:main>

<!-- Category: instruction -->
<axql:namespace
  prefix = { prefix }
  uri = { uri-reference } />

<!-- Category: instruction -->
<axql:otherwise>
  <!-- Content: instructions -->
</axql:otherwise>

<!-- Category: instruction -->
<axql:param
  name = qname
  type = schema-type />

<!-- Category: instruction -->
<axql:pi
  name = { ncname }
  value = string-expression>
  <!-- Content: instructions -->
</axql:pi>

<axql:query>
  <!-- Content: top-level-elements -->
</axql:query>

<!-- Category: instruction -->
<axql:sort
  select = string-expression
  type = schema-type
  order = ("ascending" | "descending") />

<!-- Category: instruction -->
<axql:then>
  <!-- Content: instructions -->
</axql:then>

<!-- Category: instruction-->
<axql:update
```

```
  select = node-set-expression
  new-node = node-set-expression>
  <!-- Content: instructions -->
</axql:update>

<!-- Category: instruction -->
<axql:value
  select = string-expression
  type = schema-type
  format = data-format />

<!-- Category: top-level-element -->
<!-- Category: instruction -->
<axql:var
  name = qname
  select = expression>
  <!-- Content: instructions -->
</axql:var>

<!-- Category: instruction -->
<axql:when
  test = boolean-expression>
  <!-- Content: instructions -->
</axql:when>

<!-- Category: instruction -->
<axql:where
  test = boolean-expression>
  <!-- Content: instructions -->
</axql:where>

<!-- Category: instruction -->
<axql:with-param
  name = qname
  select = expression>
  <!-- Content: instructions -->
</axql:with-param>
```


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 XML.