รายงานการวิจัย

การค้นหาสถานที่จากแผนที่

Semantic Search in Maps

สันธนะ อู่อุดมยิ่ง

ได้รับทุนสนับสนุนงานวิจัยจากเงินงบประมาณแผ่นดินหรือเงินรายได้
ประจำปีงบประมาณ 2550
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

บทคัดย่อ

ผู้วิจัยทดลองนำโครงสร้างข้อมูลเพื่อสร้างความสัมพันธ์ของบริเวณในระดับเขตของกรุงเทพมหานคร ความสัมพันธ์ที่เกิดขึ้นทำให้ความหมายของเขตเป็นประโยชน์ต่อผู้ใช้มากขึ้น กล่าวคืออำนวยให้ระบบการค้นหาสถานที่ทางเว็ปสามารถอำนวยให้ผู้ใช้ระบบสามารถเลือกบริเวณที่สถานที่ที่ผู้ใช้สนใจได้อย่างมีประสิทธิภาพทั้งในแง่ของการแสดงผลและเวลาที่ใช้ในการประมวลผล ผู้ใช้สามารถนำโปรแกรมต้นแบบที่พัฒนาขึ้นนี้ไปประยุกต์ใช้กับธุรกิจอสังหาริม-ทรัพย์มือสอง

abstract

We created semantic relationships among district for Bangkok. The information would enhance any adopting application to better find resultset for its users. The idea is to allow user to specify his area of interest which the resultset is relative to it in term of surround districts. With our example in real-estate web application, Our result shows that the system performance outperforms 3-tier web application.

สารบัญ

บทนำ.....1
งานวิจัยที่เกี่ยวข้อง.....2
วิธีการดำเนินงาน.....5
ผลการดำเนินงาน.....9
สรุปและข้อเสนอแนะ.....12

บทนำ

ในการค้นหาข้อมูลเกี่ยวกับอสังหาริมทรัพย์มีความแตกต่างจากการค้นหาข้อมูลสถานที่ทั่วไปจากอินเตอร์เนต เช่นหากเราต้องการข้อมูลสถานที่ท่องเที่ยวในภูเก็ต เราสามารถค้นหาจาก search engine ด้วยชื่อสถานที่ได้ทันที แม้ว่า search engine พยายามเดาคำใกล้เคียง เพื่อเพิ่มอำนวยความสะดวกให้ผู้ใช้ เลือก keyword ที่ดียิ่งขึ้น แต่ในโปรแกรมประยุกต์บนอินเตอร์เนตนั้นยังใช้รูปแบบของ exact match กับข้อมูลในฐานข้อมูล ซึ่งในการค้นหาบ้านมือสองนั้น keyword ที่เราต้องการใช้จะเป็นชื่อของบริเวณที่เราสนใจ เช่น ทองหล่อ บางกะปิ เป็นต้น ปัญหาคือคำเหล่านี้ไม่มีความสัมพันธ์กับบริเวณใกล้เคียง เช่น ทองหล่อ — เอกมัย หรือ บางกะปิ — รามคำแหง ซึ่งในความเป็นจริงนั้น ข้อมูลที่อยู่ในบริเวณใกล้เคียงนั้นควรได้รับการคัดเลือกให้อยู่ในเซตของคำตอบด้วย

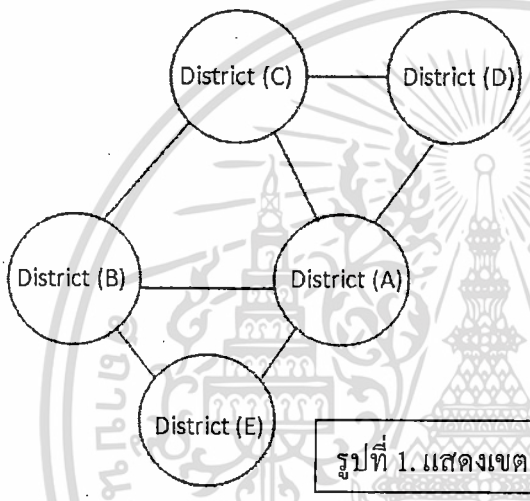
ในกรณีที่ผู้สนใจซื้อบ้าน หาข้อมูลจาก search engine เช่นค้นหาโดยใช้ keyword บ้าน บางกะปิ จะเกิดปัญหาในลักษณะ high recall, low precision กล่าวคือ ได้ผลลัพธ์มากมายนับหมื่นรายการ แต่อาจไม่มีรายการที่เป็นการขายบ้าน ที่ยังไม่ได้ขาย พร้อมรายละเอียดเลยก็ได้ ในทางตรงกันข้าม หากผู้สนใจซื้อบ้าน หาข้อมูลจากเว็ปขายอสังหาริมทรัพย์[1] จะพบปัญหาในลักษณะของ low or no recall เนื่องจาก เขาเลือกได้เพียงว่าอยู่เขตลาดพร้าว หรืออาจไม่มีตัวช่วยเลยก็ได้ ซึ่งทำให้พลาดโอกาสที่จะพบอสังหาริมทรัพย์ในบริเวณที่ต้องการ

งานวิจัยนี้จึงมุ่งพัฒนาโครงสร้างข้อมูลเพื่อเป็นเครื่องมือสำหรับการค้นหาอสังหาริมทรัพย์มือสอง ซึ่งสามารถนำไปประยุกต์กับธุรกิจอื่นๆ ได้หลากหลายจากความสัมพันธ์เชิงพื้นที่เช่นนี้ โดยพัฒนาเป็นโปรแกรมต้นแบบ

[1] http://www.homedd.com, http://www.ban4you.com, และอื่นๆ

งานวิจัยที่เกี่ยวข้อง

งานวิจัยทางด้าน web semantic searching กำลังได้รับความนิยมเป็นอย่างสูง semantic searching คือความสามารถในการตีความความหมายของคำที่ผู้ใช้ระบุเป็น keyword ในการค้นหาเพื่ออำนวยให้ผู้ใช้ได้รับความสะดวกในการใช้ระบบเนื่องจากความซับซ้อนของศัพท์ของภาษามนุษย์ และเพิ่มโอกาสในการค้นพบเว็ปที่ต้องการหาได้มากยิ่งขึ้น ในส่วนของธุรกิจอสังหาริมทรัพย์นั้น ระบบการค้นหายังไม่ได้รับการพัฒนาเท่าที่ควร เนื่องจากระบบไม่รู้จักความสัมพันธ์ของบริเวณที่ติดต่อกัน อีกทั้งการอ้างถึงบริเวณของคนกรุงเทพฯ ไม่ได้อ้างอิงตามเขตอย่างทางการ แต่อ้างอิงด้วยบริเวณที่เจาะจงมากเช่น แยกบางกะปิ รูปที่ 1. แสดงเขตในกรุงเทพฯ ซึ่งอาจมีเขตที่ติดต่อกันกี่เขตก็ได้



รูปที่ 1. แสดงเขตในกรุงเทพฯ ซึ่งอาจมีเขตที่ติดต่อกันกี่เขตก็ได้

การสร้างความสัมพันธ์ของสิ่งต่างๆนั้นสามารถใช้เทคนิค Ontology[1] ในลักษณะที่เรียกว่าการสร้างคอนเซ็ปต์ เช่นต้องการสื่อว่าแมว ปลาโลมา เป็น สัตว์เลี้ยงลูกด้วยนม ในขณะเดียวกัน แมว ช้าง เป็นสัตว์บก กล่าวคือสร้างคลาสของสัตว์เลี้ยงลูกด้วยนม และ สัตว์บก ซึ่งสามารถแสดงความสัมพันธ์ดังกล่าวด้วยกราฟ โดยมี โหนดของสัตว์ โหนดของคลาส (class) และแสดงการเป็นสมาชิกของ ด้วยเส้นเชื่อมระหว่างโหนด ซึ่งในกรณีนี้เราสามารถพิจารณาว่า

---

[1] The term of *Ontology* originates from philosophy. It is used as the name of a subfield of philosophy, namely, the study of the nature of existence, the branch of metaphysics concerned with identifying, in the most general terms, the kinds of things that actually exits, and how to describe them.

However, in more recent years, *ontology* has become one of the many words hijacked by computer science and given a specific technical meaning that is rather different from the original one. Instead of "ontology" we now speak of "*an* ontology." For our purposed, we will use T. R. Gruber's definition, later refined by R. Studer: *An ontology is an explicit and formal specification of a conceptualization.*

โหนดของคลาส เป็นอีก layer หนึ่ง (hierarchical) การ represent แบบนี้แต่ละโหนดสามารถทับซ้อน (overlapped) เป็นสมาชิกมากกว่า 1 คลาส หรือ โหนดคลาสสามารถสามารถซ้อน (recursive) อย่างอิสระ เช่น แมวเป็นสัตว์ กล้วยไม้เป็นพืช และต่างเป็นสิ่งมีชีวิต ในขณะเดียวกัน คลาสของสัตว์ 4 เท้า ไม่จำเป็นต้องส่วนหนึ่งของคลาสสัตว์บกเสมอไป

เวิร์ดเนต (Wordnet[1]) สร้างความสัมพันธ์ของคำสำหรับการใช้งานคอมพิวเตอร์ โดยใช้การ represent แบบเดียวกับที่กล่าวมาในข้างต้น โดยที่เวิร์ดเนตจะมีหลายเวอร์ชัน เนื่องจากใครที่ต้องการสร้างความสัมพันธ์ก็สามารถปรับปรุงจากที่มีอยู่ หรือสร้างขึ้นใหม่ก็ได้ อย่างไรก็ดีผู้วิจัยไม่พบว่ามีการสร้างความสัมพันธ์ของสถานที่ในกรุงเทพมหานคร

อีกแนวทางหนึ่งในการสร้างความสัมพันธ์ใช้การใช้ความน่าจะเป็น และทางสถิติ ซึ่งปัจจุบันเรียกว่า tag recommendation กล่าวคือ ปัจจุบันในแต่ละหน้าของหน้าเว็ปมีการสังเคราะห์ keyword เป็น metadata ไว้ ซึ่งในอดีตผู้เขียนมักไม่ใส่ใจที่จะสร้าง metadata เอาไว้ หรืออาจเป็นการจงใจสร้างเพื่อหวังผลให้หน้าเว็ปของตนเป็นส่วนหนึ่งของกลุ่มของหน้าเว็ปคำตอบ (hit pages) ระบบอาจปรับปรุง metadata ที่สังเคราะห์ขึ้น และ สร้าง/เรียนรู้/สังเคราะห์ความสัมพันธ์ระหว่าง keyword ที่ผู้ใช้ใช้ และ metadata ที่สังเคราะห์ขึ้น

สำหรับการวิจัยนี้ ผู้วิจัยเสนอให้ใช้แสดงข้อมูลของเขตในกรุงเทพมหานคร และความสัมพันธ์ระหว่างเขตในเชิงภูมิศาสตร์ ในรูปแบบของ XML[2] เนื่องจากโปรแกรมประยุกต์บนอินเตอร์เนตสามารถประมวลผลข้อมูลนั้นได้ทันทีด้วยเทคโนโลยีของ XSLT[3] เนื่องจากใน

---

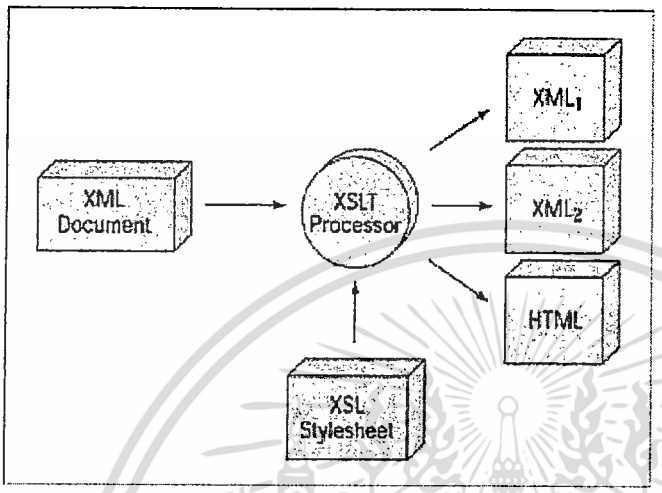
1 http://wordnet.princeton.edu

2 The Extensible Markup Language (XML) is a general-purpose *specification* for creating custom markup languages. Its primary purpose is to help information systems share structured data, particularly via the Internet, and it is used both to encode documents and to serialize data. It is a related W3C (World Wide Web Consortium) standard, and is the basis for standard information interchange. It is classified as an extensible language because it allows its users to define their own elements. XML is a generic framework for storing any amount of text or any data whose structure can be represented as a tree. The only indispensable syntactical requirement is that the document has exactly one root element (alternatively called the document element). This means that the text must be enclosed between a root start-tag and a corresponding end-tag.

3 The Extensible Stylesheet Language for Transformations (XSLT) is a language that provides the mechanism to transform and manipulate XML data while XML provides structure to information, XSLT, along with another related standard, XPath (XML path language), provides the means to extract, restructure, and manipulate that information. Another standard closely related to XSLT is Extensible Stylesheet Language (XSL). It is in charged for styling or laying out XML documents into a form that makes sense to its intended audience. XSL uses to define a set of formatting rules that are referred to when an XML document is processed.

ความเป็นจริงนั้นความสัมพันธ์ของเขตในกรุงเทพมหานครนั้นไม่ซับซ้อนและปริมาณของความสัมพันธ์ไม่มาก กล่าวคือไม่จำเป็นต้องเก็บความสัมพันธ์นี้ใน relational database

รูปที่ 2. XSLT ประมวลผลข้อมูลตามเงื่อนไขและใช้รูปแบบของ XSL เพื่อสร้างเอาท์พุตในรูปแบบ XML ซึ่งเราสามารถนำไปแสดงผลแบบ HTML ได้ทันที โดยการประมวลผลนั้น XLST จะเห็นอินน์พุตในรูปแบบ XML ตามเทคโนโลยี XPath



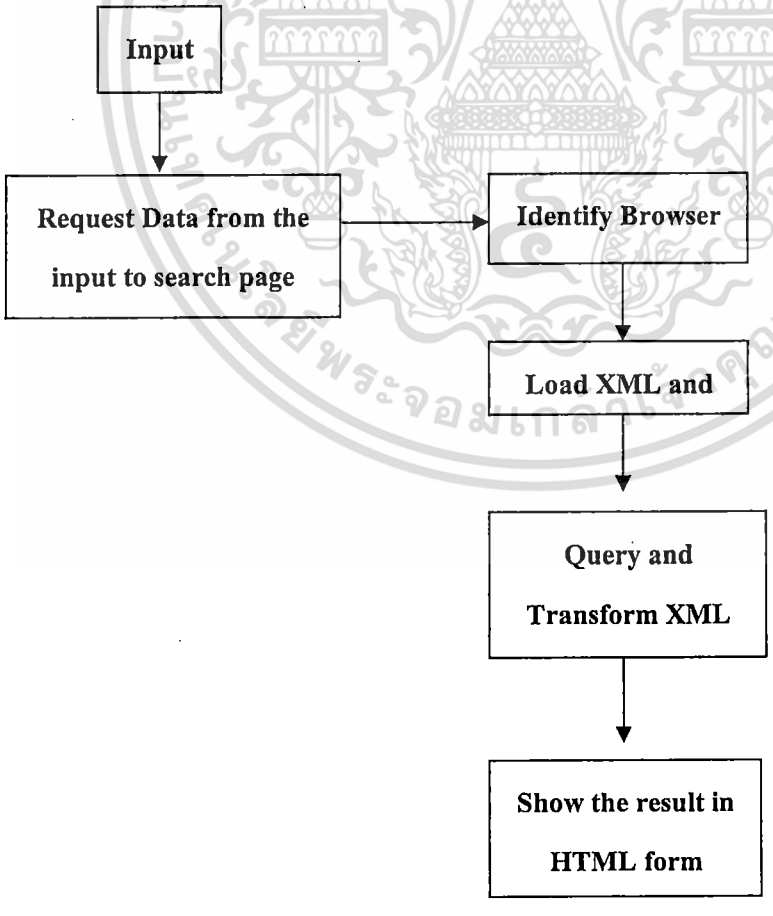
รูปที่ 2. แสดงกระบวนการทำงานในการประมวลผลข้อมูลที่อยู่ในรูป XML

วิธีการดำเนินงาน

จากที่กล่าวมาในบทนำ รูปแบบของการแสดงผลที่ต้องการคือ ความสามารถในการแสดงผลมากกว่า 1 สถานที่โดยไม่คำนึงถึงบริเวณ ตัวอย่างเช่น ปัจจุบันการใช้แผนที่ประกอบเว็ปกำลังเป็นเทคโนโลยีที่กำลังได้รับความนิยมอย่างสูง ซึ่งนอกจากจะอำนวยความสะดวกแก่ผู้ใช้เป็นอย่างมากในการเข้าใจถึงตำแหน่งของสถานที่ที่แสดงบนแผนที่ ลักษณะการแสดงตำแหน่งดังกล่าวยังสามารถแสดงสถานที่ใกล้เคียงโดยไม่คำนึงถึงบริเวณ(หรือเขต) ที่สถานที่นั้นอยู่

อย่างไรก็ดีการใช้แผนที่ประกอบเว็ปเป็นเพียงการแสดงผล สิ่งที่เราต้องการคือระบบที่อำนวยต่อการค้นหาสถานที่ที่ควรจะอยู่ในเซตของคำตอบ

รูปที่ 3. แสดงกระบวนการทำงานของโปรแกรมต้นแบบ โดยอินพุตนั้นนอกจากจะเป็นเขตที่ผู้ใช้ต้องการจะหาแล้ว ยังประกอบไปด้วยข้อมูลของตัวอสังหาริมทรัพย์ด้วยเช่น ประเภท (คอนโด บ้านเดี่ยว เป็นต้น) ช่วงราคาที่สนใจ พื้นที่ใช้สอย จำนวนห้องนอน เป็นต้น และเนื่องจาก browser IE นั้นมีไวยากรณ์สำหรับการโหลดข้อมูล XML (แสดงในรูปที่ 4.) ต่างกันจึงต้องมีการตรวจสอบตราของ browser ก่อน จากนั้นโปรแกรมต้นแบบจะโหลดข้อมูลความสัมพันธ์ของเขตในกรุงเทพมหานคร (แสดงในรูปที่ 5 6 และ 7) ซึ่งเป็นเป้าหมายของศึกษานี้ ขั้นตอนการประมวลผลข้อมูลที่เหลือนั้นไม่ต่างจากที่กล่าวมาแล้วข้างต้น



รูปที่ 3. แสดงกระบวนการทำงานของโปรแกรมต้นแบบ

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <row>
      <field name="propID">P000001</field>
      <field name="name">Grant Village</field>
      <field name="type">House</field>
      <field name="price">2550000</field>
      <field name="size">85</field>
      <field name="bedroom">3</field>
      <field name="restroom">3</field>
      <field name="disID">D008</field>
      <field name="contact">0868273437</field>
    </row>
```

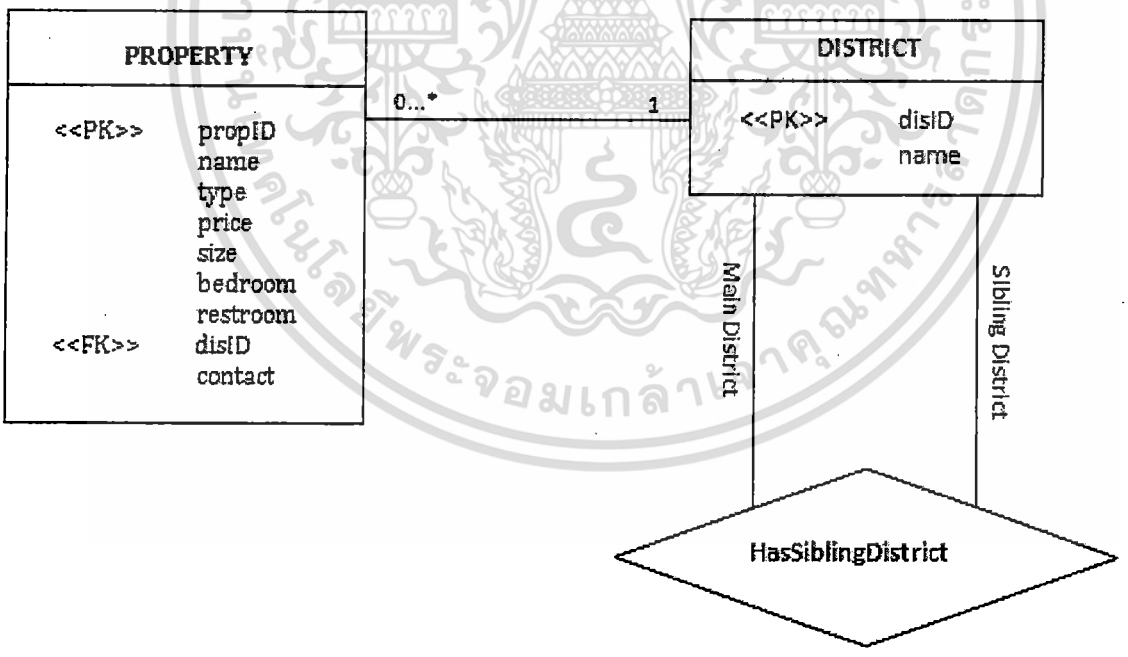
รูปที่ 4. แสดงโครงสร้างของอินน์พุตด้วยตัวอย่าง

```
    <row>
      <field name="disID">D006</field>
      <field name="name">Sounglouk</field>
    </row>
    <row>
      <field name="disID">D007</field>
      <field name="name">Bangkapi</field>
    </row>
    <row>
      <field name="disID">D008</field>
      <field name="name">Kunnawoa</field>
    </row>
    <row>
      <field name="disID">D009</field>
      <field name="name">Buengkum</field>
    </row>
    <row>
      <field name="disID">D010</field>
      <field name="name">Klongsamva</field>
    </row>
</ROOT>
```

รูปที่ 5. แสดงโครงสร้างของเขตในกรุงเทพมหานครด้วยตัวอย่าง

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
    <row>
      <field name="main">D001</field>
      <field name="sibling">D002</field>
    </row>
    <row>
      <field name="main">D001</field>
      <field name="sibling">D003</field>
    </row>
    <row>
      <field name="main">D001</field>
      <field name="sibling">D004</field>
    </row>
    <row>
      <field name="main">D001</field>
      <field name="sibling">D005</field>
    </row>
```

รูปที่ 6. แสดงโครงสร้างของความสัมพันธ์ของเขตในกรุงเทพมหานครด้วยตัวอย่าง



รูปที่ 7. แสดงความสัมพันธ์ของความสัมพันธ์ของเขตในกรุงเทพมหานครในรูป E-R

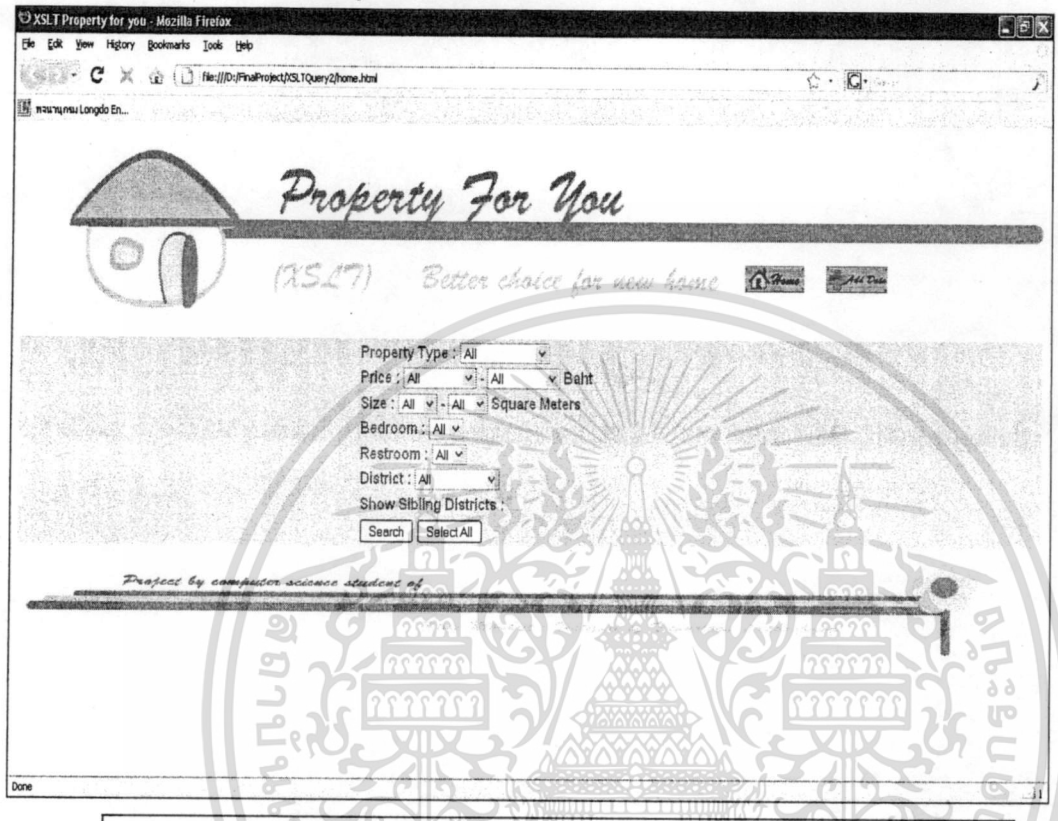
ดังที่กล่าวมาแล้วว่า XSLT ซึ่งเป็นเครื่องมือที่อำนวยให้สามารถค้นหาข้อมูลในรูปแบบ XML เสมือนการค้นหาข้อมูลด้วย SQL จาก RDBMS ซึ่งเป็นส่วนหนึ่งรูปแบบ 3-tier อันเป็นที่นิยมทั่วไปของการเขียนโปรแกรมประยุกต์บนอินเตอร์เนต ดังนั้นระบบนี้จะไม่ต้องใช้ application server ดังแสดงในรูปที่ 8 สิ่งที่ผู้จัยสนใจจากนี้คือประสิทธิภาพของระบบโดยจะแสดงประสิทธิภาพของระบบในส่วนถัดไป

MySQL	System Requirement	XSLT
✔	Web browser	✔
✔	MySQL server	
✔	Application server(Apache Tomcat), Window Server.	
✔	Driver connect to Database (mysql-connector-java)	
	Java script	✔

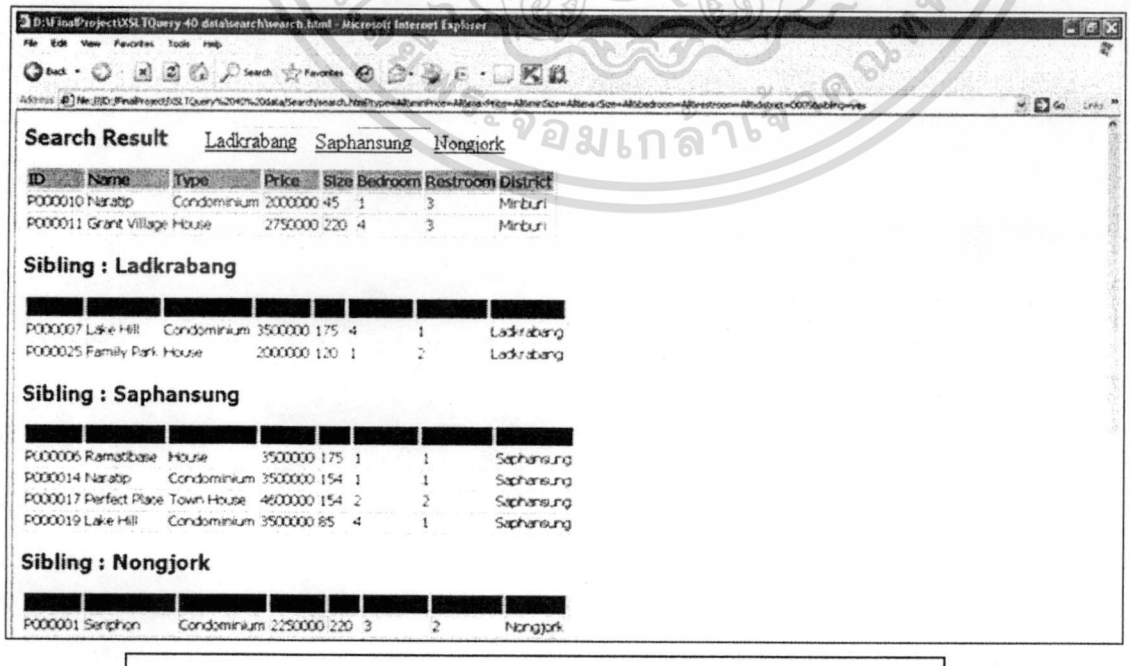
รูปที่ 8. แสดงการเปรียบเทียบ infrastructure ของระบบ กับ รูปแบบ 3-tier model (ด้วย MySQL)

ผลการดำเนินงาน

รูปที่ 9 แสดงผลการตัวอย่างของหน้าจอของโปรแกรมต้นแบบ รูปที่ 10 แสดงผลการค้นหาอสังหาริมทรัพย์ในเขตมีนบุรี โดยมี reference เพื่อให้ผู้ใช้ดูอสังหาริมทรัพย์ในเขตที่อยู่ติดกัน ได้แก่ เขตลาดกระบัง สะพานสูง และหนองจอก



รูปที่ 9. ตัวอย่างหน้าจอโปรแกรมต้นแบบการค้นหาอสังหาริมทรัพย์มือสอง



รูปที่ 10. แสดงผลการค้นหาอสังหาริมทรัพย์มือสอง ในเขตมีนบุรี

ตารางข้างล่างแสดงการเปรียบเทียบขนาดของข้อมูลจำนวน 10,000 — 100,00 รายการซึ่ง ผู้วิจัยนำมาใช้ทดลองหาประสิทธิภาพของ XSLT

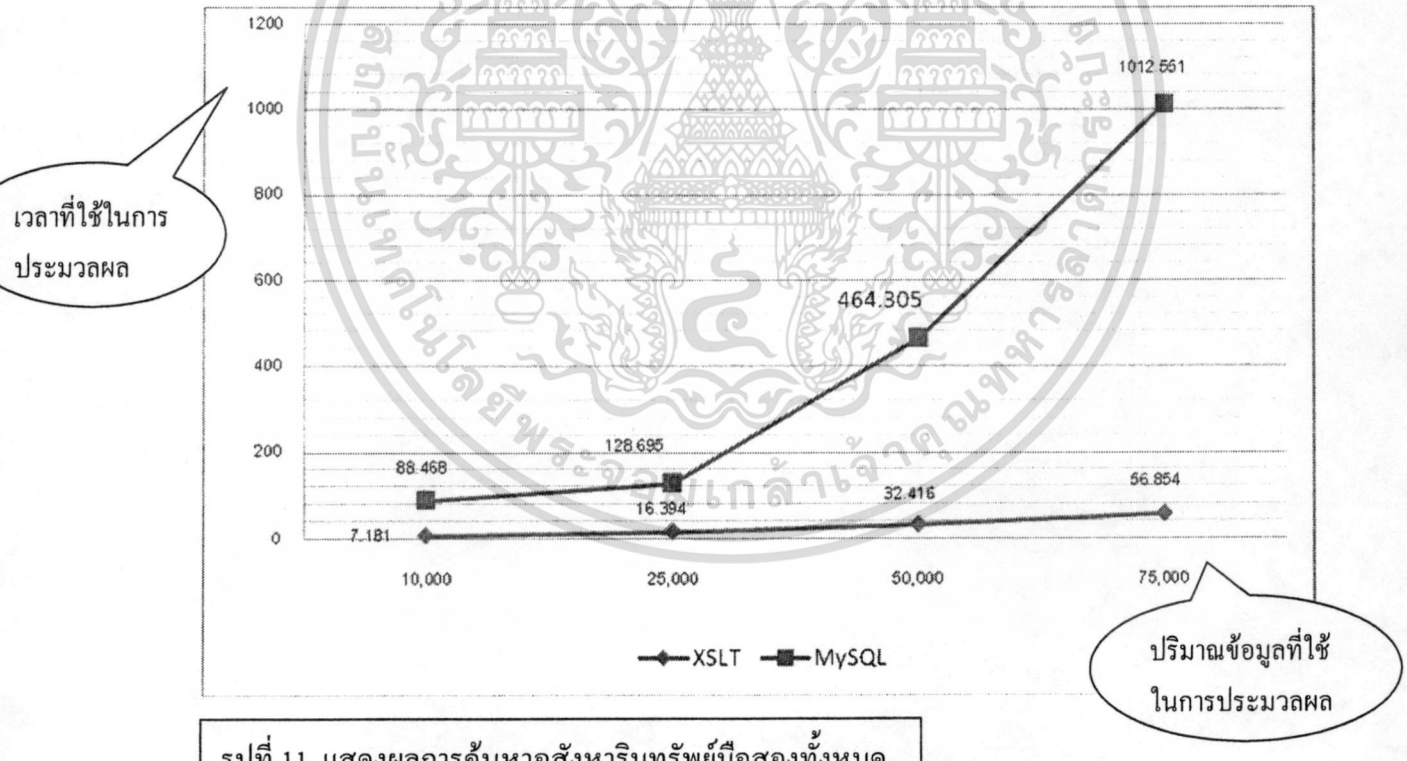
Data Size (Rows)	10,000	25,000	50,000	75,000	100,000
XSLT	3.26 MB	9.64 MB	19.20 MB	28.90 MB	38.60 MB
MySQL	744 KB	1.81 MB	3.62 MB	5.43 MB	7.24 MB

เราทำการทดลองเป็น 2 กรณี ได้แก่

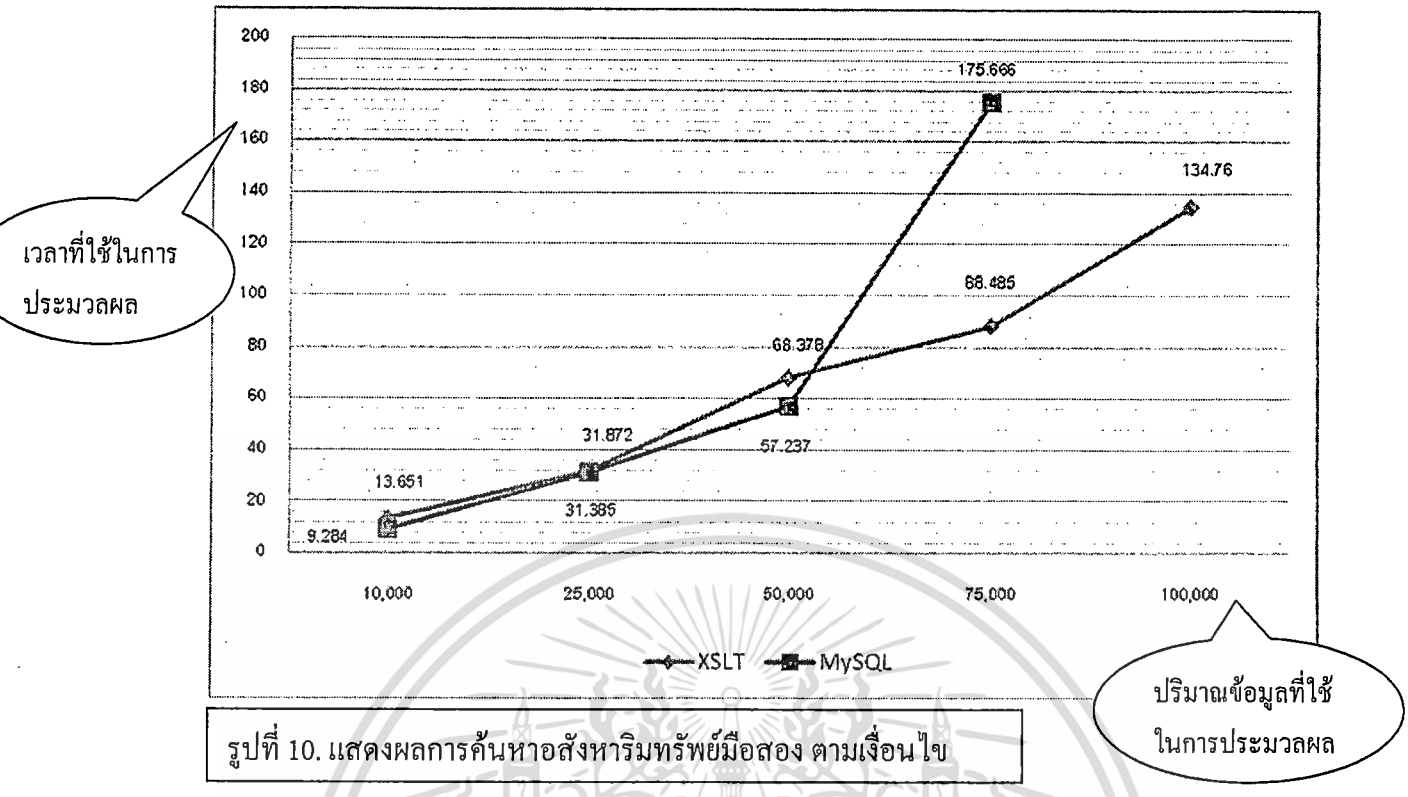
I. XSLT และ MySQL ดึงข้อมูลทั้งหมดขึ้นมาแสดง

II. XSLT และ MySQL ดึงข้อมูลตามเงื่อนไขที่ผู้ใช้ต้องการ

ซึ่งเวลาในการประมวลผลแสดงในรูปที่ 11 และ 12 บนเครื่อง CPU Inter Pentium M 1.73 GHz, Ram 2.0 GB



รูปที่ 11. แสดงผลการค้นหาอสังหาริมทรัพย์มือสองทั้งหมด



รูปที่ 10. แสดงผลการค้นหาอสังหาริมทรัพย์มือสอง ตามเงื่อนไข

จากรูปทั้งสองจะเห็นว่า

I. ระบบที่ใช้ฐานข้อมูลในกรณีนี้ใช้เวลามากกว่า XSLT เสมอ

II. ในการประมวลผลตามเงื่อนไข ทั้งสองรูปแบบใช้เวลาในการประมวลผลมากขึ้น แต่ระบบที่ใช้ฐานข้อมูลไม่สามารถให้ผลลัพธ์ได้ในกรณีที่ข้อมูลมีปริมาณมากกว่า 75,000 ข้อมูล

สรุปและข้อเสนอแนะ

สรุป

XSLT มีคุณสมบัติเด่นที่เหมาะกับโปรแกรมประยุกต์แนวการ annotate ความสัมพันธ์ของข้อมูล เช่นในกรณีในความสัมพันธ์ของเขตในกรุงเทพมหานครเพื่อค้นหาอสังหาริมทรัพย์มือสอง เนื่องจากความสัมพันธ์ที่ไม่มากและไม่ซับซ้อน จึงไม่จำเป็นต้องอาศัย RDBMS ทำให้ระบบสามารถรองรับข้อมูลได้มากกว่า และประมวลผลได้เร็วกว่า เมื่อเปรียบเทียบการการเขียนโปรแกรมประยุกต์ด้วยการใช้ RDBMS เช่น MySQL

โปรแกรมต้นแบบในการค้นหาอสังหาริมทรัพย์มือสองสามารถเอื้อให้ผู้ที่ต้องการซื้อสามารถค้นหาอสังหาริมทรัพย์ได้อย่างมีประสิทธิภาพมากขึ้น

ข้อเสนอแนะ

I. การสร้างความสัมพันธ์ของบริเวณสามารถตอบสนองความต้องการของผู้ใช้ระบบการค้นหาเชิงแผนที่ได้ เนื่องจากเขตที่กรุงเทพมหานครหรือเมืองต่างๆใช้ไม่มีรูปแบบที่ชัดเจน (แม้แต่การดูความสัมพันธ์ด้วยรหัสไปรษณีย์ก็เป็นเรื่องยากเพราะพื้นที่ครอบคลุมกว้างเกินไป) ในขณะที่ผู้ใช้มักอ้างอิงถึงสถานที่ด้วยคำที่ไม่เป็นทางการ
II. การสร้างความสัมพันธ์ของสถานที่ สามารถนำไปประยุกต์ได้หลายรูปแบบ เช่น ชื่อที่ไม่เป็นทางการ (ซอยอารีย์ = พหลโยธิน 7)
III. เราสามารถสร้างความสัมพันธ์ของพื้นที่ในระดับที่ผู้ใช้คุ้นเคย เพื่อเป็นโครงสร้างให้ประแกรมประยุกต์เชิงแผนที่ได้นำไปใช้เพื่อเพิ่มประสิทธิภาพในการค้นหาสถานที่ โดยเฉพาะอย่างยิ่งความสัมพันธ์ที่มีรูปแบบชัดเจนเช่น abstract layer ของบริเวณที่เป็นที่รู้จักดีในกรุงเทพฯ ที่มีระยะห่างเท่าๆกันเพื่อ
    - อำนวยในการหาทิศ และระยะห่างจากจุดที่ผู้ใช้สนใจ
    - นำข้อมูลไปประกอบกับค่า latitude longitude เพื่อคำนวณหาระยะห่าง จากบริเวณอ้างอิงที่สนใจได้

จะเห็นได้ว่าเราสามารถนำเทคนิคการพัฒนาโปรแกรมบนอินเตอร์เนตที่ศึกษามาต่อยอดได้ ซึ่งเพิ่มประสิทธิภาพการใช้แผนที่ได้

XSLT has a lot of commands such as elements, functions that are suitable for helping a user to query the data.

The <xsl:template>Element

An XSLT processor looks in a stylesheet for an xsl:template element that has a match attribute with value of /

The following example code stylesheet, Query.xsl, shows the xsl:template element with the match attribute of the root node of the XPath model of the source tree).

The <xsl:template> element is used to build templates. All <xsl:template> elements must have either the match or the name attribute defined. Although not common, it is also possible to create <xsl:template> elements that have both a match and a name. The match="/" attribute associates the template with the root of the XML source document. We would like to show XSL codes in Figure 3.11.

```
<xsl:template match="/"> <html> <body>
  <h2>Real Estate</h2>
  <table border="1">
    <th>ID</th>
    <th>Name</th>              <th>Type</th>
    <th>Price</th>             <th>Size</th>
    <th>Bedroom</th>           <th>Restroom</th>
    <th>District</th>          <th>Contact</th>
  <tr>
    <td><xsl:value-of select="ROOT/row/field[@name='propID']"/></td>
    <td><xsl:value-of select="ROOT/row/field[@name='name']"/></td>
    <td><xsl:value-of select="ROOT/row/field[@name='type']"/></td>
    <td><xsl:value-of select="ROOT/row/field[@name='price']"/></td>
    <td><xsl:value-of select="ROOT/row/field[@name='size']"/></td>
    <td><xsl:value-of select="ROOT/row/field[@name='bedroom']"/></td>
    <td><xsl:value-of select="ROOT/row/field[@name='restroom']"/></td>
    <td><xsl:value-of select="ROOT/row/field[@name='contact']"/></td> </tr>
</table> </body> </html> </xsl:template>
```

Figure 3.11 Query.xsl (using <xsl:template> element)

The <xsl:value-of> element

The <xsl:value-of> element can be used to extract the value of an XML element and add it to the output stream of the transformation.

<td> <xsl:value-of select=" "field[@name='propID']"/> </td>

The <xsl:value-of> element provides the value of a part of the source tree that represents the source XML document. The <xsl:value-of> element has a mandatory select attribute, whose value is an XPath location path.

The xsl:value-of element is the simplest XSLT element that extracts information from the source tree. It simply selects the value of a node-set, which might be only a single node, specified by the location path that is the value of the select attribute of the xsl:value-of element. If there is more than one node in the node-set, then the xsl:value-of element uses the value of the first node in document order only, not the values of all nodes. The xsl:value-of element is particularly useful when producing output for presentation, as in the example just below, but it can also be used when XML is being restructured. The result of the transformation from Figure 3.11 would look like Figure 3.12.

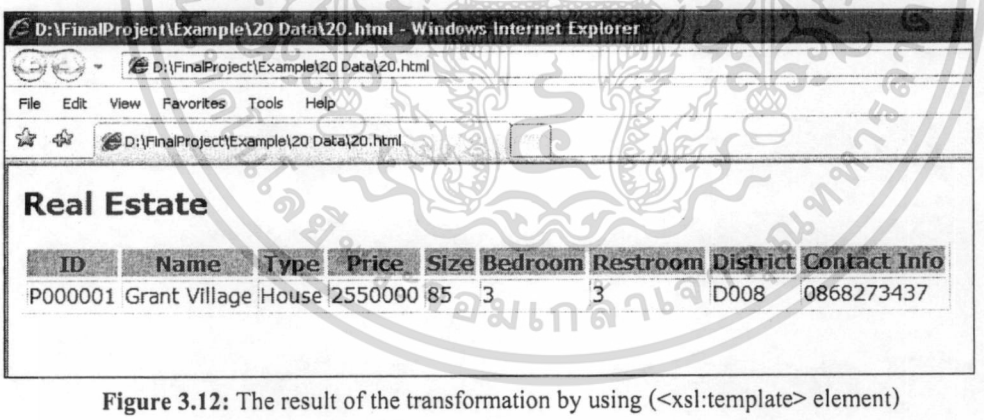


Figure 3.12: The result of the transformation by using (<xsl:template> element)

The <xsl:for-each> element

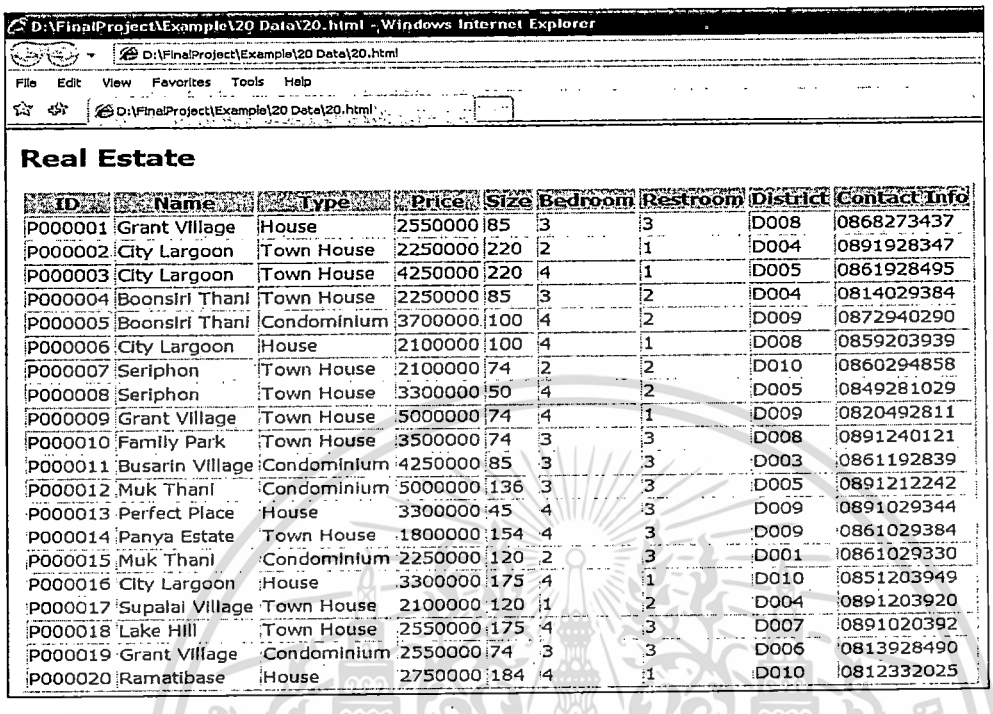
The xsl:for-each element can be used to select every XML element of a specified node-set. Inside the template, the start-tags and end-tags of an unordered list are specified using literal result elements. Between those tags you use the xsl:for-each element to create a list item for each Characteristic element node child of the context node, which is an Object element node. We would like to show XSL codes in Figure 3.13.

```
<xsl:template match="/">
    <xsl:for-each select="ROOT/row">
      <tr>
        <td><xsl:value-of select="field[@name='propID']"/></td>
        <td><xsl:value-of select="field[@name='name']"/></td>
        <td><xsl:value-of select="field[@name='type']"/></td>
        <td><xsl:value-of select="field[@name='price']"/></td>
        <td><xsl:value-of select="field[@name='size']"/></td>
        <td><xsl:value-of select="field[@name='bedroom']"/></td>
        <td><xsl:value-of select="field[@name='restroom']"/></td>
        <td><xsl:value-of select="field[@name='disID']"/></td>
        <td><xsl:value-of select="field[@name='contact']"/></td>
      </tr>
    </xsl:for-each>
  </xsl:template>
```

Figure 3.13: Query.xsl (using <xsl:for-each> element)

The xsl:for-each element loops through each node in a node set in its order of occurrence and applies the same template to each node. A node set is simply the collection of all of the same XML tags (nodes) in an XML file.

The mandatory select attribute provides an expression that specifies which node set is to be processed by the loop. This can simply be a string that is the name of the node. Only one node set can be defined. The result of the transformation from Figure 3.13 would look like Figure 3.14.



Real Estate

ID	Name	Type	Price	Size	Bedroom	Restroom	District	Contact Info
P000001	Grant Village	House	2550000	85	3	3	D008	0868273437
P000002	City Largoon	Town House	2250000	220	2	1	D004	0891928347
P000003	City Largoon	Town House	4250000	220	4	1	D005	0861928495
P000004	Boonsiri Thani	Town House	2250000	85	3	2	D004	0814029384
P000005	Boonsiri Thani	Condominium	3700000	100	4	2	D009	0872940290
P000006	City Largoon	House	2100000	100	4	1	D008	0859203939
P000007	Seriphon	Town House	2100000	74	2	2	D010	0860294858
P000008	Seriphon	Town House	3300000	50	4	2	D005	0849281029
P000009	Grant Village	Town House	5000000	74	4	1	D009	0820492811
P000010	Family Park	Town House	3500000	74	3	3	D008	0891240121
P000011	Busarin Village	Condominium	4250000	85	3	3	D003	0861192839
P000012	Muk Thani	Condominium	5000000	136	3	3	D005	0891212242
P000013	Perfect Place	House	3300000	45	4	3	D009	0891029344
P000014	Panya Estate	Town House	1800000	154	4	3	D009	0861029384
P000015	Muk Thani	Condominium	2250000	120	2	3	D001	0861029330
P000016	City Largoon	House	3300000	175	4	1	D010	0851203949
P000017	Supalai Village	Town House	2100000	120	1	2	D004	0891203920
P000018	Lake Hill	Town House	2550000	175	4	3	D007	0891020392
P000019	Grant Village	Condominium	2550000	74	3	3	D006	0813928490
P000020	Ramatibase	House	2750000	184	4	1	D010	0812332025

Figure 3.14: The result of Query.xsl (using <xsl:for-each> element)

The <xsl:sort> element

The simplest way to rearrange our XML elements is to use the <xsl:sort> element. This element temporarily rearranges a collection of elements based on criteria we define in our stylesheet. The xsl:sort element can be used together with the xsl:apply-templates element and the xsl:for-each element. The follow excerpt from Query.xsl shows in Figure 3.15 and Figure 3.16 shows the result.

The value of the select attribute of the xsl:sort element specifies the value by which the node-set is to be sorted (if you want to sort the characteristics in descending order, you need to specify the value of attribute of the xsl:sort element as descending.)

```
<xsl:template match="/">
  <xsl:for-each select="ROOT/row">
  <xsl:sort select="field[@name='price']"/>
   <tr>
     <td><xsl:value-of select="field[@name='propID']"/></td>
     <td><xsl:value-of select="field[@name='name']"/></td>
     <td><xsl:value-of select="field[@name='type']"/></td>
     <td><xsl:value-of select="field[@name='price']"/></td>
     <td><xsl:value-of select="field[@name='size']"/></td>
     <td><xsl:value-of select="field[@name='bedroom']"/></td>
     <td><xsl:value-of select="field[@name='restroom']"/></td>
     <td><xsl:value-of select="field[@name='disID']"/></td>
     <td><xsl:value-of select="field[@name='contact']"/></td>
   </tr>
  </xsl:for-each>
</xsl:template>
```

Figure 3.15: Query.xsl (using <xsl:sort> Element)

Real Estate

ID	Name	Type	Price	Size	Bedroom	Restroom	District	Contact Info
P000014	Panya Estate	Town House	1800000	154	4	3	D009	0861029384
P000006	City Largoon	House	2100000	100	4	1	D008	0859203939
P000007	Seriphon	Town House	2100000	74	2	2	D010	0860294858
P000017	Supalai Village	Town House	2100000	120	1	2	D004	0891203920
P000002	City Largoon	Town House	2250000	220	2	1	D004	0891928347
P000004	Boonsiri Thani	Town House	2250000	85	3	2	D004	0814029384
P000015	Muk Thani	Condominium	2250000	120	2	3	D001	0861029330
P000001	Grant Village	House	2550000	85	3	3	D008	0868273437
P000018	Lake Hill	Town House	2550000	175	4	3	D007	0891020392
P000019	Grant Village	Condominium	2550000	74	3	3	D006	0813928490
P000020	Ramatibase	House	2750000	184	4	1	D010	0812332025
P000008	Seriphon	Town House	3300000	50	4	2	D005	0849281029
P000013	Perfect Place	House	3300000	45	4	3	D009	0891029344
P000016	City Largoon	House	3300000	175	4	1	D010	0851203949
P000010	Family Park	Town House	3500000	74	3	3	D008	0891240121
P000005	Boonsiri Thani	Condominium	3700000	100	4	2	D009	0872940290
P000003	City Largoon	Town House	4250000	220	4	1	D005	0861928495
P000011	Busarin Village	Condominium	4250000	85	3	3	D003	0861192839
P000009	Grant Village	Town House	5000000	74	4	1	D009	0820492811
P000012	Muk Thani	Condominium	5000000	136	3	3	D005	0891212242

Figure 3.16: The result of Query.xsl (using <xsl:sort> Element)

The <xsl:if> element

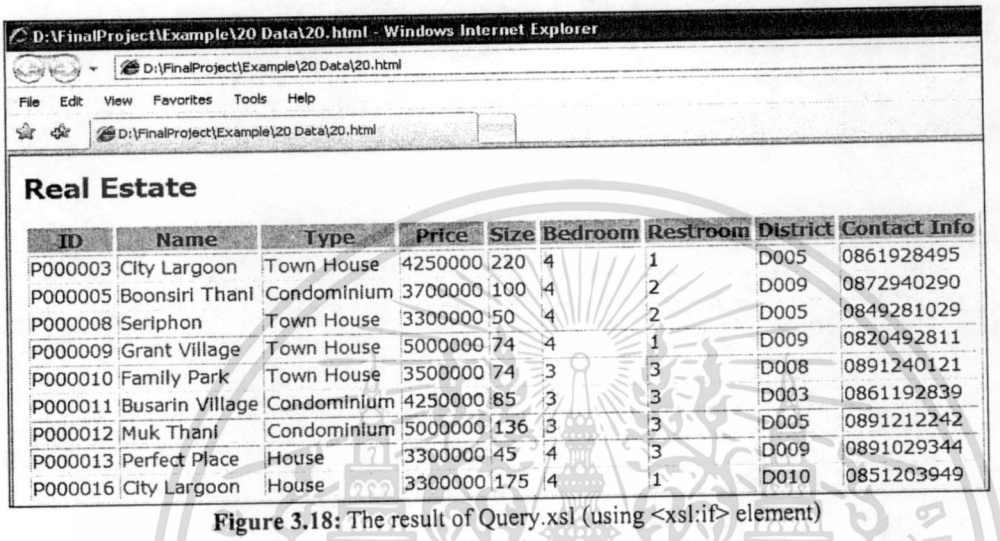
The xsl:if element tests whether a Boolean condition is true or false. If it is true, then the content of the xsl:if element is instantiated. If it is false, then nothing specified inside the xsl:if element is added to the result tree.

This excerpt stylesheet uses the xsl:if element to add to the result tree only when the value of the price attribute exceeds the specified price greater than 3,000,000 Baht. We would like to show XSL codes in Figure 3.17.

```
<xsl:template match="/">
<xsl:for-each select="ROOT/row">
  <xsl:if test="field[@name='price'] &gt; 3000000">
  <tr>
    <td><xsl:value-of select="field[@name='propID']"/></td>
    <td><xsl:value-of select="field[@name='name']"/></td>
    <td><xsl:value-of select="field[@name='type']"/></td>
    <td><xsl:value-of select="field[@name='price']"/></td>
    <td><xsl:value-of select="field[@name='size']"/></td>
    <td><xsl:value-of select="field[@name='bedroom']"/></td>
    <td><xsl:value-of select="field[@name='restroom']"/></td>
    <td><xsl:value-of select="field[@name='disID']"/></td>
    <td><xsl:value-of select="field[@name='contact']"/></td>
  </tr>
  </xsl:if>
  </xsl:for-each>
<xsl:template match="/">
```

Figure 3.17: Query.xsl (using <xsl:if> element)

The xsl:if element is a child element of the xsl:template element. Therefore, if the test attribute of the xsl:if element returns the Boolean value false, then nothing is output from the template for that Size element. The value of the required test attribute contains the expression to be evaluated.[1] The result of the transformation from Figure 3.17 would look like Figure 3.18.



Real Estate

ID	Name	Type	Price	Size	Bedroom	Restroom	District	Contact Info
P000003	City Largoon	Town House	4250000	220	4	1	D005	0861928495
P000005	Boonsiri Thani	Condominium	3700000	100	4	2	D009	0872940290
P000008	Seriphon	Town House	3300000	50	4	2	D005	0849281029
P000009	Grant Village	Town House	5000000	74	4	1	D009	0820492811
P000010	Family Park	Town House	3500000	74	3	3	D008	0891240121
P000011	Busarin Village	Condominium	4250000	85	3	3	D003	0861192839
P000012	Muk Thani	Condominium	5000000	136	3	3	D005	0891212242
P000013	Perfect Place	House	3300000	45	4	3	D009	0891029344
P000016	City Largoon	House	3300000	175	4	1	D010	0851203949

Figure 3.18: The result of Query.xsl (using <xsl:if> element)

The <xsl:choose> element

The <xsl:choose> element is used to determine one course of action based on a series of tests. Each test is done inside an <xsl:when> element. If a test succeeds, the body of the <xsl:when> element is executed. If no tests fail then a <xsl:otherwise> element can be used to specify a default action. We would like to show XSL codes in Figure 3.19.

[1] **Note:** Legal filter operators are: = (equal),! = (not equal), < (less than), > (greater than), < = (less than or equal), > = (greater than or equal), and, or.

```
<xsl:template match="/">
  <xsl:for-each select="ROOT/row">
    <xsl:choose>
     <xsl:template match="/">
  <xsl:for-each select="ROOT/row">
    <xsl:choose>
    <xsl:when test="field[@name='price'] &gt; 3000000">
      <td bgcolor="#ff00ff"><xsl:value-of  select="field[@name='propID']"/></td>
      <td bgcolor="#ff00ff"><xsl:value-of select="field[@name='name']"/></td>
      <td bgcolor="#ff00ff"><xsl:value-of select="field[@name='type']"/></td>
      <td bgcolor="#ff00ff"><xsl:value-of select="field[@name='price']"/></td>
      <td bgcolor="#ff00ff"><xsl:value-of select="field[@name='size']"/></td>
      <td bgcolor="#ff00ff"><xsl:value-of select="field[@name='bedroom']"/></td>
      <td bgcolor="#ff00ff"><xsl:value-of select="field[@name='restroom']"/></td>
      <td bgcolor="#ff00ff"><xsl:value-of select="field[@name='disID']"/></td>
      <td bgcolor="#ff00ff"><xsl:value-of select="field[@name='contact']"/></td>
     </xsl:when>
    <xsl:otherwise>
      <td><xsl:value-of select="field[@name='propID']"/></td>
      <td><xsl:value-of select="field[@name='name']"/></td>
      <td><xsl:value-of select="field[@name='type']"/></td>
      <td><xsl:value-of select="field[@name='price']"/></td>
      <td><xsl:value-of select="field[@name='size']"/></td>
      <td><xsl:value-of select="field[@name='bedroom']"/></td>
      <td><xsl:value-of select="field[@name='restroom']"/></td>
      <td><xsl:value-of select="field[@name='disID']"/></td>
      <td><xsl:value-of select="field[@name='contact']"/></td>
    </xsl:otherwise>
   </xsl:choose>
  </xsl:for-each>
 </xsl:template>
```

Figure 3.19: Query.xsl (using <xsl:choose> element)

Nested inside the xsl:choose element are an xsl:when element and an xsl:otherwise element. On the xsl:when element is a test attribute whose value is a Boolean value. If the value of the test attribute is the Boolean value true, then the content of the xsl:when element is output. If the value of the test attribute of the xsl:when attribute is false, then none of the content of the xsl:when element is output; the content of the xsl:otherwise element is output instead. The result of the transformation from Figure 3.19 would look like Figure 3.20.

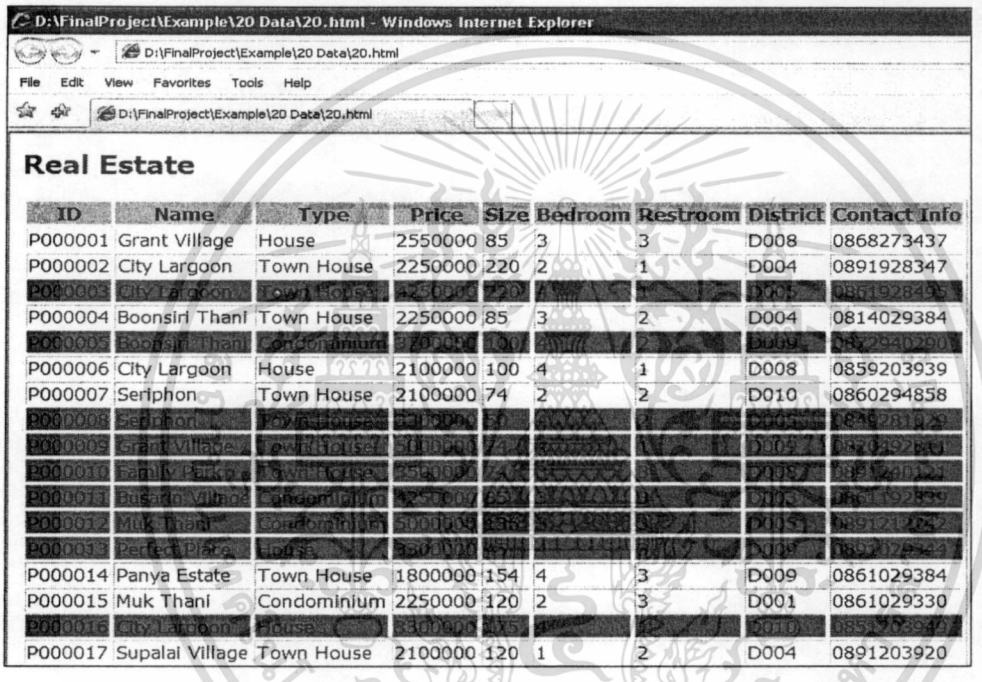


Figure 3.20: The result of Query.xsl (using <xsl:choose> element)

The <xsl:apply-templates> elemen

The xsl:apply-templates element defines a set of nodes to be processed or, by default, selects all child nodes of the current node being processed, and finds a matching template rule to apply to each node in the set.

If we add a select attribute to the <xsl:apply-templates> element it will process only the child element that matches the value of the attribute. We can use the select attribute to specify the order in which the child nodes are processed. We would like to show XSL codes in Figure 3.21.

```
<xsl:template match="row">
    <xsl:apply-templates select="field[@name='name']"/>
    <xsl:apply-templates select="field[@name='price']"/>
</xsl:template>
<xsl:template match="field[@name='name']">
    Name: <span style="color:#ff0000">
    <xsl:value-of select="."/></span> <br />
</xsl:template>
<xsl:template match="field[@name='price']">
    Price: <span style="color:#00ff00">
    <xsl:value-of select="."/></span> Baht <br />
</xsl:template>
```

Figure 3.21: Query.xsl (using <xsl:apply-templates> element)

The select attribute is set to an expression that returns a node set. This can simply be a string that is the name of a node set. The nodes in the node set are processed in the order that they occur (which is called document order). The default for omitting this attribute is to select all of the child nodes of the current node being processed. The result of the transformation from Figure 3.21 would look like Figure 3.22.

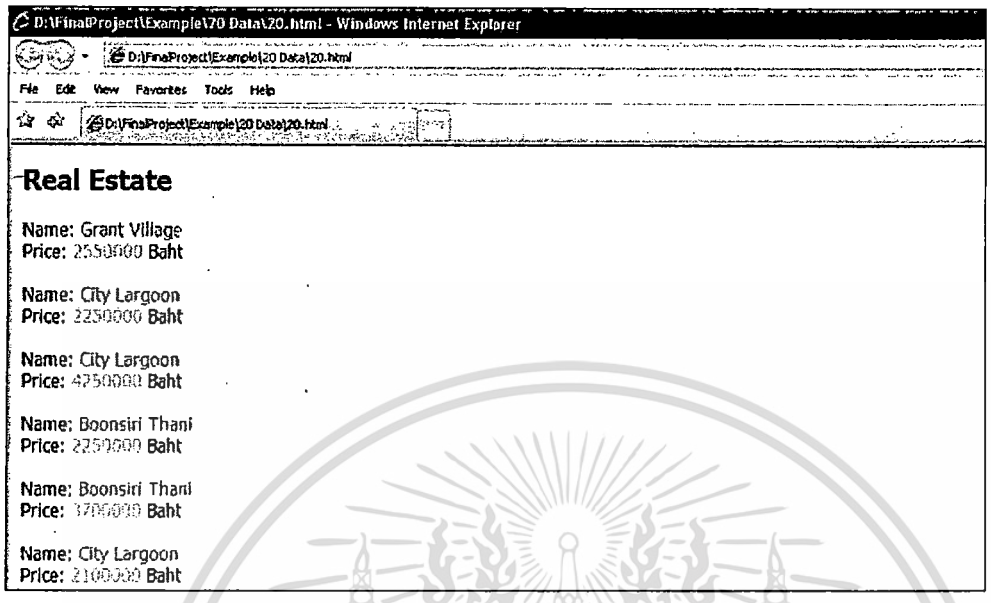


Figure 3.22: The result of Query.xsl (using <xsl:apply-templates> element)

The <xsl:variable> element

The <xsl:variable> element is used to declare a local or global variable. The variable is global if it's declared as a top-level element and local if it's declared within a template.

Once you have set a variable's value, you cannot change or modify that value. You can add a value to a variable by the content of the <xsl:variable> element by the select attribute.

We would like to show XSL codes in Figure 3.23 and the result of the transformation would look like Figure 3.24.

```
<xsl:variable name="staff_table">
 <tr>
    <td><b>Name</b></td>
    <td><b>Price</b></td>
 </tr>
 </xsl:variable>
 <xsl:template match="/">
 <table>
     <xsl:copy-of select="$staff_table" />
     <xsl:for-each select="ROOT/row">
 <tr>
     <td><xsl:value-of select="field[@name='name']" /></td>
     <td><xsl:value-of select="field[@name='price']" /></td>
 </tr>
 </xsl:for-each>
 </table>
```

Figure 3.23: Query.xsl (using <xsl:variable> Element)

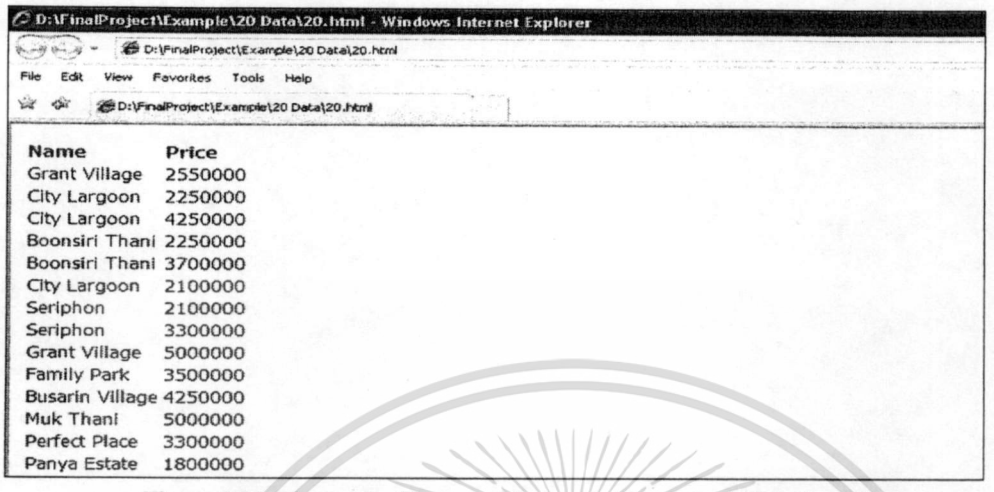


Figure 3.24: The result of Query.xsl (using <xsl:apply-variable> element)

XSL selects range of conditions

In the <xsl:if> condition, check all attributes from Property Schema which match retrieved variables from dropdowns on the first page "home.html". If all conditions are matching with dropdowns, the current element will show on the screen. So, the result is shown in Figure 3.25 and Figure 3.26 shows XSL codes using XSL selects range of conditions.

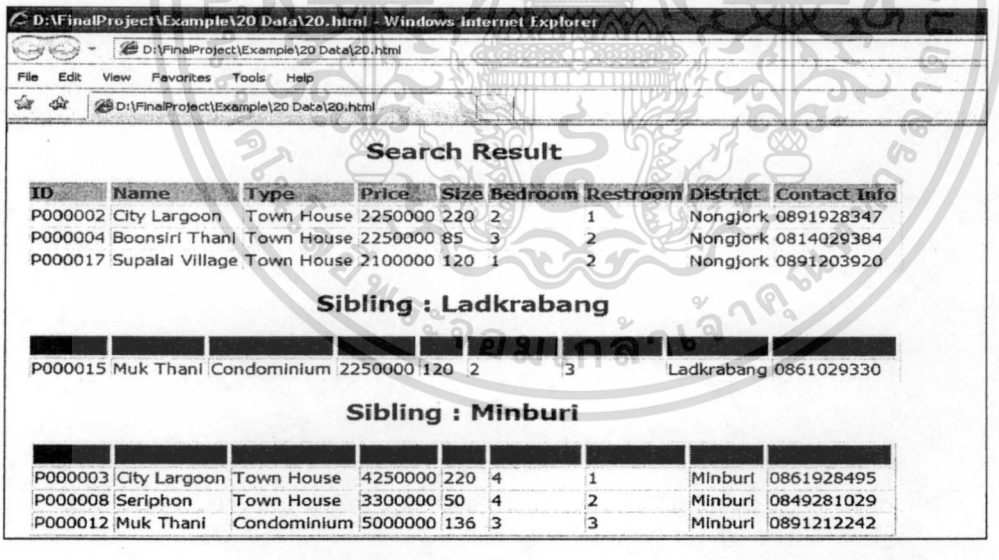


Figure 3.25: The result of the transformation by using XSL selects range of conditions

```
<xsl:if test="field[@name='type']=$myNewType
    and field[@name='price'] &gt;=$minPrice
    and field[@name='price'] &lt;=$maxPrice
    and field[@name='size'] &gt;=$minSize
    and field[@name='size'] &lt;=$maxSize
    and field[@name='bedroom'] &gt;=$minBedroom
    and field[@name='bedroom'] &lt;=$maxBedroom
    and field[@name='restroom'] &gt;=$minRestroom
    and field[@name='restroom'] &lt;=$maxRestroom
    and field[@name='disID']=$disID">
 <tr>
  <td> <xsl:value-of select="field[@name='propID']"/> <td>
  <td> <xsl:value-of select="field[@name='name']"/> <td>
  <td> <xsl:value-of select="field[@name='type']"/> </td>
  <td> <xsl:value-of select="field[@name='price']"/> </td>
  <td> <xsl:value-of select="field[@name='size']"/> </td>
  <td> <xsl:value-of select="field[@name='bedroom']"/> </td>
  <td> <xsl:value-of select="field[@name='restroom']"/> </td>
  <td>
       <xsl:call-template name="matchDistrictName">
       <xsl:with-param name="districtID" select="field[@name='disID']"/>
       </xsl:call-template>
  </td>
  <td> <xsl:value-of select="field[@name='contact']"/> </td>
</tr>
</xsl:if>
```

Figure 3.26: SearchByDistrict.xsl (using XSL selects range of conditions)

Using Keys

A key is a point of access that developers can use to connect one data set to another; similar to the way a hypertext link connects two HTML documents.

Keys are used in XSLT to cross-reference through a combination of the xsl:key element and the key() built-in function; xsl:key defines the key and key() references it. Looking closer, the xsl:key element is used to define a key for a node set. Its syntax is:

<xsl:key name="keyname" match="pattern" use="expression"/>

This element has three attributes:

- **name** defines the name of the key. This label is used to reference the key elsewhere in the stylesheet.
- **match** specifies an XPath pattern, typically returning a node set that contains the key.
- **use** is an expression that names the value of the key.

Linking with the key() function

The xsl:key element is ready, but it cannot do anything on its own. Developers need to use the key() built-in function to actually do something with it. The key() function syntax is:
key(KeyName, LookupValue)

The key() function has two parameters:

- **KeyName** specifies the name of the xsl:key element to use.
- **LookupValue** is an expression that provides a value to look up.

Using Keys with Multiple Source Documents

The preceding example illustrates how developers can use keys to link different XML structures, but it did so within a single XML document. In the real world, developers often want to

use keys to combine data from various XML files. They can do this by using the document() built-in function. This function allows developers to include node sets from external files.

In the result document, suppose we would like to provide a simple listing of the customer and his or her state as we do in the initial example. We can define a StatesLookup variable to be the returning node set of the document('key_state.xml') function:

<xsl:variable name="StatesLookup" select="document('key_state.xml')"/>

Therefore, when we plug in the StatesLookup variable in our stylesheet, it referencesthe states element and its contents. A key is defined for the state elements from the outside file:

<xsl:key name="StateKey" match="state" use="@id"/> ,

To perform the key lookup on these elements from another document, we need to set up an xsl:for-each loop to iterate through each of the state elements, calling key() each time to do the lookup. Because the lookup value for the key is the stateid attribute for the customer element, we package it all in a customer template rule:

Transforming with three schemas (Sibling of Districts)

We would like to show our XSL codes in Figures 3.27- 3.31.

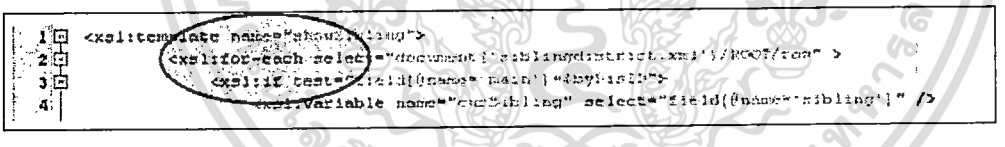


Figure 3.27: Seaching each element of sibling district

The first <xsl:for-each> , seach each element of sibling district which contained all sibling districts of a main district. For example, Figure 3.27 at line 3rd "$byDisID" is stored Ladkrabang as the main district. Then, "curSibling" will store each sibling district of Ladkrabang.

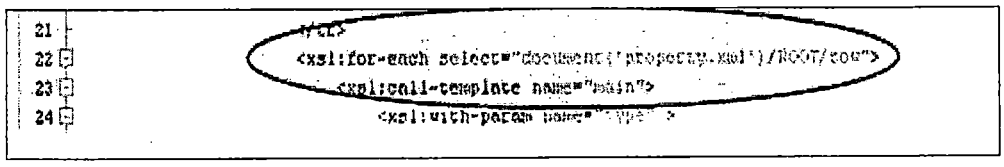


Figure 3.28: Selecting property document for calling template

From Figure 3.28, the second <xsl:for-each> at line 22nd, select property document for calling template "main" and pass a variable "curSibling" parameters at line 44th to template "main" at line 52nd for showing element of current sibling. Then, at line52nd, parameter from template "main" retrieves values from template "showsibling" is shown in Figure 3.29.

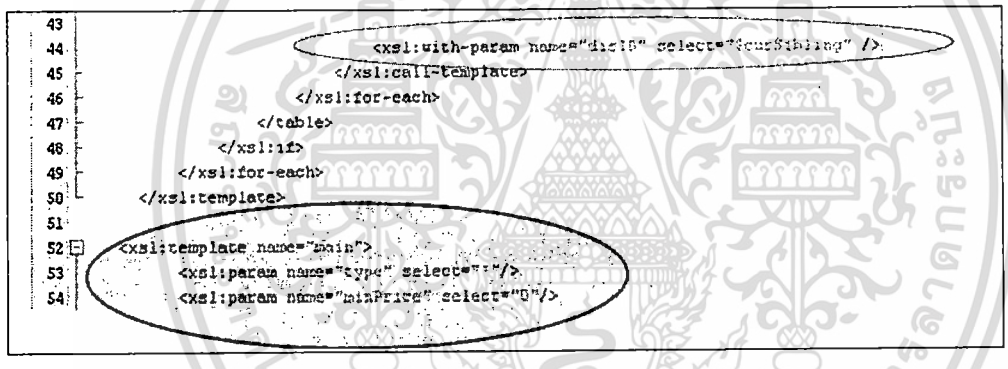


Figure 3.29: Passing a variable and showing element of current sibling

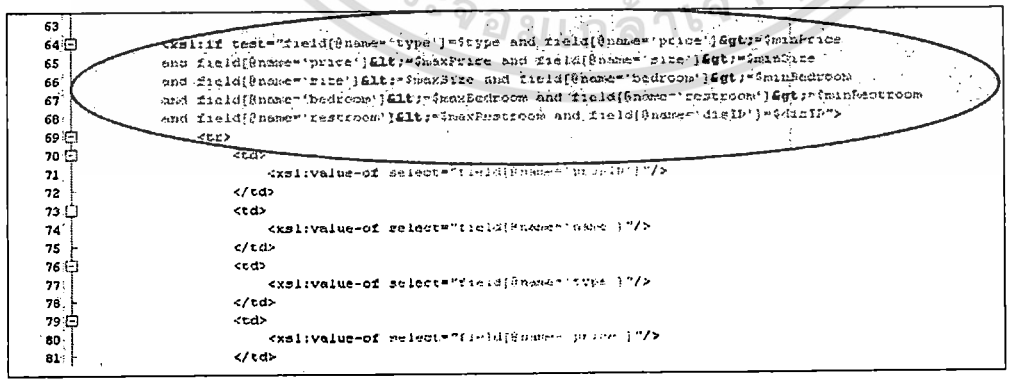


Figure 3.30: Checking if conditions with the current element

At line 64th, if all conditions match with the current element it will show on the screen. However, in the table result at colume district must be district name instead of district id is shown in Figuer 3.30. At line 145th, template "matchDistrictName" is called to query district name from district schema. Figuer 3.31 shows query district name from district schema.

```
135                 <td>
136                     <xsl:value-of select="field[@name='size']"/>
137                 </td>
138                 <td>
139                     <xsl:value-of select="field[@name='bedroom']"/>
140                 </td>
141                 <td>
142                     <xsl:value-of select="field[@name='restroom']"/>
143                 </td>
144                 <td>
145                     <xsl:call-template name="matchDistrictName">
146                         <xsl:with-param name="districtID" select="field[@name='disID']"/>
147                     </xsl:call-template>
148                 </td>
149             </tr>
150         </xsl:if>
151
152     </xsl:template>
153     <xsl:template name="...">
154         <xsl:param name="..." select="..." />
155         <xsl:for-each select="...">
156             <xsl:if test="...">
157                 <xsl:value-of select="..." />
158             </xsl:if>
159         </xsl:for-each>
160     </xsl:template>
```

Figure 3.31: Query district name from district schema