論文名稱:提升 XML 關聯式資料庫儲存模型之檢索效能－ 以 DataGuide 為路徑評估之基礎

校院系：國立暨南國際大學資訊管理研究所　　　頁數：148

畢業時間：中華民國九十五年六月　　　學位別：研究生

研究生：張哲嘉　　　指導教授：俞旭昇 博士

論文摘要

本研究將常見的 XML 查詢條件分為路徑走訪、節點型態比對、節點內容比對、多重路徑比對、節點與路徑順序比對等五種情況，並提出適當的 RDB 表格結構與系統架構，以進行查詢最佳化。我們使用 DataGuide 作為路徑走訪的依據，將 DataGuide 編號儲存於節點資訊內，以消除路徑走訪所需的表格合併。本研究將所有的節點型態儲存於單一個表格,可減少節點型態與內容比對所需的 table join。每個節點記錄上三層的節點編號，當多重路徑比對的分岔高度小於 3 時，可將 θ-join 簡化為 equal-join。針對 XML 中的順序檢索，我們以 DataGuide 與 DOM 產生節點與路徑順序，並存放於每個節點中，這使得 RDB 可以有效率的處理節點與路徑順序比對。我們的實驗結果顯示，本研究之方法在 XMark(RDB_A)上比 Edge、XParent、Monet 和 XRel 在查詢效能上分別提升 98.96 倍、57.17 倍、53.78 倍和 71.64 倍。

關鍵詞：XML 關聯式資料庫模型、DataGuide、XML 管理及檢索

Title of Thesis：Improving Query Performance for XML RDB Storage Models: A DataGuide-Based Approach for Path Evaluation

Name of Institute：National Chi Nan University Pages：148

Institute of Information Management

Graduation Time：06/2006 Degree Conferred：Master

Student Name：Che-chia Chang Advisor Name：Dr. Shiuh-Sheng Yu

ABSTRACT

We identified five types of XML query conditions, namely path evaluation, node type comparison, node content comparison, path comparison, and path ordinal comparison. We designed proper RDB tables and system architecture to boost query performance. We used a DataGuide based approach for path evaluation, which eliminates all table joins. A single table has been created to store node type and node content, which can reduce the number of table joins required for node type and node content comparison. Each node has been augmented with three fields each stores the node id of its father, grandfather, great-grandfather. This design can reduce the cost of path comparison from θ-joins to equal-joins provided the height of path comparison is less than 3. With DataGuide and DOM, path ordinal information can be generated and store in each node, which helps RDB to evaluate path ordinal comparison efficiently. Our experiment shows our approach outperforms Edge, XParent, Monet and XRel by 9896%, 5717%, 5378% and 7164% on XMark(RDB_A) respectively.

Keywords：XML RDB Storage Models, DataGuide, XML management and query

目 錄

論文摘要 .... **I**
ABSTRACT .... **II**
目 錄 .... **III**
圖 目 錄 .... **V**
表 目 錄 .... **VII**
第一章 緒論 **1**
1.1 研究背景 1
1.2 研究動機 2
1.3 研究目的、範圍與假設 3
1.4 論文架構 3
第二章 文獻探討 **4**
2.1 XML 相關知識背景 4
2.1.1 Markup 與 Metadata 4
2.1.2 SGML、XML 5
2.2 XML 資料檢索語言 12
2.2.1 Regular Path Expression 12
2.2.2 XPath 12
2.3 XML 結構索引建置 15
2.3.1 綱要淬取技術 15
2.3.2 圖形修剪技術 18
2.3.3 挖掘代表性路徑索引 19
2.4 XML 資料儲存模型 21
2.4.1 XML 文件檔案 22
2.4.2 關聯式-不需 DTD 儲存方法 22
2.4.3 關聯式-需 DTD 儲存方式 26
2.4.4 原生性(Native)資料庫儲存方式 28
2.4.5 小結 29
第三章 系統設計 **31**
3.1 引言 31
3.2 名詞定義 33
3.3 新增模組 34
3.3.1 XML 文件資料模型 34
3.3.2 DataGuide 與 Node Information 35
3.3.3 資料庫綱要設計 40
3.3.4 演算法設計 43
3.4 檢索模組 46

3.4.1 XPathCore......46
3.4.2 檢索語言轉換......48
第四章　系統實作......62
4.1 XPathCore 檢索語言剖析實作......62
4.1.1 參與者內容......63
4.1.2 參與者之間的合作方式......64
4.2 XPathCore 檢索語言轉換機制......65
4.2.1 DataGuide 重建......65
4.2.2 路徑走訪與 DataGuide 的比對......66
4.2.3 各個 Expression 的 SQL 轉換機制......67
4.2.4 小結......79
第五章　效能測試與分析......80
5.1 前言......80
5.2 實驗測試環境......82
5.3 使用的 Benchmark 簡介......82
5.3.1 Shakespeare Benchmark......82
5.3.2 XMark Benchmark......84
5.4 測試實作議題之探討......87
5.5 測試結果與分析......90
5.5.1 XML Query 與 SQL commands 的關聯性......90
5.5.2 測試 Shakespeare Benchmark......93
5.5.3 測試 XMark benchmark......100
5.5.4 小結......104
第六章　結論與未來研究方向......107
6.1 研究結果與貢獻......107
6.2 未來研究方向......108
參考文獻......109
附 錄 一......113
附 錄 二......114
附 錄 三......115
附 錄 四......122
附 錄 五......123
附 錄 六......125
附 錄 七......140
附 錄 八......147

圖 目 錄

圖 2.1-1　DOM 核心類別與介面模型圖 9
圖 2.3-1　索引建置分類圖 15
圖 2.3-2　MLB 測試資料圖形..17
圖 2.3-3　MLB 測試資料圖形的 DataGuide 17
圖 2.3-4　MLB 測試資料圖形的 schema graph 18
圖 2.4-1　XML 資料處理方法分類圖 22
圖 2.4-2　Edge schema..22
圖 2.4-3　Monet schema..23
圖 2.4-4　XParent schema 24
圖 2.4-5　XRel schema..24
圖 2.4-6　某份文件的 DTD..27
圖 2.4-7　透過 shared 所建立的 schema..27
圖 2.4-8　轉換的 DTD 圖形 27
圖 3.1-1　系統設計圖 32
圖 3.3-1　範例文件「customers.xml」的文件資料模型圖 35
圖 3.3-2　customers.xml 文件資料模型之 DataGuide 模型圖 36
圖 3.3-3　範例文件「customers.xml」的 DataGuide 與 Node Information........38
圖 3.3-4　資料庫的 E-R Model 41
圖 3.3-5　DataGuide 的資料庫表格定義 41
圖 3.3-6　Files 的資料庫表格定義 42
圖 3.3-7　Node Information 的資料庫表格定義 42
圖 3.3-8　剖析 XML 文件的虛擬碼 43
圖 3.3-9　DataGuide 類別圖 44
圖 3.3-10　DataGuide 執行的虛擬碼..45
圖 3.4-1　擴充的 XPathCore grammar..48
圖 3.4-2　EX_Q1 透過 DataGuide 與 Node Information 進行路徑比對............49
圖 3.4-3　EX_Q1 轉換後的 SQL commands 50
圖 3.4-4　EX_Q2 透過 DataGuide 與 Node Information 進行路徑比對............51
圖 3.4-5　EX_Q2 轉換後的 SQL commands 51
圖 3.4-6　EX_Q3、EX_Q4 透過 DataGuide 與 Node Information 進行路徑比對 53
圖 3.4-7　EX_Q3 轉換後的 SQL commands 53
圖 3.4-8　EX_Q4 轉換後的 SQL commands 54
圖 3.4-9　EX_Q6 轉換後的 SQL commands 55
圖 3.4-10　EX_Q7 轉換後的 SQL commands 56
圖 3.4-11　部分範例文件「customers.xml」的文件資料模型圖 56

圖 3.4-12 EX8 透過 XRel 轉換的 SQL commands..57
圖 3.4-13 EX8 透過本研究方法轉換的 SQL commands 58
圖 3.4-14 EX_Q9 轉換後的 SQL commands 58
圖 3.4-15 EX_Q10 轉換後的 SQL commands 59
圖 3.4-16 EX_Q11 走訪路徑圖 60
圖 3.4-17 EX_Q11 轉換後的 SQL commands 60
圖 4.1-1 Interpreter 結構 63
圖 4.2-1 重建 DataGuide 的虛擬碼...66
圖 4.2-2 比對路徑走訪與 DataGuide 的虛擬碼..67
圖 4.2-3 Query expression 轉換 SQL 之虛擬碼...69
圖 4.2-4 PathExpr 轉換 SQL 之虛擬碼..71
圖 4.2-5 RegularExpr 轉換 SQL 之虛擬碼 75
圖 4.2-6 Comparison 的轉換 SQL 之虛擬碼 78
圖 5.1-1 實驗測試圖 80
圖 5.5-1 比較兩方法的折線圖(省略 Q6)..96

表 目 錄

表格 2.1-1　SGML 的特性、內容、目的與方向 5
表格 2.1-2　SGML 優點與限制 6
表格 2.1-3　DOM 介面節點說明 10
表格 2.1-4　DOM 與 SAX 比較整理表 11
表格 2.2-1　與路徑表示相關的 expression 14
表格 2.2-2　較為常見的萬用符號 14
表格 2.3-1　資料節點與綱要節點 18
表格 2.3-2　person 與其子元素可能組合表 19
表格 2.3-3　檢索建置方法比較表 20
表格 2.4-1　四種不需 DTD 儲存方法比較 25
表格 2.4-2　八種儲存方法整理 30
表格 3.3-1　Node Information 的名稱、說明、與使用的檢索情況之整理 40
表格 3.4-1　XPathCore grammar 代表的語意 48
表格 5.3-1　測試資料 Shakespeare 詳細介紹 83
表格 5.3-2　8 條檢索條件表示式與其檢索涵義 84
表格 5.3-3　測試資料 XMark 詳細介紹 85
表格 5.4-1　XRel 資料庫表格定義 87
表格 5.4-2　XRel 資料庫索引建置 88
表格 5.4-3　本研究方法資料庫索引建置 89
表格 5.5-1　針對 Shakespeare Benchmark 建立資料庫欄位數量的比較 94
表格 5.5-2　針對 Shakespeare Benchmark 在資料庫建立分析與比較 95
表格 5.5-3　檢索效能比對(單位為 毫秒) 96
表格 5.5-4　兩種方法的 SQL commands 98
表格 5.5-5　XMark RDB_A 檢索效能分析表(單位為 秒，‘-’表示檢索失敗) 101
表格 5.5-6　XMark RDB_B 檢索效能分析表(單位為 秒，‘-’表示檢索失敗) 102
表格 5.5-7　本研究與文獻[16]中所有方法測試效能最高的 Shared+相比(單位：秒) 106

第一章　緒論

1.1　研究背景

二十世紀，似乎是一夜間，全球資訊網(WWW)將世界上大多數的電腦連接在一起，除了透過物理網路外，亦透過公用協定來進行訊息交換。人們突破了時空上限制，在虛擬世界中傳遞實質資訊，也因此帶動了電子商務(electronic commerce)與電子資料交換(electronic data interchange)發展。面對資訊交換的需求，各種交換標準成為重要的探討議題。XML 提供良好自我定義與描述，適時替網際網路資料的多變性與非結構性之限制，提供一個良好的解決方案。

然而面對快速膨脹且龐大的資料量，許多 XML 儲存模型紛紛被提出以管理 XML 資料，目前最為廣泛的三大類資料儲存模型：XML 文件檔案、關聯式儲存模式與原生性資料庫儲存模式。XML 文件檔案缺乏安全性與索引機制，無法看待為一個完整的資料庫儲存方式。而原生性資料庫興起時間並非很長，並沒有如關聯式資料庫系統已有成熟且強大的技術與研究支援，和關聯式儲存模型之間也存在著轉換上問題。

在關聯式儲存模型中，資料庫表格綱要則必須事先定義好。研究[37][52]提出可以使用 XML DTD 進行資料庫表格綱要設計，當 DTD 結構越複雜時，資料庫表格綱要也相對越複雜化；另一方面，由於 XML 文件皆能轉換為樹狀結構，研究[14][35][21]提出將文件結構與節點位置作為資料庫表格綱要設計。這一類方法不需要使用到 DTD，且所有處理的 XML 文件皆擁有同一份結構綱要。

除了管理 XML 儲存模型外，目前也有許多 XML 檢索語言[3][11][8][5][6]被提出。其中 XPath 為 XML 路徑表示式，XQuery 為 XML 檢索語言之主流。在檢索效能提升方面，除了儲存模型本身設計理念外，XML 結構上的索引建置亦為一個探討的議題。索引機制可分為三大類：綱要淬取技術[29][32]、圖形修剪技術[40][34]、挖掘代表性路徑技術[28][26]。

1.2 研究動機

在 XML 儲存模式中以關聯式資料庫儲存模型最為成熟，其中需要使用文件格式定義 (DTD)的研究方法面臨 DTD 結構越複雜時，資料庫表格綱要也相對的越複雜化。故此類方法，必須克服在面臨缺少文件結構資訊的情況下，依舊能快速完成檢索。

進一步探討，一份 well-formed 的 XML 文件可以拆解為文件結構資訊與節點內容資訊，故 XML 檢索包含了路徑走訪與節點內容檢索。路徑走訪通常以 Regular Path Expression 表示，它促使檢索語言呈現多樣化及高彈性，但相對也提高路徑走訪時所花費的時間成本。在以 RDBMS 為 XML 儲存模式領域中(不需 DTD)，以儲存邊資訊的方法(Edge、XParent、Monet 為代表)皆需要進行大量的表格合併運算以完成路徑走訪；而以儲存節點資訊的方法(XRel 為代表)雖以唯一邊資訊作為路徑走訪依據，企圖避開前者儲存邊資訊的方法所面臨的路徑走訪所產生的表格合併，但卻還有下列缺點：

- 路徑走訪以字串比對：針對檢索時路徑走訪部份採取和 Path 表格中的路徑資訊做字串比對，並採取 like(%)進行層級未知情況的處理。以字串比對並非一個有效率的方法。
- 表格之間的關聯：代表路徑的 Path 表格需要和其它種類節點表格(Element、Text、Attribute)進行 equal-join 以達到路徑走訪，*n* 個節點的檢索並需進行 *n* 次的 equal-join。
- 相近節點關係亦需比對：某節點與其子節點亦要進行 θ-join。

由上述可知，在以 RDBMS 為儲存模式且不使用 DTD 的研究方法中，皆有各自的優點與研究限制，文獻[16]中的 benchmark 測試數據更可證實在此領域並非有任何一個研究方法能完全勝任每一項檢索情況。

1.3 研究目的、範圍與假設

結構上 XML 文件與 RDB 表格最大的差別在於前者為階層性結構而後者為扁平式結構。如何在 XML 文件與 RDB 表格之間進行拆解與重組，一直攸關於 XML 檢索效能。為了解決上述以 RDBMS 為儲存結構、不使用 DTD 的研究限制，本研究分析 XML Query 與 SQL commands 之間的關聯性(請參考 5.5.1 節)，針對各種常見的檢索情況進行分析以找出較佳的策略。

我們透過上述分析進而設計出本研究的研究方法與系統架構，提供檢索語言與系統實作的演算法，最後再透過兩種不同類型的 XML benchmarks 實驗測試，以評估本研究當初的設計是否正確，檢索效能是否有顯著提升。

1.4 論文架構

- 第一章為背景與動機，研究目的、範圍與動機。在背景中淺談 XML 檢索語言、XML 結構索引建置與 XML 資料儲存模型。並針對 XML 資料儲存模型中以「RDBMS 為儲存模型、不使用 DTD」的研究方法，分析其研究優點與限制，以產生本研究動機。
- 第二章介紹與整理文獻探討。包含了 XML 知識背景、XML 檢索語言、XML 結構索引建置與 XML 資料儲存模型。
- 第三章為系統設計。將系統分為新增模組與檢索模組進行探討，並針對模組詳細說明設計理念與運作流程、資料庫綱要規劃與演算法設計。
- 第四章為系統實作。探討檢索模組的實作。
- 第五章為效能測試與分析。進行一系列效能上的實驗測試並且針對實驗結果進行數據報告與效能分析。
- 第六章為結論與未來研究方向。

第二章　文獻探討

為了配合本研究需求，我們將進行相關文獻探討。本章一共分為四節，第一節介紹有關 XML 的知識背景，包括 markup language 與 metadata、Namespaces、DTD(Document Type Definition)、XML Schema 及 XML 所使用的 API。第二節介紹代表 XML 資料檢索語言規則的 Regular Path Expression 與 XPath 檢索語言。第三節描述針對 XML 檢索過程所建立的索引機制，包括了網要淬取技術索引、圖形修剪技術索引、挖掘代表性路徑技術索引三大類索引機制。第四節探討 XML 資料儲存模式，包含目前最為廣泛的三大類資料儲存技術：XML 文件檔案、關聯式資料庫儲存模式與原生性資料庫儲存模式，並且針對這三大類共八種儲存方法進行多面性的整理與比較。

2.1　XML 相關知識背景

2.1.1　Markup 與 Metadata

文獻[53]中所定義 markup 為用標籤來加強或突顯文字在文件中的重要性或具有的特別意義。在機器之間的交換文件中，除了定義標籤語法外，亦須定義標籤意義。以 HTML 標記語言為例，標籤以“＜＞”表示，對於標籤中字彙則具有本身的意義與語法，如“＜TR＞”即為用以呈現表格中的「列」的效果。

全球資訊網(World Wide Web)發明人-伯納斯李(Tim Berners-Lee)對 Metadata 一詞的解釋為[42]：

"Metadata"指描述資料本身特性的資料，在全球資訊網上，它指的是讓機器看得懂的、描述資訊內容的所有相關資訊，例如描述資訊的所有權、著作權、流通權、隱私維護政策等，這可以幫助上網者更方便使用資訊。

2.1.2 SGML、XML

2.1.2.1 SGML

SGML[39] ，全名為 Standard Generalized Markup Language，是國際標準組織（ISO）於 1986 年所通過的國際標準。它用來描述一份文件中文字意義與文件所代表的結構，簡單來說就是文字編寫與編排的標準，非針對某一特定的應用軟體所設計的。SGML 目前有許多的應用層面，包括國防、航空、製造、物流與博物館…等領域。表格 2.1-1 為 SGML 的特性、內容、目的與方向，表格 2.1-2 為 SGML 優點與限制。

表格 2.1-1 SGML 的特性、內容、目的與方向

特性	內容、目的與方向
文件結構性	SGML 強烈地強調文件的結構性，因此需要使用文件格式定義（Document Type Definition）來制定文件的結構，機器便可使用文件的結構資訊與內容資訊以進行交換與分析之應用。
Metadata	SGML 可視為描述資料本身特性的資料。
Markup Language	利用編碼來代表文字涵義，編碼以標籤形式出現，類似 HTML。

表格 2.1-2 SGML 優點與限制

優點	限制
➢ 語言格式為國際標準,並非私人團體所擁有。 ➢ 文件高度結構化與標籤制定。 ➢ 文件的結構與資訊內容可分離,透過不同的樣板可以塑造出多樣的呈現方式。 ➢ 在平台與系統可擁有獨立性,不需專門的工具才能解讀。	➢ SGML 設計語法過於複雜，不易上手。 ➢ 由於語法過於複雜,亦使應用系統難以開發。 ➢ 由於 SGML 複雜難懂，文件需有 DTD 來描述文件結構。且使用未普及化,缺乏瀏覽器廠商的實作支援。

2.1.2.2 XML

XML[13][19][2][15]，全名為 eXtensible Markup Language，為 W3C 於 1996 年所制定的標準。XML 依據 SGML 規範簡化而來，其優點如下：

- ➢ 簡明。由 SGML 為出發點，去除 SGML 複雜定義缺點，將 SGML 內 500 項複雜且不適用於 Web 部分去除並簡化成 26 項規定，使 XML 較為清晰易懂且方便程式開發。
- ➢ 顯示格式與資料格式分離。使用 DTD 來描述文件的資料格式，和使用擴展樣式語言(XSL, eXtensible Stylesheet Language)加以呈現，使 XML 可以將同一份資料應用在不同的設備上加以瀏覽，如 PDA 或無線設備…等。
- ➢ 自我定義、自我描述能力的 XML 標籤。可用來描述被標記的資料，經由 XML 標籤作為搜尋索引，可以提供資料上更為精確、效率的搜尋。
- ➢ 嚴謹的資料結構與語法。XML 定義了相關的結構標準，符合此標準的 XML 文件為 well-formed 文件，並可透過剖析器檢查，以確保任何剖析器針對同一份 XML 文件所得到的元素名稱、順序、階層都為相同。

- 應用性。XML 本身為一個 Metadata，提供使用者自行定義專業領域的專屬標籤。如應用 XML 重新定義 HTML 的 XHTML[47]；應用於建築工程領域之 XML 標準的 aecXML [4]。

- XML 基本語法

依據 XML1.0[17]規定，將 XML 基本語法整理如下：

- 宣告 XML 必須出現在文件第一行，以小寫 xml 宣告並設定 Version 屬性。
- XML 文件的根節點(root)必須也只能有一個。
- 所有標籤進行巢狀排列。
- 所有標籤必須以開始和結束標籤成對出現，除了空標籤外。
- 內容為空的標籤結尾須加上「/」。
- 標籤名稱與屬性須合法，大小寫視為不同。
- 屬性值須以「" "」或「' '」括起來。
- 特殊字元須按照規定撰寫。

- Namespaces

名稱空間裡的所有標籤名稱均具有唯一性。在 XML 擁有自我描述與自我定義的特性下，標籤名稱可以由使用者定義並具有特殊意義，由人來看這些標籤是不會有問題，但若是由電腦來看，就有可能發生問題。例如一份由學校計算機中心製作的學生資料表裡的某標籤名稱為 address，代表的是該學生的電子信箱；而註冊組製作的學生資料表裡的某標籤名稱為 address，代表卻是學生的通訊住址，若將兩個 XML 文件結合在一起，所面臨的是如何解決標籤名稱衝突的問題！為了因應及確保標籤名稱的唯一性，使用 Namespaces 能解決上述的問題。

- DTD 與 XML Schema

使用者可以使用 DTD 與 XML Schema 來檢查 XML 文件格式是否正確，文件格式為何需要驗證？以 B2B 電子商務應用來看，每一筆訂單皆需要進行正確性檢驗以達到資料完整性與網站安全性，若將訂單規範成 XML 格式時，資訊交

換上即可利用剖析器做基本的格式確認。一般來說，驗證一份 XML 文件通常確認下列四點：

- 標籤（元素）與屬性都是使用規範中的名稱與內容。
- 屬性是否可屬於或應該屬於某一個標籤。
- 標籤的順序性與數量是否合法。
- 標籤與屬性的資料型態是否合法。

DTD 限制如下：

- DTD 使用一套屬於自己的語法，如「<!ELEMENT…」，和 XML 的語法不同，XML 使用者必須學習另一套語法。
- DTD 針對資料型態的定義中，只有「#PCDATA」以描述字串資料型態。
- 僅提供 0 次以上(*)、1 次以上(+)與 0 或 1 次(?) 三種重覆敘述定義。

而由微軟所發展的驗證語法 XML Schema[18]，在 2001 年已經成為 W3C 標準，其目的即在取代 DTD 作為下一代的驗證語法且擁有下列的特點：

- XML Schema 本身即為一個 well-formed XML 文件
- XML Schema 支援較多的資料型態，包括字串（String）、整數（Integer）、布林值（Boolean）、浮點數（Float）、日期（Date）…等。
- XML Schema 針對結構上提供 minOccur 與 maxOccur 兩個屬性以達到控制子元素出現次數的目的。
- XML Schema 提供命名空間，具有擴充性。
- XML Schema 採取開放式的架構，不要求所有元素與屬性必須先被宣告再使用。
- XML Schema 可以針對個別元素指定不同的 XML Schema 作為驗證語法。相較於 DTD 僅作為整份文件所使用。
- 本身為 well formed 的 XML Schema，故可以透過如 DOM 的程式介面，存取內部屬性與元素以達到修改目的。

■ DOM

DOM[45]，全名為 Document Object Model，是一套專為 well formed 和 valid 文件所設計的 API，它同時定義了這些文件的邏輯架構與存取操作方法。

DOM 是 W3C 制定的標準，不但能用來處理 XML，也能處理 HTML 或其它相關的標示語言，其目標是提供一個可以在各種程式語言、各種應用程式或作業系統都可通用的 API。所以 DOM 具有優越的相容性，不論是 C++、COM、或 Java 等都可以使用相同的 DOM 介面。圖 2.1-1 為 DOM 核心類別與可運用介面之模型，表格 2.1-3 為 DOM 介面節點說明。

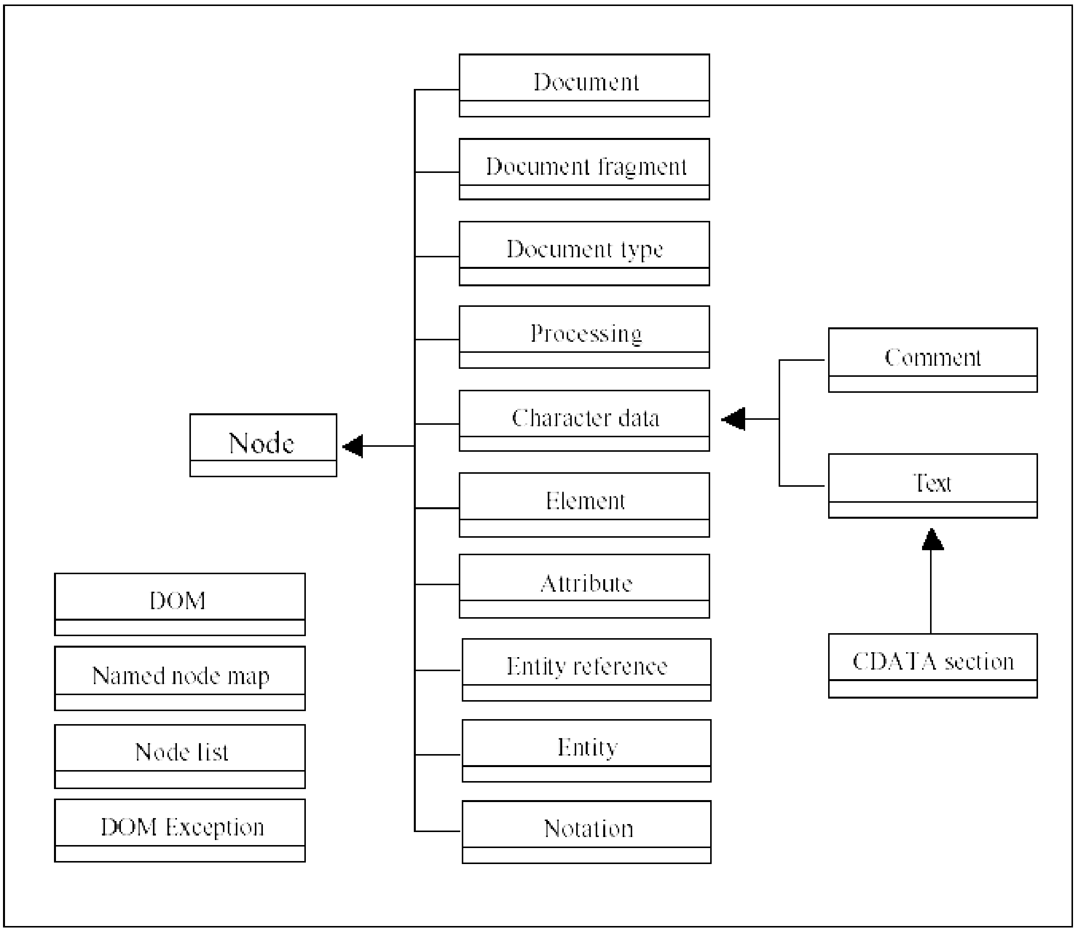


圖 2.1-1　DOM 核心類別與介面模型圖

表格 2.1-3　DOM 介面節點說明

Interface	Node Name	Node Value	Attributes
Attr	name of attribute	value of attribute	null
CDATASection	"#cdata-section"	content of the CDATA Section	null
Comment	"#comment"	content of the comment	null
Document	"#document"	null	null
Document Fragment	"#document-fragment"	null	null
Document Type	document type name	null	null
Element	tag name	null	Named NodeMap
Entity	entity name	null	null
Entity Reference	name of entity referenced	null	null
Notation	notation name	null	null
Processing Instruction	target	entire content excluding the target	null
Text	"text"	content of the text	null

■　DOM 與 SAX 之比較

用來處理 XML 或 SGML 的 API 可以分為兩大類：以樹為基礎 API(Tree-based APIs)或以事件為基礎的 API(Event-based APIs)。

在以事件為基礎的 API 在剖析 XML 文件時，會將所發生的事件以 callback 機制傳回應用程式，例如一個元素的開始與結束。使用者所需要的就是填寫遇到什麼事件會有什麼反應，故完全不需要花費到額外的記憶體，例如我們可以指定元素開始的事件會去 callback 圖形介面以產生一個按鈕。Sun 所提出的 SAX 就是一個以事件為基礎的 API。

以樹為基礎 API 在剖析 XML 文件時會將文件轉換成一個自訂的樹狀結構，利用 “樹化(Tree Structure)”的 XML 文件進行後續的操作，如新增或刪除元素。

W3C 所制定的 DOM，就是致力發展一套以樹為基礎的處理 XML 與 HTML 文件的 API。

根據上述的簡述，我們以 API 的「處理方式」、「記憶體使用」、「所需記憶體」、與「評估環境」進行整理與比較：

表格 2.1-4　DOM 與 SAX 比較整理表

API 比較	DOM	SAX
規格制定	W3C（The World Wide Web Consortium）	Xml-dev mailing list（XML 相關社群）
處理方式	以樹狀結構為基礎(Tree-based)	以事件為基礎(Event-based)
記憶體使用	將整棵樹放入記憶體	將所需要到的事件放入記憶體
所需記憶體	大	小
評估環境	➢ 須結構性地修改 XML 文件時。 ➢ 須將記憶體中的文件分享給其他應用程式時。 ➢ 當解析的 XML 文件量不是很大時。 ➢ 當應用程式想要在驗證完後才開始進行處理時。	➢ 程式需要高效能，且不能使用過多的記憶體時。 ➢ XML 文件不需要識別其結構時。當 XML 文件代表一堆屬性及其值的配對時，使用 SAX 將會非常有效率。

2.2 XML 資料檢索語言

目前有許多 XML 檢索語言相繼被提出，如 Lorel[3]、XML-QL[11]、XML-GL[8]、XPath[5]、XQuery[6]…等，而在這些所提出的方法中以 XQuery 為 XML 檢索語言之主流。在文獻[22]中針對檢索語言有一些深入探討。這小節我們介紹代表 XML 資料檢索語言規範的 Regular Path Expression 和與本研究有關的檢索語言－ XPath。

2.2.1 Regular Path Expression

目前大部分 XML 檢索語言皆支援 RPE (Regular Path Expression) [27][44]，且支援 RPE 的檢索語言或檢索技術會被視為較為有價值的檢索方式。RPE 是一種表示 XML 資料檢索語言的規則，利用一些簡單易懂的文字或符號和 like-SQL 語言的方法來表示檢索的內容與 XML 階層性資料結構。RPE 的使用不需知道檢索文件的結構全貌，只要針對所要檢索條件，利用 RPE 指定的符號或特殊字元，如「?」、「.」、「*」…等符號來與檢索條件結合，就可以輕易地表達所要尋找的 XML 資料。

以檢索子句範例：Division/*/name 進行說明，此範例由節點名稱 Division 走訪至節點名稱 name，其兩節點間透過一層未指定的節點名稱互相聯接。支援 RPE 的 XML 檢索語言有很多，如 Lorel、XPath、XQuery、XML-QL…等，當然這些檢索語言各有不同的特色與支援的程度範圍。

2.2.2 XPath

XPath[48]為 W3C Recommendation 中一環，它提供一種方法來針對 XML 文件定址，並且也提供一些操作字串、數字和布林數值的基本函式。在許多 XML 相關領域中皆有使用到 XPath，如 XSLT[50]與 XPointer[49]。XPath 將 XML 文件視為節點樹模型，這些節點依序是根節點(root)、元素節點(element)、文字節點

(text)、屬性節點(attribute)、命名空間節點(namespace)、處理函式節點(processing-instruction)和註解節點(comment)。下面我們簡略地介紹 XPath 相關語法，其詳細語法請參考 XPath 規格書[48]。

2.2.2.1 XPath 語法簡介

在 UNIX、Windows 作業系統都擁有檔案路徑的符號，用來指出一個檔案與目錄的位址，如："C:\Program Files\Real\RealOne Player\realplay.exe"。在 XPath 中，也定義了路徑符號以表示 XML 文件中某個特定的節點集合，就猶如檔案系統，此路徑可稱為位址路徑（Location path）。

如/Division/Player/Name 即為一個簡單的 XPath 表示式，會挑選出節點名稱為 Name 的元素，過程從 Division 元素下的子元素中挑選出 Player 元素，再從 Player 元素下的子元素中挑選出 Name 元素。在 XPath 中每一個被斜線分開的元素，可稱為一個位置步驟（Location step），如 Division、Player 與 Name。

XPath 與一般檔案路徑不同處在於，XPath 指的是一個節點的集合，如上述表示式表示 Player 元素下所找的元素可能為好幾個 Name 而不是單一元素，如果是單一元素，也是因為該節點集合只有單一元素。

XPath 包含了超過 100 多個內建的函數，這些函數可以包括字串（string）、數字（numeric）、日期時間比較（date and time comparison）、節點集合（node）、布林值（Boolean）...等。表格 2.2-1 擷取 XPath grammar 中較為常見與路徑表示相關的 expression，表格 2.2-2 截取較為常見的萬用符號。

表格 2.2-1　與路徑表示相關的 expression

path expressions	
nodename	Selects all child nodes of the node
/	Selects from the root node
//	Selects nodes in the document from the current node that matches the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

表格 2.2-2　較為常見的萬用符號

Wildcard	Description
*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

2.3 XML 結構索引建置

這一節我們針對可提升 XML 檢索過程效率而建置的索引進行介紹。我們將索引機制分為三大類說明[43]，分別為綱要淬取技術、圖形修剪技術、挖掘代表性路徑技術。其中第三類為針對文件的「部分路徑」進行索引建置，其餘皆對文件的「完全路徑」進行索引建置。

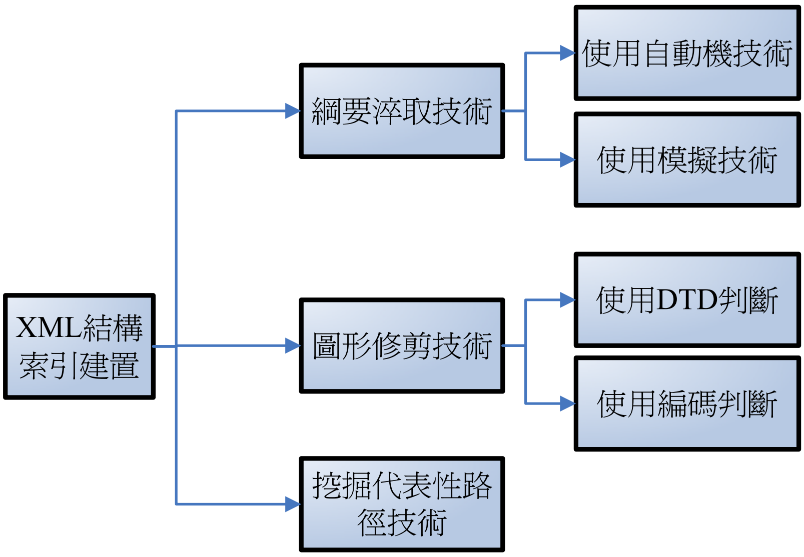


圖 2.3-1　索引建置分類圖

2.3.1 綱要淬取技術

XML 文件的非結構性或半結構性特色，往往需要 DTD 來規範其文件的結構與大小範圍，如果缺乏 DTD 的支援，檢索系統必須將 XML 文件以樹狀節點路徑全部拜訪過一次，此方法非常沒有效率。因此為了避免對 DTD 等文件型態定義的過分依賴，許多研究均對於 XML 文件提出綱要淬取技術。此研究技術可分為兩大群：使用自動機(automata)技術與使用模擬(simulations)技術。

2.3.1.1 使用自動機進行綱要淬取

文獻[29][32]中提出了使用自動機來進行綱要淬取，DataGuide 是針對半結構

化文件(semi-structure document)所建置的結構化索引機制，首次使用於 Lore XML DBMS[31]中。DataGuide 運作方式為以相同名稱的路徑合併為一條路徑，故可將 DataGuide 視為樹狀結構壓縮後的精簡結構，以提供相同路徑的比對。舉例說明，假設目前有個精簡的 MLB(Major League Baseball)資料庫，我們針對此資料進行圖形表示，如圖 2.3-2。將此圖形進行自動機程序，將節點視為狀況(state)處理，而邊視為轉換(transitions)機制。因此，有兩個轉換標籤－National 均出於狀況 1，一個轉換將會走至狀況 2；而另一個則走至狀況 14。圖 2.3-3 為圖 2.3-2 的 DataGuide。我們可以知道 DataGuide 是一個決定性有限狀態機(deterministic finite state automaton)，且它的尺寸會小於原先的來源資料庫，所以我們可以快速地透過它來進行檢索，因為它提供了路徑上的索引。請關注由 RPE 表示的檢索條件“MLB.National.player.nickname”，若不採用 DataGuide，檢索路徑勢必走訪如圖 2.3-2 中粗線部分；若使用 DataGuide，則如圖 2.3-3 中粗線部分。DataGuide 針對每一個節點均記錄對應原始文件的節點編號，稱為「target set」。故我們可以由 DataGuide 節點編號 6 的 target set 找到所對應的原始節點編號節點為 7 與 18。

然而 DataGuide 僅能應用於單一路徑規則表示法，它無法應用於較為複雜的檢索條件，多重路徑表示式：q’(b):-a(National)b, b(player.name.“Park”)c，這條表示式檢索涵義為找出球員名稱為 Park 所待的球隊名稱。這條檢索式牽扯出兩條路徑走訪，一條為 MLB.National；另一條為 National.player.name。前條路徑依據 DataGuide 節點編號 2 的 target set 對應至原始文件節點 2 與 14；另一條則為 8、12、19、22。可是我們卻無法由 DataGuide 中知道檢索答案，因為在 DataGuide 中並沒有儲存文件節點之間的關係資訊。

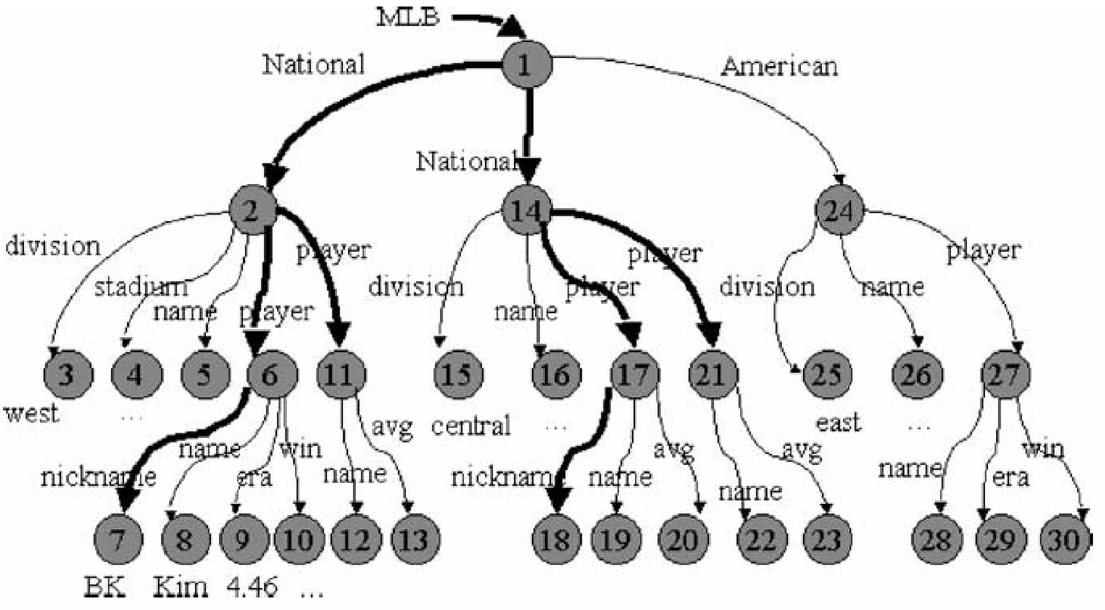


圖 2.3-2　MLB 測試資料圖形

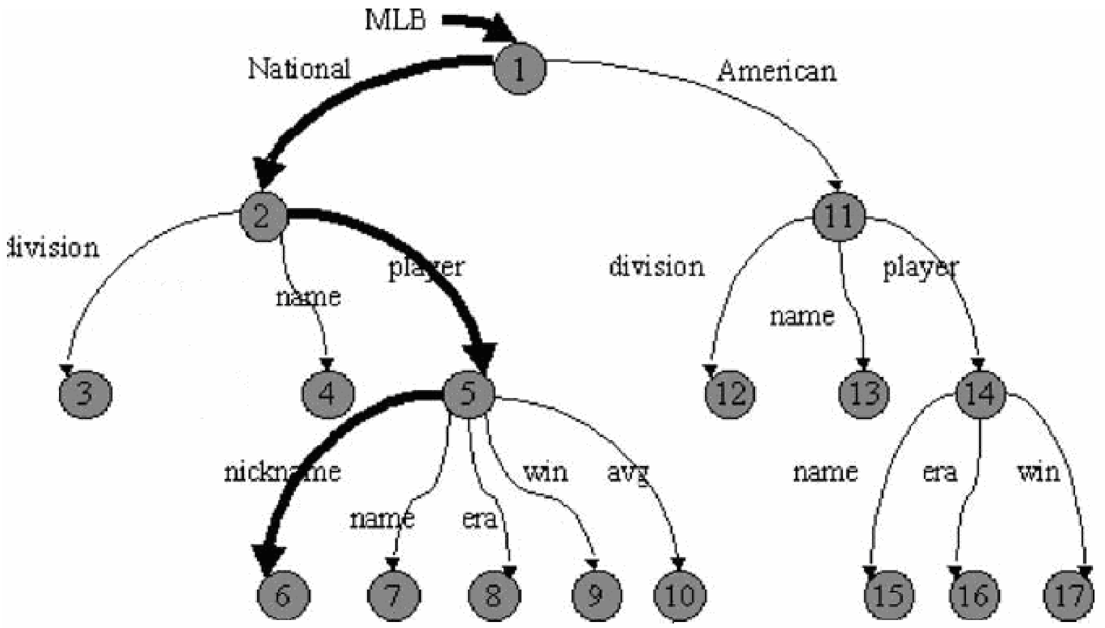


圖 2.3-3　MLB 測試資料圖形的 DataGuide

2.3.1.2 使用模擬技術進行綱要淬取

藉由建立原始文件圖形所對應的綱要圖形(schema graph)，我們能縮減在檢索過程中路徑走訪範圍。由於介於半結構化文件(或來源文件)與綱要圖形中，故此模擬技術需要可擴展性的需求，主要需求如下：

➢ 在原始文件中的 root 與建立的綱要圖形都需要出現於模擬的過程中。

➢ 由原始文件中，節點 x 到節點 y 之間邊，應該能由萬用字元或其他包含此邊的邊來替代。

圖 2.3-4 為圖 2.3-2 的綱要圖，表格 2.3-1 其記錄了兩者之間的模擬關係。

表格 2.3-1　資料節點與綱要節點

資料節點	綱要節點
1	Root
2,14	N-term
24	A-term
6,11,17,21	N-player
27	A-player
3,4,5,7,8,9,10,12,13,15,16,18,19,20,22,23,25,26,28,29,30	String

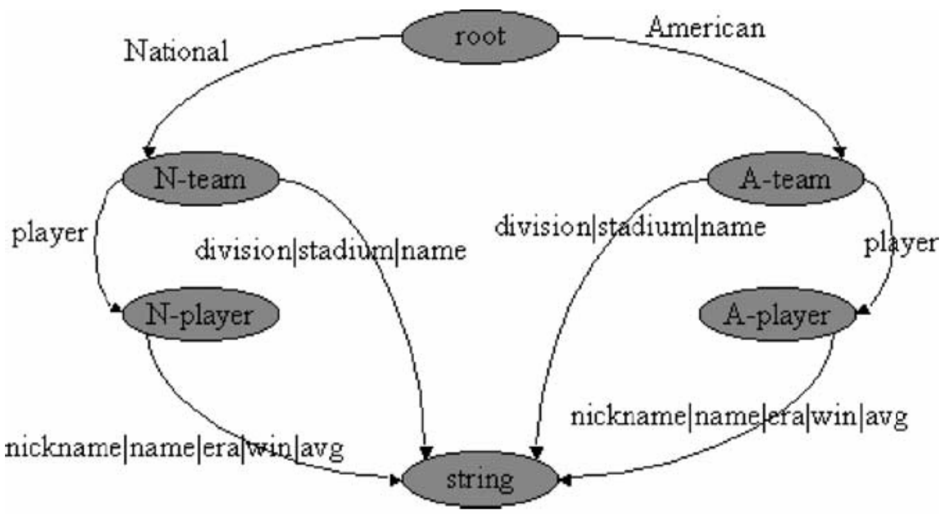


圖 2.3-4　MLB 測試資料圖形的 schema graph

2.3.2　圖形修剪技術

圖形修剪技術是將一些格外的資訊藏在文件節點上，檢索進行時可以利用這些資訊來判斷路徑是否須繼續往下走訪。以 NodeInfo[34][40]說明，假設某原始文件擁有部分 DTD 如下：

```
<!ELEMENT person (name, address, e-mail*,(school|company))>
```

我們可以分析出在元素 person 下，必然會有 name 與 address 子元素。其中

也包括 0 個以上的 e-mail 子元素，與兩者選其一的 school 和 company 子元素，故我們能以表格 2.3-2 說明 person 元素其子節點可能組合(由於 name 與 address 子節點為必要，不給予記錄)：

表格 2.3-2　person 與其子元素可能組合表

組合情況	組合值
1	{e-mail; school}
2	{e-mail; company }
3	{school}
4	{company}

假設擁有某一個 person 元素，其下面所接的子元素名為 name、address、與 school，則此 person 元素為狀況 3。那麼針對想要檢索 person 元素所接的子元素為 e-mail 且有 school 子元素，根據隱含於 person 元素上的資訊，此時則毋須針對此 person 元素下的路徑進行走訪。

其他相關研究中，亦有將 XML 文件節點加以編碼並運用於檢索過程，做為檢索過濾的條件。透過此方法能在路徑走訪的過程中，先經過編碼比對，來判斷是否要放棄目前節點下的路徑走訪以裁剪不必要的走訪，以文獻[30]為代表。

2.3.3　挖掘代表性路徑索引

以完全路徑來建立索引雖然可以全面性的因應任何檢索，但是卻需要花費大量的索引建置時間成本與面臨部分資訊的遺失問題。某些研究提出將資料勘查技術應用於 XML 索引建置中，選擇重要且具代表性的路徑以避免全面性的索引建置和路徑檢索(是優點也是限制)。此類研究以[28][26]為主要代表，[28]所挖掘的是一個連續項目集（sequential patterns），[26]亦進行連續路徑挖掘，而路徑是由使用者輸入之規則表示法的路徑。[7][33]以挖掘文件結構，將項目重新定義，以挖掘出文件之重要的綱要；[34]對文件節點進行分類以提升存取的效率。

最後我們將上述三大類索引建置方法，以「是否需要 DTD」、「所花費的建置時間成本」、「索引空間成本」與「是否支援多重路徑索引」四構面加以比較

[43]，以表格 2.3-3 表示之。由於挖掘代表性路徑索引大類是針對部分路徑進行索引建置，並不納入比較範圍內。表中我們假設文件資料圖形擁有 n 個節點與 m 條邊；而 k 與 t 則表示所使用演算法中的常數。

表格 2.3-3　檢索建置方法比較表

索引建置方法	是否需要 DTD 使用	索引建置時間成本	索引空間成本	是否支援單一路徑	是否支援多從路徑
自動機(DataGuide)	否	mlog(n)	線性	是	否
模擬技術	否	mlog(n)	線性	是	否
圖形修剪(NodeInfo)	是	log(k.n)	log(t.n)	是	是

2.4 XML 資料儲存模型

一般來說，XML 資料儲存模型可分為文字檔案架構、關聯式資料庫系統架構、與針對 XML 文件設計的原生性 XML 資料庫引擎架構三大類。

當 XML 文件儲存至關聯性系統時，資料庫表格綱要則必須事先定義好。一些研究提出可以使用 XML DTD 中元素的資訊進行資料庫表格綱要設計，當 DTD 結構越複雜時，資料庫表格綱要也相對的越複雜化，我們稱這類方法為「需 DTD 的關聯式資料庫儲存方法」；另一方面，由於 XML 文件皆能轉換為樹狀結構，有某些研究提出將文件結構與節點位置作為資料庫表格綱要設計。這一類的方法不需要使用到 DTD 的資訊，且處理的所有 XML 文件皆擁有同一份結構綱要，我們稱此類方法為「不需 DTD 的關聯式資料庫儲存方法」。XML 文件亦可使用此方法存放至物件導向式資料庫系統內。

原生性 XML 資料庫是近來非常熱門的資料處理方法，在儲存與檢索方面皆有許多有趣的議題。使用者可以使用 DOM 技術對 XML 文件進行模組化，在放置原生性資料庫引擎中。此類方法在文件儲存和檢索時，對文件的 DTD 並非必要要求，但通常都會使用 DTD 提昇檢索速度。圖 2.4-1 為 XML 資料處理方法分類圖。

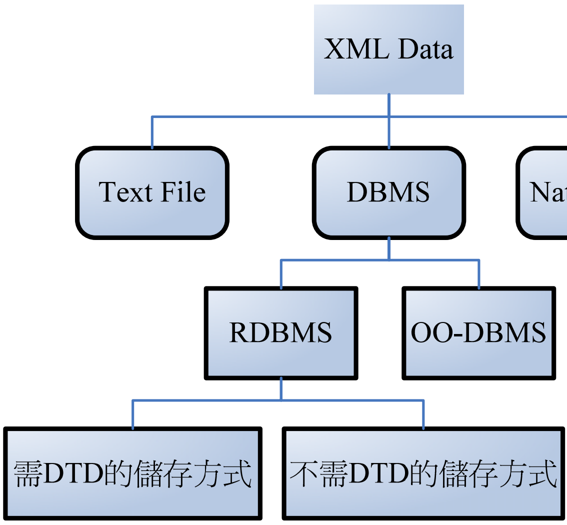


圖 2.4-1　XML 資料處理方法分類圖

基於本研究研究範圍，我們針對 RDBMS 中「不需 DTD 的儲存方法」做詳細介紹，針對其他的儲存方法，我們也會一併加以簡略介紹。

2.4.1　XML 文件檔案

XML 文件結構可視為一個樹狀結構，使用者可以應用 XML 的檢索語言，如 XPath 或是 DOM API 處理技術進行檢索。由於此檔案處理方式缺乏安全性與對檢索的索引建立等機制，故無法看待為一個完整的資料庫儲存方式。

2.4.2　關聯式-不需 DTD 儲存方法

在這類方法中我們選擇四種主要的研究方法進行介紹，分別為 Edge、Monet、XParent 和 XRel。

2.4.2.1 Edge

Edge[14]以儲存 XML 文件樹狀結構上所有邊的資訊(LabelPath)於關聯式資料庫表格內，此表格名稱為 Edge，其表格綱要如下：

Edge(Source, Target, Ordinal, Label, Flag, Value)

圖 2.4-2　Edge schema

XML 文件結構中的邊，由起始節點編號與終端節點編號所定義，分別以屬性 Source 與屬性 Target 表達。屬性 Label 記錄這個邊的名稱為何。屬性 Ordinal 記錄這個邊位於所有兄弟邊中所排列的順序。屬性 Flag 則表示邊的終端節點是否為葉節點(Atomic node)或是擁有子元素節點的節點(Complex ndoe)。屬性 Value 則記錄邊的終端節點資料內容值。

這是一種十分直覺的儲存方式，將 XML 文件內所有邊的資訊均記錄到資料庫表格內，並且記錄每一條邊的起始與終端節點編號、邊的順序與其他相關的資訊。此方法的限制在於對於路徑走訪時，需花費大量時間成本於表格合併(equal-join)上。

2.4.2.2 Monet

Monet[35]可視為 Edge 方法的變形，它將所有的邊依據可能的標籤路徑(LabelPath)情況進行分類，並且將 XML 中節點內容值部份(CDATA)獨立出來，其資料庫表格綱要如下：

```
LabelPath-1(Source, Target, Ordinal)
LabelPath-2(Source, Target, Ordinal)
…………..
LabelPath-n(Source, Target, Ordinal)


LabelPath-i-CDATA(ID, Value)
LabelPath-j-CDATA(ID, Value)
…………..
LabelPath-k-CDATA(ID, Value)
```

圖 2.4-3 Monet schema

針對每條唯一路徑，Monet 均建立一個表格且依據這條路徑名稱作為表格的名稱，以屬性 Source 與屬性 Target 建立起 XML 文件中每條唯一的路徑。針對節點內容值或是屬性內容值以另外表格 LablePath-CDATA 進行儲存。此方法會產生數量十分龐大的小型表格，通常幾條唯一路徑就會決定幾個 LabelPath 表格；LabelPath-CDATA 表格數量則需視 CDATA 與屬性所屬的唯一路徑數量而定。

2.4.2.3 XParent

不同於 Monet 方法，XParent[21]明確地儲存文件上邊的資訊(LabelPath)與節點內容路徑(DataPath)，其資料庫表格綱要如下：

```
LabelPath(ID, Len, Path)
DataPath(Pid, Cid)
Element(PathID, Did, Ordinal)
Data(PathID, Did, Ordinal, Value)
```

圖 2.4-4 XParent schema

資料庫內 LabelPath 表格儲存邊的資訊於屬性 Path。每一條邊都有自己唯一的編號：屬性 ID。屬性 Len 記錄了此 LablePath 為幾條邊所擁有。由於節點內容路徑通常走訪路徑長且資料量也十分龐大，此方法記錄了每條邊的父節點與子節點於 DataPath 表格中。而表格 Element 與 Data 則記錄了元素與內容值，以屬性 PathID 和屬性 Did 為外來鍵，分別關聯於表格 LabelPath 中的屬性 ID 與表格 DataPath 中的屬性 Cid。

上述三種方法 Edge、Monet、XParent 都是以儲存 XML 文件上邊資訊。還有一種方法是以儲存文件中節點資訊，也是接下來要介紹的第四種方法－XRel。

2.4.2.4 XRel

由於 XRel 儲存 XML 文件中節點資訊，故需要去記錄一種節點之間的關係，文獻中稱之為 region。Region 是一組成對的數字，記錄每一個節點的起始與終止位置。利用這組數字可以判斷兩個節點間是否存在祖孫上的關係，而非僅僅是父子關係的判斷。這種 region 的觀念並非在 XRel 文獻中第一次提出，先前即有許多文獻均使用到[51][23][38][9]，XRel 的資料庫表格綱要如下：

```
Path(PathID, Pathexp)
Element(DocId, PathID, Start, End, Ordinal)
Text(DocId, PathID, Start, End, Value)
Attribute(DocId, PathID, Start, End, Value)
```

圖 2.4-5 XRel schema

表格Path記錄到XML文件中唯一的路徑資訊於屬性Pathexpr中。屬性DocId代表該份文件編號。PathId則為路徑編號，關聯於表格Path中PathId屬性。屬性Start與屬性End則為該節點的起始與終止位置。屬性Ordinal記錄該節點為兄弟節點之中相同節點名稱的節點順序。而屬性Value為內容值。XRel開宗明義即表示其研究架構參考XPath模型而設計出三種節點型態：Element、Text、Attribute，並且依型態建立三個表格以儲存各自的節點資訊。

2.4.2.5 小結

我們整理了上述四種不需DTD的儲存方法，分別以「所需表格數」、「單一表格所儲存的資料量」、「設計理念」、「提出年份」、「貢獻」和「研究方向」進行比較。如表格 2.4-1：

表格 2.4-1　四種不需DTD儲存方法比較

方法	Edge	Monet	XParent	XRel
提出年份	1999	2000	2002	2001
所需表格數	1	唯一路徑數量 (包括element,attribute,text)	4	4
單一表格儲存量	大	小	中	中
設計理念	儲存邊資訊	儲存邊資訊	儲存邊資訊	儲存節點資訊
貢獻	簡單明白	改進Edge方法，將邊資訊進行分類	將邊與節點資訊分開	以節點為主，應用region縮減表格的合併運算
研究方向	需要大量的合併運算	會產生十分龐大表格數量，需要合併運算	需要合併運算	應用region會產生合併運算

由表格 2.4-1 可知，在關聯式資料庫系統中的儲存方法，若不使用 DTD 產生資料庫綱要，針對檢索時路徑走訪情況，表格之間的合併是無法避免的。假設有一個檢索條件一共會走訪 m 層，透過元素 e1、e2 到 e_n。針對 Edge 方法便需要經過 n 次表格的合併運算。

以 Monet，雖然將每一條唯一路徑設計成一個表格，路徑走訪時僅需針對所需的路徑表格進行運算即可。但是面對與 text 和 attribute 的檢索，還是需要進行合併運算且此方法會產生極大的表格數量。XParent 也擁有此限制。

以 XRel，透過 Path 表格進行路徑的走訪以找出該路徑終端節點編號，某個層次上與 Monet 和 XParent 的 LabelPath 表格雷同。而針對以節點為處理基礎的它，需利用到 region 來進行節點之間的關係比較。即使是某元素節點與該元素節點下的屬性節點關係也需進行 region 比較(會產生大量的 θ-join)。

2.4.3　關聯式-需 DTD 儲存方式

在這一類儲存方法中，以 shared 和 $shared^+$兩種作為代表。由於此類方法需要藉由 DTD 進行資料庫表格綱要設計，故複雜的 DTD 會產生出複雜的表格綱要。接下來我們簡略地介紹這兩種方法。

2.4.3.1 Shared

shared[37]主要是將 DTD 轉換為關聯式表格綱要，首先它有一些轉換原則：

- 複雜的 DTD 是需要被簡化(修剪)。
- DTD 中的元素能夠被轉換至關聯表內。

以某份文件的 DTD 為範例，如下：

```
<!ELEMENT Inquire (OrderList)>
<!ATTLIST Inquire Com_Name CDATA #INQUIRE>
<Date CDATA #INQUIRE>
<!ELEMENT OrderList (Article)*>
<!ELEMENT Article EMPTY>
<!ATTLIST Article Prod_No CDATA #INQUIRE>
<Prod_Name CDATA #INQUIRE>
<Prod_Quantity CDATA #INQUIRE>
```

圖 2.4-6　某份文件的 DTD

此方法利用 Preorder、Depth-First 走訪繪製出 DTD 的圖形如圖 2.4-8，再利用 DTD 圖形建立資料庫表格，以根元素建立一個 Inquire 表格，其元素建立將對應的欄位。由於 Article 擁有*，所以會建立一個新的表格，再建立 Article 的子元素於相對應的欄位中並且建立屬性 Order 表示同一層元素出現順序，最後以屬性 ParentID 與屬性 Inquire 表格進行關聯。其所建立的 schema 如下：

```
Inquire(ID, Com_Name, Data)
Article(ID, ParentId, Order, Prod_No, Prod_Name, Prod_Quantity)
```

圖 2.4-7　透過 shared 所建立的 schema

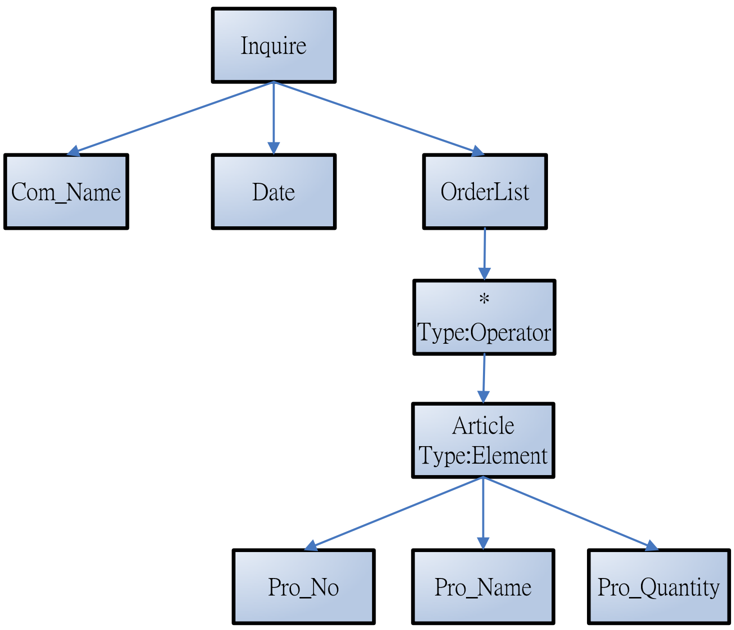


圖 2.4-8　轉換的 DTD 圖形

2.4.3.2 Shared$^+$

shared$^+$[52]設計理念如同前面介紹 XParent 方法中的 LabelPath 設計。它將原先的 shared 方法加上一個路徑上的索引。此路徑索引在文獻中稱為 PathIndex，但這個索引並不像是 B-tree 的設計，而是以一個資料庫表格 PathIndex 儲存。表格內有屬性 PathName，記錄每條唯一路徑資訊，而屬性 PathID 則為唯一識別值。在應用 DTD 圖形產生資料庫表格時，每一個表格均產生一個屬性 PathID，用來關聯 PathIndex 表格的識別值。如此的設計可以降低在路徑走訪時表格所需的合併運算次數。

我們舉個例子，假設有一個檢索條件需要走訪 XML 文件 n 層的深度，透過元素 e1、e2 到 e_n；而且假設這份 XML 文件存於 m 個表格內。假使今天沒有 PathIndex，則最多可能需要 m 個合併運算才可以完成檢索。若使用了 PathIndex，則可以透過 PathID 關聯至每一個表格內。換言之，shared 需要 m 個合併運算而 shared$^+$僅需一個合併運算與一個 Select 運算。

2.4.4 原生性(Native)資料庫儲存方式

2.4.4.1 以物件導向為基礎的原生性資料庫方法

此方法[41]利用物件導向觀念將 XML 元素視為個別的物件，如商用性的 XML 原生性系統 DDB-OO。它避免關聯式資料庫將 XML 階層性資料加以分解，而是採取更自然有彈性的解決方式，透過 XML 相關技術如 XQuery 和 XSLT 以提供自然的 XML 處理機制與檢索方法。它去除了關聯式資料庫中 XML 對應機制，也無須文件的綱要定義。由於興起時間並非很長，並沒有如關聯式資料庫系統已有成熟且強大的技術與研究支援，和使用關聯式儲存方式之間的轉換機制，也存在值得探討的議題。

2.4.4.2 以 DOM 為基礎的原生性資料庫方法

DOM 提供了一套標準來表達 HTML 與 XML 文件，使用者透過 DOM 的標準介面來管理 XML 資料。在 DOM 中提供了祖孫關係的指標如 parentNode、firstChide、lastChild；針對兄弟關係提供了 previousSibling 與 nextSibling 指標。透過這些指標便可對文件進行走訪。

2.4.5 小結

最後我們針對上述三大類共八種儲存方法進行整理，分別以「所處領域」、「代表研究文獻」、「設計理念」、「是否需要 DTD」、「貢獻」與「研究方面」等六構面進行比較。以表格 2.4-2 表示：

表格 2.4-2　八種儲存方法整理

<table>
<tr><th>領域</th><th>代表研究文獻</th><th>是否需DTD</th><th>發表年份</th><th>設計理念</th><th>貢獻</th><th>研究方向</th></tr>
<tr><td rowspan="6">關聯式資料庫系統</td><td>Edge</td><td rowspan="4">否</td><td>1999</td><td rowspan="3">儲存邊資訊</td><td>設計簡單</td><td>需要大量合併運算</td></tr>
<tr><td>Monet</td><td>2000</td><td>改進 Edge</td><td>會產生大量的表格且需要合併運算</td></tr>
<tr><td>XParent</td><td>2002</td><td>分離路徑資訊與節點資訊</td><td>需要合併運算</td></tr>
<tr><td>XRel</td><td>2001</td><td>儲存節點資訊</td><td>使用 region 技術</td><td>需要合併運算</td></tr>
<tr><td>shared</td><td rowspan="2">是</td><td>1999</td><td rowspan="2">以 DTD 設計表格綱要</td><td>應用 DTD 進行表格設計</td><td rowspan="2">需要 DTD</td></tr>
<tr><td>shared+</td><td>2001</td><td>改進 shared，新增路徑索引</td></tr>
<tr><td rowspan="2">原生性資料庫引擎</td><td>NXD (OO)</td><td rowspan="2">否
(檢索時會使用)</td><td>無</td><td>將 XML 元素視為個別的物件</td><td rowspan="2">提供自然的 XML 處理機制與檢索方法</td><td rowspan="2">較無成熟技術且和以 RDBS 為儲存模型的資料庫有轉換上的成本</td></tr>
<tr><td>NXD (DOM)</td><td>2002</td><td>DOM 可以透過 DOM 的標準介面來管理 XML 資料</td></tr>
</table>

第三章　系統設計

3.1　引言

在設計方面，本系統視為建構於使用者與關聯式資料庫之間的 gateway，以圖 3.1-1 表示之。系統內部可分為兩大模組：

I. 新增模組：此模組接受 XML 文件並且加以剖析，以建構 DataGuide 和 Node Information。DataGuide 代表所有 XML 文件在路徑上的壓縮結構，在程式執行時於記憶體內，並且隨著 XML 文件新增而不斷地修正結構，並適時儲存於關聯式資料庫的 DataGuide table 中。Node Information 則記錄所有 XML 文件中節點的內容資訊。每一個文件節點皆會對應到一個 DataGuide 節點，而每一個 DataGuide 節點則擁有一個以上的文件節點對應。

II. 檢索模組：此模組接受 XPath 檢索語言並加以剖析。將路徑走訪與 DataGuide 進行比對，以得到符合路徑條件的 DataGuide 節點編號。應用這些 DataGuide 節點編號與 Node Information table 的資料結構，便能將 XPath 中路徑走訪與內容檢索轉換為 SQL commands。最後將這些 SQL commands 合併，交給關聯式資料庫處理，以獲取檢索回傳值。

接下來小節安排依序為名詞解釋、新增模組、檢索模組，詳細說明其設計理念與運作流程、資料庫綱要規劃與演算法設計。

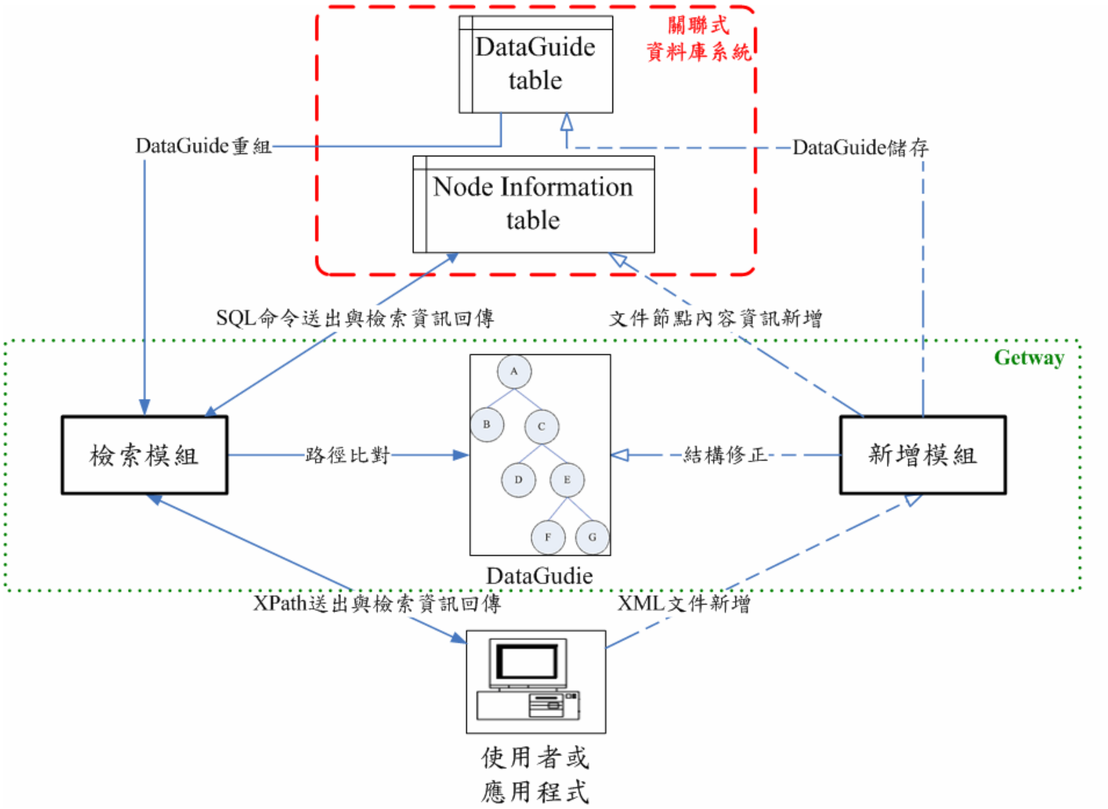

關聯式
資料庫系統
DataGuide
table
Node Information
table
DataGuide重組
DataGuide儲存
SQL命令送出與檢索資訊回傳
文件節點內容資訊新增
Getway
檢索模組
路徑比對
結構修正
新增模組
A
B
C
D
E
F
G
DataGudie
XPath送出與檢索資訊回傳
XML文件新增
使用者或
應用程式

圖 3.1-1　系統設計圖

3.2 名詞定義

- XML 原始文件(raw data)：使用者欲管理及檢索的來源資料，每一份 XML 文件由許多標籤(tag)所組成(請參考 2.1.2.2 小節)。
- DataGuide：於 1997 年在 Proceedings of the 23rd VLDB Conference[29]中提出的路徑壓縮概念，視為路徑上的索引。它擷取半結構化文件的路徑後針對相同的路徑給予合併，以樹狀結構表示。(請參考 2.3.1.1 小節)
- 路徑(Path)：樹狀結構中，路徑是由節點組成。由根節點到節點 P 之路徑型態表示為{ $N_r/N_1/N_2/N_3/\ N_4/\ldots./N_P$}，$N_r$ 為 root node。(請參考 2.2.1 小節)
- RPE(Regular Path Expression)[27][44]：對於走訪 XML 文件路徑，使用者可以使用 Regular Expression，對路徑進行彈性化與多變化的描述。(請參考 2.2.1 小節)
- 索引(Index)：針對原始資料依不同目的所設計相關的資訊。可能為關鍵字或相關資料的列表，以指向更完整的資料。在 XML 研究範疇中，使用者能針對 XML 文件中的 Text、文件結構與路徑進行索引建置，其目的在於提升檢索效能。(請參考 2.3 節)

3.3 新增模組

3.3.1 XML 文件資料模型

我們設計一個 XML 文件資料模型以描述 well-formed 的 XML 文件，但不要求是 valid。每一份文件由許多 tag 所組成，而每一個 tag 又包含多個下列三種型態的節點：

1. Element node：表示一份 XML 文件內的 tag。節點名稱預設值為 tag 所指定的 tag name，節點內容值為 Null。 Element node 下層可擁有零到多個型態為 Element node 或 Attribute node 或 Text node 的子節點。
2. Attribute node：表示一份 XML 文件內相同 tag 下的 attribute。節點名稱預設值為 attribute name，節點內容值為 attribute content。Attribute node 並無子節點，可視為 leaf node。
3. Text node：表示一份 XML 文件內相同 tag 下的 text。節點名稱預設值為 null，節點內容值為 text content。Text node 並無子節點，可視為 leaf node。

根據上述的三種節點型態，我們可以將 XML 文件透過 XML 剖析器，轉換為本研究所需的文件資料模型。圖 3.3-1 為範例文件「customers.xml」[附錄一]的文件資料模型圖。

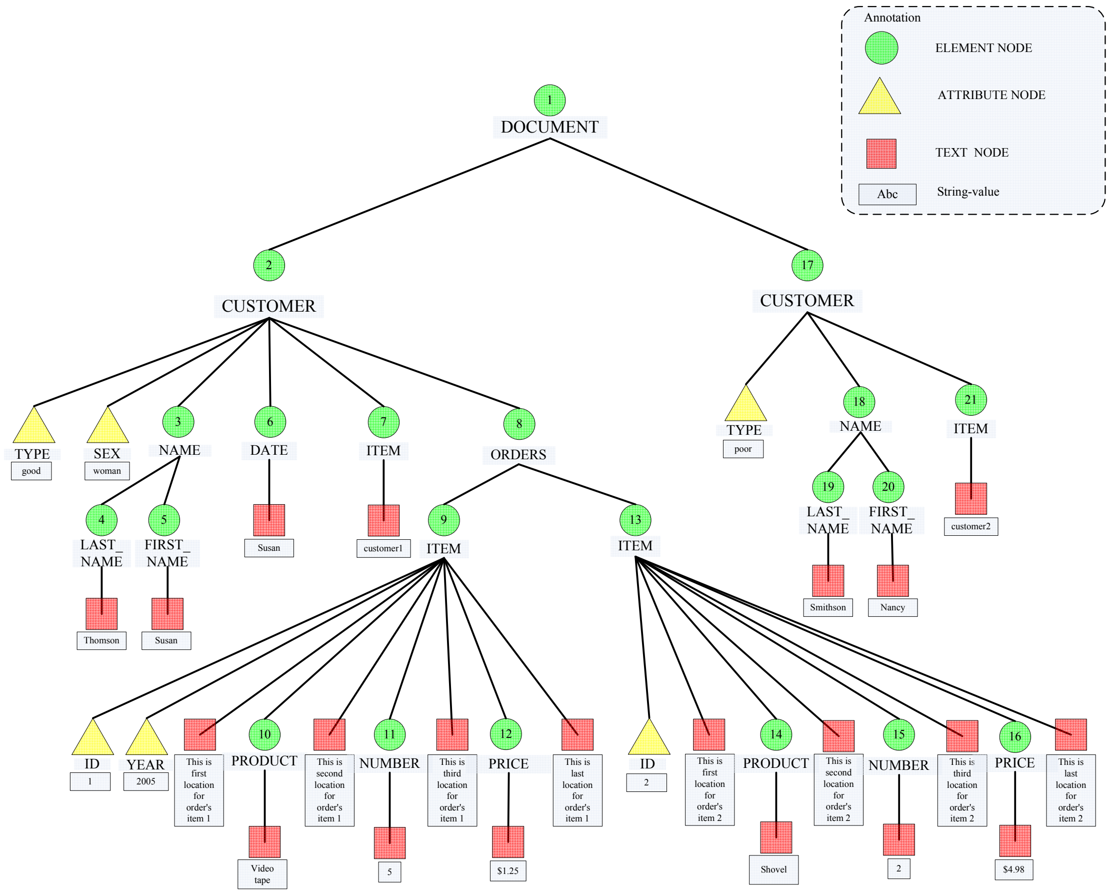


圖 3.3-1　範例文件「customers.xml」的文件資料模型圖

圖 3.3-1 中以不同的圖型表示上述的三種節點型態。數字編號為節點所對應到原始 XML 文件的標籤，進行 preorder、Depth-First 走訪編碼。Attribute node 與 Text node 對應於和父親節點(Element node)相同的標籤，故標籤編號同於父親節點，圖中此資訊省略不記。

模型中節點之間以線段相互連接以表達此份文件的結構性。以標籤編號 9 的 Element node 為例，擁有 9 個子節點—3 個 Element nodes(標籤編號為 10、11、12)、2 個 Attribute nodes(attribute name 為 ID 與 YEAR)與 4 個 Text nodes。標籤編號 9 的路徑資訊為/DOCUMENT/CUSTOMER/ORDERS/ITEM。

3.3.2　DataGuide 與 Node Information

將新增的 XML 文件透過 XML 剖析器轉換為 XML 文件資料模型後，我們對 XML 文件資料模型拆解為 DataGuide 與 Node Information 兩個部份。DataGuide

代表所有 XML 文件綱要萃取後的結構，Node Information 負責記錄所有 XML 文件中節點的內容資訊。

3.3.2.1 DataGuide

我們採用 1997 年 R. Goldman 所提出的路徑壓縮觀念並加以擴充。DataGuide 節點名稱為原始文件的 tag 名稱，而 DataGuide 節點之間組成路徑資訊。DataGuide 針對單一文件內相同路徑給予合併，對於 Text node 則不予記錄。我們將 DataGuide 從單一文件擴充至多份文件處理，並對於 Attribute node 也不予記錄(本研究將在同 tag 內的節點對應至所屬的 tag 編號上，故可共同分享路徑資訊)。圖 3.3-2 為範例文件「customers.xml」資料模型之 DataGuide 模型圖。

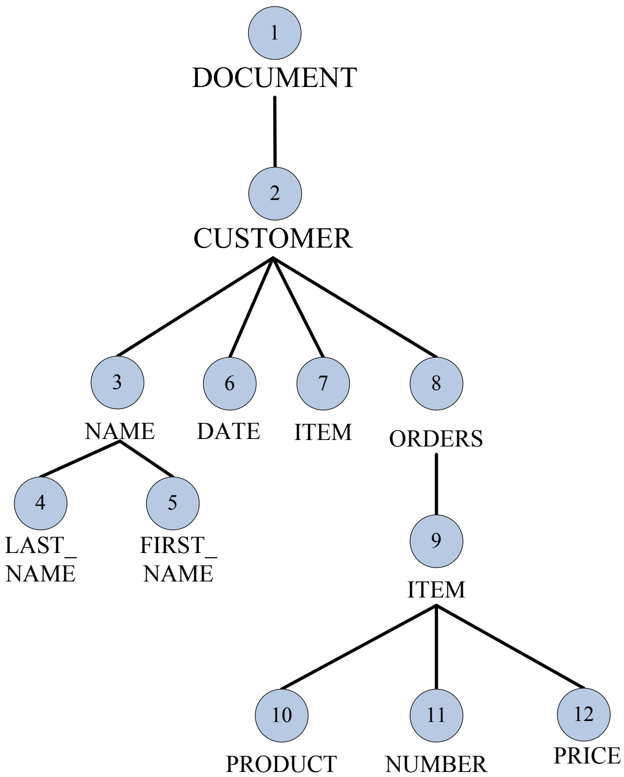


圖 3.3-2　customers.xml 文件資料模型之 DataGuide 模型圖

我們將圖 3.3-1 與圖 3.3-2 進行比對，可以很清楚地發現後者的節點數量驟減(由 48 個降為 12 個)，此優勢可降低檢索時在相同路徑上的走訪成本。在圖 3.3-1 中，標籤編號 9、13 的 Element node 擁有相同的路徑資訊

(/DOCUMENT/CUSTOMER/ORDERS/ITEM)，故在圖 3.3-2 中便將此 2 個 Element node 壓縮為編號 9 的 DataGuide node，並且將 Element node 與 DataGuide node 的關係和 Element node 的內容記錄於 Node Information 中。相同地，標籤編號 9 的 Element node 底下的 Attribute node 與 Text node 也擁有相同的路徑資訊，故一併與 DataGuide node 的關係和本身節點內容存放在 Node Information 中。

3.3.2.2 Node Information

Node Information 負責記錄所有 XML 資料模型中節點的內容資訊。配合 DataGuide，Node Information 可視為記錄某一個 DataGuide 節點所"隱含"的內容資訊，字面"隱含"意謂：1、DataGuide 節點所對應到的文件節點必定符合在 DataGuide 中走訪的路徑條件。2、對應文件節點資訊包括了 Element node、Attribute node 和 Text node。下面說明 Node Information 的資料結構：

- DGNodeid：記錄所對應到的 DataGuide 編號。
- TagId：記錄所對應的文件標籤編號。標籤編號為針對 XML 文件的標籤，進行 preorder、Depth-First 走訪編碼。
- AttributeName：主要記錄屬性節點名稱。若此節點屬於 Element node 或 Text node，則記錄 Null。
- NodeId：在同一個 TagId(即為是同一個 tag)下的節點編號。Element node 以 0 表示，Text node 以 1 開始遞增，Attribute node 以-1 遞減。
- Content：儲存節點的內容值。Element node 記錄 Null。
- PrefixContent：擷取 Content 前 256Bytes 字元。
- FileName：文件名稱。儲存此文件節點隸屬哪一份文件。
- ElementOrder of Same Name under Parent：在相同父親節點下，相同 Element node 名稱之節點順序。
- PathOrder：由根節點開始走訪，某節點處於相同路徑資訊之路徑順序。

- StartLocation：某節點在 XML 文件內的起始標籤位置。Element node 記錄此 Element node 對應的起始標籤位置與文件開頭的距離，Attribute node 與 Text node 的 StartLocation 與父節點相同(因為處於同一個 tag 內)。
- EndLocation：某節點在 XML 文件內的終點標籤位置。Element node 記錄此此 Element node 對應的終點標籤位置與文件開頭的距離，Attribute node 與 Text node 的 EndLocation 與父節點相同(因為處於同一個 tag 內)。
- Father：某節點的父親節點的 TagId，若無則記錄 0。
- Gfather：某節點的祖父親節點的 TagId，若無則記錄 0。
- Ggfather：某節點的曾祖父親節點的 TagId，若無則記錄 0。

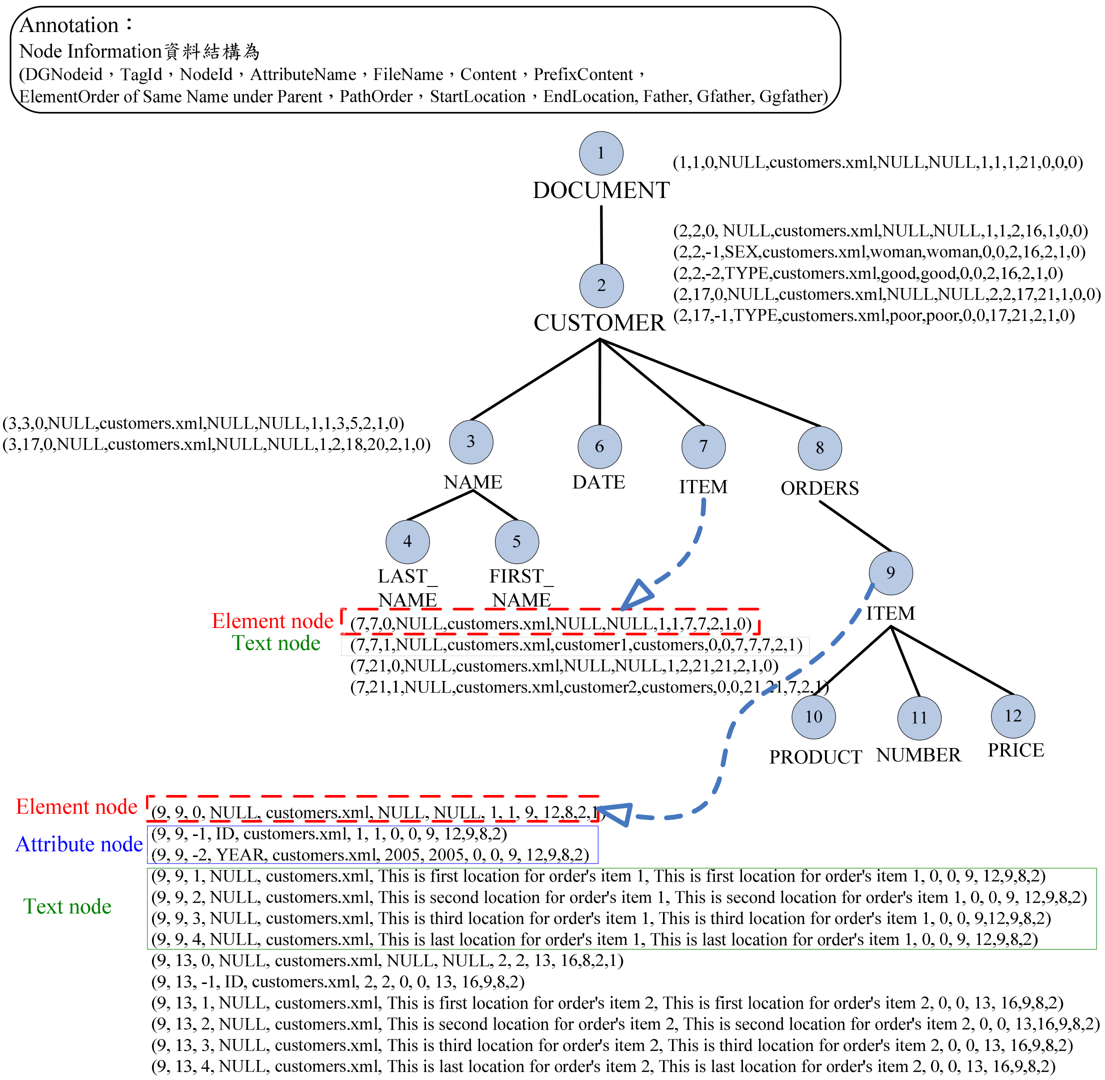


圖 3.3-3　範例文件「customers.xml」的 DataGuide 與 Node Information

透過上述所組成的資料結構，即可完整表達出一份 XML 文件中所擁有的節點內容資訊。透過 DataGuide 快速的走訪，找到符合路徑條件及節點內容值是屬於哪些文件中的哪些節點。圖 3.3-3 為範例文件「customers.xml」的 DataGuide 與 Node Information(圖中有省略部份的 Node Information 資訊)。Node Information 所呈現的資料結構為(DGNodeid，TagId，NodeId，AttributeName，FileName，Content，PrefixContent，ElementOrder of Same Name under Parent，PathOrder，StartLocation，EndLocation，Father，Gfather，Ggfather)。

以 DataGuide 編號 9 為例，所對應到的 Node Information 共有 13 筆記錄。前 7 筆記錄可以看出對應到文件標籤編號 9 的節點，其中 NodeId 為 0 表示為 Element node，NodeId 為-1、-2 表示為 Attribute node，NodeId 為 1、2、3、4 表示為 Text node，其餘節點可以參考圖 3.3-1 以此類推。表格 3.3-1 整理了上述 Node Information 所建立的值組名稱、說明，及使用於哪些檢索情況。

表格 3.3-1　Node Information 的名稱、說明、與使用的檢索情況之整理

名稱	說明	使用的檢索情況
DGNodeid	文件節點所對應到的 DataGuide 編號	A、B、C、D、E
TagId	文件節點標籤編號	A、B、C、D、E
NodeId	在同個 TagId 下的節點編號	A、B、C、D、E
AttributeName	屬性名稱	B
Content	節點內容值	C
PrefixContent	擷取 Content 的前 256Bytes 字元	C
FileName	文件名稱	A、B、C、D、E
ElementOrder of Same Name under Parent	在相同父親節點下，相同節點名稱之節點順序。	E
PathOrder	由根節點向下走訪，相同路徑資訊之路徑順序。	E
StartLocation	節點在 XML 文件內的起始標籤位置	D
EndLocation	節點在 XML 文件內的終點標籤位置	D
Father	父親節點 tagid	D
Gfather	祖父節點 tagid	D
Ggfather	曾祖父節點 tagid	D
(註)：情況 A 為路徑走訪；情況 B 為節點型態比對；情況 C 為節點內容值比對；情況 D 為多重路徑走訪；情況 E 為順序比對		

3.3.3 資料庫綱要設計

圖 3.3-4 為資料庫的個體-關係模型(E-R Model)，我們可以看到有三個個體：DataGuide、Node Information 與 Files。Node Information 與 DataGuide 之間的關係以「refers to」關聯，表示每一個 Node 都可以對應到一個 DataGuide 節點，屬多對一關係；Node Information 與 Files 之間的關係以「belongs to」關聯，表示每一個 Node 都會屬於一個 File，屬多對一關係

在 DataGuide 資料庫儲存設計採用結構對應法，將 DataGuide 中的節點，以

一筆筆 tuple value 存入，並將節點之間的關係以父子關係表示。圖 3.3-5 為 DataGuide 的資料庫表格定義。

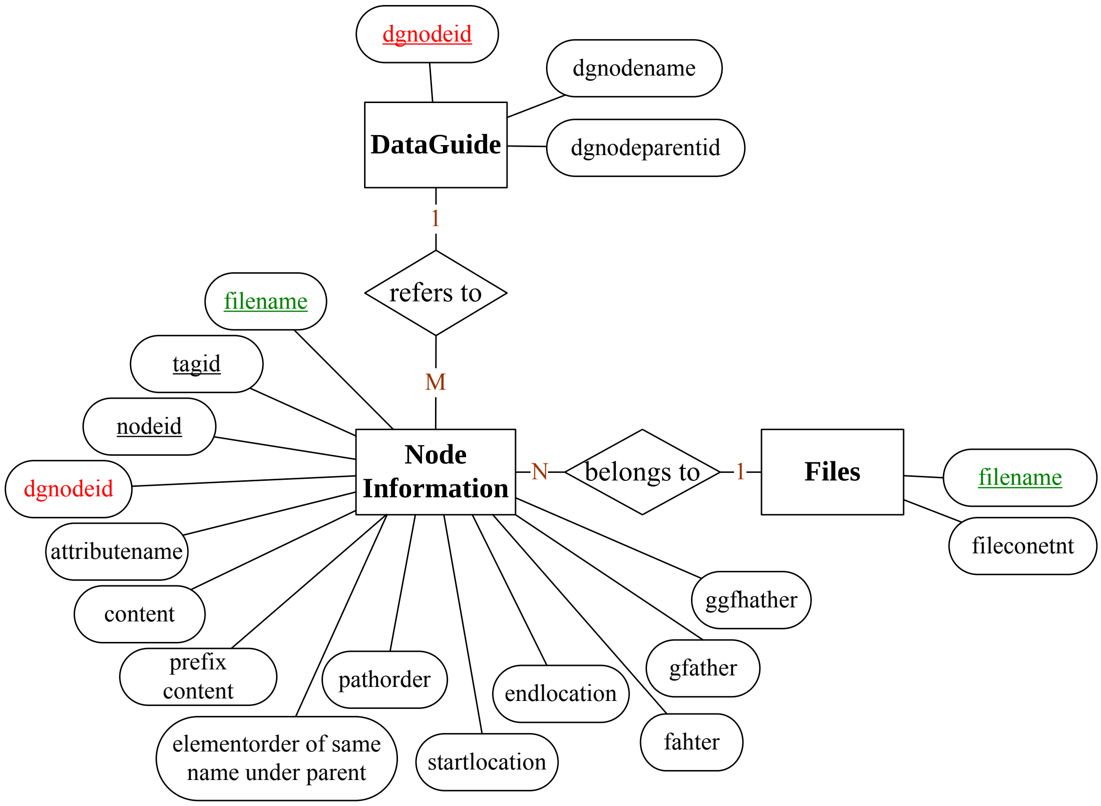


圖 3.3-4 資料庫的 E-R Model

```
create table dataguide   --建立 DataGuide 表格
(
    dgnodeid int not null,   --DataGuide 某節點編號
    dgnodename char(20) not null,   --DataGuide 某節點名稱
    dgnodeparentid int not null,   --DataGuide 某節點的父節點編號
    primary key(dgnodeid)   --設 dgnodeid 為主鍵
)
```

圖 3.3-5 DataGuide 的資料庫表格定義

在設計完 DataGuide 的資料庫表格後，我們將所輸入的 XML 文件儲存至資料庫中以方便管理，圖 3.3-6 為檔案的資料庫表格定義。

```
create table files--建立 Files 表格
(
        filename char(20) not null, --XML 文件檔名
        filecontent text null, --XML 文件內容
        primary key(filename) --設 filename 為主鍵
)
```

圖 3.3-6 Files 的資料庫表格定義

根據表格 3.3-1 中 Node Information 的名稱、說明，Node Information 表格綱要設計如圖 3.3-7。

```
create table node_information --建立 Node Information 表格
(
        dgnodeid int not null, --DataGuide 某節點編號
        tagid int not null, --某份 XML 文件內標籤編號，由 1 開始
        nodeid int not null, --某份 XML 文件中在同個 Tagid 下的節點編號
        attributename char(20) null,   -- 記錄屬性名稱，Element 和 Text 以 NULL 表示
        content varchar(1100) null, --記錄節點內容值，Element 以 NULL 表示
        prefixcontent char(256) null, --記錄節點內容值前 256Bytes 字元，Element 以 NULL 表示
        filename char(20) not null,   --XML 文件名稱
        elementorderofsamenameunderparent int not null,   --在相同父親節點下，相同節點名稱之節
                                                           --點順序
        pathorder int not null,   --由根節點向下走訪，某節點在相同路徑資訊之路徑順序
        startlocation int not null,   --節點在 XML 文件內的起始標籤位置
        endlocation int not null,   --節點在 XML 文件內的終點標籤位置
        father in not null,   --此節點的父親節點編號
        gfather in not null,   --此節點的祖父親節點編號
        ggfather in not null,   --此節點的曾祖父親節點編號
        primary key(filename, tagid, nodeid),   --複合主鍵，某一份文件的某標籤編號之某節點編號
        foreign key(filename)   --關聯 files 表格中的 filename
        references files,
        foreign key(dgnodeid)   --關聯 dataguide 表格中的 dgnodeid
        references dataguide
)
```

圖 3.3-7 Node Information 的資料庫表格定義

3.3.4 演算法設計

3.3.4.1 剖析 XML 文件

我們使用 DOM Parser 去針對所輸入的 XML 文件，進行文件中節點的獲取，圖 3.3-8 為應用 DOM 去剖析 XML 文件的虛擬碼。首先程式接收代表一份文件的 Document node（*n*），並且依據該節點的子節點型態，進行子節點的資訊取得；在資訊取得後，再檢查子節點是否有下一層子節點存在，若有則呼叫遞迴。此程式會將一份文件以 preorder、depth-first 的方式掃描完所有的節點，並且將所需要的節點資訊取出。

```
TraverseXmlFile(n)
    if GetAttribute(n) != null
        GetAttributeInf(n)
    for i ← 0 to GetChildNodesLength(n)
        if GetNodeType(i) = TEXTNODE
            GetTextInf(i)
        eles if GetNodeType(i) = ELEMENTNODE
            GetElementInf(i)
            TraverseXmlFile (i)
```

圖 3.3-8　剖析 XML 文件的虛擬碼

3.3.4.2 DataGuide

我們設計一個 DataGuide 的資料結構，圖 3.3-9 為 DataGuide 類別圖。圖 3.3-10 為 DataGuide 執行的虛擬碼。此虛擬碼接受輸入的 XML Document node（*n*）與 DataGuide（*g*），將 Document node 所擁有的子節點，依序和 DataGuide 節點的全部子節點作節點名稱上的比對，若有重複的名稱則跳過，且雙方節點遞迴呼叫；若沒有重複，則新增 DG 節點的子節點。經由不斷的遞迴呼叫，輸入的 XML 文件即可與 DataGuide 結構同時完成比對。

DataGuide
-dgNodeId : int = 0 -dgNodeName : string = null -dgNodeParent_R : DataGuide = null -dgNodeChildrenList_R : DataGuide = ArrayList() -startLocation : int = 0 -endLocation : int = 0 -elementOrder : int = 1 -pathOrder : int = 0
+addNodeChildren() : void +getNodeChildrenRef() : DataGuide +getNodeChildrenNum() : int +getDgNodeId() : int +setDgNodeId() : void +getDgNodeName() : string +setDgNodeName() : void +getDgNodeParent_R() : DataGuide +setDgNodeParent_R() : void +compareXMLandDG() : void

圖 3.3-9　DataGuide 類別圖

```
CompareXmlAndDG(n, g)
    if GetAttribute(n) != null
        GetAttributeInf(n)
    for i ← 0 to GetChildnodesLength(n)
        n_child ← GetChildRefence(n,i)
        if GetNodeType(n_child) = TEXTNODE
            GetTextInf(n_child)
        else if GetNodeType(n_child) = ELEMENTNODE
            for j ← 0 to GetChildnodesLength(g)
                g_child ← GetChildRefence (g,j)
                if GetChildNodesLength(g_child) = null
                    AddChild(g_child)
                    GetElementInf(n_child)
                    CompareXmlAndDG(n_child, g_child)
                else if NodenameEqualDGName(n_child, g_child) = = true
                    GetElementInf(n_child)
                    CompareXmlAndDG(n_child, g_child)
                else if (j + 1) = = GetChildnodesLength(g)
                    AddChild(g_child)
                    GetElementInf(n_child)
                    CompareXmlAndDG(n_child, g_child)
```

圖 3.3-10　DataGuide 執行的虛擬碼

3.4 檢索模組

檢索模組的整體流程為接受並剖析所輸入的 XML 檢索語言－XPathCore，將檢索中的路徑走訪與 DataGuide 進行比對，以得到符合路徑條件的 DataGuide 節點編號。應用這些 DataGuide 節點編號與 Node Information table 的資料結構，便能將 XPath 中路徑走訪與內容檢索轉換為 SQL commands。最後將這些 SQL commands 合併，交給關聯式資料庫系統處理以獲取檢索回傳值。

3.4.1 XPathCore

在 XML 檢索語言上我們所採用 XPathCore[24]，此檢索語言擷取 XPath 的核心部份－路徑檢索描述。因應第五章檢索效能測試的需求，我們將原先的 grammar 加以擴充(灰色網底部分)，圖 3.4-1 為擴充後的 XPathCore grammar。

```
Query               ::=  PathExpr
PathExpr            ::=  RegularExpr
                      |  '/' RegularExpr
                      |  '//' RegularExpr
                      |  ParenthesizedExpr Predicate?  '/'  RegularExpr
                      |  ParenthesizedExpr Predicate?  '//'  RegularExpr
RegularExpr         ::=  Step Predicate?
                      |  Step Predicate?  '/'  RegularExpr
                      |  Step Predicate?  '//'  RegularExpr
Step                ::=  QName
                      |  '@'  QName
                      |  '*'
                      |  'text()'
Predicate           ::=  '['  Comparison  ']'
ParenthesizedExpr   ::=  '('  PathExpr  ')'
BasicExpr           ::=  Literal
                      |  Number
                      |  NULL
Comparison          ::=  ArithExpr
                      |  ArithExpr  CompareOp  ArithExpr
CompareOp           ::=  '='
                      |  '!='
                      |  'BEFORE'
                      |  'IS'
                      |  'ISNOT'
```

ArithExpr	：：=	BasicExpr
	|	PathExpr

圖 3.4-1　擴充的 XPathCore grammar

表格 3.4-1　XPathCore grammar 代表的語意

Expression	代表的語義
Query	一條完整的 XPathCore 表示式。
PathExpr	XPathCore 中完整的路徑表示式。
RegularExpr	使用 token'/'或'//'將 Step 組合起來以表達基本路徑表示，配合上 Predicate 可以進行進階表示。
Step	表示 XML 文件中 Element node、Attribute node 與 Text node，亦可使用 token '*'表示未知名稱的 Element node。
Predicate	針對某節點或是某路徑進行 predicate 描述。
ParenthesizedExpr	針對某特定路徑的描述。
BasicExpr	基本的 literal、數值或 Null 值。
Comparison	針對兩個 ArithExpr 進行比對運算，其 ArithExpr 可以是條路徑或一個基本的比較值。
CompareOp	比對運算中的運算子。
ArithExpr	比對運算中的運算元，可為 BasicExpr 或 PathExpr。

3.4.2　檢索語言轉換

3.4.2.1 簡易路徑檢索條件

首先我們先以一個簡易的路徑檢索條件例子，說明如何使用 DataGuide 與 Node Information 將 XPathCore 轉換為 SQL commands。

EX_Q1：/DOCUMENT/CUSTOMER/ORDERS/ITEM 表示檢索路徑的條件是以根

節點往下經過 DOCUMENT、CUSTOMER、ORDERS 到 Element node 名稱為 ITEM 的節點，並找出 ITEM 節點屬於哪些文件中的哪些標籤編號，此檢索條件由 grammar 中的第 1~4 條所組合而成。我們將此檢索條件與 DataGuide、Node Information 進行比對，利用代表所有文件壓縮結構綱要的 DataGuide 進行路徑上的走訪，再將所找到的 DG 節點結合 Node Information 資料結構轉換為 SQL commands。以圖 3.4-2 表示之。

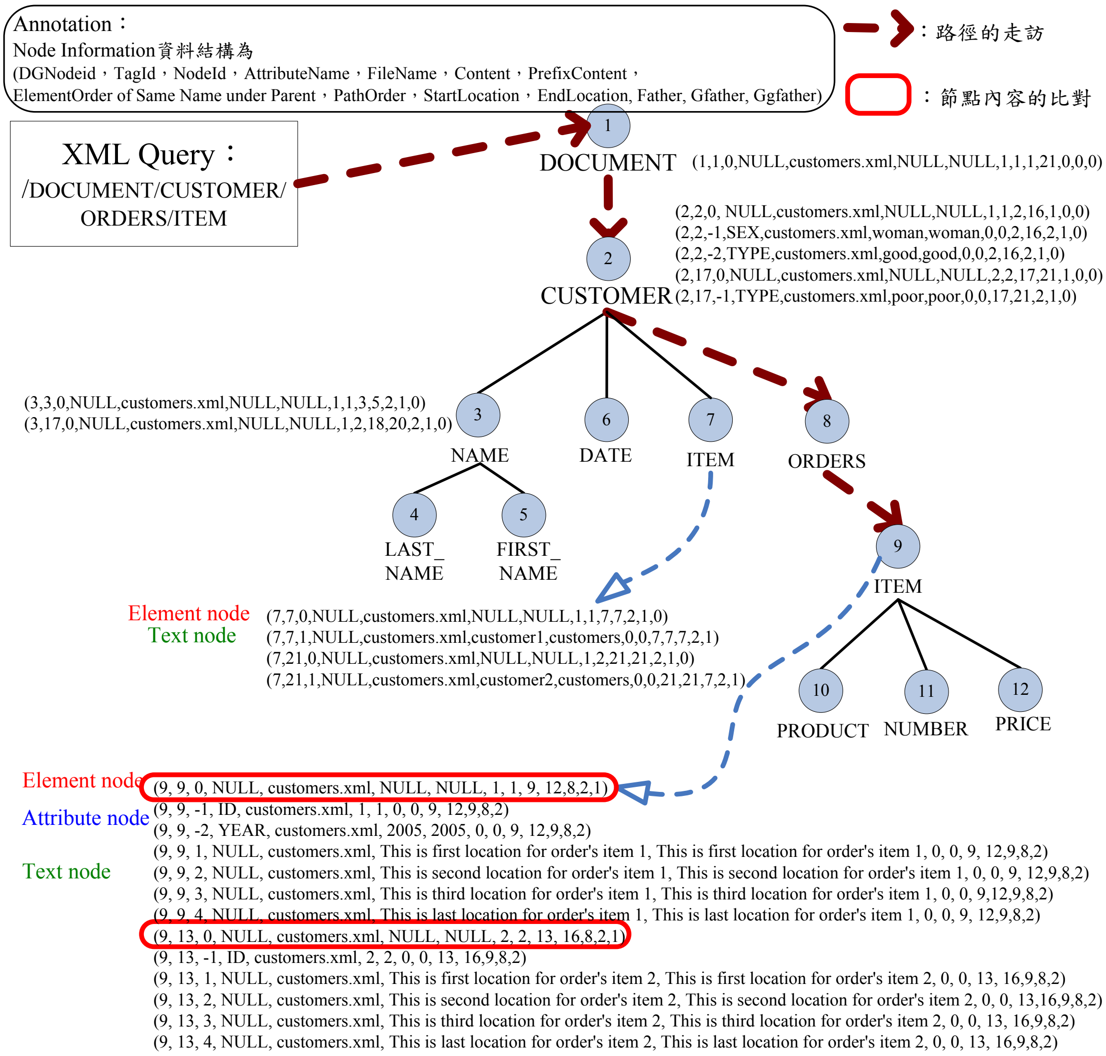


圖 3.4-2　EX_Q1 透過 DataGuide 與 Node Information 進行路徑比對

圖中粗體虛線段為針對路徑檢索所走訪的路徑，而實線框框為針對某個節點內容所比對的內容值。由於 EX_Q1 是以走訪一個單純路徑至某一個 Element node，所以

在節點內容會針對Element node進行比對。我們根據記錄節點內容的Node Information資料結構表格 3.3-1 來看，欄位 NodeId 為判斷節點型態的主要因子；欄位 DGNodeid為所對應到的 DataGuide 編號；欄位 TagId 為文件節點標籤編號；欄位 FileName 為所屬文件名稱。圖 3.4-3 為轉換後的 SQL commands。

```
select n9.filename, n9.tagid
from node_information n9
where n9.dgnodeid = 9 and n9.nodeid = 0
```

圖 3.4-3 EX_Q1 轉換後的 SQL commands

此 SQL commands 的涵義為查詢 node_information table 內，當條件為欄位dgnodeid 為 9 且欄位 nodeid 為 0(表示此筆節點與 DataGuide 節點編號 9 擁有相同的路徑資訊，且節點型態為 Element node)時，它的標籤編號與所屬的文件名稱為何。由此範例我們可以看出應用 DataGuide 可以在記憶體內，可快速地檢索至所符合路徑條件的 DataGuide 節點，且經由此 DataGuide 節點可以快速的關聯到對應的 XML 文件節點。此作法能在路徑條件為層級未知或節點名稱未知的情況下獲取更大的效益，因為所建立的 DataGuide 已經包含了所有文件的壓縮結構資訊。

3.4.2.2 複雜路徑檢索條件

根據 XPathCore 的 grammar 中 PathExpr 表示，在路徑走訪情況除了以‘/’連接不同的 Step 外，更可以利用‘//’來表示兩個 Step 中層級未知的路徑走訪。以檢索條件EX_Q2：/DOCUMENT//ITEM 為例，表示檢索路徑的條件是以根節點往下經過DOCUMENT 後，再經過 0 ~ n 層的路徑至 Element node 名稱為 ITEM 的節點，並找出 ITEM 節點屬於哪些文件中的哪些標籤編號，此檢索條件由 grammar 中的第 1~4條所組合而成。我們將此檢索條件與 DataGuide、Node Information 進行比對，以圖3.4-4 表示之。路徑 /DOCUMENT/CUSTOMER/ITEM 與/DOCUMENT/CUSTOMER/ORDERS/ITEM 皆符合此檢索條件，所對應至 DataGuide節點編號為 7 與 9。我們分別針對 DataGuide 編號 7 與 9 的情況產生個別的 SQL commands，在將這些 SQL commands 進行聯集運算(Union)以找出這些節點的集合。

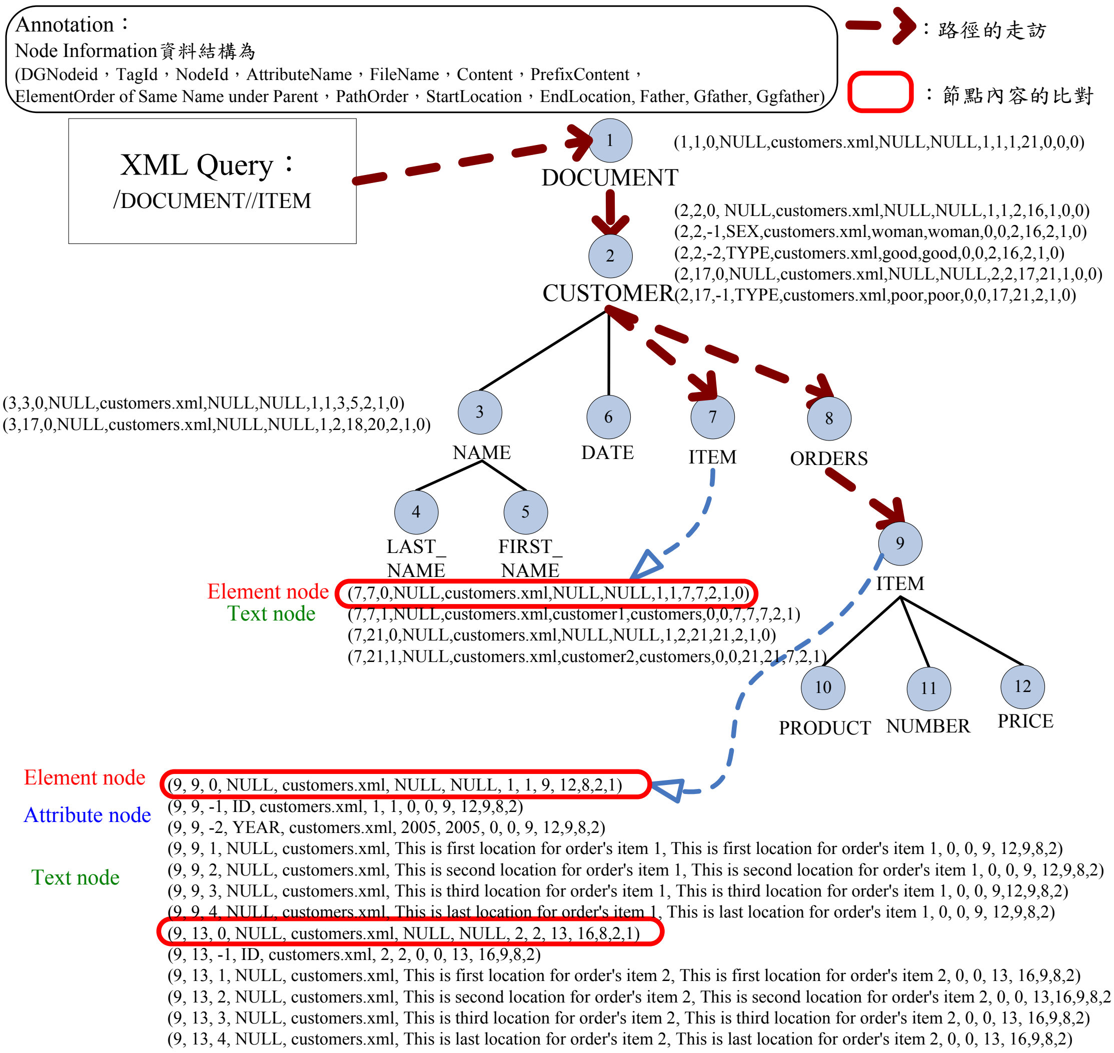


圖 3.4-4 EX_Q2 透過 DataGuide 與 Node Information 進行路徑比對

```
(select n7.filename, n7.tagid
from node_information n7
where n7.dgnodeid = 7 and n7.nodeid = 0)
UNION
(select n9.filename, n9.tagid
from node_information n9
where n9.dgnodeid = 9 and n9.nodeid = 0)
```

圖 3.4-5 EX_Q2 轉換後的 SQL commands

3.4.2.3 節點型態檢索條件

根據 XPathCore 的 grammar 中 Step 表示，節點型態的檢索除了 Element node 外，更包含了 Attribute node、Text node 與未指定節點名稱的 Element node(以‘*’表示)。針對這一方面的檢索，僅需要配合上所設計的 Node Information 資料結構，便可簡明地達到 SQL commands 轉換。以檢索條件 EX_Q3：/DOCUMENT/CUSTOMER/ORDERS/ITEM/@YEAR 為例，檢索條件為經由指定路徑走至 Element node 名稱為 ITEM 的節點後，再找到所屬的 Attribute node，其屬性名稱為 YEAR。我們將此檢索條件與 DataGuide、Node Information 進行比對，以圖 3.4-6 可見，透過 DataGuide 走訪至 DataGuide 節點編號 9 的位置後，再配合上 Node Information 即可找出所屬的 Attribute node(以實線框框表示)。由於 Attribute node 與 Element node 所屬同一標籤編號(在 XML 文件中都是屬於同一標籤)，故找出的標籤標號皆相同。

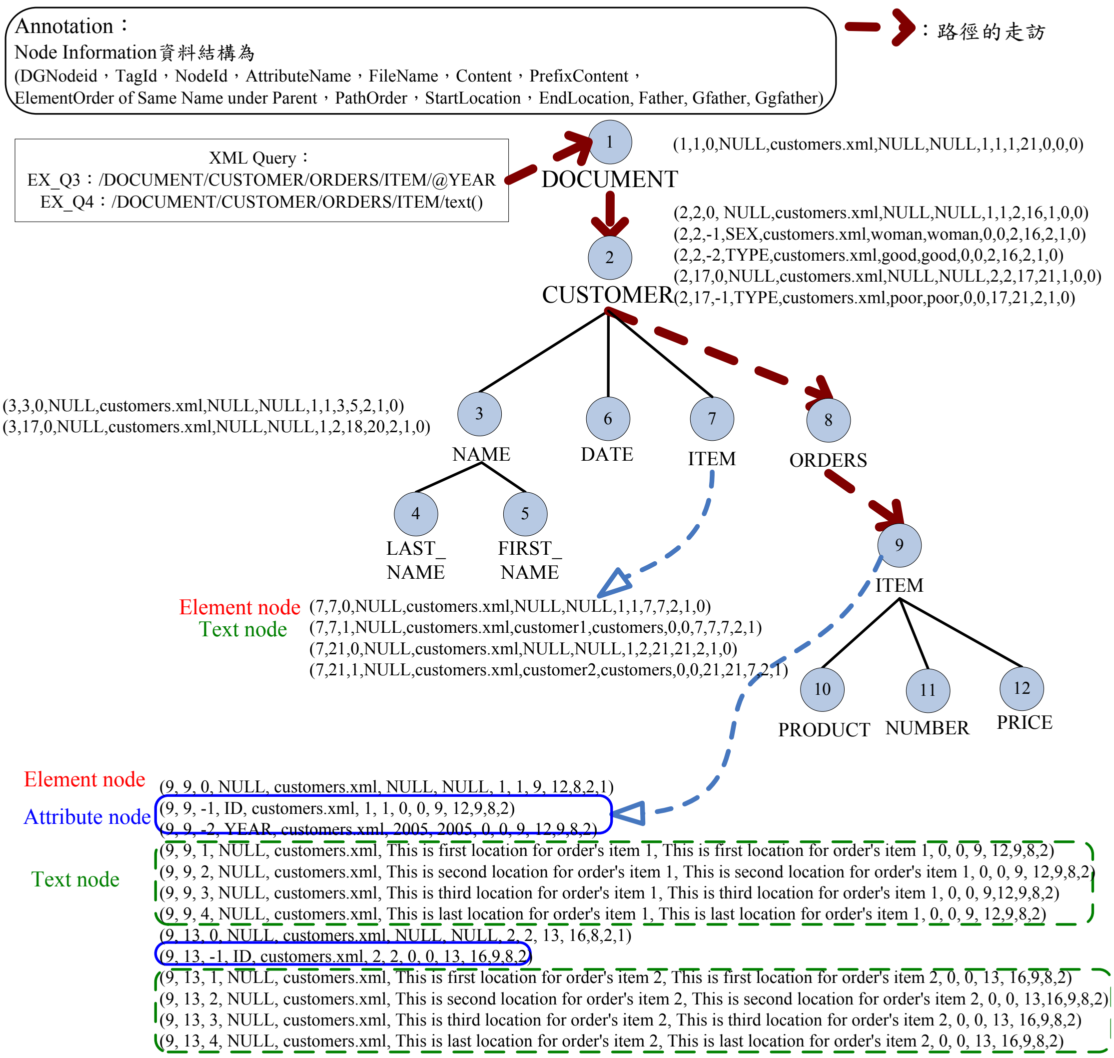


圖 3.4-6　EX_Q3、EX_Q4 透過 DataGuide 與 Node Information 進行路徑比對

我們以節點型態欄位「Attributename」作為屬性檢索的判斷。圖 3.4-7 為轉換後的 SQL commands。在針對 Text node 進行檢索時，以 EX_Q4：/DOCUMENT/CUSTOMER/ORDERS/ITEM/text()檢索條件，則針對 nodeid 改為大於 0 即可。(圖 3.4-6 中虛線框框表示)

```
select n9.filename, n9.tagid
from node_information n9
where n9.dgnodeid = 9 and n9.attributename = ' YEAR'
```

圖 3.4-7　EX_Q3 轉換後的 SQL commands

```
select n9.filename, n9.tagid
from node_information n9
where n9.dgnodeid = 9 and n9.nodeid > 0
```

圖 3.4-8 EX_Q4 轉換後的 SQL commands

在針對未指定節點名稱的 Element node(以‘*’表示)的情況下，則僅需在 DataGuide 裡，將指定未知節點的層級全部走訪過一遍後以求得可以繼續往下走訪的路徑即可。以 EX_Q5：/DOCUMENT/CUSTOMER/*/ITEM 檢索條件為例，走訪至 Element node 名稱為 CUSTOMER 的節點後，在未指定子節點名稱情況下往下走訪一層，故走訪路線會針對 CUSTOMER 節點下的每一個子節點(DataGuide 編號 3、6、7、8)皆詢問是否擁有 ITEM 的子節點，若有則繼續走訪之(見圖 3.4-6)。所以此檢索條件仍然會走訪至 DataGuide 節點編號 9 的位置。

3.4.2.4 節點內容值檢索條件

根據 XPathCore 的 grammar 中 Predicate 表示，使用者可以針對某個節點進行 predicate 檢索。Predicate 中 Comparison 由兩個 ArithExpr 與一個 CompareOp 所組成，可以進行節點內容值或節點位置的比對。以檢索條件 EX_Q6：/DOCUMENT/CUSTOMER/ORDERS/ITEM[text() = “This is last location for order's item 1”]為例，檢索條件為經由指定路徑走至 Element node 名稱為 ITEM 後，針對 ITEM 節點作 Predicate 檢索(ITEM 節點下 Text node 執行內容值比對)，最後找出此 ITEM 節點的所屬標籤編號與所屬文件名稱。將此檢索條件與 DataGuide、Node Information 進行比對，在走訪至 DG 節點編號 9 後再配合上 Node Information 所屬的 Text node(以圖 3.4-6 虛線框框表示)，針對 Text node 所記載的 prefixcontent 欄位進行內容值比對。圖 3.4-9 為轉換後的 SQL commands(這裡我們應用 Attributename is NULL 作為屬性判斷，因為 Element node 與 Text node 的 Attributename 皆為 NULL)。針對 Attribute node 的內容值檢索亦增加 content 欄位進行內容值的比對即可。

```
select n9.filename, n9.tagid
from node_information n9
where n9.dgnodeid = 9 and n9.attribtename is null and
n9.prefixcontent = ' This is last location for order item 1'
```

圖 3.4-9　EX_Q6 轉換後的 SQL commands

3.4.2.5 多重路徑檢索條件

DataGuide 會引發多重路徑的路徑資訊遺失議題。何謂多重路徑？XML 檢索條件通常會產生兩條以上(多重)路徑走訪。以檢索條件 EX_Q7：/DOCUMENT/CUSTOMER/ORDERS/ITEM[PRODUCT/text() = "Shovel"]為例，其走訪的路徑包含了 DOCUMENT/CUSTOMER/ORDERS/ITEM 與/DOCUMENT/CUSTOMER/ORDERS/ITEM/PRODUCT/text()。而路徑資訊遺失，顧名思義即為應用 DataGuide 進行文件路徑的壓縮會導致某些路徑的資訊無法保留下來。

以 EX_Q7 為例，在走訪 DataGuide 後，前者路徑可得 DataGuide 編號 9 節點，對應至 Node Information 後可得標籤編號 9 與 13 的 Element node；後者路徑可得 DataGuide 編號 10 節點，對應至 Node Information 後可得標籤編號 10 與 14 的 Text node。將標籤編號 10 與 14 的 Text node 進行內容檢索後，可得知標籤編號 14 的 Text node 符合此內容條件。但是我們卻無法由 DataGuide(圖 3.4-2)得知，標籤編號 14 Text node 的父節點是標籤編號 9 或是 13？除非回到原始的文件中(圖 3.3-1)才可視出端倪。

■　節點之間關係比對

我們針對每一個文件節點皆記錄其節點在 XML 文件內的起始標籤位置(startlocation)與終止標籤位置(endlocation)，應用文件節點之間的位置去判斷兩個節點是否擁有祖孫關係(並非侷限於父子關係的比對)與處於同一份文件上的關聯性。在 EX_Q7 中，文件節點 9 的起始位置一定小於等於文件節點 10，而終點位置反之；相同的，文件節點 13 的起始位置一定小於文件節點 14，而終點位置亦反之；文件節點

9、10、13、14 皆處於相同一份文件內(請參考圖 3.4-11)。配合上起始標籤位置與終止標籤位置，圖 3.4-10 為 EX_Q7 轉換後的 SQL commands。

```
select n9.filename, n9.tagid
from node_information n9,n10
where n9.dgnodeid = 9 and n9.nodeid = 0 and
n10.dgnodeid = 10 and n10.attributename is null and n10.prefixcontent = ' Shovel '
and
n9.startlocation <= n10.startlocation and
n9.endlocation >= n10.endlcoation
```

圖 3.4-10　EX_Q7 轉換後的 SQL commands

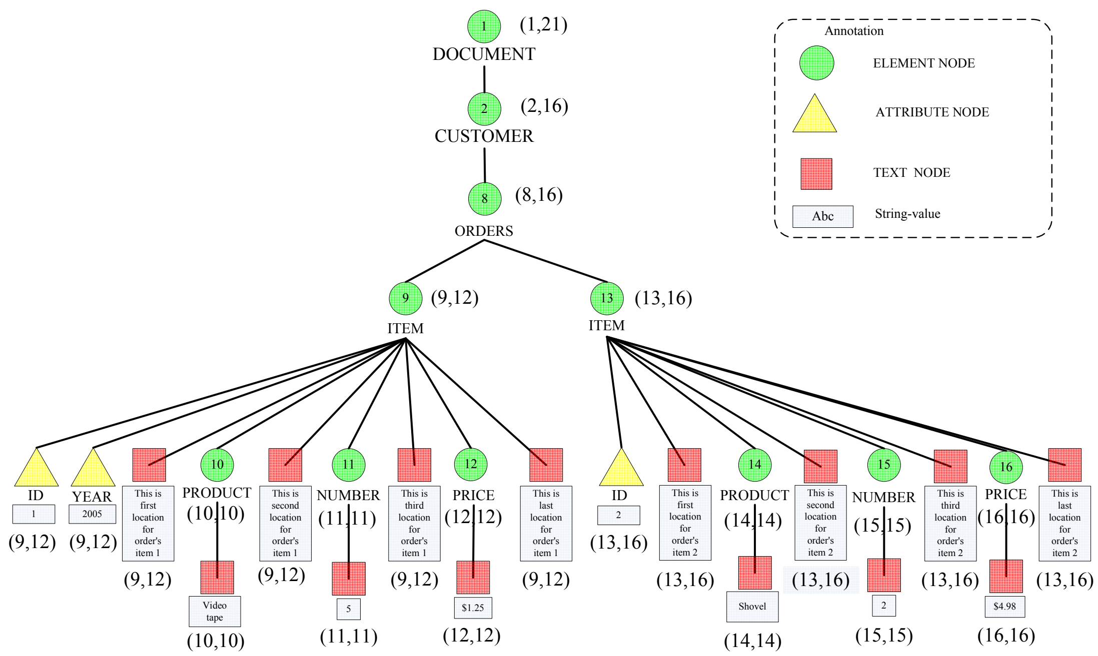


圖 3.4-11　部分範例文件「customers.xml」的文件資料模型圖

■　降低節點之間關係比對次數

在 XRel 方法中，資料庫表格 Path 儲存文件的路徑資訊，包含由根節點走訪至元素節點或屬性節點唯一的路徑資訊。根據這一些路徑資訊可以對應至其他型態的節點表格進行檢索。以檢索條件 EX8：/DOCUMENT/CUSTOMER/ORDERS/ITEM[@ID = '2']/PRICE/text()為例，檢索意義為找出訂單項目屬性編號內容值為 2，其價錢為何？此檢索條件屬多重路徑(三條路徑的結合)：

- /DOCUMENT/CUSTOMER/ORDERS/ITEM
- /DOCUMENT/CUSTOMER/ORDERS/ITEM/@ID
- /DOCUMENT/CUSTOMER/ORDERS/ITEM/PRICE/text()

圖 3.4-12 為 EX8 透過 XRel 轉換的 SQL commands 。由 SQL Commands 中可以看出某一個元素節點 e1 與其屬性節點 a2 亦需要進行 region 運算。*n* 條路徑即需要進行(*n*-1)次的 region 運算(即為 θ-join)與 *n* 次 equal-join 運算，這是 XRel 與所有「以節點為儲存資訊」的研究領域中，所要面對的一個造成檢索效能低落的議題。

```
select t3.docNAME, t3.start, t3.[end]
from Path p1, Path p2, Path p3, Element e1, Attribute a2, Text t3
where p1.pathexp = '/DOCUMENT/CUSTOMER/ORDERS/ITEM' and
p2.pathexp = '/DOCUMENT/CUSTOMER/ORDERS/ITEM/@ID' and
p3.pathexp = '/DOCUMENT/CUSTOMER/ORDERS/ITEM/PRICE' and
p1.pathID = e1.pathID and
p2.pathID = a2.pathID and
p3.pathID = t3.pathID and
e1.start < a2.start and e1.[end] > a2.[end] and
e1.start < t3.start and e1.[end] > t3.[end] and
a2.s_vaule = ‘2’
```

圖 3.4-12 EX8 透過 XRel 轉換的 SQL commands

本研究方法以 DataGuide 作為路徑走訪的索引，DataGuide 節點對應至文件標籤中的 Element node、Attribuete node 與 Text node，完全去除 equal-join 運算所花費的時間成本。在節點關係比對方面，本研究使用節點往上記錄三層的父節點標籤編號作為節點之間的關係比對。在剖析器剖析檢索子句時，同時判斷節點之間的層級是否在三層之內，應用欄位 father、gfather 與 ggfather 進行節點之間的判斷。將原先的 θ-join 轉換為 equal-join，以降低檢索時間。

```
select n12.filename, n12.tagid, n12.prefixcontent
from node_information n9, node_information n9_1, node_information n12
where n9.dgnodeid = 9 and n9.nodeid = 0 and
n9_1.dgnodeid = 9 and n9.attributename = 'id' and n9.prefixcontent = '2' and
n12.dgnodeid = 12 and n12.attributename is null and n12.nodeid > 0 and
n9.tagid = n9_1.father and
n9.tagid = n12.gfather
```

圖 3.4-13　EX8 透過本研究方法轉換的 SQL commands

3.4.2.6 節點順序(Ordinal)與節點間位置(Location)檢索條件

根據 XPathCore grammar 中 Predicate，檢索條件可以進行「兩個節點之間位置的比對」與「在相同父親節點下相同節點名稱之節點順序比對」；而 grammar 中 ParenthesizedExpr 可讓檢索條件進行「某節點由根節點向下走訪相同路徑資訊之路徑順序比對」。這小節我們說明如何應用 DataGuide 與 Node Information 支援這三種順序上與位置上的檢索比對。

以檢索條件 EX9：/DOCUMENT/CUSTOMER/ORDER/ITEM[2]為例，檢索意義為走訪至 Element node 名稱為 ORDER 的節點後，尋找底下擁有相同節點名稱且節點順序為 2 的 Element node。以圖 3.3-1 範例文件「customers.xml」的文件資料模型圖說明，所找出的標籤編號為 13 的 Element node。我們利用 Node Information 中 elementorderofsamenameunderparent 欄位支援這種節點順序上的比對。圖 3.4-14 為 EX9 轉換後的 SQL commands。

```
select n9.filename, n9.tagid
from node_information n9
where n9.dgnodeid = 9 and n9.nodeid = 0 and
n9.elementorderofsamenameunderparent = 2
```

圖 3.4-14　EX_Q9 轉換後的 SQL commands

以檢索條件 EX10：(/DOCUMENT/CUSTOMER/NAME)[2]/FIRST_NAME 為例，走訪相同路徑(/DOCUMENT/CUSTOMER/NAME)之路徑順序為 2 的 Element node

後，檢索此 Element node 下的 Element node 名稱為 FIRST_NAME 的節點標籤編號與所屬的文件名稱為何。檢索結果為圖 3.3-1 中的標籤標號 20 的 Element node，我們應用 pathorder 欄位支援這種路徑順序上的比對。圖 3.4-15 為轉換後的 SQL commands。

```
select n5.filename, n5.tagid
from node_information n3, n5
where n3.dgnodeid = 3 and n9.nodeid = 0 and n3.pathorder = 2 and
n5.dgnodeid = 5 and n5.nodeid = 0 and
n3.startlcoation <= 5.startlcoation and
n3.endlcoation >= n5.endlocation
```

圖 3.4-15　EX_Q10 轉換後的 SQL commands

至於兩個節點之間位置上的檢索為兩條路徑走訪至各自的某節點後，針對兩節點進行位置上的比對，「位置」意指節點在一份文件中所出現的位置。由於標籤編號採取 preorder traversal，故我們可以利用標籤編號來支援這一種檢索條件。以檢索條件 EX11：/DOCUMENT/CUSTOMER/NAME[[FIRST_NAME/text() =“Susan”] BEFORE [/DOCUMENT/CUSTOMER/NAME/FIRST_NAME/text() = “Nancy”]]為例，須檢索出 Element node 名稱為 NAME 節點標籤編號與所屬文件，此 NAME 節點擁有 FIRST_NAME 為 Susan 且位置要比另一個擁有 FIRST_NAME 為 Nancy 的 NAME 節點來得之前。以部份原始文件圖 3.4-16 說明，圖中粗體虛線為第一條路徑，方框為 Predicate；粗體實線為第二條路徑，方框亦為 Predicate。我們將符合 Predicate 條件的 Text 節點進行位置上的比對，圖 3.4-17 為轉換後的 SQL commands。

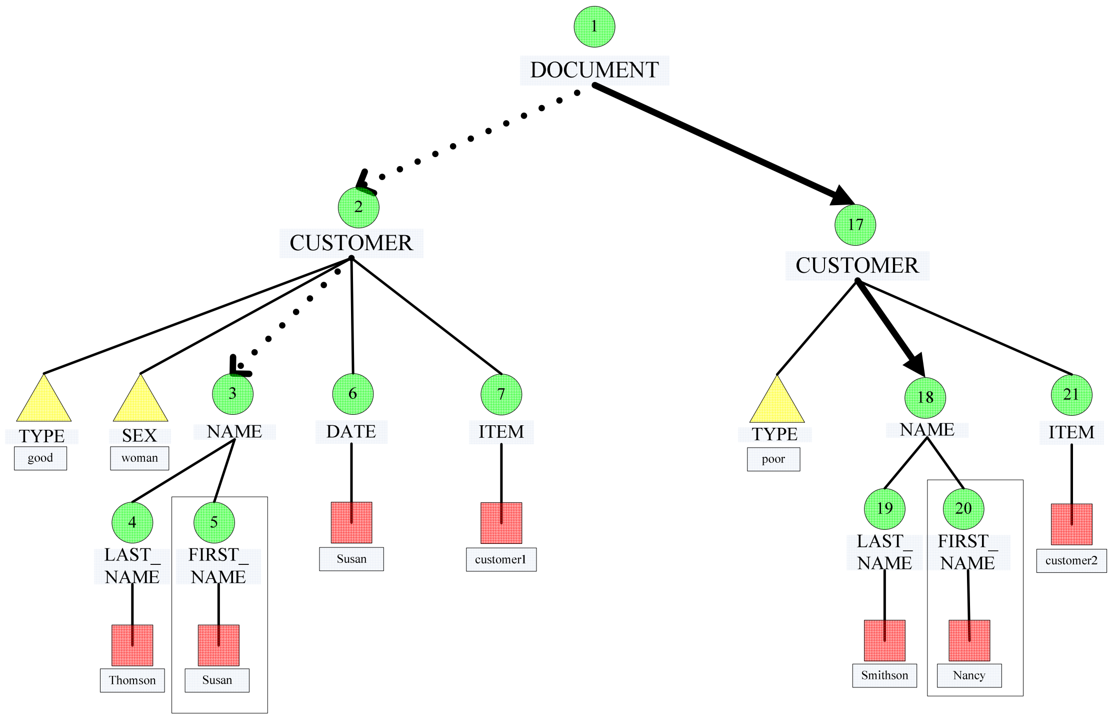


圖 3.4-16　EX_Q11 走訪路徑圖

select n3.filename, n3.tagid from node_information n3, n5_1, n5_2 where n3.dgnodeid = 3 and n3.nodeid = 0 and n5_1.dgnodeid = 5 and n5_1.attributename is null and n5_1.prefixcontent = 'Susan' and n3.startlocation <= n5_1.startlocation and n3.endlocation >= n5_1.endlocation and n5_2.dgnodeid = 5 and n5_2.attributename is null and n5_2.prefixcontent = 'Nancy' and n3.startlocation <= n5_2.startlocation and n3.endlocation >= n5_2.endlcoation **and** **n5_1.startlocation > n5_2.startlocation**

圖 3.4-17　EX_Q11 轉換後的 SQL commands

上述我們總共介紹五類檢索條件情況(路徑檢索條件、節點型態檢索條件、節點內容值檢索條件、多重路徑檢索條件、節點順序與節點間位置檢索條件)在進行語法轉換時，DataGuide 與 Node Information 的操作流程與 SQL commands 產生機制。配

合上 XPathCore grammar 規範，將適當的轉換程式撰寫入剖析器程式中，當剖析器剖析完 XPathCore 檢索子句後，即可產生所對應的 SQL commands。剖析檢索語言的實作與詳細的轉換演算法，我們將於第四章系統實作進行說明。

第四章　系統實作

本章針對檢索模組部分進行實作的探討(新增模組的實作說明請參考第三章)，其章節安排如下：

第一節闡述面臨了 XPathCore 檢索語言剖析實作，我們參考 Design Patterns 中 Interprreter 模型，針對某個標的語言定義出文法(XPathCore grammar)，以及設計出可解讀這種語句的解析器。

第二節介紹我們如何在 XPathCore 剖析的過程中，將轉換的機制安排進去，當剖析器完成剖析 XPathCore 子句後，亦將 XPathCore 子句轉換成相對應的 SQL commands。此節一共有三部分：a、將代表文件結構的 DataGuide 從關聯式資料庫重建、b、檢索時路徑走訪與 DataGuide 的比對、c、XPathCore grammar 中各個 Expression 的 SQL 轉換機制。

4.1　XPathCore 檢索語言剖析實作

針對剖析 XPathCore 檢索語言的實作方法，我們參考了 Design Patterns[10]中的 Interpreter。在 Interpreter 模式中教導使用者定義 regular expressions(之後簡稱為 regex)文法、表達某一個 regex、製作此 regex 的剖析器(或解譯器)。針對前兩部份，本研究參考文獻 XRel 制定 regex 文法與 W3C 的 XPath，而剖析器的實作架構如圖 4.1-1 所示。我們簡略地說明這個結構中的參與者內容與參與者之間的合作方式，在參與者後面以底線畫記的 expression 為對應本研究 XPathCore 的 Expression。

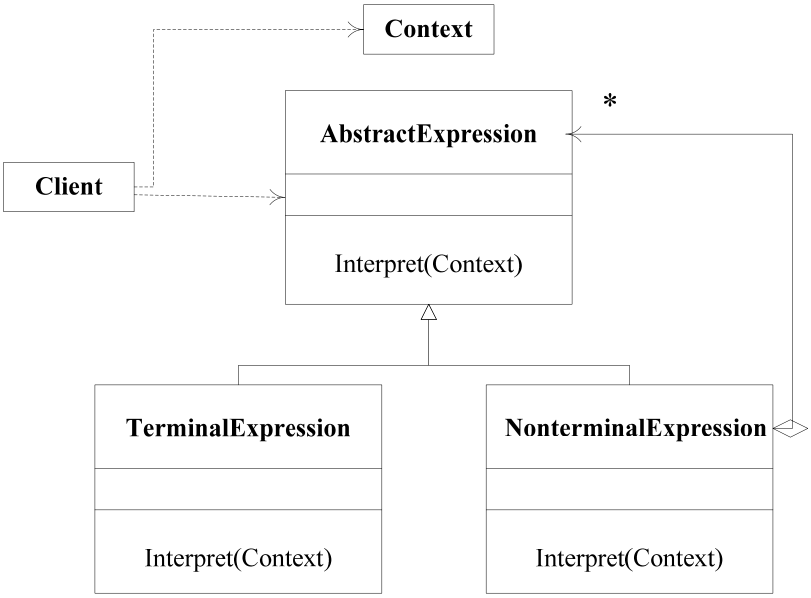


圖 4.1-1 Interpreter 結構

4.1.1 參與者內容

- **AbstructExpression** (RegularExpression)
  - 宣告抽象語法樹所有節點所共有的 Interpret()抽象操作。
- **TerminalExpression** (Step、BasicExpr、CompareOp)
  - 實作出終端符號所對應的 Interpret()操作
  - 需要持有每一個終端符號的物件個體。
- **NonterminalExpression** (Query、PathExpr、RegularExpr、Predicate、ParenthesizedExpr、Comparison、ArithExpr)
  - 針對 grammar 中每一條 Expression(R ::= R1R2…Rn)，各配上一個類別(也就是指 R)。
  - 針對 $R_1 \sim R_n$之間每一個符號，各設一個 AbstructExpression 型別的 Reference。
  - 針對每一個非終端符號，實作出對應的 Interpret()。Interpret( )通常會透過 $R_1 \sim R_n$ 的 Reference 遞迴呼叫自己。
- **Context**

- 整個剖析器所共見的全域性資訊。

- **Client**
  - 根據文法，建立(或給定)特定語句的抽象語法樹。抽象語法樹由 NonterminalExpression 與 TerminalExpression 類別的物件個體所構成。
  - 執行 Interpret()操作。

4.1.2 參與者之間的合作方式

- 針對給定的 regex，Client 負責建立起抽象語法樹並且會去對 Context 初始化，然後將抽象語法樹交給 NonterminalExpression 的 Interpret()執行。
- 每一個 NonterminalExpression 節點都會去使用到自己的子運算式的 Interpret()來製作自己的 Interpret()。每一個 TerminalExpression 的 TerminalExpression 則定義遞迴終止的情況。
- 每一個節點的 Interpret()會用到 Context 來存取剖析器的狀態。

4.2 XPathCore 檢索語言轉換機制

4.2.1 DataGuide 重建

DataGuide 代表所有 XML 文件在路徑上的壓縮結構，在程式執行時處於記憶體內，並且隨著 XML 文件新增而不斷地修正結構，並適時儲存於關聯式資料庫的 DataGuide table 中。圖 4.2-1 為 DataGuide 重建的虛擬碼，說明如何將關聯式資料庫中的 DataGuide table 重新組合於記憶體內，以供檢索模組進行路徑走訪的比對。此虛擬碼接收一個資料庫連線(*jdbc*)，回傳一個 DataGuide 結構(*dgArray_R*)。

```
RECONSTRUCTDG(jdbc)
    rs ← EXECUTEQUERY(jdbc, "select count(*) from dataguide")
    while NEXT(rs)
        count ← GET(rs , 1)    //抓第 1 個欄位
    dgArray _R[] ← DATAGUIDE_i , i ← 0 to (count + 1)
    dgArray_R[0] ← NEWDATAGUIDE("root")
    SETDGNODEID(dgArrat_R[0]) ← 0
    e ← 1
    rs1 ← EXECUTEQUERY(jdbc, "select * from dataguide")
    while NEXT(rs1)
        id ← GET (rs1 , 1)
        dgArray_R[e] = NEWDATAGUIDE (GET (rs1 , 2))
        SETDGNODEID (dgArray_R[e]) ← id
        e++
    rs2 ← EXECUTEQUERY (jdbc, "select * from dataguide")
    while NEXT (rs2)
        nodeid ← GET(rs2,1)
        parented ← GET(rs2,3)
        for m ← 0 to (count + 1)
            if GETDGNODEID(dgArray_R[m]) = = nodeid
                for j ← 0 to (count + 1)
                    if GETDGNODEID (dgArray_R[j]) = = parented
                        SETDGPARENT_R(dgArray_R[m]) ← dgArray_R[j]
            if GETDGNODEID(dgArray_R[m]) = = parented
                for k ← 0 to (count + 1)
                    if GETDGNODEID (dgArray_R[j]) = = nodeid
                        SETDGCHILD_R(dgArray_R[m]) ← dgArray_R[j]
      return dgArray_R
```

圖 4.2-1　重建 DataGuide 的虛擬碼

4.2.2　路徑走訪與 DataGuide 的比對

我們將 XML 檢索中路徑走訪部份與記憶體中的 DataGuide 進行比對，圖 4.2-2 為比對路徑走訪與 DataGuide 的虛擬碼，此虛擬碼接受起始節點(*source*)、目標節點名稱(*target*)，回傳目標節點陣列(*targetnode*)。

```
TraverseDG(source, target)   //走訪型態 type 由其它函式指定
    targetnode[]
    if type = = 1   //走訪單一層級
        if source != null && target = = "*"
            for i ← 0 to GetChildNum(source)
                targetnode[] ← GetChildRef(source , i)
        return targetnode
        else if source != null && target != null
            for j ← 0 to GetChildNum (source)
                if GetChildName(source,   j) = = target
                    targetnode[]← GetChildRef (source, i)
                    return targetnode
                else if   j = = (GetChildNum (source) + 1)
                    return null;
    else if type = =2   //走訪多層級
        if source != null && target = = "*"
            return error;
        else if source != null && target != null
            targetnode[]← DoubleTraverse(source, target)
            return targetnode

DoubleTraverse (s, t)   //source 節點與 target 節點名稱
    targetnode1[]
    for k ← 0 to GetChildNum (s)
        if GetChildName (s,k) = = target
            targetnode1[] ← GetChildRef (s, k)
            DoubleTraverse (GetChildRef (s,k),t)
        else
            DoubleTraverse (GetChildRef (s,i),t)
    return targetnode1[]
```

圖 4.2-2 比對路徑走訪與 DataGuide 的虛擬碼

4.2.3 各個 Expression 的 SQL 轉換機制

XPathCore 剖析器接受 XPathCore 檢索子句，判斷檢索子句語法(grammar)是否正確以產生抽象語法樹。在抽象語法樹中，每一個節點(expression)都會記錄節點狀態(意指在 grammar 中子 expression 的狀態)。我們根據每個節點狀態的不

同，將進行不同的 SQL 轉換機制。

接下來我們將針對每一個 expression 所要處理的 SQL 轉換機制進行虛擬碼描述與說明，而在判斷每一個 expression 語法是否正確與所處狀態則省略不寫。針對轉換機制有下列重點：

- 在 XML 文件結構中，每一個節點皆代表一條走訪的路徑；而每一條走訪的路徑皆會轉換成一個 SLQ command。實作時將所產生的 SQL command 記錄於新產生的 DataGuide 節點物件內。
- 每一個 expression 的狀態(type)編號對應至 XPathCore grammar(請參考圖 3.4-1)中的 expression 狀態。例如：某個 CompareOp expression object 狀態為 3，則表示其中內容值為“BEFORE”。
- 根據抽象語法樹結構，每一個 expression 皆會執行自己的子 expression 的 SQL 轉換函式。

4.2.3.1 Query

Query 代表一條完整的 XPathCore 表示式。在這個 expression 中僅有一個正確的狀態發生「一個 Query 就是一個子 PathExpr」(請參考圖 3.4-1)。轉換機制部分會將子 PathExpr 所有走訪的終端節點進行 SQL 轉換並且將所有完整的 SQL commands 以 union 結合。圖 4.2-3 為虛擬碼，它接受子 PathExpr(*q_pathexpr*)，回傳轉換後的 SQL commands 字串。

```
QRUEYTRANSFER(q_pathexpr)
    if q_type = = 1    //情況為 Query：：= PathExpr 時
        qkeynode ← q_pathexpr. PATHEXPRTRANSFER()    //執行子 pathexpr 的 SQL 轉換函式並且將值指派給 qkeynode
        keynodesize ← GETKEYNODESIZE(qkeynode)

        for i ← 0 to keynodesize
            nodeid ← GETKEYNODEID(qkeynode, i)    //qkeynode 中第 i 個節點編號
            ADDSELECT(nodeid) ← "n" + nodeid + ".filename, n" + nodeid + ".tagid"
            ADDFROM(nodeid) ← "node_information n" + nodeid
            ADDWHERE(nodeid) ← "n" + nodeid + ".dgnodeid = " + nodeid + "and n" + nodeid + ".nodeid = 0"

        for j ← 0 to keynodesize
            keynode ← GETKEYNODE(qkeynode,j)
            select_command ← GETSELECT(keynode)
            from_command ← GETFROM(keynode)
            where_command ← GETWHERE(keynode)
            if keynodesize = =1 | j = = (keynodesize-1)
                return "( select" + select_command + "from" + from_command + "where" + where_command + ")"
            else
                return "( select" + select_command + "from" + from_command + "where" + where_command    + ") union"
```

圖 4.2-3　Query expression 轉換 SQL 之虛擬碼

4.2.3.2 PathExpr

PathExpr 在 XPathCore grammar 中代表完整的"路徑"表示式，路徑勢必會走訪至某些節點集合(我們稱為 pkeynode)。在 PathExpr 中一共有五種不同的路徑狀況(請參考表格 3.4-1)，說明如下：

- 狀況一：表示此 PathExpr 即為一個 RegularExpr，而 RegularExpr 即是構成路徑主要的架構。

➢ 狀況二與狀況三：表示此 PathExpr 由文件根節點開始走訪，透過一層或是多層走訪。

➢ 狀況四與五：表示此 PathExpr 由 ParenthesizedExpr 開始，將進行「某節點由根節點向下走訪相同路徑資訊之路徑順序比對」之檢索條件。

接下來我們針對每種路徑情況進行不同的處理與 SQL 轉換機制。圖 4.2-4 為虛擬碼，它接受子 RegularExpr、子 ParenthesizedExpr、子 Predicates(*p_regularexpr*, *p_parenthesizedexpr*,*p_preidcate*)，回傳所走訪路徑的終端節點陣列(*pkeynode*)。

```
PATHEXPRTRANSFER(p_regularexpr, p_parenthesizedexpr,p_preidcate)
    p_source, p_target   //路徑走訪由 p_source 節點開始,至 p_target 節點
    pkeynode[]    //宣告 pkeynode 為記錄路徑走訪的節點
    if p_type = = 1
        SETSOURCENODE(p_regularexpr) ← p_source
        pkeynode[] ← p_regularexpr.REGULAREXPRTRANSFER()   //執行子
p_regularexpr 的 SQL 轉換並將其 keynode 指派給 pkeynode
        return pkeynode

    else if p_type = = 2
        p_source ← RECONSTRUCTDG    //由 DG 根節點開始走訪
        SETTRAVERSEDGTYPE(1)   //指定走訪型態為單層走訪
        SETSOURCENODE(p_regularexpr) ← p_source
        pkeynode[] ← p_regularexpr.REGULAREXPRTRANSFER()
        return pkeynode

    else if p_type = = 3
        p_source ← RECONSTRUCTDG
        SETTRAVERSEDGTYPE (2)   //指定走訪型態為層級未知走訪
        SETSOURCENODE(p_regularexpr) ← p_source
        pkeynode[] ← p_regularexpr.REGULAREXPRTRANSFER()
        return pkeynode

    else if p_type = = 4
        a[] ← p_parenthesizedexpr.PARENTHESIZEDEXPRTRANSFER()
        par_nodesize ← GETKEYNODESIZE(a)
        if GETTYPE(p_preidcate) != 1   //沒有 predicateExpr
            for i ← 0 to par_nodesize
```

```
                SetTraverseDGType(1)   //指定走訪型態為單層走訪
                SetSourceNode(p_regularexpr) ← GetKeynode(a,i)
                pkeynode[] ← p_regularexpr. RegularexprTransfer()
            return pkeynode
        else   //有 predicateExpr
            for j ← 0 to par_nodesize
                SetSourceNode (p_predicate) ← GetKeynode(a,j)
                p_predicate.PredicateTransfer()   //執行 p_predicate 轉換 SQL

                if GetType (p_preidcate.pr_comparison) = = 1   //假如[]放數字
                    par_nodeid ← GetKeynodeID(a, j)
                    AddFrom(nodeid) ← "node_information n" + nodeid
                    AddWhere(nodeid) ← "n" + nodeid + ".dgnodeid = " + nodeid + "and n" + nodeid + ".nodeid = 0"
                    AddWhere (nodeid) ← "n" + nodeid + ".pathorder = " + GetNumber(p_predicate)

                SetTraverseDGType (1)
                SetSourceNode(p_regularexpr) ←GetKeynode (a, j)
                b[] ←p_regularexpr. RegularexprTransfer() //存放 p_regularexpr 的 keynode
                //每次傳回的 keynode，皆與目前 p_source 進行位置比對
                for r ← 0 to GetKeynodeSize (b)
                    reg_nodeid ← GetKeynodeID(b, r)
                    AddWhere (reg_nodeid) ← "n" + reg_nodeid + ".filename =    n" + nodeid + ".filename and n" + reg_nodeid + ".startlocation >= n" +nodeid + ".startlocation and n" + reg_nodeid + ".endlocation <= n" + nodeid + ".endlocation"
                pkeynode[] ← b
            return pkeynode

    else if p_type = = 5
        //修正 p_type == 4 的程式中的 SetTraverseDGType ()為 2 即可，省略
```

圖 4.2-4　PathExpr 轉換 SQL 之虛擬碼

4.2.3.3 RegularExpr

RegularExpr 由一個以上的 Step 與零個以上的 Predicate 所組成，由‘ /’做為區隔。RegularExpr 涵義有二：a、一條路徑是由許多個 Step 所組成，b、應用 Predicate 可以達到支援 RE 的功能。在 RegularExpr 中一共有三種狀況，舉凡有遇到‘/’或‘//’都需要進行路徑比對。圖 4.2-5 為虛擬碼，它接受子 Step(*r_step*)、子 Predicate(*r_predicate*)和子 RegularExpr(*r_regularexpr*)，回傳路徑走訪尾端的節點集合(rkeynode)。

```
REGULAREXPRTRANSFER(r_step, r_predicate, r_regularexpr)
    r_source, r_target   //來源節點，目標節點
    r_keynode[]   //存放由 Step 找到的關鍵節點
    nodesize ← GETKEYNODESIZE(r_keynod)

    if r_type = = 1   //狀況一
        if GETTYPE(r_step) = = 1   //當 Step 為 Element node
            e[] ←TRAVERSEDG(r_source, r_step.STEPTRANSFER())   //目標節點
            SETKEYNODETYPE(e[]) ← ‘Element’ //目標節點設定型態為 Element
            r_keynode ← e
        else if GETTYPE(r_step) = = 2   //當 Step 為 Attribute node
            SETKEYNODETYPE(e[]) ← ‘ATT’
            r_keynode ← e
        else if GETTYPE(r_step) = = 3   //當 Step 為不限節點名稱時(*)
            e[] ←TRAVERSEDG(r_source, r_step.STEPTRANSFER())
            SETKEYNODETYPE(e[]) ← ‘Text’
            r_keynode ← e
        else if GETTYPE(r_step) = = 4   //當 Step 為 Text node
            SETKEYNODETYPE(e[]) ← ‘Text’
            r_keynode ← e

        //將之前存放在 r_source 的 SQL 資訊傳遞至最新節點(r_keynode)上
        for i ← 0 to nodesize
```

```
            ADDSELECT(r_keynode) ← GETSELECT(r_source)
            ADDFROM(r_keynode) ← GETFROM (r_source)
            ADDWHERE(r_keynode) ← GETWHERE (r_source)

        if GETTYPE(r_predicate) = = 1    //有 predicateExpr 情況
            for j ← 0 to nodesize
                nodeid ← GETKEYNODEID(r_keynode, j)
                SETSOURCENODE(r_predicate) ←GETKEYNODE(r_keynode,j)
                r_predicate.PREDICATETRANSFER()    //執行 r_predicate 轉換 SQL

                if GETTYPE (r_predicate.pr_comparison) = = 1//假如[]放數字
                    ADDWHERE(nodeid) ← "and n" + nodeid + ".elementorderofsameunderparent = " + GETNUMBER(r _predicate)
                else
                    //先將原先 SQL 資訊移除，在將[]內的 SQL 命令一併複製
                    CLEARFROM (nodeid)
                    CLEARWHERE (nodeid)
                    pre_ nodesize ← GETKEYNODESIZE (r_predicate)
                    for q ← 0 to pre_ nodesize
                        pre_ nodeid ← GETKEYNODEID(r_predicate, q)
                        ADDWHERE(pre_ nodeid) ← "n" + pre_ nodeid + ".filename =    n" + nodeid + ".filename and n" + pre_ nodeid + ".startlocation >= n" +nodeid + ".startlocation and n" + pre_ nodeid + ".endlocation <= n" + nodeid + ".endlocation"
                        ADDFROM(r_keynode) ← GETFROM (pre_ nodeid)
                        ADDWHERE(r_keynode) ← GETWHERE (pre_ nodeid)
        return r_keynode
    else if r_type = = 2    //狀況二，Text node 與 Attribute node 僅出現在狀況一
        if GETTYPE(r_step) = = 1    //當 Step 為 Element node
            e[] ←TRAVERSEDG(r_source, r_step.STEPTRANSFER())    //目標節點
            SETKEYNODETYPE(e[]) ← 'Element' //目標節點設定型態為 Element
```

```
            r_keynode ← e
        else if GetType(r_step) = = 3    //當 Step 為不限節點名稱時(*)
            e[] ←TraverseDG(r_source, r_step.StepTransfer())
            SetKeynodeType(e[]) ← 'Text'
            r_keynode ← e
        for i ← 0 to nodesize
            AddSelect(r_keynode) ← GetSelect(r_source)
            AddFrom(r_keynode) ← GetFrom (r_source)
            AddWhere(r_keynode) ← GetWhere (r_source)

        if GetType(r_predicate) != 1    //沒有 predicateExpr 情況
            for j ← 0 to nodesize
                SetSourceNode(r_regularexpr) ←GetKeynode(r_keynode,j)
                SetTraverseDGType(1)    //指定走訪型態為單層走訪
                pkeynode[] ← r_regularexpr. RegularexprTransfer()
            return pkeynode
        else if GetType(r_predicate) = = 1    //有 predicateExpr 情況
            for m ← 0 to nodesize
                nodeid ← GetKeynodeID(r_keynode, m)
                SetSourceNode(r_ predicate) ←GetKeynode(r_keynode,m)
                r_predicate.PredicateTransfer()    //執行 r_predicate 轉換 SQL
                if GetType (r_predicate.pr_comparison) = = 1//假如[]放數字
                    AddFrom(nodeid) ← "node_information n" + nodeid
                    AddWhere(nodeid) ← "and n" + nodeid + ".elementorderofsameunderparent = " + GetNumber(r _predicate)
                    SetSourceNode(r_regularexpr) ←GetKeynode(r_keynode,m)
                SetTraverseDGType(1)    //指定走訪型態為單層走訪
                pkeynode[] ← r_regularexpr. RegularexprTransfer()
                return pkeynode
                else    //Predicate 的其他情況
                    ClearFrom (nodeid)
                    ClearWhere (nodeid)
                    pre_ nodesize ← GetKeynodeSize (r_predicate)
```

```
                    for q ← 0 to pre_ nodesize
                        pre_ nodeid ← GETKEYNODEID(r_predicate, q)
                        ADDWHERE(pre_ nodeid) ← "n" + pre_ nodeid + ".filename =   n" + nodeid + ".filename and n" + pre_ nodeid + ".startlocation >= n" +nodeid + ".startlocation and n" + pre_ nodeid + ".endlocation <= n" + nodeid + ".endlocation"
                        ADDFROM(pre_ nodeid) ← "node_information n" + pre_ nodeid
                        ADDWHERE(pre_ nodeid) ← "n" + pre_ nodeid + ".dgnodeid = " + pre_ nodeid + "and n" + pre_ nodeid + ".nodeid = 0"
                        ADDFROM(r_keynode) ← GETFROM (pre_ nodeid)
                        ADDWHERE(r_keynode) ← GETWHERE (pre_ nodeid)
                    SETSOURCENODE(r_regularexpr) ← GETKEYNODE(r_keynode,m)
                    SETTRAVERSEDGTYPE(1)   //指定走訪型態為單層走訪
                    reg_ nodesize ← GETKEYNODESIZE (r_regularexpr)
                    for p ← 0 to reg_ nodesize
                        reg_ nodeid ← GETKEYNODEID(r_regularexpr, p)
                        ADDWHERE(r_regularexpr,) ← "n" + reg_ nodeid + ".filename =   n" + nodeid + ".filename and n" + reg_ nodeid + ".startlocation >= n" +nodeid + ".startlocation and n" + reg_ nodeid + ".endlocation <= n" + nodeid + ".endlocation"
                    pkeynode[] ← r_regularexpr.REGULAREXPRTRANSFER()
                return pkeynode
    else if r_type = = 3   //狀況三
        //修正 p_type == 2 的程式中的 SETTRAVERSEDGTYPE ()為 2，省略
```

圖 4.2-5　RegularExpr 轉換 SQL 之虛擬碼

4.2.3.4 Step

Step 在 XPath grammar 中稱為位置步驟(Location step)，可以代表元素節點、屬性節點、節點名稱未知的節點(*)與文字節點，以'/'或'//'為區隔。Step 並沒有執行轉換 SQL 的工作，它僅負責提供此 Step 是上述四種情況中的哪一種。

4.2.3.5 Predicate

Predicate 對指定節點作 predicate，由'['、']'包含 Comparison，在轉換 SQL 呼叫時直接呼叫自己的子 pr_comparison，並且將子 pr_comparison 的 keynode 指派給自己的 keynode 並且回傳。

4.2.3.6 ParthesizedExpr

表示針對一段路徑作 predicate，而在 ParthesizedExpr 後面的子 predicate 僅可以為數字。在轉換 SQL 呼叫時直接呼叫自己的子 par_pathexpr，並且將子 par_pathexpr 的 keynode 指派給自己的 keynode 並且回傳。

4.2.3.7 BasicExpr

BasicExpr 在 XPath grammar 中為基本的運算子，可以代表 literal、數字(number)與 Null。Step 並沒有執行轉換 SQL 的工作，它僅負責提供此 BasicExpr 是上述三種情況中的哪一種。

4.2.3.8 Comparison

Comparison 為一個完整的比較式，可以由單一子 arithexpr 或由兩個子 arithexpr 與一個子 compareop 所組成。在前者情況的子 arithexpr 僅可為 BasicExpr，而後者情況則依子 arithexpr 的情況進行不同條件的比較。為虛擬碼，接受兩個子 arithexpr 與一個子 compareop，回傳路徑走訪尾端的節點陣列(*arith_1keynodeid*)。

```
ComparisonTransfer(c_arithexpr1, c_arithexpr2, c_compareop)
    c_source, c_target
    c_arithkeynode1[],c_arithkeynode2[]
    if c_type = = 1
        if GetType(c_arithexpr1) = = 1    //假如 c_arithexpr1 為 BasicExpr 情況
            c_arithexpr1.ArithexprTransfer()    //執行 c_arithexpr1 的 SQL 轉換
        else
            //在 Comparison 中有錯誤，情況一僅可擺放 BasicExpr
    else if c_type = = 2
        SetSourceNode(c_arithexpr1) ← c_source
        if GetType(c_arithexpr1) = = 1
            //在 Comparison 中有錯誤，c_arithexpr1 不可以是數字
        else if GetType(c_arithexpr1) = = 2
            c_arithexpr1.ArithexprTransfer()
            c_arithkeynode1[] ←GetKeynode(c_arithexpr1)
            arith_1keynodesize ← GetKeynodeSize(c_arithkeynode1)
        c_compareop.COmpareopTransfer()    //執行子 c_compareop 的 SQL 轉換
        if GetType(c_arithexpr2) = = 1    // c_arithexpr2 為 BasicExper
            c_arithexpr2.ArithexprTransfer()
            if GetType(c_arithexpr2.a_basicexpr) = = 1    // c_arithexpr2 為 literal
                for i ← 0 to arith_1keynodesize
                    arith_1keynodeid ← GetKeynode(c_arithexpr1,i)
                    AddFrom(arith_1keynodeid) ← "node_information n" + arith_1keynodeid
                    if GetType(c_compareop) = = 1    //operator 為等號,字串比對
                        AddWhere(arith_1keynodeid) ← "n" + arith_1keynodeid + ".dgnodeid = " + arith_1keynodeid + "and n" + arith_1keynodeid + ".attributename is null and n " + arith_1keynodeid + ".preficContent = " + c_arithexpr2.literal
            else if GetType(c_arithexpr2.a_basicexpr) = = 3    // c_arithexpr2 為 literal
            for i ← 0 to arith_1keynodesize
                arith_1keynodeid ← GetKeynode(c_arithexpr1,i)
```

```
                ADDFROM(arith_1keynodeid) ← "node_information n" + arith_1keynodeid
                if GETTYPE(c_compareop) = = 4 || GETTYPE(c_compareop) = = 5   //operator 為 IS 或 ISNOT
                    ADDWHERE(arith_1keynodeid) ← "n" + arith_1keynodeid + ".dgnodeid = " + arith_1keynodeid + "and n" + arith_1keynodeid + ".attributename is null and n " + arith_1keynodeid + ".preficContent IS( | ISNOT) NULL"

        else if GETTYPE(c_arithexpr2) = = 2   // c_arithexpr2 為路徑
            c_arithexpr2.ARITHEXPRTRANSFER()
            c_arithkeynode2[] ←GETKEYNODE(c_arithexpr2)
            arith_2keynodesize ← GETKEYNODESIZE(c_arithkeynode2)
            for i ← 0 to arith_1keynodesize
                arith_1keynodeid ← GETKEYNODE(c_arithexpr1,i)
                for k ← 0 to arith_2keynodesize
                    arith_2keynodeid ← GETKEYNODE(c_arithkeynode2,k)
                    GETTYPE(c_compareop) = = 3   //  位置比較_BEFORE
                    ADDFROM(arith_1keynodeid) ← "node_information n" + arith_1keynodeid
                    ADDFROM(arith_1keynodeid) ← "node_information n" + arith_2keynodeid
                    ADDWHERE(arith_1keynodeid) ← "n" + arith_1keynodeid + ".startlocation > " + arith_2keynodeid + ".startlocation"
         return arith_1keynodeid   //將收集到的 SQL 加以回傳
```

圖 4.2-6 Comparison 的轉換 SQL 之虛擬碼

4.2.3.9 CompareOp 與 Arithexp

CompareOp 表示了一個運算式中的運算元，可以針對在 Comparison 中兩側的 Arithexpr 作不同的運算，包括路徑的比對、字串的比對、內容值是否為空。CompareOp 在 SQL 轉換僅提供運算元所出現的情況為何。而 Arithexpr 而針對自己的子 a_basicExpr 或子 a_pathexpr 進行 SQL 轉換呼叫，並且回傳所走訪路徑的末端節點集合(akeynode)。

4.2.4 小結

在檢索語言轉換機制這一節，我們介紹了如何將 DataGuide 從關聯式資料庫重新組合於記憶體內以供檢索模組進行路徑走訪的比對。在路徑比對後會回傳符合指定路段的節點集合，集合內為空表示沒有符合指定路段的節點集合。而在 SQL 轉換機制，我們依每一個 expression 所代表語意，適時的設計相對應的 SQL 轉換語法於其中， 當剖析器剖析完 XPathCore 子句後即轉換為相對應的 SQL commands，最後再將代表此 XPathCore 子句的 SQL commands 交給關聯式資料庫系統處理，即可獲取回傳值。

第五章　效能測試與分析

5.1　前言

為了驗證本研究方法在檢索時是否具有可行性與高效能性，本章將進行一系列效能上的實驗測試並且針對實驗結果進行數據報告與效能分析。本章的實驗測試分為兩個 benchmarks：

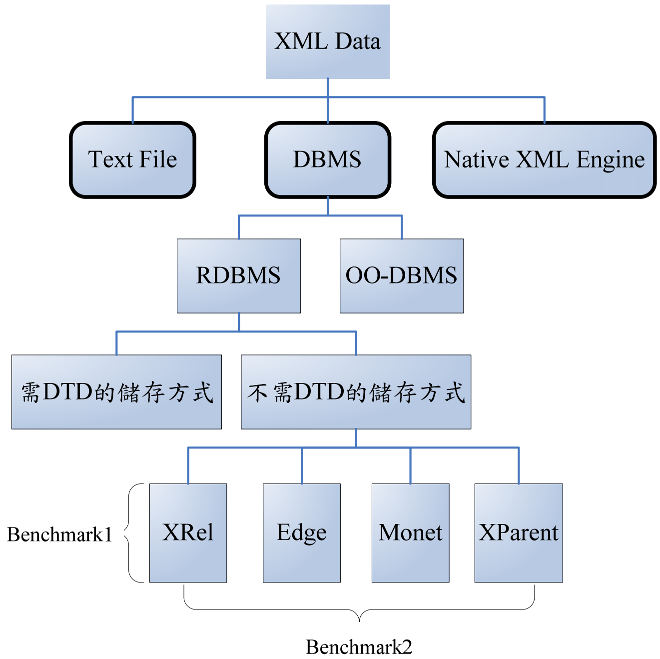


圖 5.1-1　實驗測試圖

- Shakespeare Benchmark：與 XRel 研究相比。我們選擇了由 YOSHIKAWA, M. 所提出的 XRel 方法，XRel 方法所處研究的領域(以 RDBMS 為儲存架構)、假設(不使用 DTD)和設計精神(儲存文件節點為主)皆和本研究較為相關 [請參考 2.4.2.5　節]。我們實作本研究方法與 XRel 方法，並且針對 XRel 文獻中的 Benchmark 進行效能測試與比對，藉由與 XRel 方法比較以突顯本研究設計的精髓與執行高效能性。

- XMark Benchmark ：與相同研究領域中所代表的方法(Edge、Monet、XParent、XRel)相比。將評估的範圍擴展至相同的研究領域(以 RDBMS 為儲存架構)與研究假設(不使用 DTD)。我們參考了文獻[16]中所提出的檢索效能數據，利用本研究方法進行文獻[16]中的 Benchmark 測試，測試環境採取相似於文獻[16]的測試環境，並根據所測試的結果進行討論。

接下來章節架構如下：第二節為針對實驗測試環境進行描述；第三節介紹了兩個實驗測試所使用到的 Benchmarks；第四節探討在實驗測試時，一些實作上的議題；第五節說明 XML 檢索語言與 SQL 語言的關係，並展示兩個 benchmarks 的測試結果並進行效能上的評估與說明。

5.2 實驗測試環境

測試機器方面，所有的實驗皆建立於單一顆 1.7GHz CPU、1.0G Memory 的個人電腦上，作業系統為 Windows XP。開發工具方面，程式語言採用 Java SDK 5.0[20]與 Apache 的 Xerces DOM Parser[46]搭配 Eclipse Platform 3.0.1[12]整合開發環境，關聯式資料庫管理系統則為 Microsoft SQL Sever 2000，使用 JDBC 進行資料庫連結。檢索語言則以 XPathCore grammar 為主。

5.3 使用的 Benchmark 簡介

5.3.1 Shakespeare Benchmark

Shakespeare Benchmark[17]被引述於 XRel 文獻[16]中，將文學家莎士比亞的作品集以 XML 格式進行表示。每一本作品即為一個 XML 文件檔案，一共有 37 份 XML 文件檔。文件資料型態以 Document-based 為主，強調文章結構的先後順序。附錄四為此測試資料的 DTD。表格 5.3-1 為 Shakespeare 測試資料詳細的描述， 表中可以看出此測試資料並無屬性的存在，整份文件結構呈現扁平狀(分支數最大為 406 而深度僅有 5)。

表格 5.3-1　測試資料 Shakespeare 詳細介紹

文件數量(份)	37
總容量(MB)	7.65
平均容量(KB)	506.71
文件中標籤節點總數量(個)	179,689
文件中屬性節點總數量(個)	0
文件中文字節點總數量(個)	147,442
唯一路徑數量(條)	57
文件中最大的分支數	406
文件中最小的分支數	1
文件中最大的深度	5
文件中最小的深度	1

針對檢索條件部分，XRel 文獻提供 8 條檢索條件以對效能進行分析。表格 5.3-2 為 8 條檢索條件表示式與其檢索涵義：

表格 5.3-2　8 條檢索條件表示式與其檢索涵義

檢索條件表達式(以 XPath 表示)	檢索涵義
Q1　/PLAY/ACT	單一路徑表示式(短)
Q2　/PLAY/ACT/SCENE/SPEECH/LINE/STAGEDIR	單一路徑表示式(長)
Q3　//SCENE/TITLE	一個層及未知走訪
Q4　//ACT/*/TITLE	一個層級未知走訪與 一個未指定節點名稱走訪
Q5　/PLAY/ACT[2]	相同父親節點下相同名稱之節點順序比對
Q6　(/PLAY/ACT)[2]/TITLE	由根節點向下走訪相同路徑資訊之路徑比對
Q7　/PLAY/ACT/SCENE/SPEECH[SPEAKER = ‘CURIO’]	節點內容值檢索
Q8　/PLAY/ACT/SCENE[*/SPEAKER = 'Steward']/TITLE	一個未指定節點名稱走訪與節點內容值檢索

Q1~Q4 主要是針對路徑的走訪，分別為簡短路徑走訪、長路徑走訪、層級未知的情況下走訪與未指定節點名稱的走訪。Q5 與 Q6 則為節點順序上的比對。Q7、Q8 都以節點內容值為檢索條件並且有多重路徑的走訪。

5.3.2　XMark Benchmark

此 Benchmark[1]被引述於文獻[16]中，資料來源為真實世界的一個拍賣網站資訊。文件呈現階層式元素結構、references 與 text。附錄五為此 XMark 的 DTD。表格 5.3-3 為 XMark 測試資料的詳細描述。由表格 5.3-3 中可知，XMark 產生一個大的 XML 文件；而 Shakespeare 則為 37 份較小的 XML 文件所組成。XMark 文件結構呈現扁平狀，更甚於 Shakespeare(因為分支度數量高達 25,500 而深度最深為 12)。

表格 5.3-3　測試資料 XMark 詳細介紹

文件數量(份)	1
總容量(MB)	113
平均容量(KB)	113
文件中標籤節點總數量(個)	1,666,315
文件中屬性節點總數量(個)	381,878
文件中文字節點總數量(個)	1,173,732
唯一的路徑數量(含屬性) (條)	548
文件中最大的分支數	25,500
文件中最小的分支數	1
文件中最大的深度	12
文件中最小的深度	3

XMark 定義了 20 條檢索條件以對效能進行分析，下列為簡略的介紹：

1. Return the name of the item with ID 'item20748' registered in North America.**(exact match)**
2. Return the initial increases of all open auctions. **(ordered access)**
3. Return the first and current increases of all open auctions whose current increase is a least twice as high as the initial increase. **(ordered access)**
4. List the reserves of those open auctions where a certain person issued a bid before another person. **(ordered access)**
5. How many sold items cost more than 40? **(casting)**
6. How many items are listed on all continents? **(regular path expression)**
7. How many pieces of prose are in our database? **(regular path expression)**
8. List the names of persons and the number of items they bought. **(chasing references, join)**
9. List the names of persons and the names of the items they bought in Europe. **(chasing references, join)**
10. List all persons according to their interest; use French markup in the result. **(construction of complex results)**
11. For each person, list the number of items currently on sale whose price does not exceed 0.02% of the person's income.
12. For each richer-than-average person, list the number of items currently on sale

whose price does not exceed 0.02% of the person's income.

13. List the names of items registered in Australia along with their descriptions.
14. Return the names of all items whose description contains the word "gold". **(full text)**
15. Print the keywords in emphasis in annotations of closed auctions. **(path traversals)**
16. Return the IDs of the sellers of those auctions that have one or more keywords in emphasis. **(path traversal)**
17. Which persons don't have a homepage? **(missing elements)**
18. Convert the currency of the reserve of all open auctions to another currency. **(function application)**
19. Give an alphabetically ordered list of all items along with their location. **(sorting)**
20. Group customers by their income and output the cardinality of each group.

在每條檢索條件後的圓括弧粗體字為檢索涵義的說明。此二十條檢索條件包含了許多情況的檢索，有精準的節點內容值比對、節點順序比對、路徑走訪、多重路徑走訪、reference 比對、複雜版面展示、函式轉換測試、排序…等。透過這 20 條檢索條件，實驗研究者能較為客觀且全面性地測試各種 XML 研究方法的可行性與效能性。

5.4 測試實作議題之探討

本節探討在實驗測試時所面臨一些實作上的議題，分為兩個部份：

- XRel 實作：為了進行第一個實驗測試，我們以相同開發工具與測試環境實作 XRel 方法。表格 5.4-1 為 XRel 資料庫表格定義，其演算法請參考 XRel 文獻。
- 關聯式資料庫管理系統中索引的建置：對於所處研究的領域以 RDBMS 為儲存架構、研究假設為不使用 DTD 的 XRel 方法與本研究方法，我們均針對資料庫會使用到的屬性建立資料庫索引以提升檢索效率。表格 5.4-2 為 XRel 索引建置。表格 5.4-3 為本研究方法索引建置。但由於在 Microsoft SQL Sever 2000 中索引值允許的最大長度為 900 個位元組，所以我們針對 text 內容值將擷取前 256Bytes 獨立為一個新的欄位並且對它作索引，如 XRel 中的 s_value 屬性與本研究中的 prefixContent 屬性。

表格 5.4-1 XRel 資料庫表格定義

```
create table Path ( --建立 Path 表格
        pathID int not null,
        pathexp varchar(500) not null,
        primary key(pathID)
)

create table Element ( --建立 Element 表格
        docNAME varchar(50) not null,
        pathID    int not null,
        start     int not null,
        [end]         int not null,
        [order]     int not null
        primary key(docNAME, [start], [end]),
        foreign key(pathID)    --關聯 Path 表格中的 pathID
        references Path
)

create table Attribute    ( --建立 Attribute 表格
        docNAME varchar(50) not null,
        pathID    int not null,
        start     int not null,
        [end]     int not null,
        [value] varchar(1100) not null,
```

```
        s_value char(256) not null,
        primary key(docNAME, pathID, start, [end]),
        foreign key(pathID)    --關聯 Path 表格中的 pathID
        references Path
)

create table Text ( --建立 Text 表格
        docNAME varchar(50) not null,
        pathID int not null,
        start int not null,
        [end] int not null,
        [value] varchar(1100) not null,
        s_value char(256) not null,
        primary key(docNAME, start, [end]),
        foreign key(pathID)    --關聯 Path 表格中的 pathID
        references Path
)
```

表格 5.4-2　XRel 資料庫索引建置

```
create index index1 on Element(pathID,[order],start,[end])
create index index2 on Path(pathID, pathexp)
create index index3 on [Text](pathID, start,[end],s_value)
create index index4 on Attribute(pathID, start,[end],s_value)
```

表格 5.4-3　本研究方法資料庫索引建置

```
create index index1 on node_information
(dgnodeid,nodeid,filename,startlocation,endlocation)--針對 Element node(路徑)作檢索

create index index2 on node_information
(dgnodeid,attributename,prefixcontent,filename,startlocation,endlocation)--針對 Element node 底下的 Attribute node 與 text node 作檢索

create index index3 on node_information
(dgnodeid, nodeid, pathorder,filename,startlocation,endlocation)--針對由根節點向下走訪相同路徑資訊之路徑的節點順序作檢索

create index index4 on node_information
(dgnodeid,nodeid,elementorderofsameunderparent,filename,startlocation,endlocation)--針對相同父親下相同節點名稱的節點順序作索引

create index index5 on node_information
(dgnodeid,nodeid,filename,father,gfather,ggfather)--針對 Element node(路徑)作檢索

create index index6 on node_information
(dgnodeid,attributename,prefixcontent,filename,father,gfather,ggfather)--針對 Element node 底下的 Attribute node 與 text node 作檢索

create index index7 on node_information
(dgnodeid, nodeid, pathorder,filename,father,gfather,ggfather)--針對由根節點向下走訪相同路徑資訊之路徑的節點順序作檢索

create index index8 on node_information
(dgnodeid,nodeid,elementorderofsameunderparent,filename,father,gfather,ggfather)--針對相同父親下相同節點名稱的節點順序作索引
```

5.5 測試結果與分析

5.5.1 XML Query 與 SQL commands 的關聯性

在進入測試結果分析前，我們先說明 XML Query 與 SQL commands 之間的關聯性以幫助我們更了解測試所顯示結果為何如此。舉凡 XML Query Benchmark 皆脫離不了下列檢索情況：

- 路徑走訪：由節點 a 走訪至節點 b 的路徑檢索。
- 節點型態與內容值比對：走訪至某節點後，此節點型態與內容值之檢索。
- 多重路徑走訪：路徑之間必須確定是否有關聯性(更詳述說明即為兩路徑的終端節點是否屬於同一個祖先節點)。
- 節點順序比對：以樹狀結構為主的 XML 文件有順序的觀念在其中，包括了節點的順序，路徑的順序，節點位置的比對。
- 其它：函式應用、排序、版面的呈現、全文檢索。

我們針對上述五種檢索情況，以資料庫綱要設計與轉換相對應的 SQL commands 說明四種方法(Edge、XParent、Monet、XRel)的優缺點以及探討本研究採取策略。

5.5.1.1 路徑走訪

XML 文件與 RDB 表格最大的差別在於前者為階層性結構而後者為扁平式結構。如何在 XML 文件與 RDB 表格之間進行拆解與重組，一直攸關於路徑走訪的檢索效能。

以 XRel 而言，將唯一路徑資訊儲存至 Path 表格中，使用字串的比對以達到路徑的走訪(可透過資料庫索引建立增進檢索效能)。對 XParent 與 Edge 儲存邊資訊為主的研究，進行路徑走訪時必須透過屬性 Lable(在 Edge 中)或屬性 Path(在 XParent 中)進行路徑資訊比對，再利用屬性 Source 與屬性 Target(在 Edge 中)或屬

性 Pid 與屬性 Cid(在 XParent)進行路徑層級的走訪。上述兩種方法在進行路徑走訪都需付出大量的表格合併成本(equal-join)。Monet 採取以路徑資訊為表格名稱的策略，針對路徑走訪時僅需針對某一個符合此路徑的表格進行檢索即可。

由上述可知，在面臨較長路徑走訪時，以儲存邊的 XParent 與 Edge 需要花費在大量的 equal-join。而在使用屬性值比對的 XRel 或直接以路徑資訊為表格名稱的 Monet 則可以直接走訪到符合路徑資訊的節點上。

本研究將文件結構資訊從 RDB 中抽離，使用 DataGuide 在記憶體中進行路徑走訪，以求得較佳的路徑走訪效能。

5.5.1.2 節點型態比對

在 XML Query Benchmark 中，某路徑走訪終端節點型態一共有三：Element 節點、Text 節點與 Attribute 節點，如何記錄這三種型態的節點內容值是每種研究方法所需討論的議題。

以 Egde 而言，記錄每條邊的表格 Edge 中，以屬性 Flag 記錄這條邊的終端節點是否為葉節點(即為 Text 節點與 Attribute 節點)，所以要進行節點型態比對需針對屬性 Flag 進行比對以獲取記錄節點內容值的屬性 Value 的值為何。XParent 則是以屬性 Path 記錄路徑是否走訪至 Text 節點與 Attribute 節點，再透過表格 Element 與表格 Text 中的屬性 PathID 進行 equal-join。Monet 將路徑資訊為表格名稱的策略，也同時將路徑資訊記錄至最後的節點型態，如 Element 節點、Text 節點或 Attribute 節點，檢索時僅需針對某一個符合此路徑的表格進行檢索即可。至於 XRel 採取以表格 Path 中的屬性 Pathexp 進行路徑資訊比對，路徑資訊記錄至最後的節點型態，包括 Element 節點與 Text 節點，再透過屬性 PathID 與其他三個表格 Element、Text、Attribute 中的屬性 PathID 進行 equal-join 以找出符合的節點。

由上述可知在節點型態比對方面，Egde 透過屬性 Flag 內容值的比對。XParent 透過屬性 Path 內容值的比對與其他表格進行 equal-join。Monet 透過表格名稱的

比對。XRel 則透過的屬性 Pathexp 與其他表格進行 equal-join。

本研究方法將三種節點型態以屬性 nodeid 與屬性 attributename 巧妙地安排在表格 node_information 中，以減少節點之間內容值比對所需 table join。更利用 RDB 中的檢索建置加快檢索的速度。當 DataGuide 走訪至 DG 節點時，即透過屬性 nodeid 與屬性 attributename 和屬性 dgnodeid 對表格 node_information 進行檢索，避免像 XRel 或 XParent 的 equal-join 所花費的時間成本。

5.5.1.3 多重路徑走訪

檢索情況常常發生走訪路徑不僅僅為單一條的路徑，而是走訪多條的路徑，而在路徑之間必須確定是否有關聯性(更詳述說明即為兩路徑的終端節點是否屬於同一個祖先節點)。

以 Edge、XParent 和 Monet 而言，路徑之間關聯性以來源節點與目標節點的編號(在 Edge 中以屬性 Source 和屬性 Target；在 XParent 中以屬性 Pid 和屬性 Cid；在 Monet 中以屬性 Source 和屬性 Target)加以判別。透過來源節點與目標節點的比對需要經過 equal-join。

而以 XRel 來看，則需去判斷代表走訪至路徑末端節點之間的關係為何，透過兩個節點的 start 和[end]比對去判別是否屬於同一個祖先節點。透過 start 與[end]的比對需要 θ-join。

由上述可知，equal-join 可以使用 index join 或 sort-merge join 以達到加速檢索的速度；而 θ-join 必須使用 nested-loop join，這就是不同方法在面臨多重路徑走訪時所採取的策略。

本研究是以儲存邊資訊為主的方法，勢必會面臨如 XRel 在多重路徑所產生的節點關係比對。為了避免 θ-join 所產生大量的時間成本，我們考慮了檢索時多重路徑之間節點層級數與現實生活中的需求，記錄了每個節點向上三層的父親、祖父、曾祖父的節點編號，透過這三個屬性的比對，便能將原先的 θ-join 轉換為 equal-join(當節點之間層級數超過三層時，我們依舊採取 region 比對)。

5.5.1.4 順序比對

在 RDB 中表格的內容並沒有順序的觀念，而以樹狀結構為主的 XML 文件則有順序的觀念在其中，包括了節點的順序，路徑的順序，節點位置的比對。

在 Edge、XParent 和 Monet，雖然都有屬性 Ordinal，但此資訊記錄這個邊在自己所處的兄弟邊當中的順序，而 XRel 則以屬性[order]記錄在相同父親下相同節點名稱的節點順序。

本研究除了記錄相同父親下相同節點名稱的節點順序外，更透過屬性 nodeid 記錄 Attribute 節點與 Text 節點在相同父親下的節點順序，pathorder 屬性則記錄以根節點至某一節點的相同路徑資訊的節點順序。

5.5.1.5 其它

在其它方面，檢索情況可能會去測試函式應用、排序、版面的呈現、全文檢索等等，這些情況皆和研究設計較無絕對的關聯性。我們不予討論。

透過上述針對 XML Query 的檢索情況進行分析後，我們進行實驗測試並且驗證本研究方法在不同的檢索情況是否已採取最佳的策略。

5.5.2 測試 Shakespeare Benchmark

5.5.2.1 資料庫建立分析

我們首先探討本研究方法與 XRel 方法針對 Shakespeare Benchmark 建立資料庫所花費的時間成本與空間容量。其中測試構面有：

- 程式執行的時間：測試內容為研究方法處理 XML 文件並且將文件轉換資料庫欄位資料的整個過程，此階段並無將資料新增至資料庫中。
- 資料新增的時間與容量：測試其研究方法處理 XML 文件並將資料新增至資料庫中所需的時間與資料庫容量。

- 索引建置時間：測試其研究方法建立索引所需的時間與資料庫容量。
- 資料庫欄位數量：表示資料庫表格中所記錄的資料筆數。

表格 5.5-1 為資料庫欄位數量的比較，表格 5.5-2 為本研究方法與 XRel 方法在資料庫建立分析與比較。由表格 5.5-1 可以看出，本研究在 dataguide 表格與 XRel 在 path 表格所儲存的資料筆數皆為 57 筆，而本研究所儲存的是 DataGuide 的節點資訊，而 XRel 所儲存的是文件唯一的路徑資訊。在本研究的 node_information 表格所儲存的資料筆數為 XRel 的其他表格(element、[text]、attribute)的總和資料筆數。本研究將節點資訊結合在同一個表格內，利用欄位 nodeid 與 attributeneme 巧妙地判別節點型態，利用資料庫索引的建置可以加快同一個表格內資料之間的比對。

由表格 5.5-2 可以看出，本研究所花費的總執行時間與空間較多，其主要原因在於本研究在索引方面的建置較多，而欄位資料也相對 XRel 多一些。但此花費是值得的，因為在檢索時本研究方法可以提供更多的資訊以降低檢索上的時間花費。

表格 5.5-1　針對 Shakespeare Benchmark 建立資料庫欄位數量的比較

方法	欄位名稱	tuple 數量
本研究方法	dataguide	57
	node_informarion	327,131
XRel	path	57
	element	179,689
	[text]	147,656
	attribute	0

表格 5.5-2　針對 Shakespeare Benchmark 在資料庫建立分析與比較

方法	構面	花費時間(seconds)	花費空間(MB)
本研究方法	程式執行	6.828	0
	新增資料	1357.875	596.44
	建立索引	175	408.25
	總執行	1532.875	1004.69
XRel	程式執行	3.469	0
	新增資料	1226.344	278.63
	建立索引	50	113.68
	總執行	1276.344	392.31

5.5.2.2 檢索測試分析

此小節我們針對每一條檢索子句皆連續測試執行 1000 次後求平均值的時間，而時間以千分之一秒(毫秒，millisecond)為單位。表格 5.5-3 為本研究方法與 Xrel 方法針對 Shakespeare Benchmark 中的 8 條檢索條件所測試的檢索時間。圖 5.5-1 為比較兩種方法的比較折線圖。 表格 5.5-4 為兩種方法詳細的 SQL commands 描述。由圖 5.1-1 可以得知本研究的整體檢索效能均表現得比 XRel 卓越，其最主要的差別為本研究方法在某些檢索條件下可以縮減 XRel 一半的表格合併的時間。我們以這八條檢索子句進行說明。

表格 5.5-3 檢索效能比對(單位為 毫秒)

檢索條件	本研究方法	XRel	效能提升(倍)	改善策略
Q1	1.687	2.65	0.570835803	A
Q2	3.953	4.69	0.186440678	A
Q3	5.5	19.69	2.58	A
Q4	6.125	19.69	2.214693878	A
Q5	0.64	1.41	1.203125	C
Q6	2.391	1693.91	707.4525303	C
Q7	0.937	179.37	190.4300961	A,B
Q8	1.281	253.75	197.0874317	A,B
Sum	22.514	2175.16	95.61366261	-
(註)：改善策略 A：DataGuide；B：多重路徑(轉換為 equal-join)；C：ordinal 資訊儲存；D：儲存所有節點資訊於同個表格內				

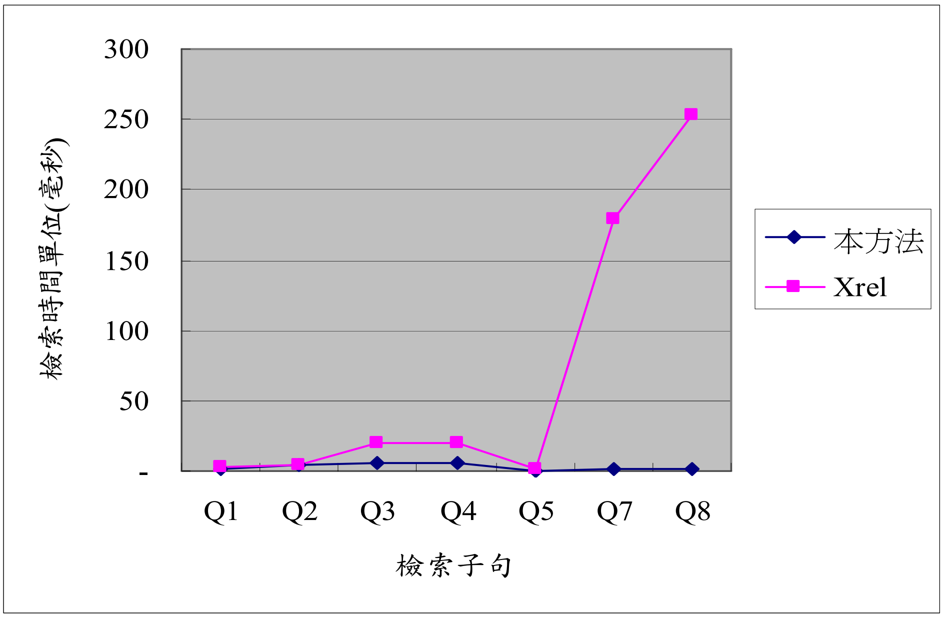


圖 5.5-1 比較兩方法的折線圖(省略 Q6)

針對 Q1、Q2 路徑走訪檢索條件，而 XRel 方法針對路徑檢索是採取 Path 表

格中的“字串”比對，比對後再將 Path 表格與其他表格進行 equal-join 動作，故整體執行成本偏高。以表格 5.5-4 的 Q1 說明，XRel 需要針對 Path 表格進行路徑：#/PLAY#/ACT 的字串比對，再將 Path 表格中的 pathID 欄位與 Element 表格中的 pathID 欄位進行 equal-join 以找出符合此路徑的元素節點。對於透過 XRel 處理的 Q1，需要一個字串比對與一個 equal-join。而本研究透過 DataGuide 進行路徑走訪，配合 Node_Information 表格產生 SQL commands。

針對 Q3、Q4 層級未知的路徑走訪檢索條件，本研究方法利用 DataGuide 產生所有符合路徑條件的 SQL commands 並且使用 Union 進行所有子句的聯集。而 XRel 方法針對路徑檢索採用“like”以找出所有可能的值組後，再與其他表格進行 join 動作。.

針對 Q5 相同父親節點下相同節點名稱的節點順序檢索條件，本研究方法與 XRel 方法檢索時間相近，雖然前者減少一個 join 動作，但檢索時間相差不大。

針對 Q6 由根節點向下走訪相同路徑資訊之路徑的節點順序的檢索條件，本研究將此路徑順序的資訊記載至每一個文件節點內，故在檢索時僅需針對 pathorder 屬性進行條件的比對即可。而 XRel 僅儲存相同父親節點下相同節點名稱的節點順序資訊，對於由根節點開始走訪的相同路徑名稱順序的檢索條件並無幫助，故使用子檢索(subquery)配合 NOT EXISTS 進行這類型的檢索操作。

針對 Q7 與 Q8 多重路徑與節點內容值的檢索條件，本研究方法掌握減少 join 操作的優勢，可以完全的降低檢索的時間成本。在 Q7，XRel 方法需要三個 equal-join 與兩個 θ-join(共五個)；在 Q8，XRel 方法需要五個 equal-join 與四個 θ-join(共九個)。而本研究方法對於 Q7 與 Q8 分別僅需兩個與四個 equal-join 操作，所以在檢索上可以大幅的降低時間成本(請參考表格 5.5-4)。

表格 5.5-4　兩種方法的 SQL commands

	本研究方法	XRel
Q1	select n13.filename, n13.tagid from node_information n13 where n13.dgnodeid = 13 and n13.nodeid = 0	select e1.docname, e1.start, e1.[end] from Element e1, Path p1 where p1.pathexp = '#/PLAY#/ACT' and e1.pathid = p1.pathid
Q2	select n21.filename, n21.tagid from node_information n21 where n21.dgnodeid = 21 and n21.nodeid = 0	select e1.docname, e1.start, e1.[end] from Element e1, Path p1 where p1.pathexp = '#/PLAY#/ACT#/SCENE#/SPEECH#/LINE#/STAGEDIR' and e1.pathid = p1.pathid
Q3	(select n16.filename,n16.tagid from node_information n16 where n16.dgnodeid = 16 and n16.nodeid = 0) UNION (select n51.filename,n51.tagid from node_information n51 where n51.dgnodeid = 51 and n51. nodeid = 0)	select e1.docname, e1.start, e1.[end] from Element e1, Path p1 where p1.pathexp like '#%/SCENE#/TITLE' and e1.pathid = p1.pathid
Q4	(select n16.filename,n16.tagid from node_information n16 where n16.dgnodeid = 16 and n16.nodeid = 0) UNION (select n24.filename,n24.tagid from node_information n24 where n24.dgnodeid = 24 and n24. nodeid = 0) UNION (select n38.filename,n38.tagid from node_information n38 where n38.dgnodeid = 38 and n38. nodeid = 0)	select e1.docname, e1.start, e1.[end] from Element e1, Path p1 where p1.pathexp like '#%/ACT#/%/TITLE' and e1.pathid = p1.pathid
Q5	select n13.filename,n13.tagid from node_information n13 where n13.dgnodeid = 13 and n13.nodeid = 0 and n13.elementorderofsamenameunderparent = 2	select e1.docname, e1.start, e1.[end] from Path p1, Element e1 where p1.pathexp = '#/PLAY#/ACT' and e1.pathid= p1.pathid and e1.[order] = 2
Q6	Select n14.filename,n14.tagid from node_information n14, node_information n13 where n14.dgnodeid = 14 and n14.nodeid = 0 and n13.dgnodeid = 13 and n13.nodeid = 0 and n13.pathorder = 2 and n14.filename = n13.filename and n14.father = n13.tagid	select e1.docname, e1.start, e1.[end] from Path p1, Path p3, Element e1, Element e3 where p1.pathexp = '#/PLAY#/ACT' and p3.pathexp = '#/PLAY#/ACT#/TITLE' and e1.pathid = p1.pathid and e3.pathid = p3.pathid and exists(select * from Path p11, Element e11 where p11.pathexp = '#/PLAY#/ACT' and e11.pathid = p11.pathid and e11.docname = e1.docname and e11.start < e1.start) and

		not exists(select * from Path p11, Element e11,Path p12, Element e12 where p11.pathexp = '#/PLAY#/ACT' and p12.pathexp = '#/PLAY#/ACT' and e11.pathid = p11.pathid and e12.pathid = p12.pathid and e11.docname = e1.docname and e12.docname = e1.docname and e11.start < e12.start and e12.start < e1.start) and e1.docname = e3.docname and e1.start < e3.start and e1.[end] > e3.[end]
Q7	select n18.filename,n18tagid from node_information n18, node_information n19 where n18.dgnodeid = 18 and n18.nodeid = 0 and n19.dgnodeid = 19 and n19.attributename is null and n19.prefixcontent = 'CURIO' and n19.filename = n18.filename and n18.tagid = n19.gfather	select e1.docname, e1.start, e1.[end] from Path p1, Path p3, Element e1, Text t3 where p1.pathexp = '#/PLAY#/ACT#/SCENE#/SPEECH' and p3.pathexp = '#/PLAY#/ACT#/SCENE#/SPEECH#/SPEAKER' and e1.pathid = p1.pathid and t3.pathid = p3.pathid and e1.start < t3.start and e1.[end] > t3.[end] and e1.docname = t3.docname and t3.s_value = 'CURIO'
Q8	select n16.filename, n16.tagid from node_information n16, node_information n15, node_information n19 where n16.dgnodeid = 16 and n16.nodeid = 0 and n15.dgnodeid = 15 and n15.nodeid = 0 and n19.dgnodeid = 19 and n19.attributename is null and n19.prefixcontent = 'Steward' and n16.filename = n15.filename and n19.filename = n15.filename and n16.father = n15.tagid and n19.ggfather = n15.tagid	select e5.docname, e5.start, e5.[end] from Path p1, Path p3,Path p5, Element e1, Element e5, Text t3 where p1.pathexp = '#/PLAY#/ACT#/SCENE' and p3.pathexp like '#/PLAY#/ACT#/SCENE#/%#/SPEAKER' and p5.pathexp = '#/PLAY#/ACT#/SCENE#/TITLE' and e1.pathid = p1.pathid and t3.pathid = p3.pathid and e5.pathid = p5.pathid and e1.start < t3.start and e1.[end] > t3.[end] and e1.docname = t3.docname and e1.start < e5.start and e1.[end] > e5.[end] and e1.docname = e5.docname and t3.s_value = 'Steward'

5.5.3 測試 XMark benchmark

我們將評估的範圍擴展至相同的研究領域(以 RDBMS 為儲存架構)與研究假設(不使用 DTD)。由於本實驗並無完全實作該領域中代表的四種方法(Edge、Monet、XParent、XRel)，故我們利用本研究方法進行文獻[16]中的 Benchmark 測試，再參考了文獻[16]所提出的檢索效能數據以進行比較。

文獻[16]提出的檢索效能數據分別實驗於兩個商業性資料庫：RDB_A 與 RBB_B。為了使測試方法可以達到最佳的檢索效能，文獻[16]應用了 RDB_A 中的 Fix-point operator 以支援路徑走訪時未知的 join；RBB_B 中的 Connect BY 子句以應付同一個表格的自我遞迴性的關聯。

本研究在檢索語言實作 XPathCore，而文獻[16]則是以實作文獻[21]中的 XQuery。XPath 與 XQuery 在檢索本質上皆有路徑走訪、多重路徑走訪、內容值比對..等，兩者主要差別在於 XQuery 可允許使用者自訂回傳的版面與迴圈的使用(XQuery 稱之為 FLWR：for、let、where、return)。我們將測試結果與文獻[16]合併討論，目標主要有兩個：

- 測試本研究方法面對 XMark Benchmark 中 20 條檢索子句時，是否可以全方位完整達到各種不同情況的檢索需求。
- 利用較低階的實驗機器，將本研究所測試的數據，用轉譯後的 SQL 語法進行全面性的效能評估。

在實驗測試環境設備(本研究使用單一顆 1.7GHz CPU、1G Memory、作業系統為 Windows XP；文獻[16]使用單一顆 2.0GHz CPU、1G Memory、作業系統為 Windows XP)。

5.5.3.1 檢索測試分析

表格 5.5-5 中 Thesis 為本研究方法進行文獻[16]中的 Benchmark 二十條檢索測試所得的時間；而其他四種方法為文獻[16]測試 RDB_A 所提供的檢索時間數

據。表格 5.5-6 是與文獻[16]測試 RDB_B 所提供的檢索時間數據相比。附錄六為 XMark Benchmark 檢索子句與相對應的 DataGuide 圖形。附錄七為本研究方法在 XMark Benchmark 所產生的 20 條 SQL commands。附錄八為文獻[16]所測試的全部方法的檢索時間數據。

表格 5.5-5 XMark RDB_A 檢索效能分析表(單位為 秒，'-'表示檢索失敗)

Query	Thesis	XRel	XParent	Edge	Monet	排名	改善策略
Q1	0.197	0.39	0.54	4.46	0.76	1	A,B,D
Q2	0.286	0.57	15.09	20.85	141.72	1	A,B,C,D
Q3	1.900	-	33.58	36.06	579.28	1	A,B,C,D
Q4	0.083	4,485.52	82.73	411.81	2.68	1	A,B,D
Q5	0.01	0.50	0.33	30.06	0.10	1	A,B,D
Q6	0.390	6.50	0.79	13.92	0.19	2	A
Q7	2.160	4.50	2.44	12.12	0.52	2	A
Q8	0.742	76.20	39.60	892.86	1.24	1	A,B,D
Q9	2.147	64.34	149.85	-	367.87	1	A,B,D
Q10	60.307	556.87	1,735.84	4,547.90	1,337.84	1	A,B,D
Q11	0.950	21.96	1,182.53	1,33.59	1,065.43	1	A,B,D
Q12	0.515	18.57	462.48	501.11	383.52	1	A,B,D
Q15	0.002	0.02	0.02	5.04	0.12	1	A
Q16	0.697	4.05	493.67	720.76	1.07	1	A,B,D
Q17	0.279	0.32	0.59	1.87	1.22	1	A,B
Q18	0.02	0.09	0.07	0.80	0.01	2	A
Q19	1.548	6.88	2.53	20.10	73.57	1	A
Q20	0.015	1.02	0.11	2.26	0.36	1	A,B
Sum	72.248	5,248.28	4,202.79	8,555.99	3,957.52	1	
(註)：改善策略 A：DataGuide；B：多重路徑(轉換為 equal-join)；C：ordinal 資訊儲存；D：儲存所有節點資訊於同個表格內(減少 table-join)							

表格 5.5-6　XMark RDB_B 檢索效能分析表(單位為 秒，‘-’表示檢索失敗)

Query	Thesis	XRel	XParent	Edge	Monet	排名	改善策略
Q1	0.197	5.04	1.12	3.74	1.53	1	A,B,D
Q2	0.286	12.54	1.33	3.48	3.4	1	A,B,C,D
Q3	1.900	6.01	5.47	7.82	15.53	1	A,B,C,D
Q4	0.083	5.21	116.65	9.12	2.4	1	A,B,D
Q5	0.01	0.44	0.04	4.27	0.01	1	A,B,D
Q6	0.390	1.8	1.76	-	0.27	2	A
Q7	2.160	2.09	2.54	-	0.24	2	A
Q8	0.742	9717.71	25.24	-	1.03	1	A,B,D
Q9	2.147	4298.98	18421.2	21.97	8.62	1	A,B,D
Q10	60.307	96.61	127.13	43.98	206.97	5	A,B,D
Q11	0.950	16.2	4.56	9708.62	573.84	1	A,B,D
Q12	0.515	10.75	0.55	0.89	410.2	1	A,B,D
Q15	0.002	0.27	0.27	0.1	0.03	1	A
Q16	0.697	1.08	-	95.65	0.7	1	A,B,D
Q17	0.279	3.97	0.4	31.42	0.92	1	A,B
Q18	0.02	0.01	0.02	2.32	0	2	A
Q19	1.548	179.22	2.46	-	2.15	1	A
Q20	0.015	0	1.45	3.5	21.2	2	A,B
Sum	72.248	14,357.93	18,712.19	9,936.88	1,249.04	1	
(註)：改善策略 A：DataGuide；B：多重路徑(轉換為 equal-join)；C：ordinal 資訊儲存；D：儲存所有節點資訊於同個表格內(減少 table-join)							

接下來我們透過上述的檢索效能測試所得到的數據，將本研究方法與其它方法進行分析與比較(以)。XRel、XParent、Edge 與 Monet 的設計理念我們在第二章文獻探討與第 5.5.1　節有介紹過。XRel 與本研究較為相同，是「以儲存節點資訊」，而 XParent、Edge 與 Monet 則是「以儲存邊資訊」。

■　本研究方法與 XRel

由於本研究方法與 XRel 是「以儲存節點資訊」為設計理念(請參考圖 2.4-5)，在有關「節點順序」的檢索子句中，皆有不錯的效能表現。以 Q2 為例(請參考附錄六)，檢索內容為：Return the initial increase of all open auctions. 此檢索子句有

節點順序的觀念，而本研究與 XRel 皆有對每個節點記錄相同父親節點下的相同節點名稱的節點順序，故對這一類型的檢索皆能勝任。

在有關「長路徑走訪」、「多重路徑比對(比對的路徑為兩條)」的檢索子句中，本研究方法與 XRel 也有不錯的效能表現，如 Q1、Q11、Q12。

而在 XRel 表現較差的 Q3、Q4、Q8，Q3 主要原因在於資料庫產生大量的暫存結果(temporal results)，而 Q4 與 Q8 則需要大量的 θ-join 以處理節點之間的關係比對。而特別一提的是在 Q9、Q10 的情況，這兩條檢索子句皆為所有研究方法需克服的瓶頸。這兩條檢索子句皆有「走訪路徑長」、「多重路徑檢索」的情況，所以在專門於長路徑走訪的 XRel，也需花費大量的 θ-join 來確定節點之間的關係。本研究以儲存每個節點上三層的節點標籤標號，以父親節點標籤標號進行節點之間的關係確定，將 θ-join 轉換為 equal-join 以降低檢索成本。

■ 本研究方法與 XParent

XParent 明確地儲存文件上邊的資訊(LablePath)與節點內容值路徑(DataPath)於兩個表格中(請參考圖 2.4-4)，故在面臨「走訪路徑長」、「多重路徑比對(比對的路徑為兩條)」的檢索子句能有不錯的效能表現，如：Q3、Q5、Q6、Q7、Q15、Q17、Q18、Q19、Q20。

以 Q3 為例：Return the IDs of all open auctions whose current increase is at least twice as high as the initial increase. 檢索 open_auction 節點下的第一個 bidder 節點與目前最新的 bidder 節點之間的 increase 節點內容比對。這種針對儲存一段一段邊資訊的 XParent 與 Edge 皆有不錯的表現，可以直接透過幾次的 equal-join 即可找到符合路徑資訊的節點進行內容值的比對。本研究方法針對「走訪路徑長」、「少多重路徑比對」的檢索情況也有優異的效能表現，主要原因在於運用 DataGuide 進行路徑走訪。

而在 XParent 表現較差的 Q16，主要原因在於它需要花費大量的 equal-join 來處理多重路徑的路徑走訪。針對 Q16，本研究透過 DataGuide 進行路徑走訪以

處理長路徑的走訪；再透過屬性 father 來進行節點之間的關係，能更快速地找到符合路徑資訊的節點以進行內容值的比對。

■ 本研究方法與 Monet

Monet 可視為 Edge 方法的變形，它將所有的邊依據可能的標籤路徑(LabelPath)情況進行分類，並且將 XML 中節點內容值部份(CDATA)獨立出來。這樣的做法可以在確定路徑資訊與內容值比對的檢索情況有較佳的表現，如 Q4、Q8、Q16。本研究方法利用 DataGuide 依舊可以達到唯一邊的路徑走訪，並且更快速的針對 node_information 表格進行節點內容值的比對，所以在 Q4、Q8、Q16 的檢索效能表現亦很好。

特別在於 Q6、Q7 的優異表現，主要在於這兩條檢索子句皆以檢索符合某路徑的節點總數，針對這類的檢索情況，Monet 僅需找到符合路徑的表格在將它們加總即可，其 SQL commands 如：select ((select count(*) from R_1) + … +(select count(*) from R_n))。故針對這樣的特例，本研究方法在效能上是比 Monet 差一點。

至於 Monet 表現差的 Q19，主要原因在於需要進行大量的 union 以處理所有不同 regoins 的 locations 結果。針對 Q19，本研究方法利用 DataGuide 產生所有符合路徑條件的 SQL commands 並且使用 Union 進行所有子句的聯集，檢索效能表現良好。

5.5.4 小結

本研究採取以儲存節點為資訊的設計理念，可以在面臨長路徑的走訪、層級未知走訪的情況下，有良好的效能表現(XRel 亦有此優勢，但它需要使用 equal-join 以關聯節點表格)。以 Q16 說明，多重路徑為 2 條但節點之間關係比對(closed_auction 節點與 keyword 節點)是屬祖孫關係，使用記錄節點起始位置(屬性 startlocation)與終點位置(屬性 endlocation)進行節點關係比對，即可解決此類問題。

但是面臨較鄰近的節點之間的關係，XRel 依舊採取上述節點關係比對策略，大大降低檢索效能，如 Q3、Q4、Q8、Q9、Q10。本研究採取透過父親節點標籤標號來進行節點之間的關係，將 XRel 所需的 θ-join 運算轉換為 equal-join，這樣的設計理念為在儲存路徑資訊與儲存節點資訊中取得一個平衡點，也在「以關聯式資料庫，不需 DTD」的研究領域中，得到一個十分良好的效能表現。

值得一提的是在 Q10 的檢索情況，若「以關聯式資料庫，需 DTD」或是 DOM^+ 的研究方法，效能表現即為突出(請參考文獻[16])。其中主要是因為以關聯式資料庫為儲存模型的研究，皆須將文件的結構與內容拆散至資料庫中，即使針對某一節點下的部分子節點的內容擷取，也需透過 join 的運算。這裡我們針對此類檢索情況提出了一個「後置處理」的觀念，以 Q10 的檢索情況，SQL command 檢索出所有 person 的節點編號且使用已存在欄位中的 Startlocation 與 endlocation 將 person 標籤中的所有內容抓出交給 SAX API 進行資料的擷取。實驗證明使用後置處理檢索時間為 14.5 秒(走訪至 person 節點時間為 516ms, 將 person 節點下的子節點擷取時間為 13984ms)，

針對 XMark Benchmark 中的 Q11、Q12 與 Q18，這三條檢索子句皆有數值(value)的觀念，而在「不需 DTD」的研究方法中(本研究、XRel、XParent、Edge 與 Monet)，節點內容值皆以字串為欄位儲存型態。故本實驗針對 Q11、Q12 與 Q18 採取字串比對，Q18 若以 SQL 的 cast(value as float)轉換處理，則檢索時間為 276.6ms。

表格 5.5-7 為本研究與文獻[16]中所有方法測試效能最高的 Shared+相比，Shared+採用 DTD 進行路徑走訪與比對，並且建立了路徑的索引來縮短走訪的距離。而本研究不受需要 DTD 的限制，而表現的效能也十分優異。

表格 5.5-7　本研究與文獻[16]中所有方法測試效能最高的 Shared+相比(單位：秒)

Query	Thesis	Shared+ (RDB_A)	Shared+ (RDB_B)
Q1	0.197	1.54	0.27
Q2	0.286	0.65	0.00
Q3	1.900	0.16	1
Q4	0.083	0.25	0.03
Q5	0.01	0.19	0.01
Q6	0.390	0.25	4.74
Q7	2.160	0.46	0.27
Q8	0.742	1.42	0
Q9	2.147	2.27	1.56
Q10	14.5	1.16	0.57
Q11	0.950	1.62	0.24
Q12	0.515	0.95	0.26
Q15	0.002	1.01	0.34
Q16	0.697	9.37	1.57
Q17	0.279	0.66	0.84
Q18	0.02	0.13	0
Q19	1.548	0.93	17.69
Q20	0.015	0.19	0.22
Sum	26.441	23.26	29.62

第六章　結論與未來研究方向

6.1　研究結果與貢獻

為了解決以 RDBMS 為儲存結構、不使用 DTD 假設的檢索速度問題，本研究分析 XML Query 與 SQL commands 之間的關聯性並針對各種常見的檢索情況進行分析以找出較佳的策略。透過兩種不同類型的 XML benchmarks 實驗測試，證實本研究能比其他四種方法(Edge、XParent、Monet、XRel)在效能上有顯著的提升。整理本研究的貢獻如下：

- 利用代表文件結構的 DataGuide 作為路徑走訪的依據。在記憶體內進行路徑比對，避開 XRel 利用字串找尋路徑的缺陷；並 DataGuide 節點編號直接儲存於節點資訊內，以減少結合路徑與節點資訊所需的 equal-join。此法也可省去其他 Edge、XParent、Monet 等處理路徑走訪時所需的表格合併。
- 將所有的節點型態儲存在同一個表格中以減少節點之間內容值比對所需的 table join。
- 面對多重路徑檢索情況時，使用記錄節點上三層的節點編號進行節點之間的關係比較，可將 θ-join 轉換為 equal-join。
- 以 DataGuide 與 node_information 表格相互配合，將節點順序與路徑順序資訊儲存於節點資訊內，能處理順序方面的檢索。
- 無需複雜的結構轉換設計。運用符合 SQL-92 之 RDBMS，便可達到有效率之 XML 文件儲存與檢索。
- 無需使用 DTD，可適用於 XML 文件結構多變的環境。
- 實驗結果證實本研究能比其它在同研究領域中的 Edge、XParent、Monet、XRel，在效能上分別提升 17.526 倍、8.098 倍、7.569 倍、10.364 倍。

6.2 未來研究方向

我們所採用 XPathCore 作為本研究的檢索語言，此語言擷取 XPath 的核心部份－路徑檢索描述。然而在文獻[16]所提供的 XMark benchmark 測試數據為實作 XQuery 檢索語言，XPath 與 XQuery 在檢索本質上皆有路徑走訪、多重路徑走訪、內容值比對..等，其主要差別在於 XQuery 可允許使用者自訂回傳的版面與迴圈的使用(XQuery 稱之為 FLWR：for、let、where、return)。為求得能更進一步提供強大的檢索功能，未來可以實作出 XQuery。

參考文獻

[1]. A Schmidt, F Waas, ML Kersten, "XMark:A Benchmark for XML Data Management", 28th VLDB, 2002

[2]. A. Zisman. "An overview of XML", Computing and Control Engin. J., Volume 11, Aug. 2000, pages 165-167.

[3]. ABITEBOUL, S.,QUASS, D.,MCHUGH, J.,WIDOM, J., ANDWIENER, J. L. 1997. "The Lorel query language for semistructured data", International Journal on Digital Libraries 1, 1, 68–88.

[4]. AecXML, http://www.iai-na.org/aecxml/mission

[5]. BERGLUND, A., BOAG, S., CHAMBERLIN, D., FERNANDEZ, M. F., KAY, M., ROBIE, J., AND SIMEON, J. 2002, "XML path language (XPath) 2.0."

[6]. BOAG, S., CHAMBERLIN, D., FERNANDEZ, M. F., FLORESCU, D., ROBIE, J., AND SIMEON, J. 2002, "XQuery1.0: An XML query language", In W3C Working Draft 16 August 2002.

[7]. C. Moh, E. Lim, and W. Ng, "DTD-Miner: a tool for mining DTD from XML document," Proc. of the 2nd IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, 2000, pp.144-151.

[8]. CERI, S., COMAI, S., DAMIANI, E., FRATERNALI, P., PARABOSCHI, S., AND TANCA, L. 1999. "XML-GL: a graphical language for querying and restructuring XML documents". In Proceedings of the 8th International World Wide Web Conference. (Toronto, Canada). 1171–1187.

[9]. CHIEN, S. Y., VAGENA, Z., ZHANG, D., TSOTRAS, V., AND ZANIOLO, C. 2002. "Efficient structural joins on indexed XML documents". In Proceedings of the 28th International Conference on Very Large Data Bases. Hong Kong, China. 263–274.

[10]. Design Patterns, Gamma.Johnson, Helm.Vlissides, ISBN 957-2054-11-2

[11]. DEUTSCH, A., FERNANDEZ, M., AND FLORESCU, D. 1999. "A query language for XML". In Proceedings of the 8th International World Wide Web Conference. Toronto, Canada. 1155–1169.

[12]. Eclpise IDE, http://www.eclipse.org/

[13]. Extensible Markup Language (XML) 1.0 http://www.w3.org/XML

[14]. FLORESCU, D. AND KOSSMANN, D. 1999. "A performance evaluation of alternative mapping schemes for storing XML data in a relational database. Survey report".

[15]. Hakon Wium Lie and Janne Saarela, "Multipurpose Web publishing using HTML, XML, and CSS", Commun. ACM 42, 10(Oct. 1999), Page 95 – 101

[16]. Hongjun Lu, Jeffrey Xu Yu, Guoren Wang, Shihui Zheng, Haifeng Jiang, Ge Yu, Aoying Zhou, “What Makes the Differences: Benchmarking XML Database Implementations”, in: ACM Transactions on Internet Technology Vol. 5, No. 1, February 2005, Pages 154-194
[17]. http://lists.xml.org/archives/xml-dev/199703/msg00083.html
[18]. Imamura,M.;Nagahama,R.;Suzuki,K.;Watable,A.;Tsuji,H.;“metadata representation in XML for Internet-based electronic XML application from business to government”,s, Seventh International Conference on Parallel and Distributed Systems: Workshop 2000,2000, pages 387-392
[19]. J. Roy, A. Ramanujan. “XML: data's universal language,” IT Professional, Volume 2, Issue 3, May-June 2000 pages 32-36.
[20]. Java Technology, http://java.sun.com/
[21]. JIANG, H., LU, H., WANG, W., AND YU, J. X. 2002b. “XParent: An efficient RDBMS-based XML database system”. In Proceedings of the 18th International Conference on Data Engineering. (San Jose, CA). 335–336.
[22]. LEE, D. AND CHU, W. W. 2000. “Comparative analysis of six XML schema languages”. SIGMOD Record 29, 3, 76–87.
[23]. LI, Q. AND MOON, B. 2001. “Indexing and querying XML data for regular path expressions.” In Proceedings of the 27th International Conference on Very Large Data Bases. (Rome, Italy). 361–370.
[24]. M Yoshikawa, T Amagasa, T Shimura, S Uemura - ACM Transactions on Internet Technology, “XRel: a path-based approach to storage and retrieval of XML documents using relational databases”，Vol. 1, No. 1, August 2001, Pages 110-141
[25]. M. Fernandez, D. Suciu, “Optimizing regular path expressions using graph schemas”, in: IEEE International Conference on Data Engineering, 1998.
[26]. N. Minos, R. Rajeev, and K. Shim, “SPIRIT: Sequential pattern mining with regular expression constraints”, Proc. of the 25th VLDB Conference, 1999, pp. 223-234.
[27]. P. Buneman, S. Davidson, G. Hillebrand and D. suciu. “A Query Language and Optimization Techniques for Unstructured Data,”
[28]. R. Agrawal and R. Srikant, “Mining sequential patterns,” Proc. of the 7th IEEE International Conference on Data Engineering, 1995, pp.3-14.
[29]. R. Goldman, J. Widom, “DataGuides: enabling query formulation and optimization in semistructured databases”, in: Proceedings of the Conference on Very Large Data Bases, 1997.
[30]. S Park, HJ Kim – DASFAA，“A New Query Processing Technique for XML Based on Signature”, 7th International Conference on DASFAA, pages

22–29,April 2001. 11.
[31]. S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. L. Wiener. “The Lorel query language for semistructured data,” International Journal on Digital Libraries, Volume 1, Issue 1, 1997, pages 68
[32]. S. Nestorov, J. Ullman, J. Wiener, S. Chawathe, “Representative objects: concise representations of semistructured, hierarchical data”, in: IEEE International Conference on Data Engineering, 1997.
[33]. S. Nestorov, S. Biteboul, and R. Motwani, “Inferring structure in semistructured data,” Proc. of the SIGMOD Conference, 1997, pp.39-43.
[34]. S. Park and H. J. Kim, “SigDAQ: an enhanced XML query optimization technique,” Journal of Systems and Software Vol. 61, Issue: 2, pp.91-103, March 15, 2002
[35]. SCHMIDT, A., KERSTEN, M. L., WINDHOUWER, M., ANDWAAS, F. 2000. “Efficient relational storage and retrieval of XML documents”. In International Workshop on the Web and Databases (Informal Proceedings). 47–52.
[36]. Shaks200,http://www.oasis-open.org/cover/bosakShakespeare200.html
[37]. SHANMUGASUNDARAM, J., TUFTE, K., ZHANG, C., GANG, H., DEWITT, D. J., AND NAUGHTON, J. F. 1999. “Relational databases for querying XML documents: Limitations and opportunities”. In Proceedings of the 25th International Conference on Very Large Data Bases. (Edinburgh, Scotland) UK. 302–314
[38]. SRIVASTAVA, D., AL-KHALIFA, S., JAGADISH, H. V., KOUDAS, N., PATEL, J. M., AND YUQING, W. U. 2002. “Structural joins: A primitive for efficient XML query pattern matching”. In Proceedings of the 18th International Conference on Data Engineering. (San Jose, CA). 141–152.
[39]. Standard Generalized Markup Language,http://www.w3.org/MarkUp/SGML/
[40]. T.-S. Chung, H.-J. Kim, “Extracting indexing information from XML DTDs”, Information Processing Letters 81 (2)(2002) 97–103.
[41]. TIAN, F., DEWITT, D. J., CHEN, J., AND ZHANG, C. 2000. “The design and performance evaluation of altervative XML storage strategies”. Tech. rep., Computer Science Department, University of Wisconsin, Madison, WI.
[42]. Tim Berners-Lee, http://www.w3.org/People/Berners-Lee/
[43]. TS Chung, HJ Kim, “Techniques for the evaluation of XML queries: a survey,” Data & Knowledge Engineering, (2003)225-246
[44]. V. Christophides, S. Abiteboul, S. Cluet and M. Scholl, “From Structured documents to Novel Query Facilities,” SIGMOD, 1994.
[45]. W3C DOM Level 1, http://www,23.org/TR/REC-DOM-Level-2/
[46]. Xerces Java Parser 1.4.4 Release, http://xerces.apache.org/xerces-j/

[47]. XHTML, http://www.w3.org/MarkUp/
[48]. XML Path Language, http://www.w3.org/TR/xpath.
[49]. XML Pointer Language (XPointer)，http://www.w3.org/TR/xptr/
[50]. XSL Transformations (XSLT)，http://www.w3.org/TR/xslt
[51]. ZHANG, C.,NAUGHTON, J. F., DEWITT, D. J., LUO, Q., AND LOHMAN, G.M. 2001. “On supporting containment queries in relational database management systems”. In Proceedings of the ACM SIGMOD International Conference on Management of Data. (Santa Barbara, CA). 425–436.
[52]. ZHOU, A., LU, H., ZHENG, S., LIANG, Y., ZHANG, L., JI, W., AND TIAN, Z. 2001. “VXMLR: A visual XMLrelational database system”. In Proceedings of the 27th International Conference on Very Large Data Bases. (Rome, Italy). 719–720.
[53]. 陳高榮，SGML、XML、RDF 文件標準比較 metadata 資料模式設計，輔仁大學圖書資訊學系碩士論文，民 88 年 6 月

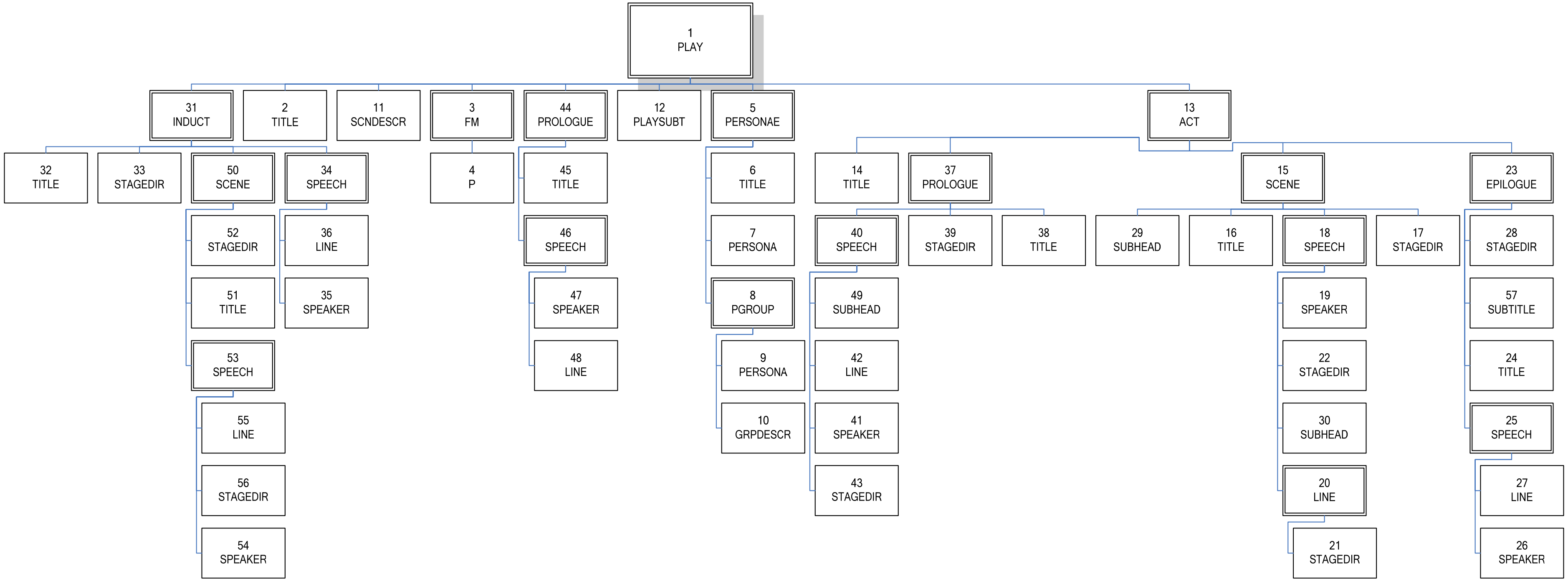
附 錄 一

附錄 1　範例文件「customers.xml」


```xml
<?xml version = "1.0"?>
<DOCUMENT>
	<CUSTOMER TYPE="good" SEX="woman">
		<NAME>
			<LAST_NAME>Thomson</LAST_NAME>
			<FIRST_NAME>Susan</FIRST_NAME>
		</NAME>
		<DATE>September 1, 2001</DATE>
		<ITEM>customer1</ITEM>
		<ORDERS>
			<ITEM ID = "1" YEAR = "2005">
				This is first location for order's item 1
				<PRODUCT>Video tape</PRODUCT>
				This is second location for order's item 1
				<NUMBER>5</NUMBER>
				This is third location for order's item 1
				<PRICE>$1.25</PRICE>
				This is last location for order's item 1
			</ITEM>
			<ITEM ID = "2">
				This is first location for order's item 2
				<PRODUCT>Shovel</PRODUCT>
				This is second location for order's item 2
				<NUMBER>2</NUMBER>
				This is third location for order's item 2
				<PRICE>$4.98</PRICE>
				This is last location for order's item 2
			</ITEM>
		</ORDERS>
	</CUSTOMER>
	<CUSTOMER TYPE="poor">
		<NAME>
			<LAST_NAME>Smithson</LAST_NAME>
			<FIRST_NAME>Nancy</FIRST_NAME>
		</NAME>
		<ITEM>customer2</ITEM>
	</CUSTOMER>
</DOCUMENT>
```

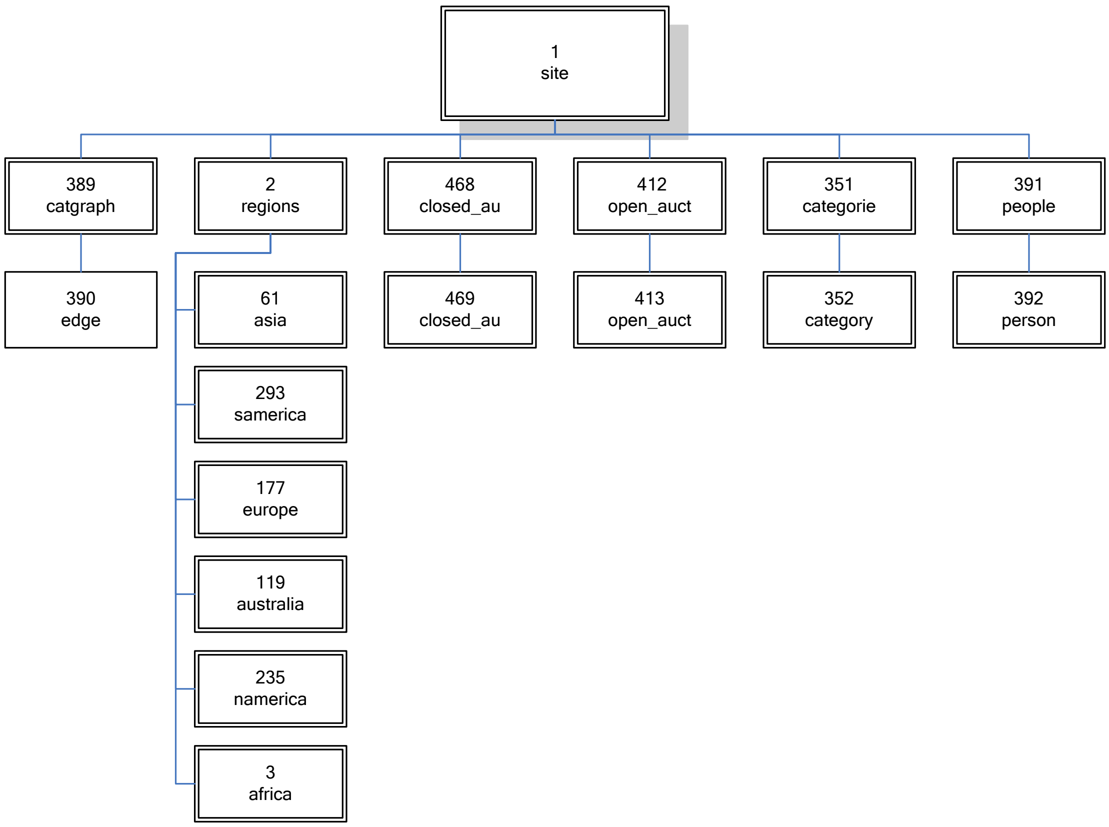
附 錄 二

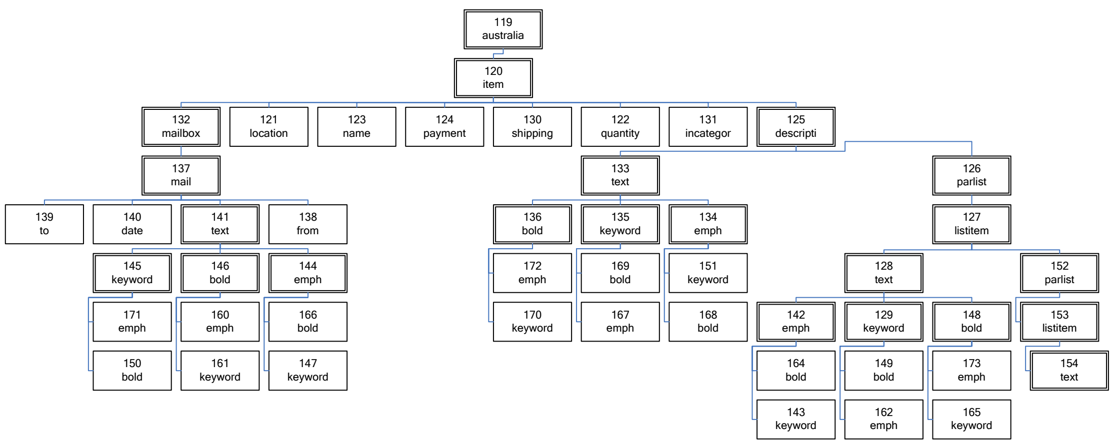
附錄 2　測試資料 Sharks200 之 DataGuide

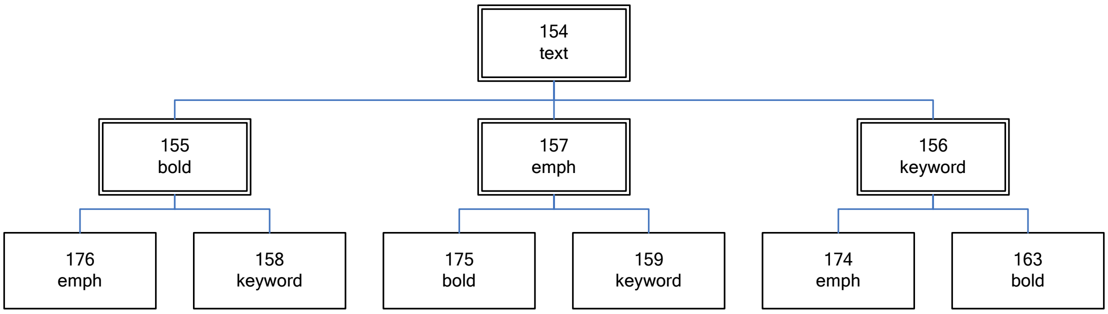


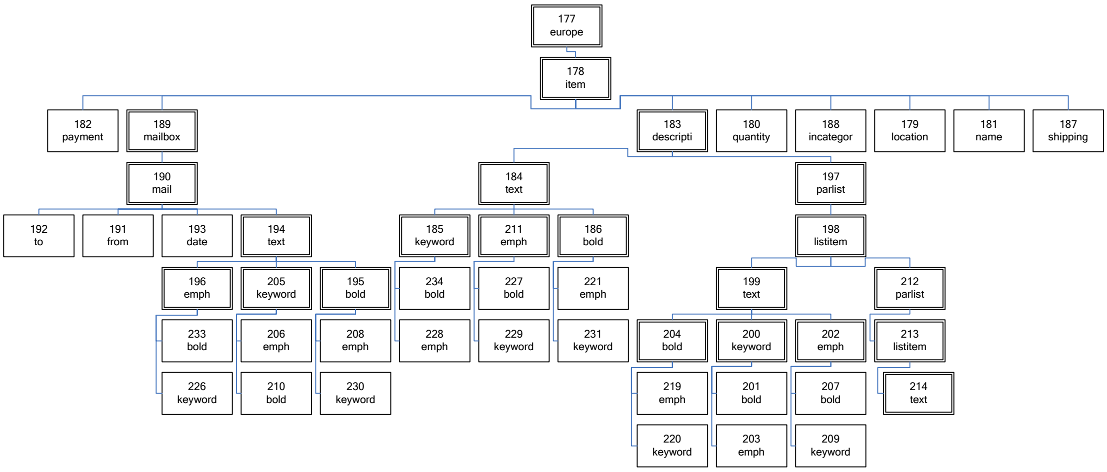
附 錄 三

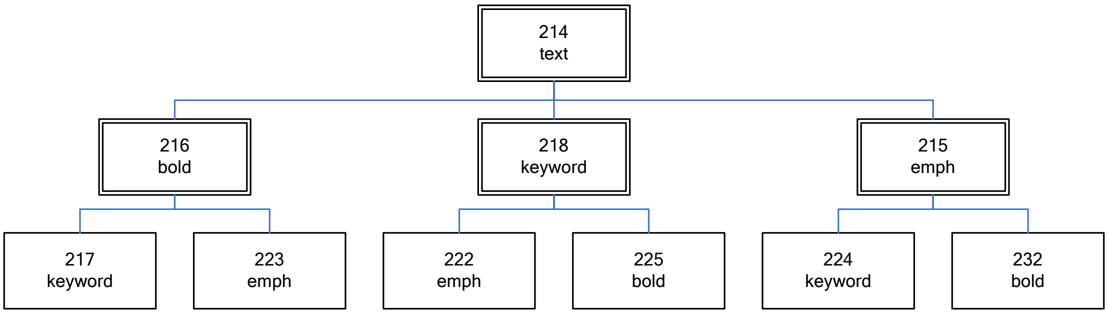
附錄 3 測試資料 XMark 之 DataGuide

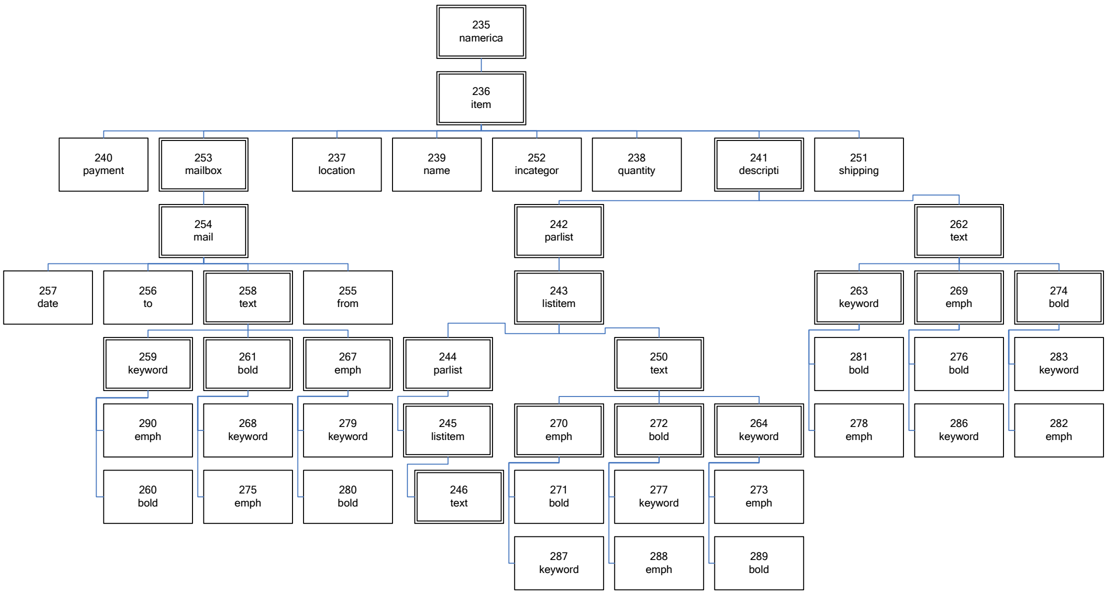


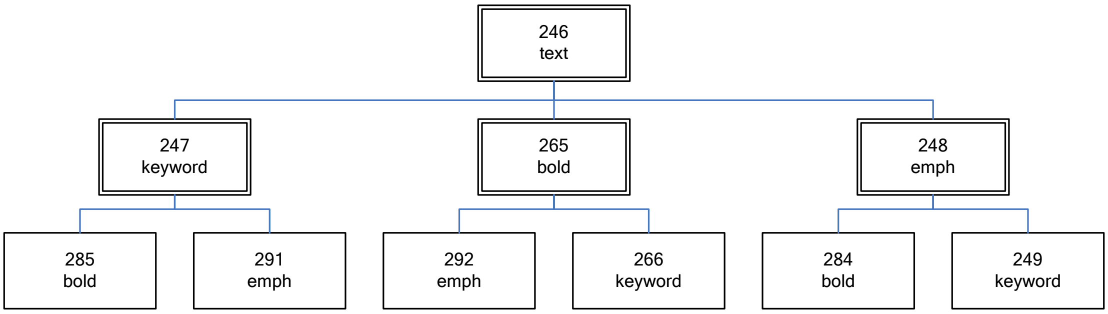


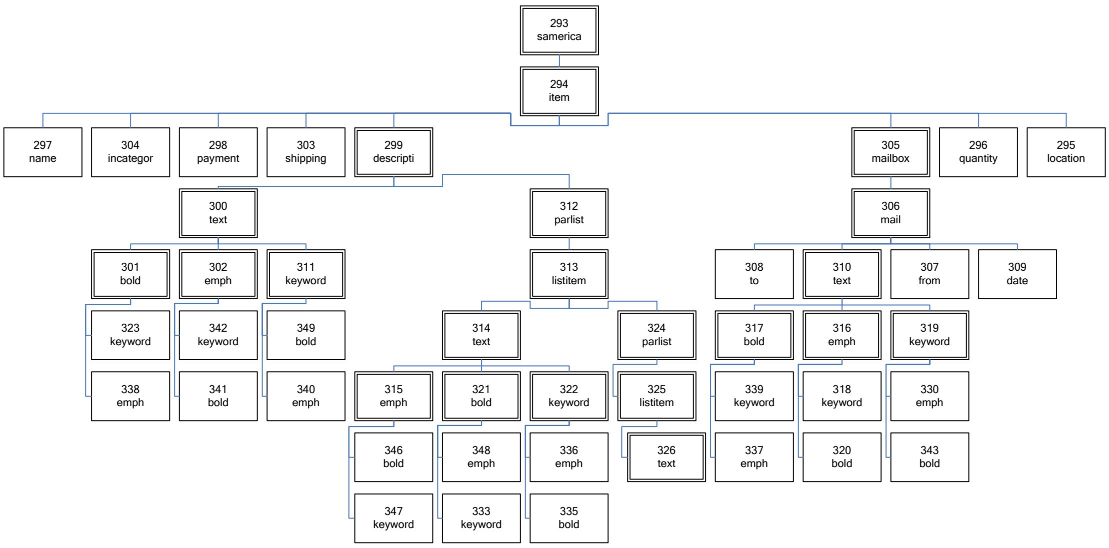

154
text
155
bold
157
emph
156
keyword
176
emph
158
keyword
175
bold
159
keyword
174
emph
163
bold

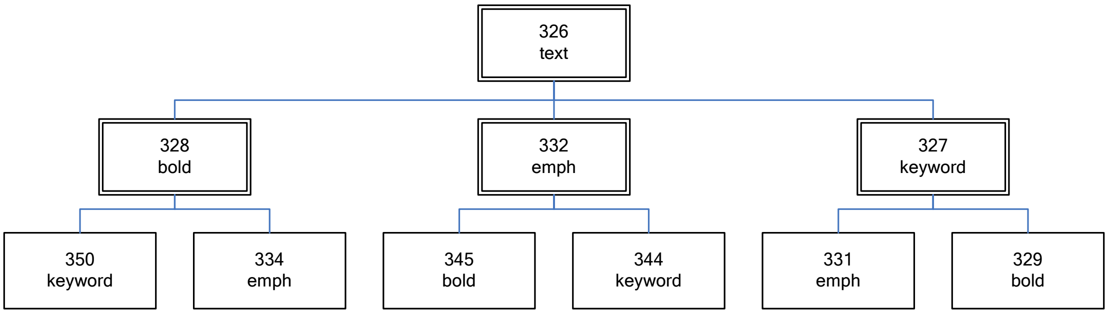

177
europe
178
item
182
payment
189
mailbox
183
descripti
180
quantity
188
incategor
179
location
181
name
187
shipping
190
mail
184
text
197
parlist
192
to
191
from
193
date
194
text
185
keyword
211
emph
186
bold
198
listitem
196
emph
205
keyword
195
bold
234
bold
227
bold
221
emph
199
text
212
parlist
233
bold
206
emph
208
emph
228
emph
229
keyword
231
keyword
204
bold
200
keyword
202
emph
213
listitem
226
keyword
210
bold
230
keyword
219
emph
201
bold
207
bold
214
text
220
keyword
203
emph
209
keyword

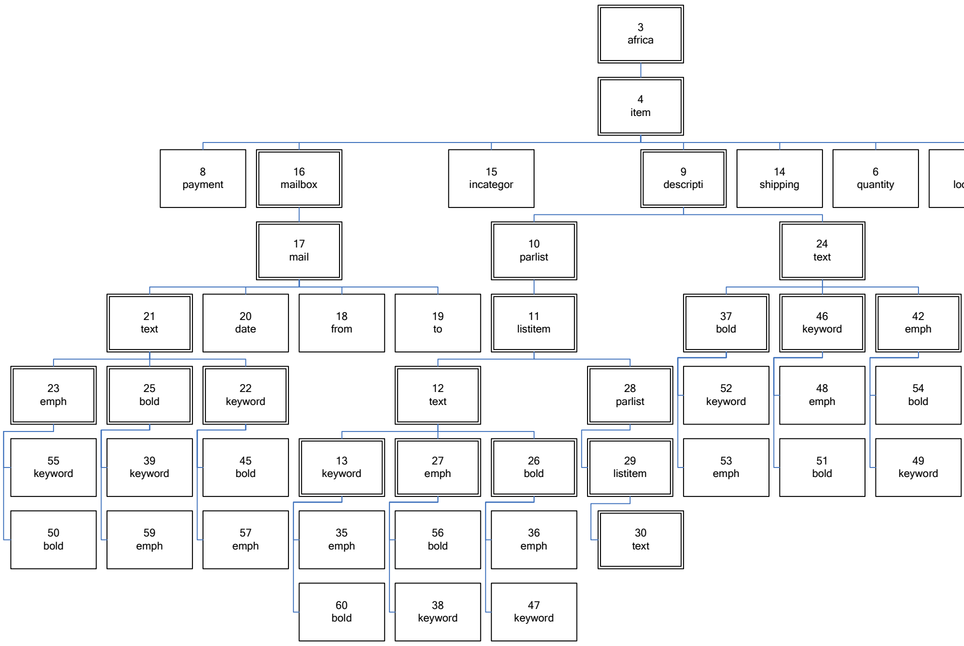

214
text
216
bold
218
keyword
215
emph
217
keyword
223
emph
222
emph
225
bold
224
keyword
232
bold

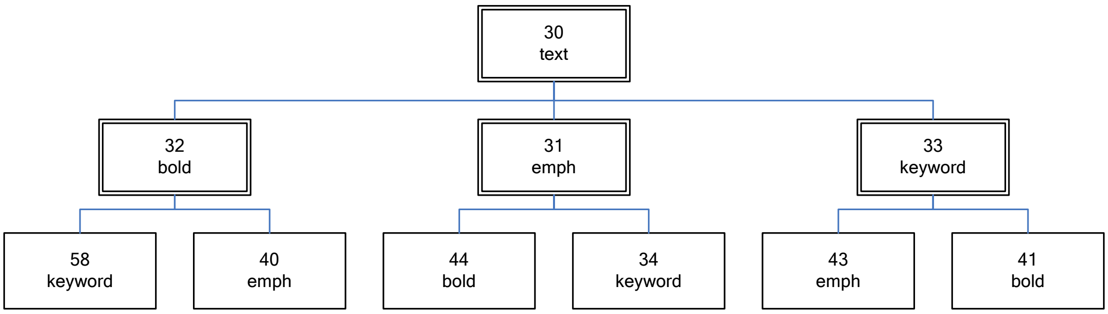

235 namerica
236 item
240 payment
253 mailbox
237 location
239 name
252 incategor
238 quantity
241 descripti
251 shipping
254 mail
242 parlist
262 text
257 date
256 to
258 text
255 from
243 listitem
263 keyword
269 emph
274 bold
259 keyword
261 bold
267 emph
244 parlist
250 text
281 bold
276 bold
283 keyword
290 emph
268 keyword
279 keyword
245 listitem
270 emph
272 bold
264 keyword
278 emph
286 keyword
282 emph
260 bold
275 emph
280 bold
246 text
271 bold
277 keyword
273 emph
287 keyword
288 emph
289 bold

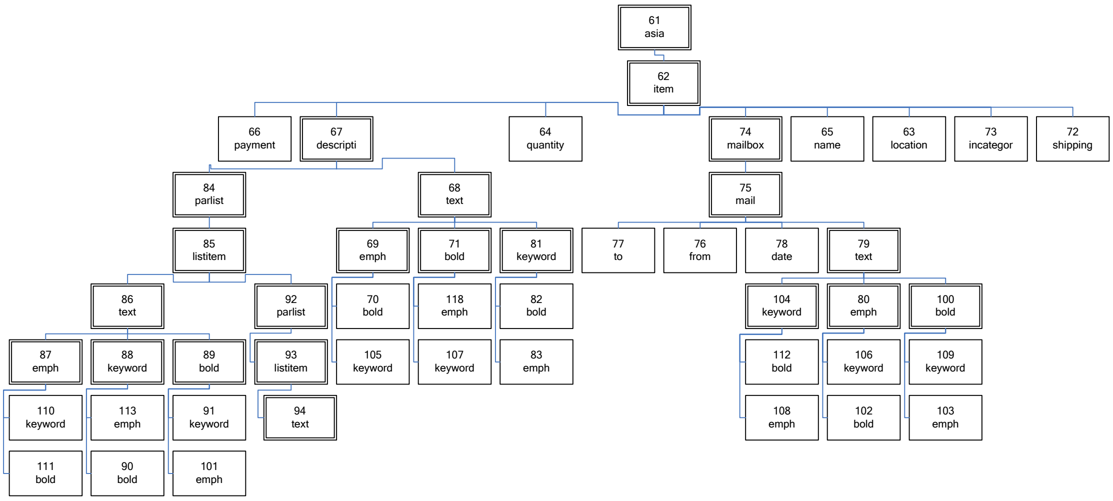

246 text
247 keyword
265 bold
248 emph
285 bold
291 emph
292 emph
266 keyword
284 bold
249 keyword

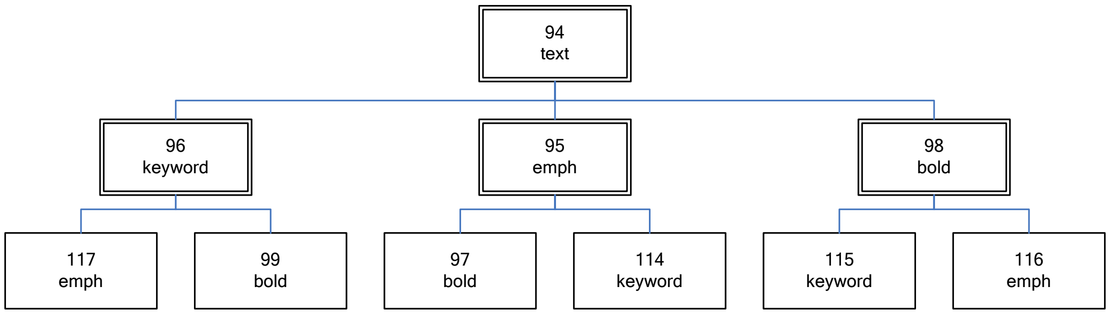

293 samerica
294 item
297 name
304 incategor
298 payment
303 shipping
299 descripti
305 mailbox
296 quantity
295 location
300 text
312 parlist
306 mail
301 bold
302 emph
311 keyword
313 listitem
308 to
310 text
307 from
309 date
323 keyword
342 keyword
349 bold
314 text
324 parlist
317 bold
316 emph
319 keyword
338 emph
341 bold
340 emph
315 emph
321 bold
322 keyword
325 listitem
339 keyword
318 keyword
330 emph
346 bold
348 emph
336 emph
326 text
337 emph
320 bold
343 bold
347 keyword
333 keyword
335 bold

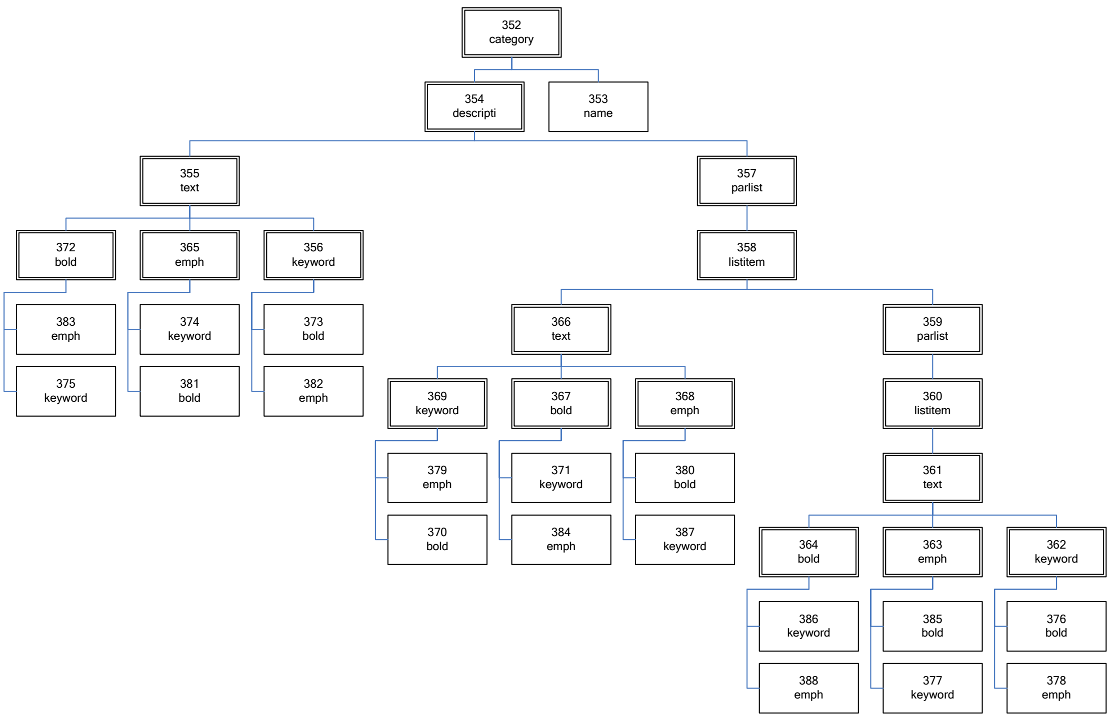

326 text
328 bold
332 emph
327 keyword
350 keyword
334 emph
345 bold
344 keyword
331 emph
329 bold

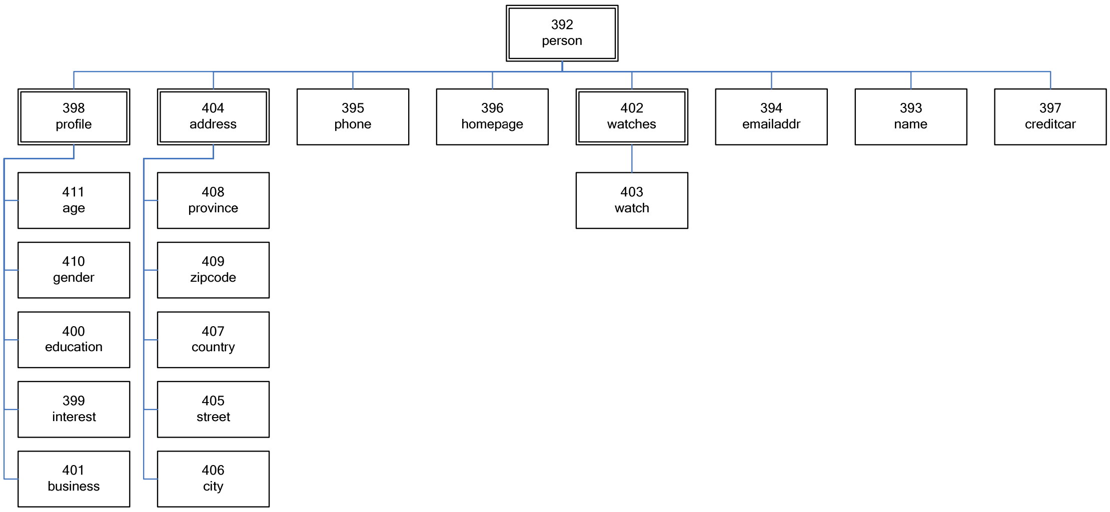

3 africa
4 item
8 payment
16 mailbox
15 incategor
9 descripti
14 shipping
6 quantity
5 location
7 name
17 mail
10 parlist
24 text
21 text
20 date
18 from
19 to
11 listitem
37 bold
46 keyword
42 emph
23 emph
25 bold
22 keyword
12 text
28 parlist
52 keyword
48 emph
54 bold
55 keyword
39 keyword
45 bold
13 keyword
27 emph
26 bold
29 listitem
53 emph
51 bold
49 keyword
50 bold
59 emph
57 emph
35 emph
56 bold
36 emph
30 text
60 bold
38 keyword
47 keyword

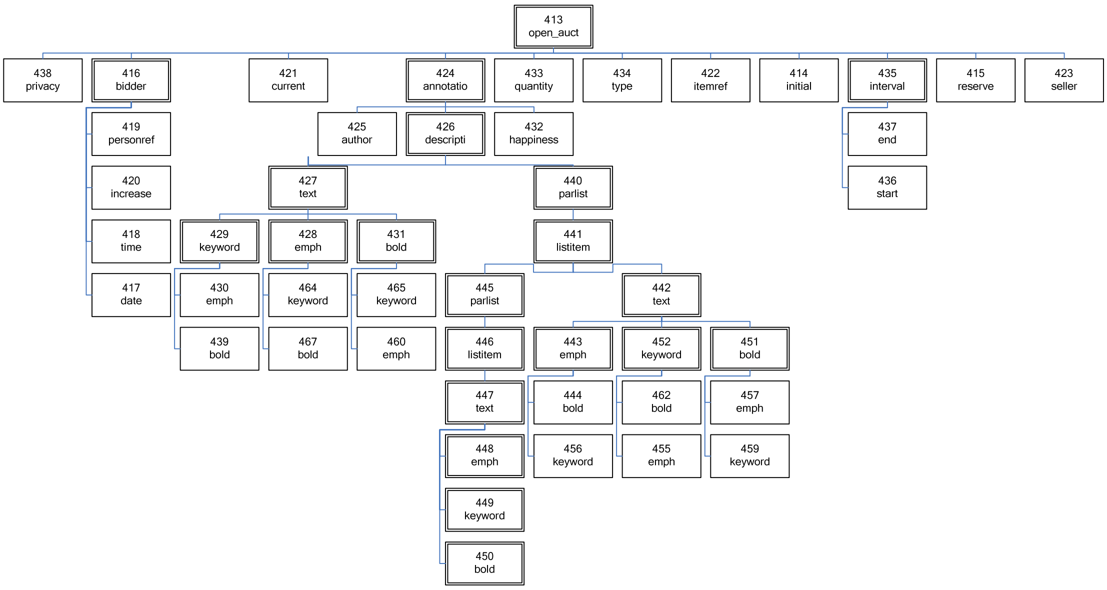

30 text
32 bold
31 emph
33 keyword
58 keyword
40 emph
44 bold
34 keyword
43 emph
41 bold

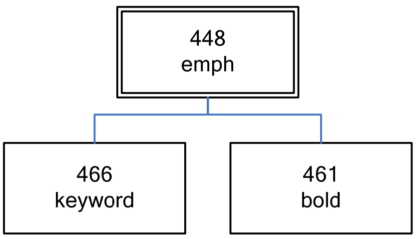

61 asia
62 item
66 payment
67 descripti
64 quantity
74 mailbox
65 name
63 location
73 incategor
72 shipping
84 parlist
68 text
75 mail
85 listitem
69 emph
71 bold
81 keyword
77 to
76 from
78 date
79 text
86 text
92 parlist
70 bold
118 emph
82 bold
104 keyword
80 emph
100 bold
87 emph
88 keyword
89 bold
93 listitem
105 keyword
107 keyword
83 emph
112 bold
106 keyword
109 keyword
110 keyword
113 emph
91 keyword
94 text
108 emph
102 bold
103 emph
111 bold
90 bold
101 emph

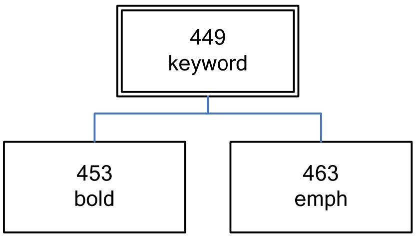

94 text
96 keyword
95 emph
98 bold
117 emph
99 bold
97 bold
114 keyword
115 keyword
116 emph

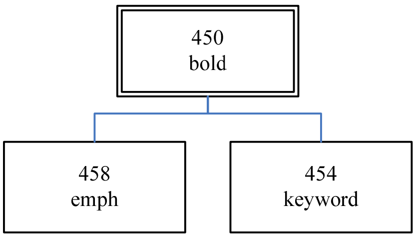

352 category
354 descripti
353 name
355 text
357 parlist
372 bold
365 emph
356 keyword
358 listitem
383 emph
374 keyword
373 bold
366 text
359 parlist
375 keyword
381 bold
382 emph
369 keyword
367 bold
368 emph
360 listitem
379 emph
371 keyword
380 bold
361 text
370 bold
384 emph
387 keyword
364 bold
363 emph
362 keyword
386 keyword
385 bold
376 bold
388 emph
377 keyword
378 emph

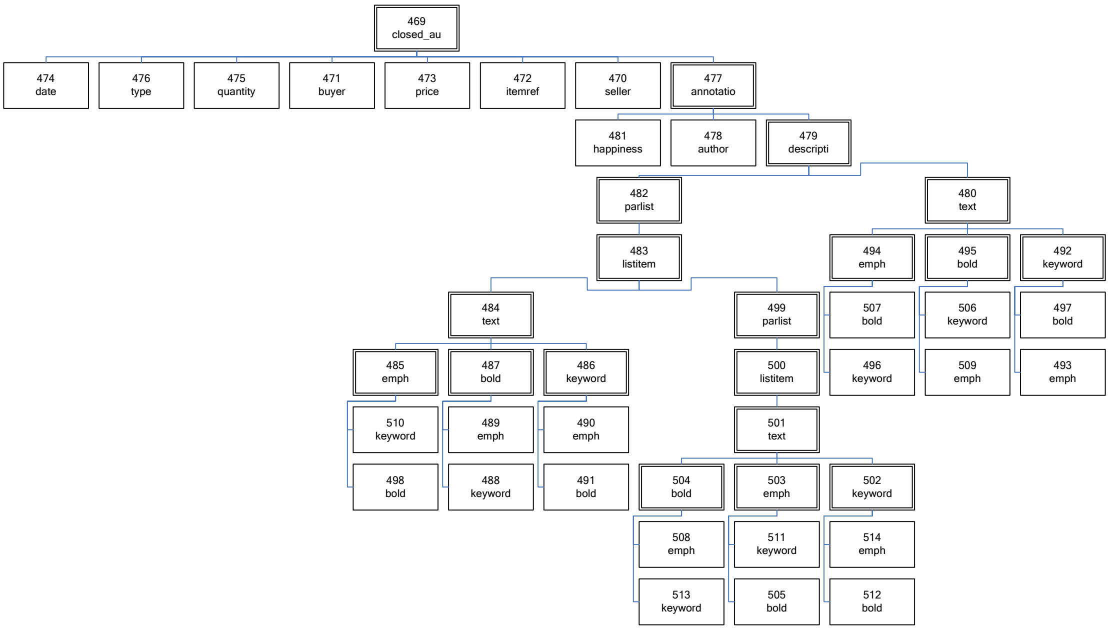

392 person
398 profile
404 address
395 phone
396 homepage
402 watches
394 emailaddr
393 name
397 creditcar
411 age
410 gender
400 education
399 interest
401 business
408 province
409 zipcode
407 country
405 street
406 city
403 watch


413 open_auct
438 privacy
416 bidder
421 current
424 annotatio
433 quantity
434 type
422 itemref
414 initial
435 interval
415 reserve
423 seller
419 personref
420 increase
418 time
417 date
425 author
426 descripti
432 happiness
437 end
436 start
427 text
440 parlist
429 keyword
428 emph
431 bold
441 listitem
430 emph
464 keyword
465 keyword
445 parlist
442 text
439 bold
467 bold
460 emph
446 listitem
443 emph
452 keyword
451 bold
447 text
444 bold
462 bold
457 emph
448 emph
456 keyword
455 emph
459 keyword
449 keyword
450 bold


448 emph
466 keyword
461 bold


449
keyword
453
bold
463
emph


450
bold
458
emph
454
keyword


469
closed_au
474
date
476
type
475
quantity
471
buyer
473
price
472
itemref
470
seller
477
annotatio
481
happiness
478
author
479
descripti
482
parlist
480
text
483
listitem
494
emph
495
bold
492
keyword
484
text
499
parlist
507
bold
506
keyword
497
bold
485
emph
487
bold
486
keyword
500
listitem
496
keyword
509
emph
493
emph
510
keyword
489
emph
490
emph
501
text
498
bold
488
keyword
491
bold
504
bold
503
emph
502
keyword
508
emph
511
keyword
514
emph
513
keyword
505
bold
512
bold

附 錄 四

附錄 4　測試資料 Shakespeare 的 DTD

```
<!-- DTD for Shakespeare        J. Bosak        1994.03.01, 1997.01.02 -->
<!-- Revised for case sensitivity 1997.09.10 -->
<!-- Revised for XML 1.0 conformity 1998.01.27 (thanks to Eve Maler) -->

<!ENTITY amp "&#38;#38;">
<!ELEMENT PLAY         (TITLE, FM, PERSONAE, SCNDESCR, PLAYSUBT, INDUCT?,
                        PROLOGUE?, ACT+, EPILOGUE?)>
<!ELEMENT TITLE        (#PCDATA)>
<!ELEMENT FM           (P+)>
<!ELEMENT P            (#PCDATA)>
<!ELEMENT PERSONAE (TITLE, (PERSONA | PGROUP)+)>
<!ELEMENT PGROUP       (PERSONA+, GRPDESCR)>
<!ELEMENT PERSONA      (#PCDATA)>
<!ELEMENT GRPDESCR (#PCDATA)>
<!ELEMENT SCNDESCR (#PCDATA)>
<!ELEMENT PLAYSUBT (#PCDATA)>
<!ELEMENTINDUCT        (TITLE, SUBTITLE*,
                        (SCENE+|(SPEECH|STAGEDIR|SUBHEAD)+))>
<!ELEMENT ACT          (TITLE, SUBTITLE*, PROLOGUE?, SCENE+, EPILOGUE?)>
<!ELEMENT SCENE        (TITLE, SUBTITLE*, (SPEECH | STAGEDIR | SUBHEAD)+)>
<!ELEMENT PROLOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
<!ELEMENT EPILOGUE     (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
<!ELEMENT SPEECH       (SPEAKER+, (LINE | STAGEDIR | SUBHEAD)+)>
<!ELEMENT SPEAKER      (#PCDATA)>
<!ELEMENT LINE         (#PCDATA | STAGEDIR)*>
<!ELEMENT STAGEDIR     (#PCDATA)>
<!ELEMENT SUBTITLE     (#PCDATA)>
<!ELEMENT SUBHEAD      (#PCDATA)>
```

附 錄 五

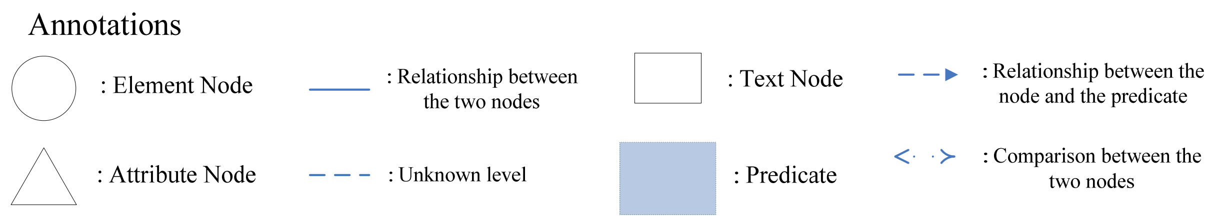
附錄 5　測試資料 XMark 的 DTD

```
<!ELEMENT site (regions, categories, catgraph, people, open_auctions,closed_auctions)>
<!ELEMENT categories (category+)>
<!ELEMENT category (name, description)> <!ATTLIST category id ID #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT description (text | parlist)>
<!ELEMENT text (#PCDATA | bold | keyword | emph)*>
<!ELEMENT bold (#PCDATA | bold | keyword | emph)*>
<!ELEMENT keyword (#PCDATA | bold | keyword | emph)*>
<!ELEMENT emph (#PCDATA | bold | keyword | emph)*>
<!ELEMENT parlist (listitem)*>
<!ELEMENT listitem (text | parlist)*>
<!ELEMENT catgraph (edge*)>
<!ELEMENT edge EMPTY> <!ATTLIST edge from IDREF #REQUIRED to IDREF
#REQUIRED>
<!ELEMENT regions (africa, asia, australia, europe, namerica, samerica)>
<!ELEMENT africa (item*)> <!ELEMENT asia (item*)>
<!ELEMENT australia (item*)> <!ELEMENT namerica (item*)>
<!ELEMENT samerica (item*)> <!ELEMENT europe (item*)>
<!ELEMENT item (location, quantity, name, payment, description, shipping, incategory+,
mailbox)>
<!ATTLIST item id ID #REQUIRED featured CDATA #IMPLIED>
<!ELEMENT location (#PCDATA)> <!ELEMENT quantity (#PCDATA)>
<!ELEMENT payment (#PCDATA)> <!ELEMENT shipping (#PCDATA)>
<!ELEMENT reserve (#PCDATA)>
<!ELEMENT incategory EMPTY> <!ATTLIST incategory category IDREF #REQUIRED>
<!ELEMENT mailbox (mail*)>
<!ELEMENT mail (from, to, date, text)>
<!ELEMENT from (#PCDATA)> <!ELEMENT to (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT itemref EMPTY> <!ATTLIST itemref item IDREF #REQUIRED>
<!ELEMENT personref EMPTY> <!ATTLIST personref person IDREF #REQUIRED>
<!ELEMENT people (person*)>
<!ELEMENT person (name, emailaddress, phone?, address?, homepage?, creditcard?,
profile?, watches?)>
<!ATTLIST person id ID #REQUIRED>
<!ELEMENT emailaddress (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT address (street, city, country, province?, zipcode)>
<!ELEMENT street (#PCDATA)> <!ELEMENT city (#PCDATA)>
<!ELEMENT province (#PCDATA)> <!ELEMENT zipcode (#PCDATA)>
<!ELEMENT country (#PCDATA)> <!ELEMENT homepage (#PCDATA)>
<!ELEMENT creditcard (#PCDATA)>
<!ELEMENT profile (interest*, education?, gender?, business, age?)>
<!ATTLIST profile income CDATA #IMPLIED>
<!ELEMENT interest EMPTY> <!ATTLIST interest category IDREF #REQUIRED>
<!ELEMENT education (#PCDATA)> <!ELEMENT income (#PCDATA)>
<!ELEMENT gender (#PCDATA)> <!ELEMENT business (#PCDATA)>
<!ELEMENT age (#PCDATA)> <!ELEMENT watches (watch*)>
<!ELEMENT watch EMPTY> <!ATTLIST watch open_auction IDREF #REQUIRED>
<!ELEMENT open_auctions (open_auction*)>
<!ELEMENT open_auction (initial, reserve?, bidder*, current, privacy?, itemref, seller,
```

```
annotation, quantity, type, interval)>
<!ATTLIST open_auction id ID #REQUIRED>
<!ELEMENT privacy (#PCDATA)> <!ELEMENT initial (#PCDATA)>
<!ELEMENT bidder (date, time, personref, increase)>
<!ELEMENT seller EMPTY> <!ATTLIST seller person IDREF #REQUIRED>
<!ELEMENT current (#PCDATA)> <!ELEMENT increase (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT interval (start, end)>
<!ELEMENT start (#PCDATA)> <!ELEMENT end (#PCDATA)>
<!ELEMENT time (#PCDATA)> <!ELEMENT status (#PCDATA)>
<!ELEMENT amount (#PCDATA)>
<!ELEMENT closed_auctions (closed_auction*)>
<!ELEMENT closed_auction (seller, buyer, itemref, price, date, quantity, type, annotation?)>
<!ELEMENT buyer EMPTY> <!ATTLIST buyer person IDREF #REQUIRED>
<!ELEMENT price (#PCDATA)>
<!ELEMENT annotation (author, description?, happiness)>
<!ELEMENT author EMPTY> <!ATTLIST author person IDREF #REQUIRED>
<!ELEMENT happiness (#PCDATA)>
```

附 錄 六

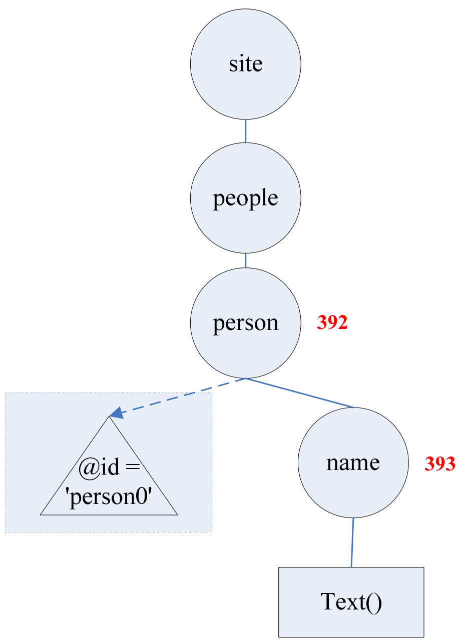
附錄 6　XMark Benchmark 檢索子句與相對應的 DataGuide 圖形



Q1.Return the name of the person with ID `person0' registered in North America. 【Exact Match】

FOR $b IN document("auction.xml")/site/people/person[@id="person0"]
RETURN $b/name/text()

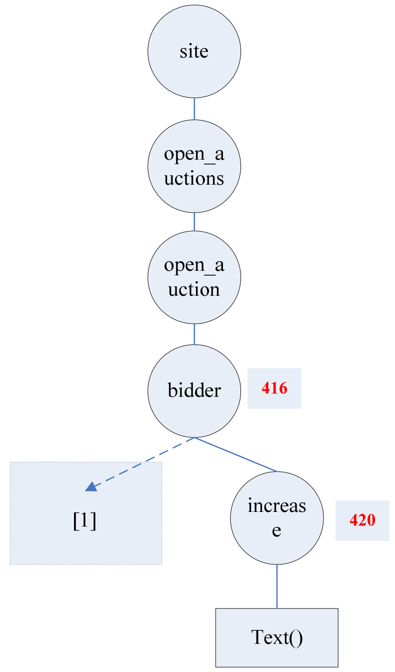
/site/people/person[@id = "person0"]/name/text()



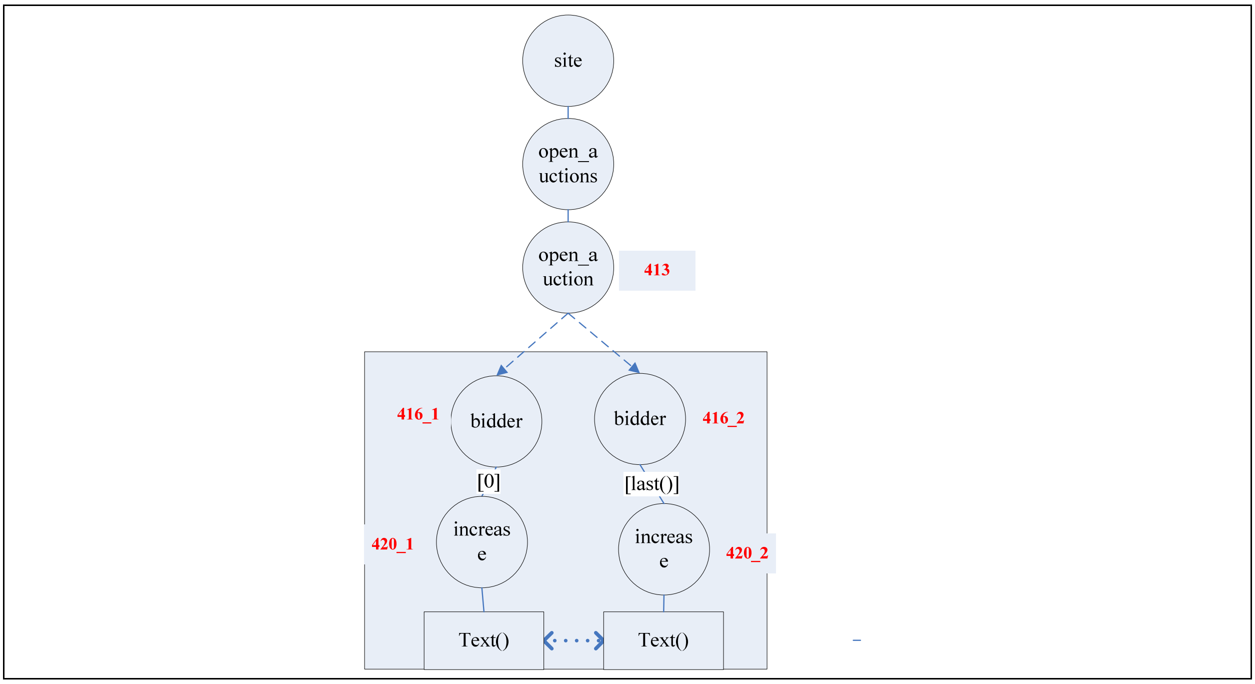
Q2. Return the initial increases of all open auctions. 【Ordered Access】

FOR $b IN document("auction.xml")/site/open_auctions/open_auction
RETURN <increase> $b/bidder[1]/increase/text() </increase>

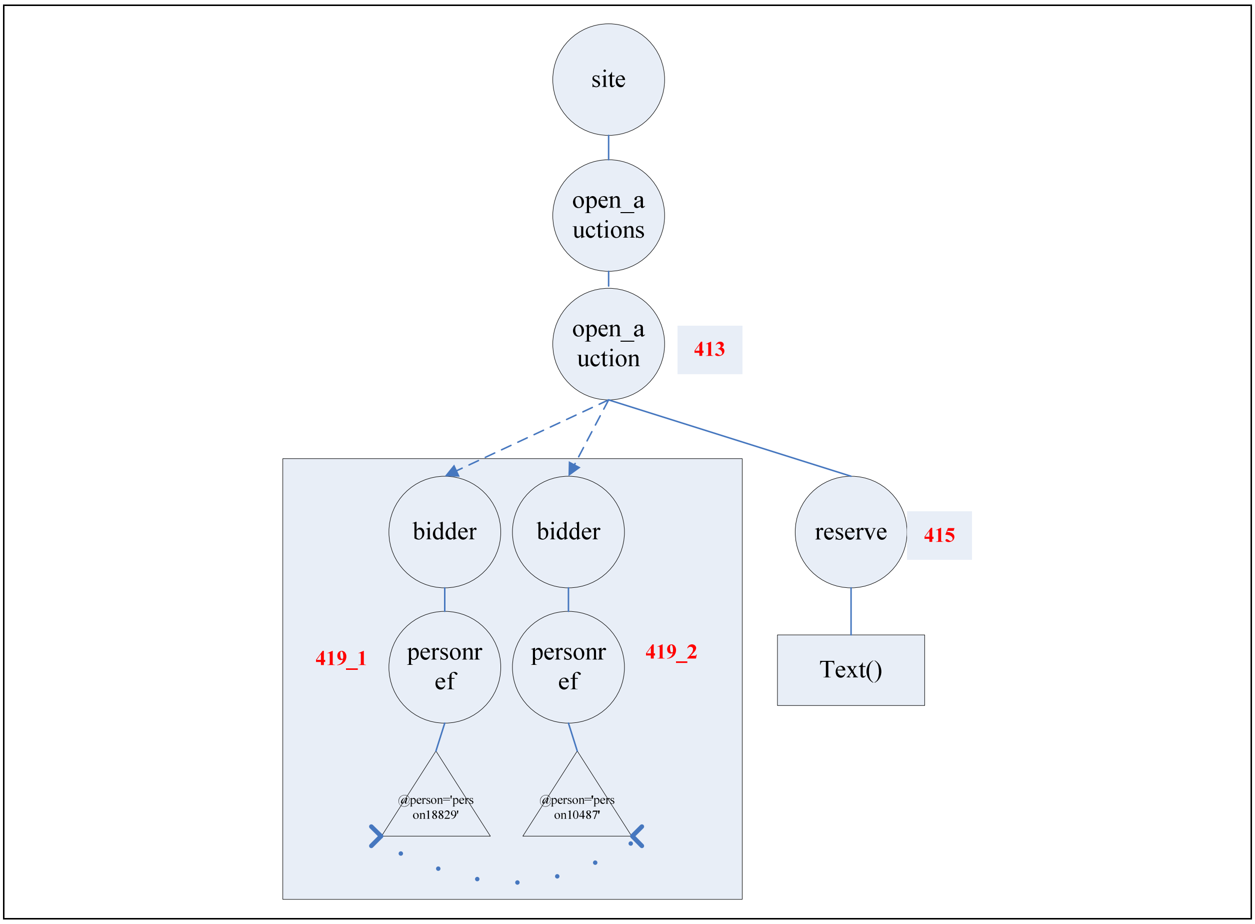
/site/open_auctions/open_auction /bidder[1]/increase/text()



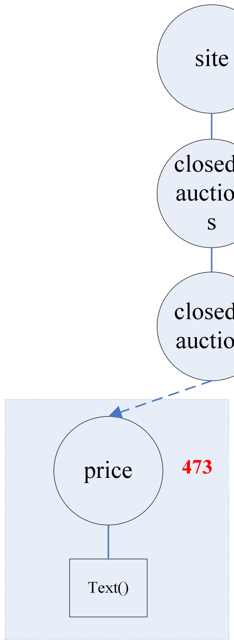
Q3. Return the IDs of all open auctions whose current increase is at least twice as high as the initial increase. 【Ordered Access】
FOR $b IN document("auction.xml")/site/open_auctions/open_auction WHERE $b/bidder[0]/increase/text() *2 <= $b/bidder[last()]/increase/text() RETURN <increase first=$b/bidder[0]/increase/text() last=$b/bidder[last()]/increase/text()/>
/site/open_auctions/open_auction[bidder[0]/increase/text()*2 <= bidder[last()]/increase/text()]



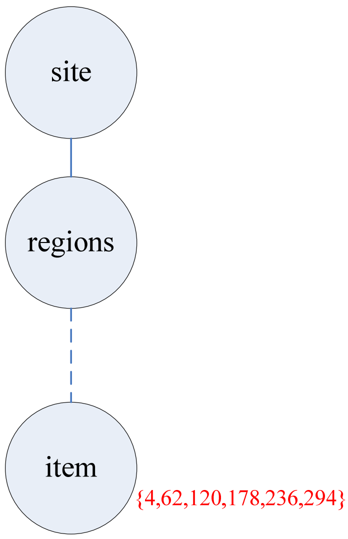
Q4. List the reserves of those open auctions where a certain person issued a bid before another person. 【Ordered Access】
FOR $b IN document("auction.xml")/site/open_auctions/open_auction WHERE $b/bidder/personref[@person="person18829"] BEFORE $b/bidder/personref[@person="person10487"] RETURN <history> $b/reserve/text() </history>
/site/open_auctions/open_auction[bidder/ personref[@person="person18829"] BEFORE bidder/personref[@person="person10487"]]/reserve/text()



Q5. How many sold items cost more than 40? 【Costing】
COUNT(FOR $i IN document("auction.xml")/site/closed_auctions/closed_auction WHERE $i/price/text() >= 40 RETURN $i/price)
/site/closed_auctions/closed_auction[price >= 40]

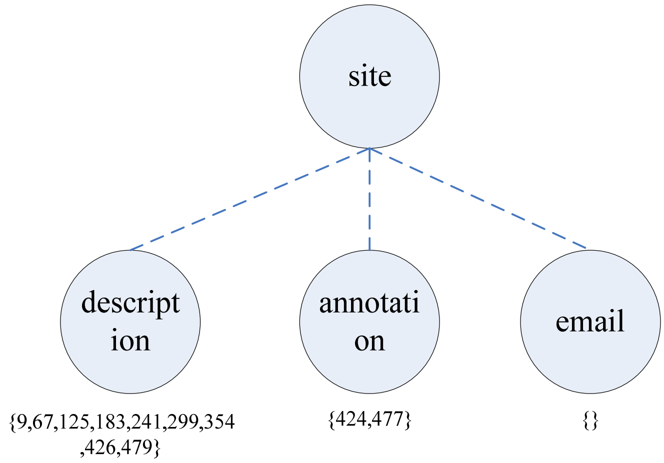


Q6. How many items are listed on all continents? 【Regular Path Expressions】
FOR $b IN document("auction.xml")/site/regions RETURN COUNT ($b//item)
/site/regions//item



Q7. How many pieces of prose are in our database? 【Regular Path Expressions】
FOR $p IN document("auction.xml")/site RETURN count($p//description) + count($p//annotation) + count($p//email);
/site// description /site// annotation

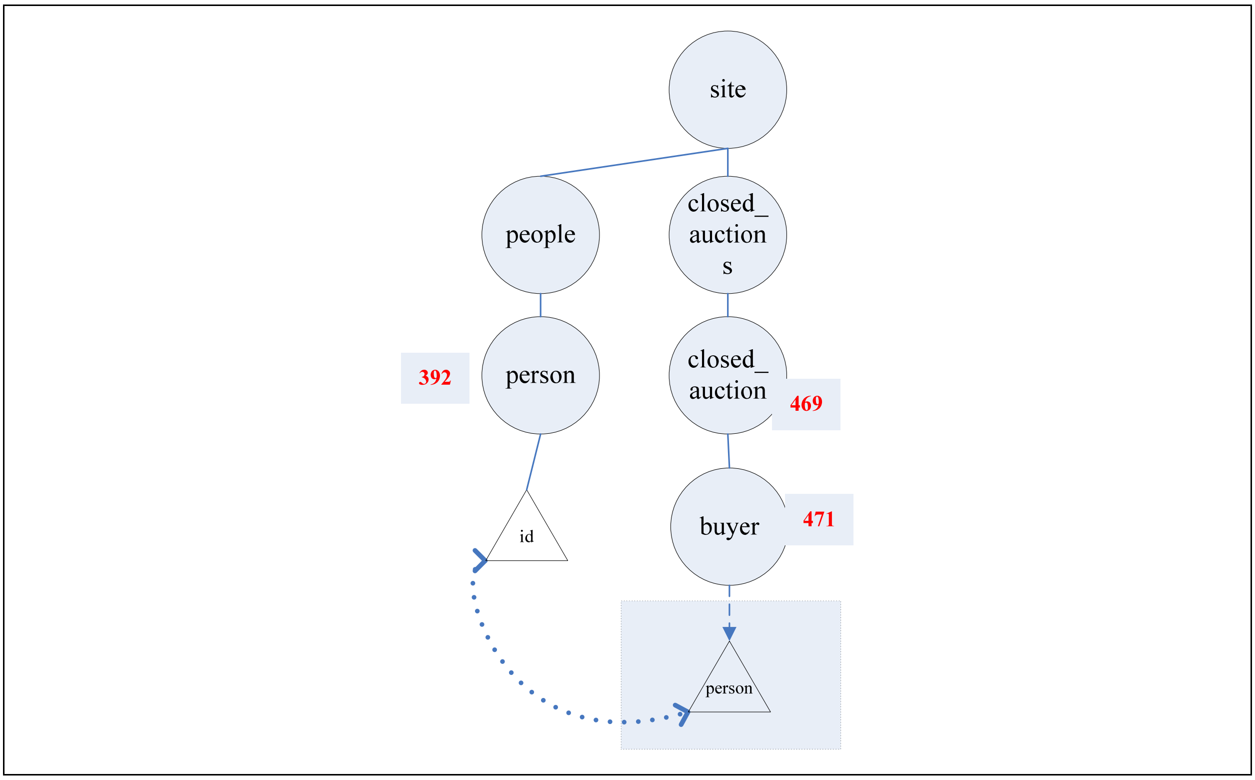
/site// email



Q8. List the names of persons and the number of items they bought. (joins person 、 closed_auction)【Chasing Reference】

```
FOR $p IN document("auction.xml")/site/people/person
LET $a := FOR $t IN document("auction.xml")/site/closed_auctions/closed_auction
              WHERE $t/buyer/@person = $p/@id
              RETURN $t
RETURN <item person=$p/name/text()> COUNT ($a) </item>
```

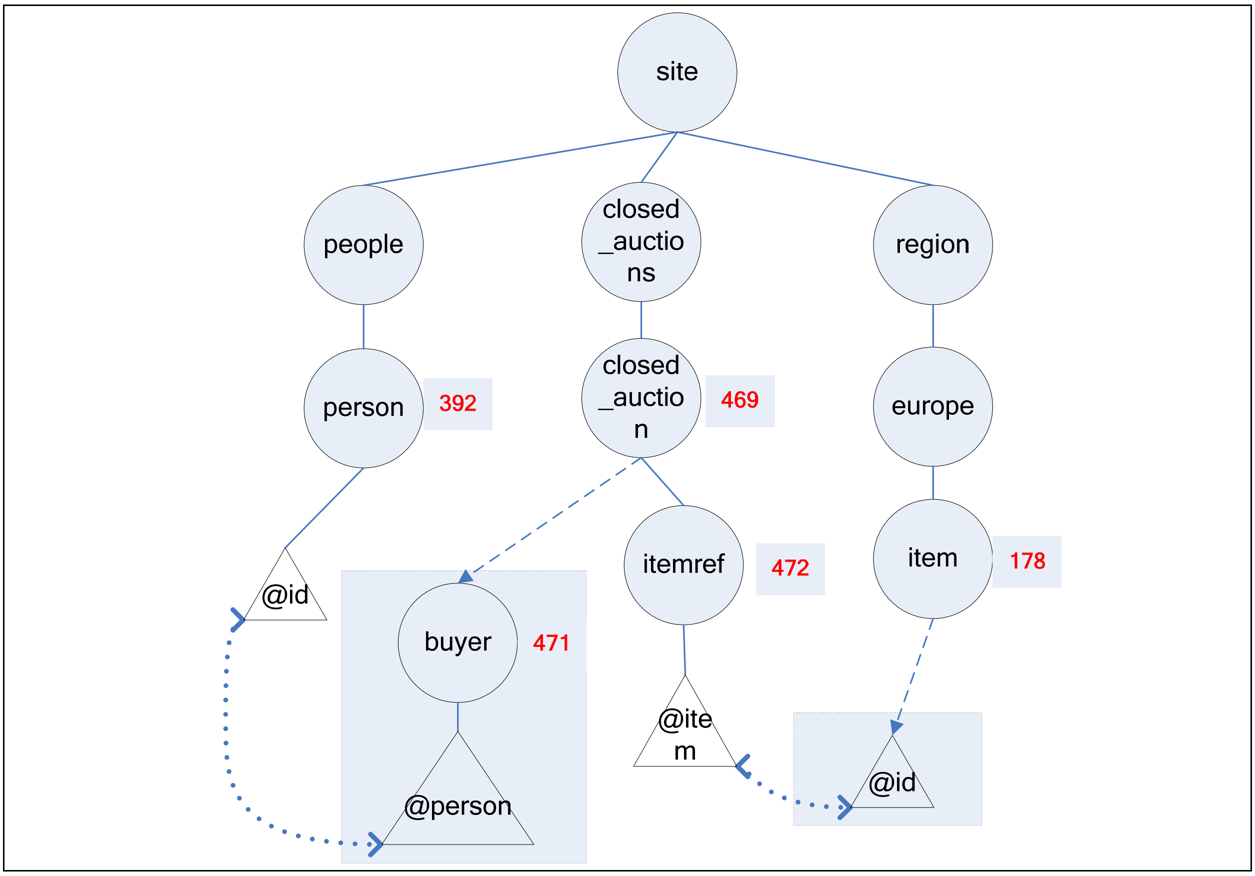
/site/closed_auctions/closed_auction [buyer/ @person = /site/people/person/@id]



Q9. List the names of persons and the names of the items they bought in Europe. (joins person、closed_auction、item) 【Chasing Reference】

```
FOR $p IN document("auction.xml")/site/people/person
LET $a := FOR $t IN document("auction.xml")/site/closed_auctions/closed_auction
                 LET $n := FOR $t2 IN document("auction.xml")/site/regions/europe/item
                                WHERE $t/itemref/@item = $t2/@id
                                RETURN $t2
                 WHERE $p/@id = $t/buyer/@person
                 RETURN <item> $n/name/text() </item>
RETURN <person name=$p/name/text()> $a </person>
```

```
/site/closed_auctions/closed_auction[buyer/@person =
/site/people/person/@id]/itemref[@item = /site/regions/europe/item/@id]
```

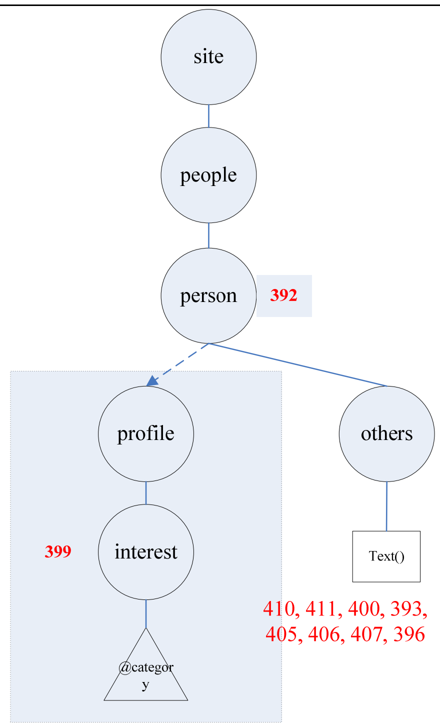


Q10. List all persons according to their interest; use French markup in the result.【Construction of Complex Results】

```
FOR $i IN DISTINCT
            document("auction.xml")/site/people/person/profile/interest/@category
LET $p := FOR     $t IN document("auction.xml")/site/people/person
            WHERE    $t/profile/interest/@category = $i
              RETURN <personne>
                   <statistiques>
                            <sexe> $t/gender/text() </sexe>,
                            <age> $t/age/text() </age>,
                            <education> $t/education/text()</education>,
                            <revenu> $t/income/text() </revenu>
                   </statistiques>,
                   <coordonnees>
                            <nom> $t/name/text() </nom>,
                            <rue> $t/street/text() </rue>,
                            <ville> $t/city/text() </ville>,
                            <pays> $t/country/text() </pays>,
```

```
                                <reseau>
                                        <courrier> $t/email/text() </courrier>,
                                        <pagePerso> $t/homepage/text()</pagePerso>
                                </reseau>,
                        </coordonnees>
                        <cartePaiement> $t/creditcard/text()</cartePaiement>
                  </personne>
RETURN <categorie>
            <id> $i </id>,
            $p
        </categorie>
```

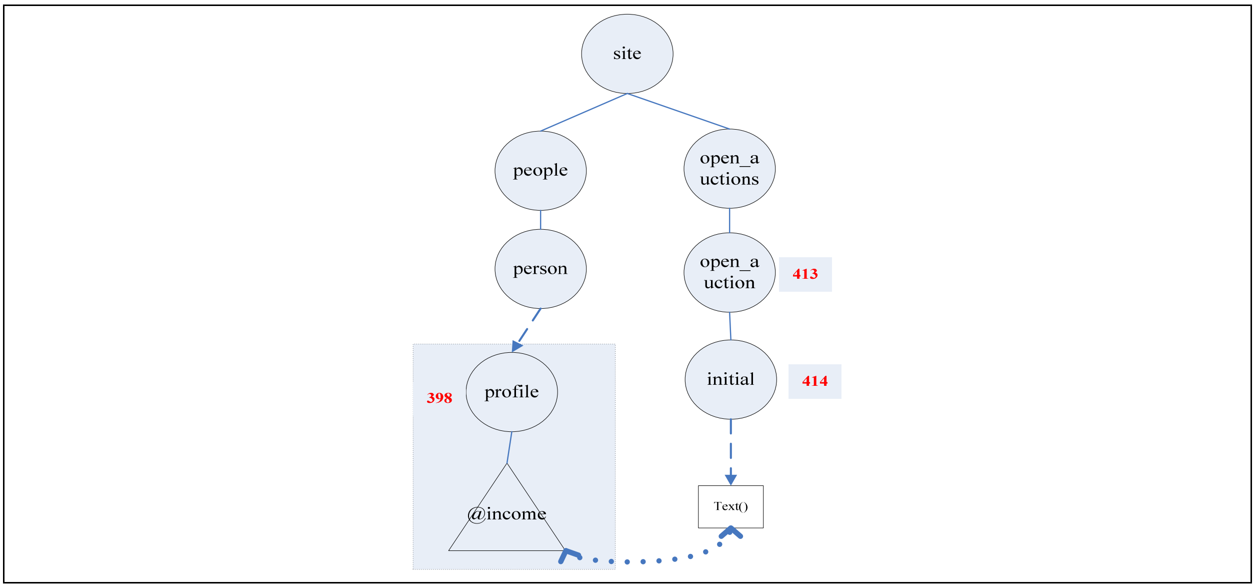
/site/people/person[profile/interest/@category =/site/people/person/profile/interest/@category]



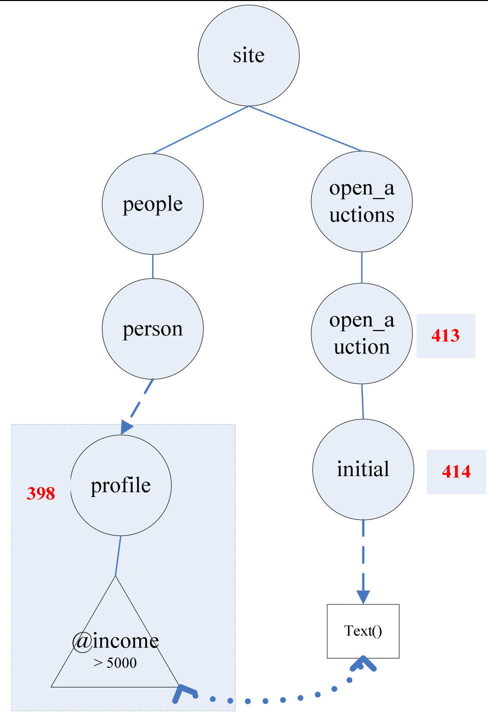
Q11. For each person, list the number of items currently on sale whose price exceeds 0.02% of the person's income. 【Joins on Value】

```
FOR $p IN document("auction.xml")/site/people/person
LET $l := FOR $i IN document("auction.xml")/site/open_auctions/open_auction/initial
          WHERE $p/profile/@income < (5000 * $i/text())
          RETURN $i
RETURN <items name=$p/name/text()> COUNT ($l) </items>
```

/site/people/person[profile/@income = /site/open_auctions/open_auction/initial/text()]



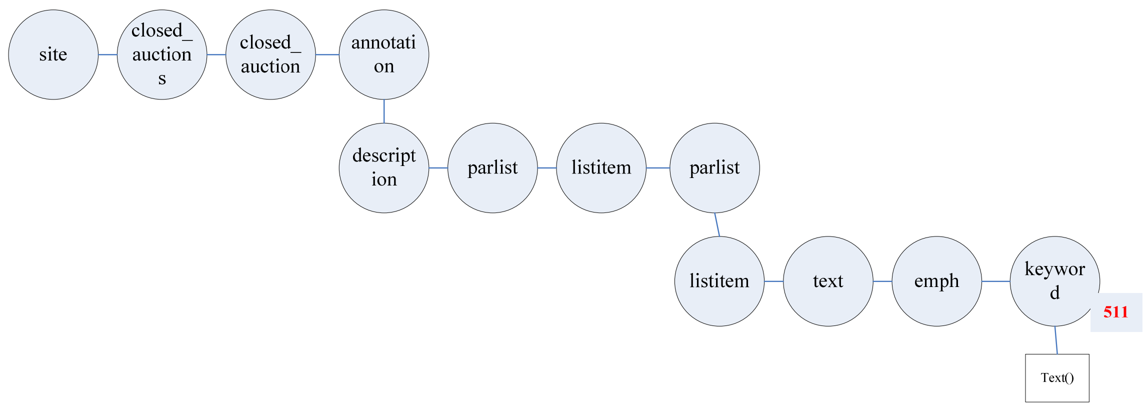
Q12. For each richer-than-average person, list the number of items currently on sale whose price exceeds 0.02% of the person's income. 【Joins on Value】
FOR $p IN document("auction.xml")/site/people/person LET $l := FOR $i IN document("auction.xml")/site/open_auctions/open_auction/initial WHERE $p/profile/@income < (5000 * $i/text()) RETURN $i WHERE $p/profile/@income > 50000 RETURN <items person=$p/name/income/text()> COUNT ($l) </person>
/site/people/person[profile[@income > 5000]/@income = /site/open_auctions/open_auction/initial/text()]



Q15. Print the keywords in emphasis in annotations of closed auctions. 【Path Traversals】

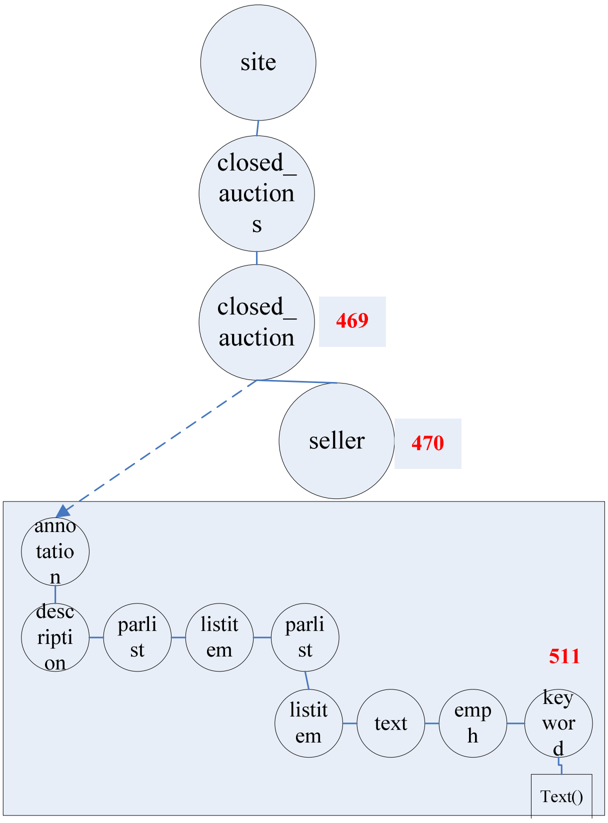
FOR $a IN document("auction.xml")/site/closed_auctions/closed_auction/annotation/
description/parlist/listitem/parlist/listitem/text/emph/keyword/text()
RETURN <text> $a <text>

/site/closed_auctions/closed_auction/annotation/description/parlist/listitem/parlist/listitem/text/emph/keyword/text()

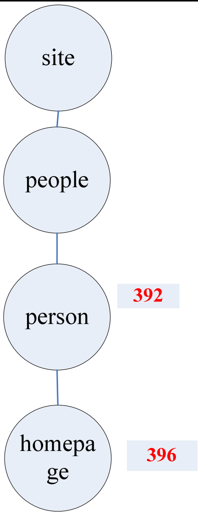


Q16. Return the IDs of those auctions that have one or more keywords in emphasis. (cf. Q15) 【Path Traversals】

FOR $a IN document("auction.xml")/site/closed_auctions/closed_auction
WHERE NOT EMPTY
($a/annotation/description/parlist/listitem/parlist/listitem/text/emph/keyword/text())

RETURN <person id=$a/seller/@person />
/site/closed_auctions/closed_auction[annotation/description/parlist/listitem/parlist/listitem/text/emph/keyword/text() ISNOT Null]


Q17. Which persons don't have a homepage? 【Missing Element】
FOR $p IN document("auction.xml")/site/people/person WHERE EMPTY($p/homepage/text()) RETURN <person name=$p/name/text()/>
/site/people/person[homepage IS Null]

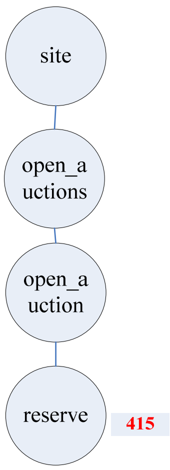


Q18.Convert the currency of the reserve of all open auctions to another currency. 【Function Application】

```
FUNCTION CONVERT ($v)
{
   RETURN 2.20371 * $v -- convert Dfl to Euro
}

FOR      $i IN document("auction.xml")/site/open_auctions/open_auction/
RETURN CONVERT($i/reserve/text())
```

/site/open_auctions/open_auction/reserve

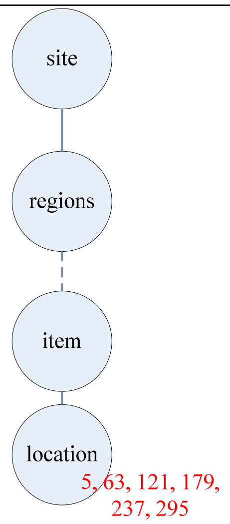


Q19. Give an alphabetically ordered list of all items along with their location. 【Sorting】

```
FOR      $b IN document("auction.xml")/site/regions//item
LET      $k := $b/name/text()
RETURN <item name=$k> $b/location/text() </item>
```

SORTBY (.)

/site/regions//item/location



Q20. Group customers by their income and output the cardinality of each group.
【Aggregation】

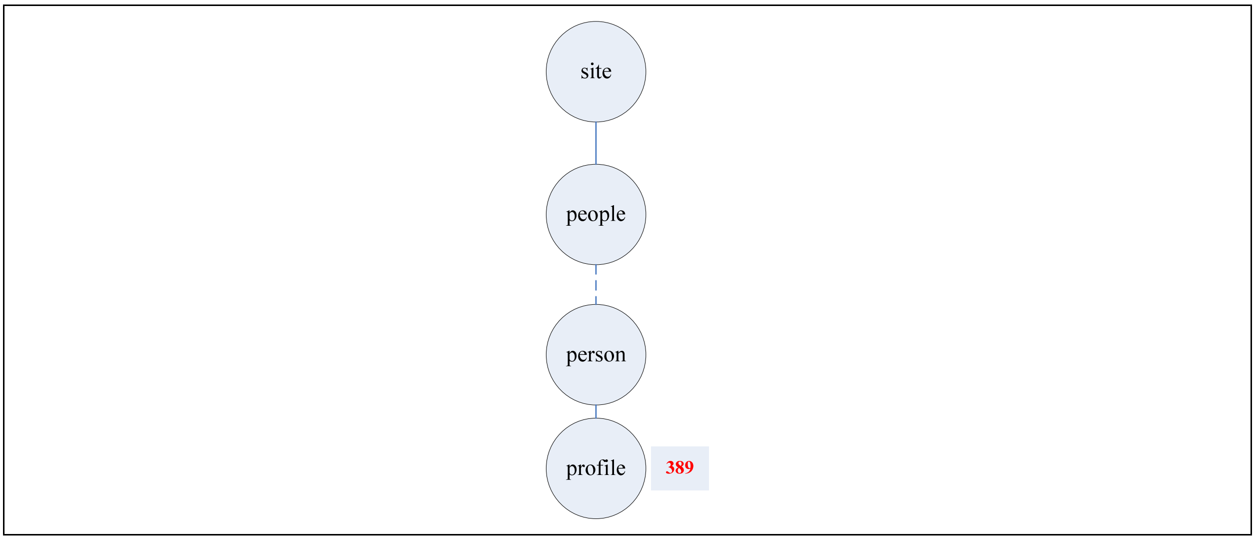
```
<result>
  <preferred>
    COUNT (document("auction.xml")/site/people/person/profile[@income >= 100000])
  </preferred>,
  <standard>
    COUNT (document("auction.xml")/site/people/person/profile[@income < 100000 and
@income >= 30000])
  </standard>,
  <challenge>
    COUNT (document("auction.xml")/site/people/person/profile[@income < 30000])
  </challenge>,
  <na>
    COUNT (FOR $p in document("auction.xml")/site/people/person
               WHERE    EMPTY($p/@income)
               RETURN $p)
  </na>
</result>
```

/site/people/person/profile[@income >= 100000]

/site/people/person/profile[@income < 100000 and @income > 30000]

/site/people/person/profile[@income < 30000]

/site/people/person/profile[@income IS Null]


site
people
person
profile
389

附 錄 七

附錄 7　本研究方法在 XMark Benchmark 所產生的 20 條 SQL commands

檢索子句	SQL commands
Q1	select n393.filename, n393.tagid from node_information n392, node_information n393 where n392.dgnodeid = 392 and n392.attributename = 'id' and n392.prefixcontent = 'person0' and n393.dgnodeid = 393 and n393.nodeid > 0 and n392.father = n393.gfather
Q2	select n420.filename, n420.tagid from node_information n416, node_information n420 where n416.dgnodeid = 416 and n416.nodeid = 0 and n416.elementorderofsamenameunderparent = 1 and n420.dgnodeid = 420 and n420.nodeid > 0 and n416.tagid = v420.gfather
Q3	select n420_1.cuttingcontent as bidder1_increase, n420_2.cuttingcontent as bidder_new_increase from node_information n413, node_information n416_1, node_information n420_1, node_information n416_2, node_information n420_2 where n413.dgnodeid = 413 and n413.nodeid = 0 and n416_1.dgnodeid = 416 and n416_1.nodeid = 0 and n416_1. elementorderofsamenameunderparent = 1 and n413.tagid = n416_1.father and n420_1.dgnodeid = 420 and n420_1.nodeid > 0 and n416_1.tagid = n420_1.gfather and n416_2.dgnodeid = 416 and n416_2.nodeid = 0 and n413.tagid = n416_2.father and n416_2. elementorderofsamenameunderparent = (select MAX(n416_3. elementorderofsamenameunderparent) from node_information n416_3 where n416_3.dgnodeid = 416 and n416_3.nodeid = 0

	and n413.tagid = n416_3.father) and n420_2.dgnodeid = 420 and n420_2.nodeid > 0 and n416_2.tagid = n420_2.gfather and n420_2.prefixcontent >= n420_1.prefixcontent
Q4	select n415.filename, n415.tagid from node_information n413, node_information n419_1, node_information n419_2, node_information n415 where n413.dgnodeid = 413 and n413.nodeid = 0 and n419_1.dgnodeid = 419 and n419_1.attributename = 'person' and n419_1.prefixcontent = 'person18829' and n419_2.dgnodeid = 419 and n419_2.attributename = 'person' and n419_2.prefixcontent = 'person10487' and n413.tagid = n419_1.ggfather and n413.tagid = n419_2.ggfather and n419_1.startlocation < n419_2.startlocation and n415.dgnodeid = 415 and n415.nodeid > 0 and n413.tagid = n415.gfather
Q5	select n473.filename, n473.tagid from node_information n473 where n473.dgnodeid = 473 and n473.attributename is null and n473.prefixcontent >= '40'
Q6	select n4.filename,n4.tagid from node_information n4 where n4.dgnodeid = 4 and n4.nodeid = 0 union select n62.filename,n62.tagid from node_information n62 where n62.dgnodeid = 62 and n62.nodeid = 0 union select n120.filename,n120.tagid from node_information n120 where n120.dgnodeid = 120 and n120.nodeid = 0 union select n178.filename,n178.tagid from node_information n178 where n178.dgnodeid = 178 and n178.nodeid = 0 union select n236.filename,n236.tagid from node_information n236 where n236.dgnodeid = 236 and n236.nodeid = 0 union select n294.filename,n294.tagid

	from node_information n294 where n294.dgnodeid = 294 and n294.nodeid = 0
Q7	select n9.filename,n9.tagid from node_information n9 where n9.dgnodeid = 9 and n9.nodeid = 0 union select n67.filename,n67.tagid from node_information n67 where n67.dgnodeid = 67 and n67.nodeid = 0 union select n125.filename,n125.tagid from node_information n125 where n125.dgnodeid = 125 and n125.nodeid = 0 union select n183.filename,n183.tagid from node_information n183 where n183.dgnodeid = 183 and n183.nodid = 0 union select n241.filename,n241.tagid from node_information n241 where n241.dgnodeid = 241 and n241.nodeid = 0 union select n299.filename,n299.tagid from node_information n299 where n299.dgnodeid = 299 and n299.nodeid = 0 union select n354.filename,n354.tagid from node_information n354 where n354.dgnodeid = 354 and n354.nodeid = 0 union select n426.filename,n426.tagid from node_information n426 where n426.dgnodeid = 426 and n426.nodeid = 0 union select n479.filename,n479.tagid from node_information n479 where n479.dgnodeid = 479 and n479.nodeid = 0 union select n424.filename,n424.tagid from node_information n424 where n424.dgnodeid = 424 and n424.nodeid = 0

	union select n477.filename,n477.tagid from node_information n477 where n477.dgnodeid = 477 and n477.nodeid = 0
Q8	select n393.prefixcontent,count(n469.tagid) from node_information n471, node_information n392, node_information n469, node_information n393 where n469.dgnodeid = 469 and n469.nodeid = 0 and n471.dgnodeid = 471 and n471.attributename = 'person' and n469.tagid = v471.gfather and n392.dgnodeid = 392 and n392.attributename = 'id' and n471.prefixcontent = n392.prefixcontent and n393.dgnodeid = 393 and n393.nodeid > 0 and n393.gfather = n392.father group by n393.prefixcontent
Q9	select n393.prefixcontent, n181.prefixcontent from node_information n469, node_information n471, node_information n472, node_information n392, node_information n178 , node_information n393, node_information n181 where n469.dgnodeid = 469 and n469.nodeid = 0 and n471.dgnodeid = 471 and n471.attributename = 'person' and n469.tagid = n471.gfather and n392.dgnodeid = 392 and n392.attributename = 'id' and n471.prefixcontent = n392.prefixcontent and n472.dgnodeid = 472 and n472.attributename = 'item' and n469.tagid = n472.gfather and n178.dgnodeid = 178 and n178.attributename = 'id' and n472.prefixcontent = n178.prefixcontent and n393.dgnodeid = 393 and n393.nodeid > 0 and n393.gfather = n392.father and n181.dgnodeid = 181 and n181.gfather = n178.father
Q10	select n399.cuttingcontent, n410.prefixcontent, n411.prefixcontent, n400.prefixcontent, n393.prefixcontent, n405.prefixcontent, n406.prefixcontent, n407.prefixcontent, n396.prefixcontent from node_information n392, node_information n399 , node_information n410,

	node_information n411, node_information n400, node_information n393, node_information n405, node_information n406, node_information n407, node_information n396 where n392.dgnodeid = 392 and n392.nodeid = 0 and n399.dgnodeid = 399 and n399.attributename = 'category' and n392.tagid = n399.ggfather and n410.dgnodeid = 410 and n410.nodeid > 0 and n410.ggfather =* n392.tagid and n411.dgnodeid = 411 and n411.nodeid > 0 and n411.ggfather =* n392.tagid and n400.dgnodeid = 400 and n400.nodeid > 0 and n400.ggfather =* n392.tagid and n393.dgnodeid = 393 and n393.nodeid > 0 and n393.gfather =* n392.tagid and n405.dgnodeid = 405 and n405.nodeid > 0 and n405.ggfather =* n392.tagid and n406.dgnodeid = 406 and n406.nodeid > 0 and n406.ggfather =* n392.tagid and n407.dgnodeid = 407 and n407.nodeid > 0 and n407.ggfather =* n392.tagid and n396.dgnodeid = 396 and n396.tagid > 0 and n396.gfather =* n392.tagid order by n399.cuttingcontent
Q11	select n398.tagid,count(n414.tagid) from node_information n414, node_information n398 where n398.dgnodeid = 398 and n398.attributename = 'income' and n414.dgnodeid = 414 and n414.nodeid > 0 and n398.prefixcontent = n414.prefixcontent group by n398.tagid
Q12	select n398.tagid,count(n414.tagid) from node_information n414, node_information n398 where n398.dgnodeid = 398 and n398.attributename = 'income' and n414.dgnodeid = 414 and n414.nodeid > 0 and n398.prefixcontent > '5000' and n398.prefixcontent = n414.prefixcontent group by n398.tagid
Q15	select n511.filename, n511.tagid from node_information n511 where n511.dgnodeid = 511 and n511.nodeid > 0

Q16	select n470.filename, n470.tagid, n470.prefixcontent from node_information n469, node_information n511, node_information n470 where n469.dgnodeid = 469 and n469.nodeid = 0 and n511.dgnodeid = 511 and n511.attributename is null and n511.prefixcontent is not null and n469.startlocation <= n511.startlocation and n469.endlocation >= n511.endlocation and n470.dgnodeid = 470 and n470.attributename = 'person' and n469.tagid = n470.gfather
Q17	select n392.tagid , n393.prefixcontent from node_information n392, node_information n393 where n392.dgnodeid = 392 and n392.nodeid = 0 and n393.dgnodeid = 393 and n393.nodeid > 0 and n393.gfather = n392.tagid and not exists(select * from node_information n396 where n396.dgnodeid = 396 and n396.nodeid = 0 and n396.father = n392.tagid)
Q18	select n415.filename, n415.tagid from node_information n415 where n415.dgnodeid = 415 and n415.nodeid = 0
Q19	select n5.filename, n5.tagid, n5.prefixcontent as location from node_information n5 where n5.dgnodeid = 5 and n5.nodeid > 0 union select n63.filename,n63.tagid, n63.prefixcontent as location from node_information n63 where n63.dgnodeid = 63 and n63.nodeid > 0 union select n121.filename,n121.tagid, n121.prefixcontent as location from node_information n121 where n121.dgnodeid = 121 and n121.nodeid > 0 union select n179.filename,n179.tagid, n179.prefixcontent as location from node_information n179 where n179.dgnodeid = 179 and n179.nodeid > 0 union select n237.filename,n237.tagid, n237.prefixcontent as location from node_information n237 where n237.dgnodeid = 237 and n237.nodeid > 0 union select n295.filename,n295.tagid, n295.prefixcontent as location

	from node_information n295 where n295.dgnodeid = 295 and n295.nodeid > 0 order by location, tagid
Q20	select count(n398.tagid) from node_information n398 where n398.dgnodeid = 398 and n398.attributename = 'income' and n398.prefixcontent >= '100000' go select count(n398.tagid) from node_information n398 where n398.dgnodeid = 398 and n398.attributename = 'income' and n398.prefixcontent < '100000' and n398.prefixcontent > '30000' go select count(n398.tagid) from node_information n398 where n398.dgnodeid = 398 and n398.attributename = 'income' and n398.prefixcontent < '30000' go select count(n398.tagid) from node_information n398 where n398.dgnodeid = 398 and n398.attributename = 'income' and n398.prefixcontent is null go

附 錄 八

附錄 8　文獻[7]中所有方法的 RDB_A 測試數據表(單位 秒 ‘-’表示檢索失敗)

Query	XRel	XParent	Edge	Monet	Shared	Shared+	XDB-OO	DOM	DOM+
Q1	0.39	0.54	4.46	0.76	1.01	1.54	8.07	4.96	0.02
Q2	0.57	15.09	20.85	141.72	1.20	0.65	6.18	6.06	5.27
Q3	-	33.58	36.06	579.28	30.33	0.16	19.12	12.58	16.06
Q4	4,485.52	82.73	411.81	2.68	0.24	0.25	9.54	63.81	0.06
Q5	0.50	0.33	30.06	0.10	0.21	0.19	5.22	1.15	0.01
Q6	6.50	0.79	13.92	0.19	0.34	0.25	12.69	3.67	0.01
Q7	4.50	2.44	12.12	0.52	0.70	0.46	21.14	63.34	0.03
Q8	76.20	39.60	892.86	1.24	138.98	1.42	5,954.51	20.65	1.10
Q9	64.34	149.85	-	367.87	162.14	2.27	8,910.99	26.04	1.18
Q10	556.87	1,735.84	4,547.90	1,337.84	0.56	1.16	-	188.95	21.86
Q11	21.96	1,182.53	1,33.59	1,065.43	277.71	1.62	-	42.39	11.45
Q12	18.57	462.48	501.11	383.52	1,046.53	0.95	-	38.71	8.14
Q15	0.02	0.02	5.04	0.12	796.79	1.01	0.85	1.62	0.00
Q16	4.05	493.67	720.76	1.07	19.35	9.37	12.31	3.64	0.28
Q17	0.32	0.59	1.87	1.22	0.78	0.66	50.89	44.70	0.47
Q18	0.09	0.07	0.80	0.01	0.38	0.13	13.83	9.27	0.01
Q19	6.88	2.53	20.10	73.57	1.31	0.93	55.93	96.92	0.99
Q20	1.02	0.11	2.26	0.36	0.19	0.19	0.22	86.55	0.43
Sum	5,248.28	4,202.79	8,555.99	3,957.52	2,478.76	23.26	14,948.09	715.01	68.05

文獻[7]中所有方法的 RDB_B 測試數據表(單位 秒 ‘-’表示檢索失敗)

Query	XRel	XParent	Edge	Monet	Shared	Shared+	XDB-OO	DOM	DOM+
Q1	5.04	1.12	3.74	1.53	0.39	0.27	8.07	4.96	0.02
Q2	12.54	1.33	3.48	3.4	0.06	0.00	6.18	6.06	5.27
Q3	6.01	5.47	7.82	15.53	0.14	1	19.12	12.58	16.06
Q4	5.21	116.65	9.12	2.4	0.01	0.03	9.54	63.81	0.06
Q5	0.44	0.04	4.27	0.01	0.02	0.01	5.22	1.15	0.01
Q6	1.8	1.76	-	0.27	0.02	4.74	12.69	3.67	0.01
Q7	2.09	2.54	-	0.24	5.82	0.27	21.14	63.34	0.03
Q8	9717.71	25.24	-	1.03	0.94	0	5,954.51	20.65	1.10
Q9	4298.98	18421.2	21.97	8.62	1.58	1.56	8,910.99	26.04	1.18
Q10	96.61	127.13	43.98	206.97	2.49	0.57	-	188.95	21.86
Q11	16.2	4.56	9708.62	573.84	643.49	0.24	-	42.39	11.45
Q12	10.75	0.55	0.89	410.2	154.42	0.26	-	38.71	8.14
Q15	0.27	0.27	0.1	0.03	2.1	0.34	0.85	1.62	0.00
Q16	1.08	-	95.65	0.7	2.09	1.57	12.31	3.64	0.28
Q17	3.97	0.4	31.42	0.92	0.4	0.84	50.89	44.70	0.47
Q18	0.01	0.02	2.32	0	0.01	0	13.83	9.27	0.01
Q19	179.22	2.46	-	2.15	0.58	17.69	55.93	96.92	0.99
Q20	0	1.45	3.5	21.2	0.41	0.22	0.22	86.55	0.43
Sum	14,357.93	18,712.19	9,936.88	1,249.04	814.96	29.62	14,948.09	715.01	68.05