

多元對應之 SQL 與 XQuery 雙向
查詢句轉換之研究

Bidirectional Query Translation Between
SQL And XQuery In Multiple Mappings

研 究 生：劉邦鑑

Student：Pang-Chien Liu

指導教授：張雅惠

Advisor：Ya-Hui Chang

國 立 臺 灣 海 洋 大 學
資 訊 工 程 學 系
碩 士 論 文

A Thesis

Submitted to Department of Computer Science and Engineering
College of Electrical Engineering and Computer Science

National Taiwan Ocean University

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science and Engineering

July 2006

Keelung, Taiwan, Republic of China

中華民國 95 年 7 月



摘要

由於電子商務及數位內容產業的蓬勃發展，延伸式標記語言XML已被提出來且發展為資料交換的標準，而關聯式資料與XML資料之間，如何進行有效地管理、檢索、交換與查詢等議題已被多加討論。

關聯式資料庫的標準查詢語言為SQL，而針對XML格式的資料，則是以W3C組織制訂的XQuery來做查詢處理。本論文的目的，在提出一個雙向查詢句轉換系統，使用者可方便的依不同類型需求來下達查詢句，再進一步進行資料的交換與使用。為了達到此目的，本論文必須探討關聯式綱要和XML綱要表示法的差異性。但是，我們亦特別針對當關聯式綱要與XML綱要的對應關係為多元的情況，討論如何有效與正確的進行雙向查詢句的轉換。本論文設計了數個對應表格來記錄關聯式綱要和XML綱要之間的屬性、集合與結構的多元對應關係，並實做一個轉換系統，依據對應表格所提供的資訊來進行查詢句轉換。

我們設計一系列的實驗，以不同類型的對應關係和查詢句，來分析本論文做法的正確性及效率。實驗證明我們提出的對應表格與轉換系統，可以有效的轉換出正確的查詢句。在效率方面，當有複雜的結構對應時，會花費較多的時間轉換，但整體而言是相當有效率的。

Abstract

Due to the development of Electronic Commerce and Digital Content Industry, extensible markup language (XML) has recently emerged as the standard of data exchange. How to achieve interoperability between relational databases and XML databases is therefore an important issue.

The standard query language for relational database is SQL. However, the standard query language for XML data is XQuery. The purpose of this thesis is to propose a bi-directional query translation system, so that users can easily query different types of schemas and get the required information. To achieve this goal, we need to discuss the representational conflicts between the relational schema and the XML schema. Besides, we also focus on the situation when the relational schema and the XML schema are presented in multiple mappings. We have designed a set of mapping tables to record the complex mappings among attributes, collections and structural conflicts. We also implement a translation system based on these mapping tables.

We have performed a series of experiments to evaluate the correctness and efficiency of our translation system. Experimental results show that our mapping tables and translation system can produce correct queries effectively. On the other hand, the translation system takes more time where there exist complex structural conflicts, but the translation process is quite efficient overall.

誌 謝

首先，特別感謝指導教授 張雅惠博士 於專業領域上的耐心指導與解惑，以及細心逐字逐句的審閱論文，使我的論文得以更趨完善、順利付梓。過程中學習到許多原本不懂的觀念與研究方法，在處事及專業領域上也有所啟發，感恩之情難以言喻。承蒙 柯佳伶博士 與許奮輝博士 於百忙中抽空參與本論文的口試審查，並給予寶貴的意見。謝謝各位恩師的指導，學生都會謹記在心。

感謝父母與家人給我精神及實質上的支持與鼓勵，讓我能無後顧之憂的完成此論文。還有平日互相提攜的實驗室成員 智宏、書賢、介璋、俊賢、政儀 及常協助與照顧我的系助理們；另外，感謝在研究所時期認識的貴人 小哈波 林主任、新光牙科 簡醫師、與菽怡，有你們的陪伴，增添了很多溫馨、愉快的回憶，感謝你們。

研究是無止境的，謹以此成果獻給一路上所有關心我的師長與朋友們，與您一同分享這份喜悅☺

邦鑑 2006.7.31

章 節 目 錄

中文摘要.....	i
英文摘要.....	ii
誌謝.....	iii
章節目錄.....	iv
圖目錄.....	vii
表目錄.....	ix

第一章 序論.....	1
-------------	---

1.1 研究動機與目標.....	1
1.2 相關研究.....	2
1.3 論文架構.....	8

第二章 相關定義.....	9
---------------	---

2.1 關聯式資料與 SQL.....	9
2.2 XML 資料與 XQuery.....	12
2.3 資料表示法差異比較.....	18
2.3.1 內容表示法差異.....	18
2.3.2 集合表示法差異.....	19
2.3.3 結構對應差異.....	20
2.4 查詢句轉換文法範圍與定義.....	22

第三章 系統架構與對應表格.....	25
3.1 轉換系統架構.....	25
3.2 對應表格之內容.....	25
3.2.1 解決基本差異的對應表格.....	26
3.2.2 解決結構差異的對應表格.....	31
3.3 對應表格之建立.....	37
第四章 轉換模組與演算法.....	40
4.1 轉換模組架構.....	40
4.2 解決基本差異的演算法.....	43
4.2.1 Collection 的對應處理.....	43
4.2.2 Value 的對應處理.....	48
4.3 解決結構差異的演算法.....	56
4.3.1 SQL 至 XQuery 的結構轉換處理.....	56
4.3.2 XQuery 至 SQL 的結構轉換處理.....	61
4.3.3 選擇對應結構輸出的方式與演算法.....	64
4.4 查詢句驗證模組.....	68
4.5 查詢句組合模組.....	71
4.6 其它轉換範例與特殊情況探討.....	74
4.6.1 其他轉換範例.....	74

4.6.2 SQL 轉換至 XQuery 可不需路徑代換的情形...	82
4.6.3 關係表格與扁平或巢狀轉換結構選擇之說明...	85
第五章 正確性分析與轉換效率評估.....	95
5.1 正確性分析.....	95
(A) Value 的對應轉換.....	95
(B) Collection 的對應轉換.....	98
(C) Structure 的對應轉換.....	100
5.2 查詢結果分析.....	110
5.3 轉換效率評估.....	114
5.3.1 控制相同的查詢句輸入，改變 Mapping 的對應個數...	114
5.3.2 針對個別的Schema Mapping，控制輸出(改變Input)的個數	116
5.3.3 改變 Collection 與 Join 的個數.....	119
第六章 結論與未來展望.....	121
參考文獻.....	122
【附錄】	
附錄 A.....	125
附錄 B.....	126
附錄 C.....	128
附錄 D.....	131

圖目錄

圖 2.1	關聯式資料範例.....	10
圖 2.2	Order-Ship DTD Graph 範例.....	13
圖 2.3	Cust-Info DTD Graph 範例.....	14
圖 2.4	SQL 文法範圍.....	23
圖 2.5	XQuery 文法範圍.....	23
圖 3.1	雙向查詢句轉換系統架構圖.....	25
圖 4.1-1	SQL 至 XQuery 轉換模組架構.....	40
圖 4.1-2	XQuery 至 SQL 轉換模組架構.....	42
圖 4.2	FromProcessor 演算法.....	44
圖 4.3	ForProcessor 演算法.....	45
圖 4.4	ProjectProcessor 演算法.....	49
圖 4.5	SubstituteExp 演算法.....	49
圖 4.6	GetVarBy_ForSet 演算法.....	49
圖 4.7	ReturnProcessor 演算法.....	51
圖 4.8	usedFromChcek 演算法.....	51
圖 4.9	S2XWhereprocessor_sel 演算法.....	53
圖 4.10	X2SWhereprocessor_sel 演算法.....	55
圖 4.11	PMT_RefPreCheck 演算法.....	57
圖 4.12	S2XWhereProcessor_join 演算法.....	58
圖 4.13	ReconStructFor 演算法.....	59
圖 4.14	AddFlatJoinToWhere 演算法.....	59
圖 4.15	JMT_RefPreCheck 演算法.....	61

圖 4.16	X2SWhereProcessor_FlatJoin 演算法.....	62
圖 4.17	NestedToWhereJoin 演算法.....	64
圖 4.18	S2XSelectStructureList 演算法.....	65
圖 4.19	選擇輸出結構的示意圖.....	66
圖 4.20	X2SSelectStructureList 演算法.....	67
圖 4.21	XQueryValidator 演算法.....	69
圖 4.22	SQLValidator 演算法.....	70
圖 4.23	XQueryConstructor 演算法.....	71
圖 4.24	XQueryConstructor 演算法.....	72
圖 4.25	Flat 結構的 DTD.....	85
圖 4.26	E-R 及結構對應示意圖.....	89
圖 5.A	關聯式表格架構.....	106
圖 5.B	對應 Flat 結構的 DTD.....	106
圖 5.C	結構選擇最佳化 範例與示意圖.....	107
圖 5.1	(S2X) Structure 一對多個數與轉換時間的影響.....	115
圖 5.2	Mapping of Attribute 個數與轉換時間的影響.....	116
圖 5.3	輸出 Attribute 個數與轉換時間的影響.....	116
圖 5.4	Collection 對應個數與轉換時間的影響.....	117
圖 5.5	(SQL to XQuery) Structure 輸出個數與轉換時間的影響.....	118
圖 5.6	(XQuery to SQL) Structure 輸出個數與轉換時間的影響.....	119
圖 5.7	Join 個數與轉換時間的影響.....	120

表目錄

表格 3.1	Collection Mapping Table.....	27
表格 3.2	Internal Join Table.....	28
表格 3.3	附錄 B 中的部份 Internal Join Table 內容.....	29
表格 3.4	部分 Value Mapping Table 內容.....	30
表格 3.5	Join Mapping Table.....	33
表格 3.6	Path Mapping Table.....	33
表格 3.7	Structure Mapping Table.....	33
表格 3.8	部分 Value Mapping Table 內容.....	34
表格 3.9	Join Mapping Table.....	34
表格 3.10	Path Mapping Table.....	34
表格 3.11	Structure Mapping Table.....	35
表格 4.1	範例 4.1 經 FromProcessor 演算法產生之 ForSet 內容.....	46
表格 4.2	範例 4.2 經 ForProcessor 演算法產生之 FromSet 內容.....	47
表格 4.3	範例 4.1 經 ProjectProcessor 演算法更新之 ForSet 內容.....	50
表格 4.4	範例 4.1 經 ProjectProcessor 演算法產生之 ReturnSet 內容.....	50
表格 4.5	範例 4.2 經 ReturnProcessor 演算法更新之 FromSet 內容.....	52
表格 4.6	範例 4.2 經 ReturnProcessor 演算法產生之 SelectionSet 內容.....	52
表格 4.7	範例 4.1 經 S2XWhereprocessor_sel 演算法更新之 ForSet 內容.....	54
表格 4.8	範例 4.1 經 S2XWhereprocessor_sel 演算法產生之 S2XWhereSet 內容.....	54
表格 4.9	範例 4.2 經 X2SWhereprocessor_sel 演算法更新之 FromSet 內容.....	55
表格 4.10	範例 4.2 經 X2SWhereprocessor_sel 演算法產生之 X2SWhereSet 內容.....	56
表格 4.11	範例 4.1 經 S2XWhereProcessor_join 演算法更新之 S2XWhereSet 內容.....	60

表格 4.12	範例 4.2 經 X2SWhereProcessor_join 演算法更新之 X2SWhereSet 內容..	63
表格 4.13	範例 4.1 經 XQueryValidator 演算法更新之 S2XWhereSet 內容.....	70
表格 4.14	範例 4.2 經 SQLValidator 演算法更新之 X2SWhereSet 內容.....	71
表格 4.15	範例 4.1 經轉換模組處理後產生的內容..	72
表格 4.16	範例 4.2 經轉換模組處理後產生的內容.....	73
表格 4.17	範例 4.11 經 FromProcessor 演算法產生之 ForSet 內容.....	75
表格 4.18	範例 4.11 經 ProjectProcessor 演算法更新之 ForSet 內容.....	76
表格 4.19	範例 4.11 經 ProjectProcessor 演算法產生之 ReturnSet 內容.....	76
表格 4.20	範例 4.11 經 Whereprocessor_sel 演算法產生之 S2XWhereSet 內容.....	76
表格 4.21	範例 4.11 經 ReconStructFor 演算法更新之 ForSet 內容..	77
表格 4.22	範例 4.11 經 XQueryValidator 演算法更新之 ForSet 內容.....	78
表格 4.23	範例 4.13 經 FromProcessor 演算法產生之 ForSet 內容.....	79
表格 4.24	範例 4.13 經 ProjectProcessor 演算法更新之 ForSet 內容.....	79
表格 4.25	範例 4.13 經 ProjectProcessor 演算法產生之 ReturnSet 內容.....	79
表格 4.26	範例 4.13 經 XQueryValidator 演算法更新之 ForSet 內容.....	80
表格 4.27	範例 4.15 經 ForProcessor 演算法產生之 FromSet 內容.....	81
表格 4.28	範例 4.15 經 ReturnProcessor 演算法更新之 FromSet 內容.....	82
表格 4.29	範例 4.15 經 ReturnProcessor 演算法產生之 SelectionSet 內容.....	82
表 5.1	性質歸納.....	114

第一章序論

1.1 研究動機與目標

自從 1998 年 W3C 組織正式頒佈了可擴充式標註語言(eXtensible Markup Language; XML)，由於該語言允許使用者針對所需要的資料，自訂標註(Markup)來描述資訊內容意義，所以已經成為網際網路資料交換的格式標準。XML 不僅被應用於企業間的電子資訊交換，用以發展電子商務，目前推動的數位典藏國家型計畫，也設定專責的後設資料工作小組，針對不同性質的文物、器皿及珍貴文史檔案等，訂定以 XML 描述的後設資料 (Metadata)，以便利資料的儲存、交換及查詢。商用資料庫系統如：MS SQL 2005、Oracle 10g、IBM DB2 9 等亦已支援 XML 資料型態的儲存及查詢。

隨著 XML 格式文件的日益增長，當資料分別以關聯式格式和 XML 格式儲存時，會因為所使用的資料庫格式不同，增加了資料管理上的複雜度，也造成了資料整合的困難，這是因為關聯式資料具有固定的結構，而 XML 資料則是半結構化資料(Semi-Structured Data)，資料的表示法有較多的彈性，特別是巢狀結構可以直接表示出集合之間的關係，因此在進行關聯式資料與 XML 資料整合時，必須考慮兩者資料表示法與結構上的差異性；如何解決兩類型資料的不一致性，以達到資料共享的目的，是個重要的課題。所以，我們提出建立一個雙向查詢語言的轉換系統，來達到方便資料交流的目的。關聯式資料的查詢語言，是以結構化的查詢語言 SQL (Structured Query Language) 為主，本論文處理 SQL 的查詢句將著重在 SPJ (Selection-Projection-Join) 類別；XML 資料的查詢語言，則是 SPJ 句型對應的 XQuery 語法為主。本系統可以適用於下述幾個方面：

- 假設一部分的資料是以關聯式表格儲存，另一部分的資料是以 XML 格式儲存，因此使用者下達的 SQL 查詢句或是嵌入在應用軟體中的查詢句，可經由本轉換系統轉換成對應的 XQuery 查詢句，至 XML 的資料庫中查詢到另一部分的資料。

- 在僅有一者資料庫中的實際資料會不斷更新或是綱要對應有所變動時，我們可經由修改對應表格內的對應資訊，利用此轉換系統將輸入之查詢句轉換成對應的查詢句，便可至該資料庫中查詢到最新的資料。
- 若相同的資料是分別以關聯式表格與 XML 格式儲存，經由雙向查詢句轉換系統，使用者可方便的依不同類型需求來下達查詢句。

研究目的，除了建立一個介於關聯式資料庫與 XML 資料之間的轉換系統，透過查詢語言的轉換來方便的達成使用者的需求之外，特別考量當關聯式資料與 XML 資料之間的屬性、集合與結構對應為更複雜的多元對應關係時，如何轉換出正確的查詢句作深入的探討。本論文的主要貢獻，總結如下所述：

1. **探討資料表示法的差異：**分析關聯式資料與 XML 格式資料表示法的差異，其中又可分為屬性、集合與結構的表示法差異。
2. **對應表格的設計與對應資訊的建立：**設計數個對應表格（Mapping Tables），來記錄表示法的差異，將屬性、集合、結構等相關的多元對應資訊建立。
3. **實作雙向查詢句的轉換系統：**實際開發此轉換系統，經由實驗證明，利用對應表格所提供的資訊，我們的轉換模組可正確與有效率的轉換出對應的查詢句。

1.2 相關研究

建立雙向的查詢句轉換系統，會牽涉的相關研究大致可區分為資料格式的轉換、Schema 對應關係的建立與整合維護、查詢句轉換三大方面。後續將依此三項分類介紹相關的研究。

(1) 不同格式資料間的轉換

從關聯式資料庫中將資料取出成為 XML 文件方面的研究，請參考[BCF+02, FKS+02]。其中[BCF+02]使用 Attribute Translation Grammar (簡稱 ATG) 將關聯

式資料庫的資料依循 DTD 定義取出，再將資料組成一份 XML 文件。首先分析定義好要輸出的 DTD 的語意，由 DTD 語意產生 ATG Graph，將 ATG Graph 和關聯式資料庫的資料輸入到系統中，配合 XML 結構分析函式產生出 XML 文件。此研究對於非決定論的 DTD 會先做正規化的處理，而有遞迴定義時，會套入統計運算，算出適當的深度後，再將資料依循遞迴定義聚合起來。[FKS+02]論文討論將關聯式資料以 XML View 的方式 Publish 之後，使用者會再下達 XQuery 的查詢句，取出所欲之資料。但是由於 XQuery 和 SQL 的表現能力不同，可能必須重複的下達同一個 SQL 查詢句，所以該篇論文提出一個 middleware 的架構，以達到此功能。

將 XML 文件資料轉入關聯式資料庫方面，需要參考 DTD 資訊的如[ML02, SKZ+99]，不需使用 DTD 資訊的請參考[SYU+99, YAS+01]。[SKZ+99]根據 DTD 建立 DTD Graph，以簡化後的 DTD Graph 結構來定義資料庫的表格定義，並提出 Basic Inlining、Shared Inlining 和 Hybrid Inlining 三種方法：Basic 對每個節點產生一個對應表格，並將子孫節點全部包含進來變成自己的欄位，除了節點是*型態和有遞迴性質的；Shared 依循 Basic 的方法，但是有多個父節點的元素（in-degree > 1），也需自成一個表格；Hybrid 是和 Shared 相似，但是有多個父節點的元素，只要不具有遞迴關係亦可被包含進來變成自己的欄位。[ML02] 為了讓關聯式 Schema 能清楚明確地在 XML Schema 上表示其限制式，使用規則樹文法（Regular Tree Grammar）轉換 XML Schema 成自行定義的 XSchema，從 XSchema 對應到關聯式 Schema 之間，會先做 Schema 的簡化和套用 Hybrid Inlining 方法，而 IDREF 和 IDREFs 的限制式，則使用和 inlining 類似的做法，將 IDREFs 屬性用 IDREFsinling 函式處理。對於遞迴結構則使用 Foreign key 關係來判斷，最後對於每個表格做主鍵值包含性關係處理，合併或減少不必要的表格和欄位。

[SYU99]忽略 Schema 將 XML 文件資料轉入關聯式資料庫，將 XML 文件所有節點分成元素、屬性和文字節點，並取出所有路徑。首先分析元素節點，取出

順序關係、在 XML 檔案中的絕對位置和對應的路徑；屬性節點記錄文字值和絕對位置；文字節點記錄文字值、在檔中的範圍和對應路徑；最後是所有可能路徑，將此四類資料匯入資料庫中成為各別的表格中。[YAS+01]是延伸[SYU99]的做法，主要著重在 XPath 查詢句轉換成 SQL 查詢句，由於記錄 XML 文件的所有不重複路徑，在處理 AD (Ancestor-Descendant) 表示式，將‘//’轉成‘%/’，利用 SQL 的 LIKE 限制式，將原本需要多條結合限制的查詢句，變成簡單的字符串比對。

另外，[BDH04]說明XML Document (or view) 在目前某些應用程式直接被用來作為輸出或修改資料的介面，此篇論文提出如何從XML View直接去Update關聯式資料庫中的資料。作法為定義Query Tree用來建立XML View，並定義其中的5種Abstract Type。還有提出一個演算法能偵測從XML View對資料Insert、Delete、Modify是否正確。最後提出能對資料做Insertion、Modification、Deletion運算的相關演算法。

(2) Schema 對應關係的建立與整合維護

Schema 的對應和整合，[CR03] 提出一種具彈性、有延伸性且可使用性的轉換模組化平台—Sangam。使用者可以輸入 XML DTD 或關聯式資料庫 Schema，接著依據 Sangam 的定義轉成 Sangam Graph，使用以圖形化方式定義的運算子以整合 Sangam 圖形，之後再將整合好的 Sangam 圖形套入模組轉換的執行策略中，最後可以輸出成 XML DTD 或關聯式資料庫 Schema。[XE03]挑戰自動對應，除了從一份來源 Schema 直接對應目標 Schema 的元素外，亦可間接對應元素。在間接對應上會遇到一般化、特殊化、合併數值、分割數值和元素名稱為數值的問題，之後在對應技術上，分別有 1.分析來源端和目標端的元素名稱字詞語意分；2.比較名稱長度和文數字比率；3.用輕量的 ontology domain 分析間接對應出現的問題，對一般化和特殊化用 Union 和 Selection 運算子解決、合併數值和分割數值用 Composition 和 Decomposition 解決、元素名稱為數值用 Boolean 來表示，

前三種做法中都會給予積分，並計算出期預值；4.比較 Schema 的結構。

[LYY02] 討論如何計算DTDs的相似度，以達到有效率的文件整合。考慮的部分有1.Semantics similarity：計算元素名稱、限制式還有對應路徑的相似度 2. Immediate descendents similarity：比較子孫的相似度（attributes and subelements） 3. Leaf-context similarity：比較各Subtree中，leaf nodes的相似度。針對上述項目，分別給予權重並相加計算，以提供最後的判斷。[MH03] 則是針對分別當A對應到B，B對應C的時，如何產生一份A對應C的直接對應且等價於原來的對應關係。在Schema的對應上，對結構編碼，編碼出來的定式建立與資料端的關聯性，再進行限定大小的合併。而在查詢句對應上，查詢句是限制在沒有比較式的SPJ範圍內，和Schema對應相仿，最後會建立Query-Rewrite Graph，並保證重新撰寫後合併產生出來的查詢句最佳化。[YLL03] 則是將XML Schema中的語意轉換成ORA-SS（Object-Relationship-Attribute Model for Semi-Structured Data），如主鍵值、兩個物件之間的關係和作用依賴性等，配合此研究提出的演算法，解決整合時所會遇到的衝突，主要四個步驟為：1.解決屬性—物件衝突、一般化和特殊化物件包含關係；2.計算每一條關係上的權重，並先忽略物件上的屬性建立整合的圖形；3.轉換圖形，移除權重較高和多餘的關係、移除循環關係和多餘物件，以及移除多重父節點的物件；4.將每個物件的屬性增加到整合的圖形上。

[AL05]說明因為不同的需要可能將原本的XML資料改成其它DTD結構的XML資料，允許直接在新的XML文件中查詢，而查詢結果必須與原始的XML檔案查詢結果一致。作法為用Tree Pattern表示source-to-target dependence，由STDs計算得到canonical pre-solution cps(T)，再利用target DTD計算 T^* ， T^* 是T的canonical solution，並提出query language(CTQ//)，用來查詢答案。由於檢查data exchange setting是否為consistent需要EXPTIME-complete，所以此篇作法皆在tractable case下執行。[WXD04] 介紹利用data frames與domain ontology snippets來直接或間接的建立Schema的對應關係，且可處理多元的對應，此處的對應關係是例如一個Object Set中的Address字串，可能包含了Street、City與State資訊，

但另一個Schema中，Street、City與State分別為3個不同的Object Set，而Address則是組合這三個Object Set的資訊，此情況稱為一個1對N的對應關係。與本論文較不同的是，我們考量的一對多對應，是例如相同欄位，會對應到DTD中不同的葉節點，不是將同一個欄位的內容細分後再建立對應。

在schema change方面[JB04]說明有時資料需要做schema change，而會造成intended schema of tuples與recorded schema of tuples mismatch的問題，本篇論文即在處理此問題，使查詢能得到正確答案。作法為定義evolving schema為其基本架構。然後提出add與delete的邏輯運算式，能正確更改attribute。並提出4種Mismatch Types (M1~M4)，用以記錄intended schema與recorded schema之間關係。最後提出自己的MECS系統，在查詢上運用projection、replacement、exclusion三種邏輯運算式，以得到正確答案。

(3) 查詢句轉換

查詢句轉換的問題，[DTC+03]研究如何轉換巢狀結構、並包含複雜路徑的FLWR表示式。它使用動態區間編碼來輔助轉換和資料查詢的動作，對XML文件用深度搜尋演算法(DFS)對每個元素、屬性和文字節點編碼，依此編碼可以找出以某個元素為頭下面整個區間的資料，利用區間編碼的特性，找出元素屬性之間的層次關係，以解決PC路徑、AD路徑和Wildcard的問題。[KCK+04]針對Recursive的XML Schema進行XML-to-SQL的查詢句轉換，將XML to Relational的對應schema S轉換成automaton AS，並將對應的XQuery轉換成Finite automaton AQ，而最後出來的SQL，利用SQL99中的with子句將各SQL子句結合起來。[JLS+04]則是對XML的架構，利用顏色來區分XML Tree中不同的屬性及其層次關係，如此一來可以清楚的劃分及查詢資料，並可減輕扁平結構DTD查詢句的join次數。

[KKN04]此篇論文提出在Fixed-Schema Mapping且XML-to-relational mappings是bijective column mapping的前題下，經由判斷Least Distinguish

Ancestor，來接續進行下列兩個主要步驟 (1)Prefix-Elimination：化簡 DTD 中，重覆且不必要的路徑 (2)Grouping：判斷並將前述步驟轉換成 SQL 後，所產生相同的 relation 合併起來，避免產生 duplicate 的結果；經由上述步驟後，即可對 Simple Path Expression 做最佳化的動作。Branch Path Expression (with a single predicate)部分，則是經由 branching node、result node、condition node 去判斷及做類似 Simple Path Expression 化簡的動作。

由於 XML 常利用來當作資料交換的格式，在某些狀況下 如：分散式系統，它可以 Stream 的方式傳輸，[PC03]主要目的在不需要等待大量完整的 XML 文件傳輸完成，即可針對所需要的部份，進行 XQuery 查詢的動作。其所利用的技巧為使用 Pushdown Transducer，針對 Query 設計 Pushdown automata，state 至 state 之間的轉換關係可以是屬性或是 element 名稱，轉換至 accept state 即代表一個查詢的結果輸出。Basic Pushdown Transducer 可以組成階層式的 Pushdown Transduce，下層的 BPDT 經由 Q.upload() 函式，將符合的資料輸入至上層的 BPDT，如此一來，可分段簡化複雜的查詢句處理。

當合併許多資料來源時，可能就會造成資料不一致性，然而要確保資料在 RDB 中符合一致性是需要花費很多的時間，[FFM05]作法是去改寫使用者的 Query，使回傳的資料能符合一致性。主要提出 ConQuer System，讓使用者可以在 SQL Query 中下 key constraints，進而將 Query 改寫，使得回傳結果在原本資料中符合一致性，並且可以處理 Query 中包含 Aggregation。

[PBM04]介紹一個 SUCXENT system，可將 XML data 利用 Path-Based 方法將 xml 文件資訊儲存在幾個 RDB 表格中，並且將 XQuery 轉換為 SQL Query 在 RDB 中查詢，回傳結果為 xml data。其作法為提出轉換方式，將 xml data 分別儲存在 RDB 中的 Document、PathValue、Path、AncestorInfo 表格中。利用 AQT (Abstract Query Tree)，將 XQuery 轉成 SQL Query，然後在 RDB 中找出符合的欄位資料。最後藉由 Extraction algorithm 將找出的欄位資料重組回 xml data 回傳給使用者。

大多論文處理將XML儲存在RDB中而進行查詢的研究，皆無法處理DTD存在Recursive結構的關係，[FYL+05]在挑戰DTD結構為Recursive時，對應的SQL查詢句轉換，與其目的較相似的論文[KCK+04]是使用SQL中的with子句來達成。而[FYL+05]提出Oracle、IBM DB2、Microsoft SQL Server皆有支援的 Simple Least Fixpoint Operator (LFP)： $\Phi(R)$ 來進行轉換，且經過實驗證明效率比利用With語法佳。

1.3 論文架構

本篇論文架構如下：在第二章中，將針對關聯式資料與XML格式資料的表示法差異先做介紹與比較，接著介紹我們雙向查詢句轉換可處理的語法範圍。第三章說明所建立對應表格的方式，以及其相關的定義與內容，其中分為屬性、集合與結構的對應關係。第四章是轉換系統核心的模組介紹並進一步說明相關的演算法及轉換過程。針對正確性與效率的分析，我們分析數個不同類型的查詢句轉換，還有進行數個不同對應關係的實驗於第五章中。最後在第六章提出本篇論文的結論與未來展望。

第二章相關定義

在本章中，首先介紹關聯式綱要與 XML 綱要的範例及查詢句，進而比較資料表示法和結構上的不同，探討結構化查詢語言 SQL (Structured Query Language) 與 XQuery (XML Query) 查詢句的語法差異及所衍生的對應問題，以及正式定義本論文要解決的問題。

2.1 關聯式資料與 SQL

圖 2.1 是依據 TPC-H Benchmark (美國交易處理效能評議會建立的一項業界標準決策支援測試)[TPCH] 所適度修改與簡化欄位個數的關聯式資料範例。共有九個表格，其中實體表格 SUPPLIER 表格記錄供應商資訊，包含了供應商的名稱與地址；PART 表格記錄零件的名稱與種類型式 (如 “LCD” 的型式有 “17 吋、19 吋” 等)；CUSTOMER 表格記錄顧客的名稱與附註資訊；而 CUSTEL 為從顧客資料中經過正規化獨立出來的表格，其中記錄了顧客的電話資料；ORDER 表格記錄訂單的內容，包含了訂單的處理狀態與總金額；NATION 與 REGION 表格則是記錄了供應商與顧客的地理位置；此外，關係表格 PARTSUPP 表格記錄供應商表格與零件表格間的關係及庫存量；LINEITEM 表格則是記錄供應商、零件與訂單間的關係，欄位 LINENUMBER 代表出貨單的編號，而 SHIPMODE 為運送方式。

表格中的主鍵 (Primary Key) 是尋找或查詢資料的依據，具有唯一性，我們在鍵值下方加上實線底線表示，其中又可分為單鍵 (Simple Key) 如 PART 表格中的 PARTKEY 或是組合鍵 (Composite Key) 如 PARTSUPP 表格中的 SUPPKEY 與 PARTKEY；而外來鍵 (Foreign Key) 則是參照其它表格的 Primary Key 來建立資料表之間的關聯，以虛線底線表示。在關聯式資料庫中，表格間彼此的關聯性是利用相同屬性的鍵值來建立連結，如實體表格 SUPPLIER 與關係表格 PARTSUPP 表格是利用相同屬性的鍵值 SUPPKEY 來建立連結。實體表格與關係

表格的區分如定義 2.1 中所示，其中 PARTSUPP 表格利用 SUPPKEY 與 PARTKEY 建立 PART 與 SUPPLIER 表格之間的關聯，為二元的關係表格；而 LINEITEM 表格則是由 ORDERKEY、SUPPKEY 與 PARTKEY 建立 ORDER、SUPPLIER 及 PART 三的表格間的關聯，為一個三元的關係表格。

SUPPLIER (SUPPKEY , NATIONKEY , NAME , ADDRESS)
PART (PARTKEY , NAME , TYPE)
CUSTOMER (CUSTKEY , NATIONKEY , NAME , COMMENT)
CUSTEL (CUSTKEY , TELEPHONE)
ORDER (ORDERKEY , CUSTKEY , ORDERSTATUS , TOTALPRICE)
NATION (NATIONKEY , REGIONKEY , NAME)
REGION (REGIONKEY , NAME , COMMENT)
PARTSUPP (PARTKEY , SUPPKEY , AVAILQTY)
LINEITEM (ORDERKEY , LINENUMBER , SUPPKEY , PARTKEY , SHIPMODE)

圖 2.1：關聯式資料範例

【定義 2.1】實體表格與關係表格

- ◆ **實體表格**：實體可藉由一組屬性的集合來表示，實體表格可視為多組相同型態的實體所形成的集合；可用特定的鍵值編號來代表一個實體。
- ◆ **關係表格**：記錄不同實體表格內的鍵值，連結實體表格間的關係，可為多元的關係。

結構化查詢語言 SQL 是關聯式資料庫管理系統的標準查詢語言，一個基本的 SQL 資料查詢句，主要可分為 SELECT、FROM、WHERE 三子句，其中 SELECT 子句表示選擇輸出的資料表格欄位名稱，FROM 子句表示提供資料的來源表格，

WHERE 子句則允許對資料表格中的欄位做相關限制，如定義 2.2 又可分為條件限制式和連結限制式，例如範例 2.1 的 WHERE 子句中，PART.NAME = 'dvd' 即為條件限制式，而 SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY 為連結限制式。本論文處理 SQL 的查詢句將著重在 SPJ (Selection-Projection-Join) 類別，該類別的特色是針對一個資料表格或將數個資料表格結合，並可對某些欄位限定條件，最後將滿足條件限制的資料列予以回傳。

【定義 2.2】：SQL 之條件限制式與連結限制式

- ◆ 條件限制式 (Selection Condition)：在 SQL 的 WHERE 子句中，限制式為文數值限制，表示法為：(表格欄位 + “比較運算子”+ 運算元) 或 (表格欄位 + “比較運算子”+ 表格欄位)。
- ◆ 連結限制式 (Join Condition)：在 SQL 的 WHERE 子句中，限制式代表兩個表格間的連結關係，表示法為：(表格的鍵值欄位 “ = ” 表格的鍵值欄位)。

以下列出兩個針對圖 2.1 關聯式資料所設計的 SQL 查詢句範例，範例 2.1 的查詢句目的在找出零件名稱為 dvd 的供應商名稱與零件型式，資料的來源為 SUPPLIER、PART 與 PARTSUPP 三個表格，由連結限制式中可看出其利用 SUPPKEY 與 PARTKEY 來建立表格間的關聯，並由條件限制式限定零件的名稱為 'dvd'。而範例 2.2 查詢句的目的在找出在台灣顧客相關資料，資料的來源為 CUSTOMER、CUSTEL 與 NATION 三個表格，由 Where 子句中可看出其利用 NATIONKEY 與 CUSTKEY 來作為表格間的連結。

【範例 2.1】

```
SELECT  SUPPLIER.NAME , PART.TYPE
FROM    SUPPLIER , PART , PARTSUPP
WHERE   SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY
        AND
        PART.PARTKEY = PARTSUPP.PARTKEY
        AND
        PART.NAME = 'dvd'
```

【範例 2.2】

```
SELECT  CUSTOMER.NAME, CUSTEL.TELEPHONE,
        CUSTOMER.COMMENT
FROM    CUSTOMER , CUSTEL, NATION
WHERE   CUSTOMER.NATIONKEY = NATION.NATIONKEY
        AND
        CUSTOMER.CUSTKEY = CUSTEL.CUSTKEY
        AND
        NATION.NAME = '台灣'
```

2.2 XML 資料與 XQuery

XML (eXtensible Markup Language) 源屬於 SGML (Standard Generalized Markup Language)，其特點是屬於半結構化資料 (Semi-Structured Data)，資料的表示法有較多的彈性，允許使用者自行定義元素標籤以說明內容資料之意涵，為一個階層式的樹狀結構，多用來做為資料交換的格式，其亦可用來表示資料庫的

資料；相較之下關聯式資料具有較固定的格式結構，為結構化資料 (Structured Data)。

為了驗證 XML 文件的結構是否合乎使用者需求、標籤是否正確，W3C (World Wide Web Consortium) 組織頒訂文件型別定義 DTD (Document Type Definition) 予以檢查其文件格式，將每一個元素包含哪些子元素、屬性以及各個元素出現的順序等，清楚地加以定義與規範。在論文中我們將複雜的 DTD Schema 定義，以 DTD Graph 表示，如圖 2.2 與 2.3 所示。它們是針對加拿大多倫多大學 Clio 計畫中的 TPC-H Nested XML Schema [CLIO] 範例稍加修改而來，其記錄顧客與供應商間的貨品訂購資訊。圖 2.3 是額外設計用來說明本論文針對結構對應處理的範例，它用來記錄顧客的資料與地理資訊，針對 VIP 客戶，我們改將其附註資訊記錄於中子樹中；此外，右子樹記錄地理位置資訊是針對實際 TPC-H 關聯式綱要所設計的项目，也就是讓關聯式表格 NATION 與 REGION 分別對應到 nation 與 region、region_popu 元素。

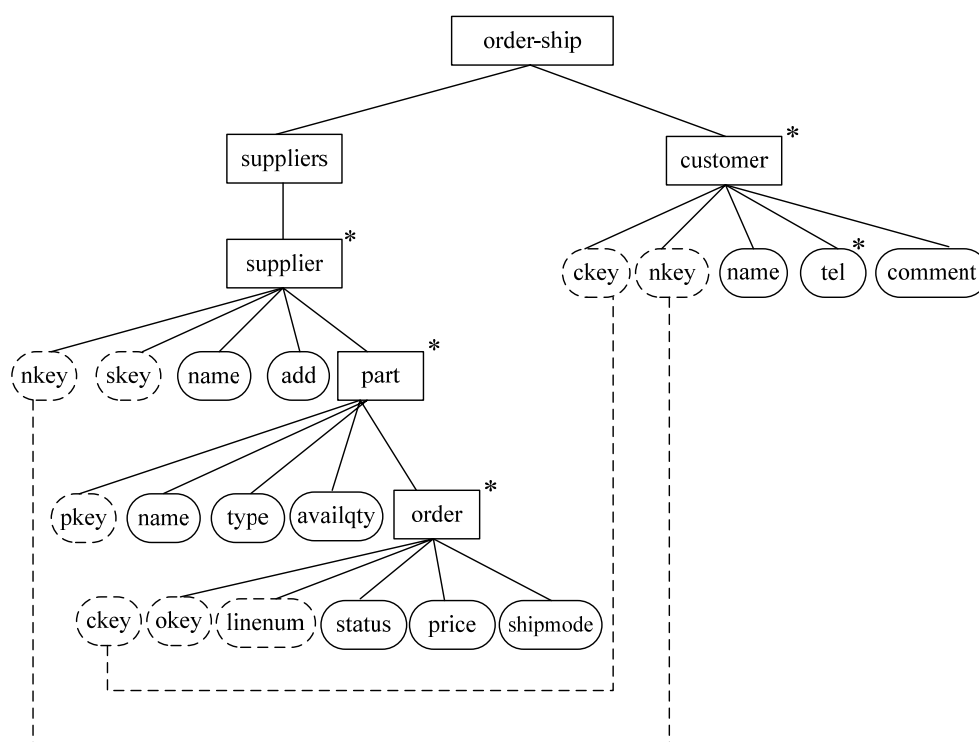


圖 2.2：Order-Ship DTD Graph 範例

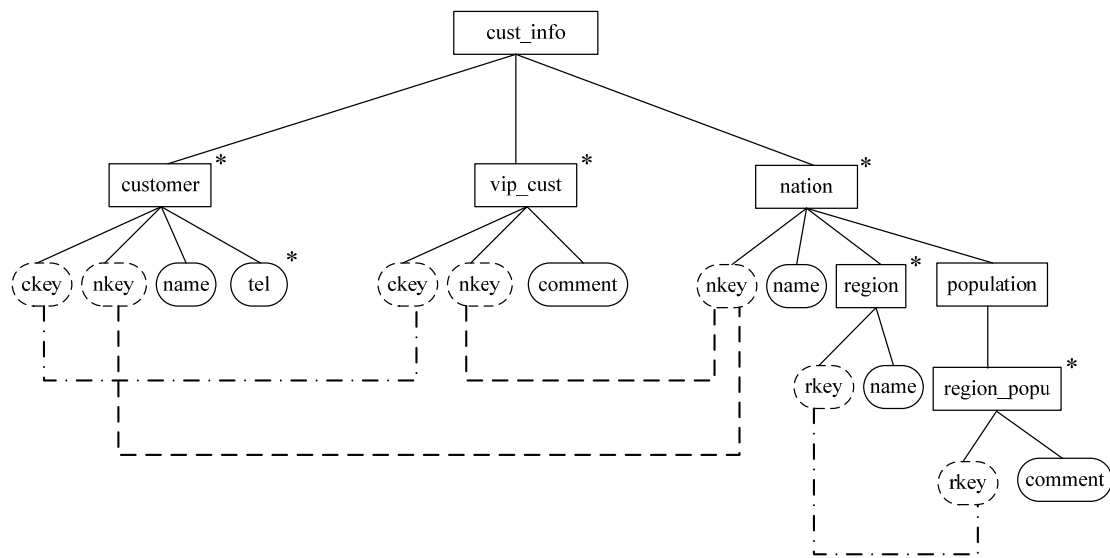


圖 2.3：Cust-Info DTD Graph 範例

以下為 DTD Graph 中，相關節點及連接線定義：

【定義 2.3】 DTD Graph 節點定義

◆ 根節點 (Root Node)：

- 根節點是對應到 DTD 中的根元素，圖中以單實線方塊表示。
- 樹上其它所有元素節點都是它的子節點或後代節點。
- 可以沒有對應的關聯式表格。

◆ 內部節點 (Internal Node)：

■ 可重覆元素 (Repeatable Node)：

- 在 XML 文件中，針對同一個父元素，可出現零次以上。
- DTD 中以單實線方塊、右上方加上星號 “*” 表示。
- 對應於關聯式資料庫中的表格。

■ **空元素 (Dummy Node) :**

- 在 XML 文件中，針對同一個父元素，只能出現零次或一次。
- DTD 中以單實線方塊表示。
- 僅用來包覆子元素的資料。
- 其對應的表格名稱與第一個有對應的子孫元素其對應到關聯式表格相同。

◆ **葉節點 (Leaf Node) :**

■ **內容節點 (Value Node) :**

- 為一個有值的元素，DTD 中以單實線橢圓表示。
- 若為多值情形，則於橢圓右上方加上星號 “*” 表示。
- 對應至關聯式表格中的欄位。

■ **屬性節點 (Attribute Node) :**

- 為一個屬性。
- DTD 中以虛線橢圓表示。
- 對應至關聯式表格中的鍵值欄位。

◆ **實線 :**

- 連結元素與其子元素或屬性，為一個父子 (parent/child) 關係。

◆ **虛線 :**

- 連結相同意義的屬性節點，其中又分為：

意義不同集合之間的連結 -----

意義相同集合之間的內部連結 -----

以圖 2.2 為例，該範例是在該文件根元素 (order-ship) 下分為兩個子樹，左子樹囊括了所有供應商 (supplier)、零件 (part) 與訂單 (order) 相關的資料，其中 supplier 為可重覆元素，其子元素 name 為一個內容節點。內部節點 supplier 與 part 亦為一個父子關係，我們以實線來連結。order-ship 的右子樹記錄了顧客

(customer) 的資料，與左子樹之間的連結關係，便是以虛線來連結與其相對應的屬性 ckey。另外，特別注意到圖 2.3 中右子樹的 population 元素為一個空元素，所對應的表格名稱，將會與 region_popu 元素相同。

XML 文件內的元素之間有著階層關係，W3C 組織定義了相關技術來處理 XML 資料，譬如以路徑表示法 (Path Expression) 為基礎的 XQuery。路徑表示法自根元素 (Root Node) 透過位址步驟 (Location Step) 結合 XML 文件中的元素名稱所組成，指出資料在 XML 文件裡的位置，結果為一序列的元素集合。在本論文中，位址步驟僅考慮一序列的父子關係，以 “/” 符號來表示，其中屬性是以 “@” 符號表示。如圖 2.2 中可重覆元素 supplier 的子元素 name 的路徑表示法是 /order-ship/suppliers/supplier/name，而屬性節點 skey 路徑表示法則為 /order-ship/suppliers/supplier@skey。

【定義 2.4】：XQuery 之條件限制式與連結限制式

- ◆ 條件限制式 (Selection Condition)：在 XQuery 的 WHERE 子句中，限制式為文數值限制，表示法為：(葉節點路徑 + “比較運算子”+ 運算元) 或 (葉節點路徑 + “比較運算子”+ 葉節點路徑)。
- ◆ 連結限制式 (Join Condition)：在 XQuery 的 WHERE 子句中，限制式代表兩個內部節點間利用其下相同意義的屬性來建立連結關係，表示法為：(屬性節點路徑 “=” 屬性節點路徑)。

W3C 定義的 XQuery 具有複雜的結構，在此我們僅考慮轉換核心的表示式，其中，FOR 以遞迴方式取得一個路徑表示法的集合結果，並給予一個變數名稱；WHERE 則允許對變數所代表的集合內容做條件的限制，根據下達的限制式內容，過濾取出所需要的資料，其中可分為條件限制式與連結限制式，連結限制式中，我們以相同意義的屬性來建立連結；RETURN 子句則將建構好的資料回傳給使用者。以下範例 2.3 為針對圖 2.2 的一個 XQuery 範例，會對應到範例 2.1：

【範例 2.3】

```
FOR   $t0 in /order-ship/suppliers/supplier ,  
      $t1 in $t0/part  
WHERE $t1/name = “dvd”  
RETURN $t0/name , $t1/type
```

參考圖 2.2，其中 FOR 子句 `$t0 in /order-ship/suppliers/supplier`，結尾元素 `supplier` 為一個可重覆元素，我們以 `$t0` 變數來代表取得路徑表示法 `/order-ship/suppliers/supplier` 之下的所有元素內容集合。此外，之後定義的變數，還可接續之前變數所取得之路徑結果繼續延伸，如 `$t1 in $t0/part`，此寫法是在建立 `$t0` 與 `$t1` 的關聯，表示我們要求 `$t1` 的內容，是取得在 `$t0` 變數所代表路徑其子元素 `part` 之下的所有元素內容集合。而 XQuery 中的 `WHERE $t1/name = “dvd”` 子句，`$t1/name` 代表取得路徑 `/order-ship/suppliers/supplier/part` 之下的所有 `name` 元素內容，此條件限制式在限定取出供應零件名稱為 “dvd” 的資料。最後以 `RETURN $t0/name , $t1/type` 子句，將符合條件的供應商名稱與零件型式輸出。

以下的範例 2.4 為針對圖 2.3 所設計的一個 XQuery 範例，會對應到範例 2.2。值得注意的是 WHERE 子句中的 `$t0@nkey = $t1@nkey` 是限定 `$t0` 所代表路徑 `/cust_info/customer` 下的 `nkey` 值，必須與 `$t1` 所代表路徑 `/cust_info/nation` 下的 `nkey` 值相同，也就是利用相同的屬性值來建立連結關係，其餘的連結限制式 `$t2@nkey=$t1@nkey`、`$t0@ckey=$t2@ckey` 亦是如此。另外，條件限制式 `$t1/name = “台灣”`，則是限定取得 `$t1` 所代表路徑 `/cust_info/nation` 之下的葉節點 `name` 內容為 “台灣”。最後以 `RETURN $t0/name, $t0/tel, $t2/comment` 子句，將符合條件的顧客名稱、電話與附註資訊輸出。

【範例 2.4】

```
For $t2 in /cust_info/nation,  
    $t0 in /cust_info/customer,  
    $t1 in /cust_info/vip_cust  
Where $t2/name= '台灣'  
  
    And    $t0@nkey=$t2@nkey  
    And    $t1@nkey=$t2@nkey  
    And    $t0@ckey=$t1@ckey  
  
Return $t0/name, $t0/tel, $t1/comment
```

2.3 資料表示法差異比較

本節中我們比較關聯式綱要與 XML 綱要表示法的差異與對應關係。

2.3.1 內容表示法差異

關聯式資料庫的資料是表示於表格的欄位中，取得資料的表示法可用 SELECT Relation.Attribute 來表示；其中 Relation 為表格的名稱，Attribute 為欄位的名稱，如 SUPPLIER 表格中的 NAME 欄位即是以 SUPPLIER.NAME 表示。但在 XML 文件中，元素之間有著階層關係，形成一樹狀結構，資料是表示於內容節點或是屬性節點中。此部分的表示法差異有以下的情形：

- A. 意義相同但表示的名稱不同：**關聯式綱要與 XML 綱要，欄位與元素的對應名稱並不一定一致，如前述 SUPPLIER 表格中的 SUPPKEY 欄位，對應的屬性節點為 supplier 元素下的 skey。
- B. 資料的選擇性：**有時資料取捨上的需要，XML 文件中可能僅記錄了部分資料，因此不一定欄位就會有對應到的葉節點，反之亦然。
- C. 欄位與葉節點一對多的關係：**表格中的同一個欄位，於對應的 DTD 中，對應到兩個以上位於不同路徑的葉節點。如圖 2.1 中 CUSTOMER 表格的 CUSTKEY

欄位，同時對應到圖 2.3 中路徑/cust_info/customer 與/cust_info/vip/vip_cust 下的屬性節點 ckey。

D.欄位與葉節點多對一的關係：不同表格中的相同語意欄位，於對應的 DTD 中，對應到相同路徑的葉節點。如圖 2.1 中 CUSTOMER 與 CUSTEL 表格中的 CUSTKEY 欄位，同時對應到圖 2.2 中路徑/order-ship/customer 下的屬性節點 ckey。

2.3.2 集合表示方式差異

關聯式資料庫中，是以表格來儲存一組資料列的集合 (Collection)；但在 XML 文件中，相關的一組資料集合，通常是被建立包含於某個內部節點之下，以路徑表示法來取得。此部分的差異情形除了與內容表示法的前兩項類似之外，尚有以下情形：

A.表格與內部節點多對一的關係：DTD 中的同一個內部節點，同時對應到兩個表格。如圖 2.1 中，CUSTOMER 表格與 CUSTEL 表格皆代表顧客的資訊，在我們的設計中皆會對應於圖 2.2 的可重覆元素 customer，路徑表示法皆為 /cust_info/customer。

B.表格與內部節點一對多的關係：除了同一路徑上的可重覆元素與空元素有可能對應到相同的表格名稱之外 (如定義 2.3 所示)，可能有不同路徑上的內部節點，對應到相同的表格名稱情形。也就是說同一份表格的欄位，於對應的 DTD 中是分散放置到不同的內部節點之下，例如圖 2.1 中的 CUSTOMER 表格，其中 NAME 與 COMMENT 欄位，分別對應於圖 2.3 內部節點路徑為 /cust_info/customer 下的 name 元素與/cust_info/vip/vip_cust 下的 comment 元素，所以 CUSTOMER 表格會對應到路徑表示法 /cust_info/customer 與 /cust_info/vip/vip_cust。

2.3.3 結構對應差異

在關聯式資料庫中，儲存資料的結構是以扁平（Flat）的表格方式存在，表格之間的連結關係是用相同的鍵值來達成。如 NATION 與 REGION 表格，是以 REGIONKEY 來建立表格之間的關聯。在 XML 方面，對於資料集合間的關聯性，基本上大致可分為扁平結構與巢狀結構，因為集合對應方式亦有可能不是一對一的關係，使得巢狀與扁平結構的對應關係亦更加複雜，我們將此差異衍生而出的對應問題分述如下：

A.關聯式連結與扁平結構一對一的關係：扁平結構資料集合間的關聯性，是類似於關聯式資料的處理方式，間接地以相同意義的屬性值來連結資料間的關係；如圖 2.2 中可重覆元素 customer 與 order 元素，便是利用相同意義的屬性值 ckey，來達成連結，此結構是對應於關聯式 CUSTOMER 與 ORDER 表格利用鍵值 CUSTKEY 來建立連結的關係，連結限制式為 CUSTOMER.CUSTKEY = ORDER.CUSTKEY。

B.關聯式連結與巢狀結構一對一的關係：DTD 中的一條路徑，可以由兩個以上的可重覆元素形成的一組巢狀結構，在關聯式資料庫中則是對應到關聯式表格間的連結限制式。如圖 2.1 關聯式資料表格 NATION 與 REGION 表格間的關聯是利用 REGIONKEY 來做為連結，於圖 2.3 中可重覆元素 nation 與 region 為一巢狀結構，取得可重覆元素 nation 下的 region 相關資訊，我們可以直接以路徑表示法/cust_info/nations/nation/region 表示即可，不需額外其它的連結資訊。

C.關聯式連結與扁平結構一對多的關係：由於表格與內部節點一對多的關係，可能有一個關聯式表格對應到不同路徑上的內部節點情形，因此會有相同的一組連結限制式對應到不同的扁平結構。如圖 2.1 關聯式表格 CUSTOMER 與 NATION 表格利用 NATIONKEY 來連結，但在本論文圖 2.3 的範例中，CUSTOMER 表格會對應到 customer 與 vip_cust 兩個可重覆元素，因此關聯式連結會對應到 (nation 元素與 customer 元素) 或是 (nation 元素與 vip_cust 元素)

的兩組扁平結構的關係。

D.關聯式連結與巢狀結構一對多的關係：

- ◆ 兩個內部節點所形成的一組巢狀結構間，若有一個空元素，由 2.2 節中得知空元素所對應的表格名稱，會與其最接近的子孫層節點為可重覆元素的元素相同，可能因此會有相同的一組連結限制式對應到不同層次的巢狀關係，如關聯式表格 NATION 與 REGION 的連結限制式會對應到圖 2.3 中 (nation 元素與 population 元素) 或是 (nation 元素與 region_popu 元素) 的巢狀關係。
- ◆ 圖 2.1 的關聯式表格 REGION，在本論文中將會對應到圖 2.3 可重覆元素 region 與 region_popu，而此兩元素皆為可重覆元素 nation 的子元素，因此關聯式表格 NATION 與 REGION 的連結限制式會對應到圖 2.3 中 (nation 元素與 region 元素) 或是 (nation 元素與 region_popu 元素) 的巢狀關係。

E.關聯式連結與扁平結構多對一的關係：不同的關聯式連結，會對應到同一組扁平結構，此種情形會發生在有一個可重覆元素對應到兩個以上的關聯式表格時發生。如圖 2.1 關聯式表格 ORDER 與 CUSTOMER 或 CUSTEL 表格之間皆是利用 CUSTKEY 來連結，但在本論文圖 2.2 的範例中，ORDER 表格會對應到可重覆元素 order，而 CUSTOMER 與 CUSTEL 在本論文中皆對應到可重覆元素 customer，因此連結限制式 ORDER.CUSTKEY = CUSTOMER.CUSTKEY 與 ORDER.CUSTKEY = CUSTEL.CUSTKEY 皆對應到 order 元素與 customer 元素此組扁平結構關係。

F.關聯式連結與巢狀結構多對一的關係：不同的關聯式連結，會對應到同一組巢狀結構，此種情形會發生在有一個可重覆元素對應到兩個以上的關聯式表格時發生。如附錄 D(V-2)中圖 D(V-2)-1 的關聯式表格架構與對應圖 D(V-2)-3 Nested 結構的 DTD，其中可重覆元素 a 會對應到表格 A1，而可重覆元素 b 會對應到表格 B1~B4，因此/root/a 與/root/a/b 此組巢狀結構，會對應到 A1 表格

與 B1~B4 表格之間的 4 個連結限制式。

G.關係表格與巢狀結構的關係：可重覆元素間所形成的巢狀結構，除了可能對應到兩個實體表格之間利用相同屬性的鍵值來連結外；尚有可能另外經由一個關係表格來建立兩個以上的實體表格連結，我們稱此關係表格為一個多元的關係表格。如圖 2.1 中 SUPPLIER、PART 與 ORDER 此三表格的關聯性，是經由另一個三元關係表格 LINEITEM 中的共同鍵值 PARTKEY、SUPPKEY 與 ORDERKEY 來連結。在後續建立對應表格的章節中會說明 SUPPLIER 與 PART 表格會分別對應到內部節點 supplier 與 part，而 ORDER 與 LINEITEM 表格會同時對應到內部節點 order。連結限制式 ORDER.ORDERKEY = LINEITEM.ORDERKEY 中，由於 ORDERKEY 皆會對應到圖 2.2 的屬性節點 okey，因此此連結條件式不會對應到任何扁平或巢狀結構。如同先前說明，SUPPLIER 表格與 LINEITEM 表格分別對應到 supplier 與 order 元素，因此連結條件式 SUPPLIER.SUPPKEY = LINEITEM.SUPPKEY 會對應到 supplier 與 order 的巢狀結構。類似的 PART.PARTKEY = LINEITEM.PARTKEY 會對應到 part 與 order 的巢狀結構。

2.4 查詢句轉換的文法範圍與定義

先前章節已介紹過關聯式資料庫的查詢語言 SQL 基本是由 SELECT、FROM、WHERE 三子句所組成，而針對 XML 文件的查詢語言 XQuery，基礎是由 FOR、WHERE、RETURN 所組成。XQuery 由 FOR 子句以遞迴方式取得一個路徑表示法的集合結果，相當於 SQL 中的 FROM 子句；WHERE 子句則允許對變數所代表的集合內容做條件的限制，相當於 SQL 中的 WHERE 子句，根據下達的限制句內容，過濾取出所需要的資料；RETURN 子句將建構好的資料回傳給使用者，相當於 SQL 中的 SELECT 子句。本論文可轉換的 SQL 與 XQuery 之精簡 Backus-Naur Form (BNF) 文法範圍如圖 2.4 與圖 2.5 所示。進行等價的雙向查詢句轉換定義，

如定義 2.5 所示，我們以此三個定義來規劃設計轉換系統，並驗證其是否能轉換出等價的查詢句。

```
SQL ::= SelectClause + FromClause + WhereClause?
SelectClause ::= SELECT {Ei}
FromClause ::= FROM {Ri}
WhereClause ::= WHERE  $\varphi(Wi)$ 
Ri ::= Relation
Ei ::= Relation.Attribute
Wi ::= Ei CompOP Value
      | Ei CompOp Ei
CompOp ::= ">" | "=" | "<"
LogicalOp ::= "and"
```

圖 2.4：SQL 文法範圍

```
XQuery ::= ForClause + WhereClause? + ReturnClause
ForClause ::= FOR {$Var IN Pi}
WhereClause ::= WHERE  $\varphi(Wi)$ 
ReturnClause ::= Return {Ei}
Pi ::= Expr
Expr ::= PathExpr
      | Var "/" PathExpr
Var ::= $fvn
Ei ::= PathExpr
PathExpr ::= RegularExpr | "/" RegularExpr
RegularExpr ::= Step | RegularExpr "/" Step
Step ::= NameTest | "@" NameTest
NameTest ::= element | attribute
Wi ::= Ei CompOP Value
      | Ei CompOp Ei
CompOp ::= ">" | "=" | "<"
LogicalOp ::= "and"
```

圖 2.5：XQuery 文法範圍

【定義 2.5】：查詢句等價的定義

【定義一】 若轉換前與轉換後的查詢句，其欄位或有值的葉節點 (Value)、集合 (Collection) 與結構 (Structure) 的表示式個數相同，且各別有正確的對應關係，則為一個等價的查詢句轉換。

【定義二】 若轉換後的查詢句，其 Collection 個數與轉換前的 Collection 個數不同，但轉換後的 Collection 之間彼此有適當的結構關係，且輸出的屬性個數相同且等價，則亦為一個等價的查詢句轉換。

【定義三】 若轉換前與轉換後的查詢句，至任一組具有相同對應資料的資料庫中，查詢所得到的結果，經由 Distinct 指令去除重覆的資訊後，輸出的結果相同，則代表兩查詢句等價。

第三章 系統架構與對應表格

在本章中將概括介紹我們雙向轉換系統的架構，並說明對應表格建立的目的與設計方式。我們利用數個對應表格來記錄於 2.3 節中所描述的差異問題，進而利用其所提供的資訊，進行查詢句轉換。

3.1 轉換系統架構

SQL 與 XQuery 雙向查詢句轉換系統的架構如圖 3.1 所示，其中對應表格 (Mapping Tables) 包含了數個對應表格，記錄了關聯式資料與 XML 資料間彼此的資料與結構對應關係。雙向查詢句的轉換模組 (Transformation Module) 的轉換法則，便是利用對應表格所提供的資訊，做適當的查詢句轉換。詳細的流程與演算法將於第 4 章中敘述。

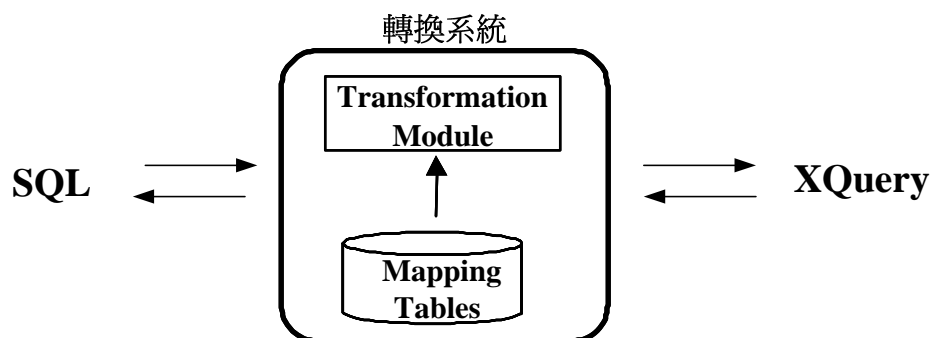


圖 3.1：雙向查詢句轉換系統架構圖

3.2 對應表格之內容

為了提供雙向轉換模組所必需的資訊，我們設計了幾個對應表格來記錄關聯式綱要 (Relational Schema) 與 XML 綱要 (XML Schema) 的對應資訊。我們將 2.3.1 節的內容表示法差異與 2.3.2 節的集合表示方式差異歸類稱為基本差異 (Basic Conflict)，與 2.3.3 節的結構差異 (Structural Conflict) 區別探討。對應表格即是用來記錄上述的差異對應。

3.2.1 解決基本差異的對應表格

為了記錄資料在Relational Schema與XML Schema資料表示法的對應，我們在本小節中定義了三個表格，來記錄每個關聯式資料表格、欄位與XML端的元素、路徑對應關係。首先，關聯式資料庫中，是以表格的型式來儲存一組相關的資料；但XML文件中，相關的一組資料集合，是被建立表示於某個內部節點之下。我們會先以定義3.1的Collection Mapping Table記錄2.3.2節中，關聯式資料庫表格與XML端內部節點的路徑對應關係。

【定義 3.1】：Collection Mapping Table （簡稱:CMT）

$CMT = \{(Rname, XPath, Type)\}$ 由三個欄位組成，其中：

- ◆ Rname: 關聯式資料庫表格的名稱。
- ◆ XPath: 此表格對應於XML端的內部節點路徑。
- ◆ Type: 此內部節點的類型名稱：
 - Repeatable：代表此內部節點為一個可重覆元素。
 - Dummy：代表此內部節點為一個空元素。
 - Nested：關係表格所關聯的實體表格之間，對應到巢狀結構。

建立CMT的方式，是以PreOrder順序，參考DTD中內部節點與表格的集合相關性，將含意相近的集合對應建立，若為內部節點與表格一對多的情形，建立對應至CMT的時候，實體表格優先於關係表格。參考第2章中，圖2.1關聯式表格與圖2.2的DTD範例，可重覆元素supplier為其葉節點nkey、skey、name與add的集合，因此會與SUPPLIER表格的相同，所以可重覆元素supplier會對應到SUPPLIER表格。另外，若相同內部節點其下的內容節點是對應到不同表格中的欄位，則我們將此相同內部節點與不同表格的對應亦一同記錄至CMT中，此為記錄表格與內部節點多對一的情況，反之一對多情況亦然；如可重覆元素customer下的葉節點name與tel，由於其會對應到CUSTOMER表格的NAME欄位與CUSTEL表格的

TELEPHONE欄位，因此在本論文建立集合對應時，/order-ship/customer會對應到CUSTOMER與CUSTEL表格。CMT中的Type欄位是用來區分此內部節點為可重覆元素 (Repeatable Node) 或是空元素 (Dummy Node)，其中空元素所對應的關聯式表格名稱，會與其最接近的子孫層元素為可重覆元素的對應表格名稱相同。如圖2.2中suppliers為一個空元素，其子元素supplier為一個可重覆元素，所以此時suppliers對應的表格名稱，會與子元素supplier對應的表格相同。

在此須特別注意到關係表格為建立實體表格之間的關聯，若實體表格之間為巢狀關係，此時，我們會指定關係表格對應的可重覆元素，會與其相關的實體表格對應是最深層的可重覆元素相同，這是因為關係表格的集合相關性，會與實體表格所對應最深層的的可重覆元素最接近。如關係表格LINEITEM記錄了SUPPLIER、PART與ORDER表格間的關聯，因此LINEITEM表格對應最深層的可重覆元素，會與ORDER表格對應的路徑/order-ship/suppliers/supplier/part/order相同。此外，為了使轉換後的結果符合XML的階層結構關係，我們會在關係表格對應為巢狀結構的情況，於其CMT中的Type欄位，增加註明Nested的特性，此目的將於演算法中再做說明。由圖2.1關聯式表格與圖2.2範例所建立的CMT內容如表格3.1所示：

Rname	XPath	Type
SUPPLIER	/order-ship/suppliers	Dummy
SUPPLIER	/order-ship/suppliers/supplier	Repeatable
PART	/order-ship/suppliers/supplier/part	Repeatable
PARTSUPP	/order-ship/suppliers/supplier/part	Repeatable & Nested
ORDER	/order-ship/suppliers/supplier/part/order	Repeatable
LINEITEM	/order-ship/suppliers/supplier/part/order	Repeatable & Nested
CUSTOMER	/order-ship/customer	Repeatable
CUSTEL	/order-ship/customer	Repeatable

表格 3.1：Collection Mapping Table

而2.3.2節A中在表格與內部節點多對一的狀況 (排除Dummy Node)，也就是由CMT中可看出位於相同路徑的可重覆元素，對應到不同表格名稱的情形，若所選擇輸出此內部節點下的數個葉節點，其對應的欄位是分別來自於不同表格的話，則這些表格間的關聯資訊亦需建立，我們用定義3.2中的IJT表格來記錄此內部連結的關係。如圖2.2的DTD中，若選擇的葉節點為內部節點customer下的name與tel，資料的來源皆取自於路徑表示法/order-ship/customer所代表集合，但其所對應的關聯式表格與欄位，由稍後介紹的VMT中可獲知為CUSTOMER表格中的NAME欄位與CUSTEL表格中的TELEPHONE欄位，此時於對應的關聯式綱要，需產生CUSTOMER表格與CUSTEL表格的連結關係，建立資料的關聯性。由第2章中圖2.1所介紹的關聯式表格與圖2.2 的DTD，所建立的IJT如表格3.2所示。

【定義 3.2】：Internal Join Table (簡稱:IJT)

$IJT = \{(Direction, Collection1, Collection2, InternalJoinCondition)\}$ 由四個欄位組成：

- ◆ Direction: 適用此條件的轉換類型對應。
 - S2X: 適用於SQL轉換為XQuery。
 - X2S: 適用於XQuery轉換為SQL。
- ◆ Collection1 和 Collection2: 資料集合的來源，可同為DTD中的可重覆元素或是同為關聯式表格名稱。
- ◆ InternalJoinCondition: 此兩資料集合間的連結條件式。

Direction	Collection1	Collection2	InternalJoinCondition
X2S	PART	PARTSUPP	PART.PARTKEY = PARTSUPP.PARTKEY
X2S	ORDER	LINEITEM	ORDER.ODERKEY = LINEITEM.ORDERKEY
X2S	CUSTOMER	CUSTEL	CUSTOMER.CUSTKEY = CUSTEL.CUSTKEY

表格 3.2：Internal Join Table

2.3.2節B中所述表格與內部節點一對多的狀況 (排除Dummy Node)，也是同樣需要建立連結資料。我們觀察到圖2.1關聯式表格CUSTOMER的欄位分別分布在圖2.3中，內部節點customer與vip_cust之下的葉節點，因此需利用相同意義的屬性值ckey來做為連結，以表示所取得的資料為關聯式表格中的同一筆資料列。我們將此兩個內部節點所對應的路徑及其內部連結限制式/cust_info/customer@ckey=/cust_info/vip/vip_cust@ckey記錄於IJT中。由第2章中圖2.1所介紹的關聯式綱要與圖2.3的DTD，所建立的部份IJT表格內容如表格3.3中所示。由以上說明，可解決2.3.2節中的集合表示法差異問題。

Direction	Collection1	Collection2	Internal Join Condition
S2X	/cust_info/customer	/cust_info/vip_cust	/cust_info/customer@ckey= /cust_info/vip_cust@ckey

表格 3.3：附錄 B 中的部份 Internal Join Table 內容

針對內容表示法差異，我們以定義3.4的Value Mapping Table表格記錄關聯式資料庫欄位與XML端的葉節點路徑的對應關係，若欄位無對應的葉節點，則不記錄至VMT中。建立VMT的方式是依序由CMT中的表格，考量表格中的每個欄位與葉節點的關係，將符合集合與屬性特性的對應關係建立。而記錄XPath (定義3.3) 的目的，是為了取得其對應之集合。由圖2.1關聯式表格與圖2.2的DTD範例，所建立的部份VMT內容如表3.4所示，詳細請參考附錄A。其中SUPPLIER表格中的SUPPKEY欄位，會對應到DTD端的屬性節點skey，因此XPath是以/order-ship/suppliers/supplier@skey表示，而NAME欄位對應到的是一個內容節點，所以對應路徑XPath是以/order-ship/suppliers/supplier/name表示。我們由DTD中不難發現到此兩元素祖先層節點最接近的可重覆元素皆為supplier，因此XPath皆為/order-ship/suppliers/supplier。以上說明可解決2.3.1節的A點意義相同但表示的名稱不同與B點資料選擇性的差異。

【定義 3.3】：可重覆路徑 (Repeatable XPath)

DTD Graph 自根元素開始，走訪至內部節點的某個可重覆元素，所經過元素組成的路徑，稱之為可重覆路徑(Repeatable XPath)，簡稱 RXPath。

【定義 3.4】：Value Mapping Table (簡稱:VMT)

$VMT = \{(Rname, Aname, XPath, RXPath)\}$ 由四個欄位組成：

- ◆ Rname: 關聯式資料庫表格的名稱。
- ◆ Aname: 此表格內的欄位名稱。
- ◆ XPath: 此欄位對應於DTD中的葉節點所代表的路徑。
- ◆ RXPath: 與此葉節點最接近的祖先層之可重覆路徑。

Rname	Aname	XPath	RXPath
SUPPLIER	SUPPKEY	/order-ship/suppliers/supplier@skey	/order-ship/suppliers/supplier
SUPPLIER	NATIONKEY	/order-ship/suppliers/supplier@nkey	/order-ship/suppliers/supplier
SUPPLIER	NAME	/order-ship/suppliers/supplier/name	/order-ship/suppliers/supplier
SUPPLIER	ADDRESS	/order-ship/suppliers/supplier/add	/order-ship/suppliers/supplier
PART	PARTKEY	/order-ship/suppliers/supplier/part@pkey	/order-ship/suppliers/supplier/part
PART	NAME	/order-ship/suppliers/supplier/part/name	/order-ship/suppliers/supplier/part
PART	TYPE	/order-ship/suppliers/supplier/part/type	/order-ship/suppliers/supplier/part
CUSTOMER	CUSTKEY	/order-ship/customer@ckey	/order-ship/customer
CUSTEL	CUSTKEY	/order-ship/customer@ckey	/order-ship/customer

表格 3.4：部分 Value Mapping Table 內容

此外，針對2.3.1節D中所述欄位與葉節點多對一的情形，我們將其多個對應關係亦建立至VMT中，如CUSTOMER表格的CUSTKEY欄位與CUSTEL表格的CUSTKEY欄位，皆是對應到路徑表示法為/order-ship/customer@ckey的屬性節點。而針對2.3.1節C中所述欄位與葉節點一對多的情形，也是會以同樣的方式建立進VMT中。

3.2.2 解決結構差異的對應表格

針對 2.3.3 節中所述結構上的差異，我們在此設計了三個表格，Join Mapping Table 記錄關聯式表格間具有結構意義的連結關係，Path Mapping Table 記錄於 DTD 中，內部節點彼此間具有結構意義的關聯，而 Structure Mapping Table 則是記錄上述兩表格中的結構對應關係。以下為相關定義：

【定義3.5】：Join Mapping Table (簡稱:JMT)

$JMT = \{(RID, Condition1, Condition2)\}$ 由三個欄位組成，解釋如下：

- ◆ RID: 此連結限制式的編號，不同的編號代表不同的意義。
 - R_i : 代表表格間的連結。
 - RR_i : 代表關係表格與關係表格的連結
- ◆ Condition1 和 Condition2: 兩個表格間，用來連結的鍵值。

【定義3.6】：Path Mapping Table (簡稱:PMT)

$PMT = \{(XID, XPath1, XPath2)\}$ 由三個欄位組成：

- ◆ XID: DTD端此結構關係的編號，不同的編號代表不同的意義。
 - X_{Fi} : 代表於DTD中是屬於扁平 (Flat) 結構。
 - X_{Ni} : 代表於DTD中是屬建立於兩個可重覆元素的巢狀 (Nested) 結構。
 - X_{NDi} : 代表於DTD中是屬於巢狀 (Nested) 結構，且其中一個為空元素。
- ◆ XPath1 和 XPath2:
 - XID為 X_{Fi} 時，XPath1與XPath2分別代表兩個葉節點之路徑。
 - XID為 X_{Ni} 時，XPath1與XPath2分別代表兩個可重覆元素之路徑。
 - XID為 X_{NDi} 時，XPath1與XPath2分別代表空元素與可重覆元素之路徑。

【定義3.7】：Structure Mapping Table (簡稱:SMT)

$SMT = \{(RID, XID)\}$ 由兩個欄位組成：

- ◆ RID：JMT中的連結限制式的編號。
- ◆ XID：PMT中的結構關係編號。

(RID, XID) ：為一組連結限制式與 XML 結構關係的對應

關聯式資料庫中，表格與表格間的關聯是以相同的鍵值來做連結，我們以 JMT 來記錄關聯式表格間的連結關係。DTD 中內部節點具有結構意義的關聯記錄於 PMT 中，JMT 與 PMT 兩者的關聯性，則是建立在 SMT 之中。例如由圖 2.1 關聯式表格中可得知，SUPPLIER 與 PART 表格的關聯，可以經由表 3.5 JMT 中編號為 R1 的關係式所構成；由 CMT 中我們知道 SUPPLIER 與 PART 表格分別對應到可重覆元素 supplier 與 part，從圖 2.2 中我們可看出可重覆元素 part 是 supplier 所對應路徑/order-ship/suppliers/supplier 之子元素，此父子關係為一巢狀結構，此結構於表 3.6PMT 中編號為 X_{N1} ，因此我們會將此組對應記錄於表 3.7 的 SMT 表格中。為了便利後續的處理，我們預先規範 PMT 中的巢狀結構中祖先元素表示於 XPath1，而子孫元素表示於 XPath2。

另外，ORDER 與 CUSTOMER 表格的關聯，可以經由表 3.5 JMT 中編號為 R4 的關係式所構成；由 CMT 中我們發現 ORDER 與 CUSTOMER 表格分別對應到可重覆元素 order 與 customer，從圖 2.2 中我們可看出此兩可重覆元素是利用相同屬性 ckey 來建立連結，為一扁平狀結構，此結構於表 3.6PMT 中編號為 X_{F1} ，此組對應關係如表 3.7 的 SMT 中所示。我們以上述說明來記錄 2.3.3 節中的 A、B 點關聯式連結與巢狀或扁平結構的對應問題。

RID	Condition1	Condition2
R1	SUPPLIER.SUPPKEY	PARTSUPP.SUPPKEY
R2	SUPPLIER.SUPPKEY	LINEITEM.SUPPKEY
R3	PART.PARTKEY	LINEITEM.PARTKEY
R4	ORDER.CUSTKEY	CUSTOMER.CUSTKEY
R5	ORDER.CUSTKEY	CUSTEL.CUSTKEY
R6	SUPPLIER.NATIONKEY	CUSTOMER.NATIONKEY
RR1	PARTSUPP.SUPPKEY	LINEITEM.SUPPKEY
RR2	PARTSUPP.PARTKEY	LINEITEM.PARTKEY

表格 3.5 : Join Mapping Table

XID	XPath1	XPath2
X _{ND1}	/order-ship/suppliers	/order-ship/suppliers/supplier/part
X _{N1}	/order-ship/suppliers/supplier	/order-ship/suppliers/supplier/part
X _{ND2}	/order-ship/suppliers	/order-ship/suppliers/supplier/part/order
X _{N2}	/order-ship/suppliers/supplier	/order-ship/suppliers/supplier/part/order
X _{N3}	/order-ship/suppliers/supplier/part	/order-ship/suppliers/supplier/part/order
X _{F1}	/order-ship/customers/customer@ckey	/order-ship/suppliers/supplier/part/order@ckey
X _{F2}	/order-ship/customer@nkey	/order-ship/suppliers/supplier@nkey

表格 3.6 : Path Mapping Table

RID	XID
R1	X _{ND1}
R1	X _{N1}
R2	X _{ND2}
R2	X _{N2}
R3	X _{N3}
R4	X _{F1}
R5	X _{F1}
R6	X _{F2}
RR1	X _{N1} & X _{N2}
RR2	X _{N3}

表格 3.7 : Structure Mapping Table

針對 2.3.3 節C中所述關聯式連結與扁平結構一對多的關係，同一個關聯式連結，會對應到不同的扁平結構，也就是於JMT表格中，同一個RID會對應到不同的XID。由圖 2.1 關聯式資料範例與圖 2.3 的DTD結構對應，考慮CUSTOMER與NATION表格的關聯(CUSTOMER.NATIONKEY = NATION.NATIONKEY)，在表格 3.9 的JMT中的編號為R1。我們由表格 3.8 的VMT中可發現CUSTOMER表格的 NATIONKEY 欄位，會對應到 /cust-info/customer@nkey 與 /cust-info/vip/vip_cust@nkey兩個屬性節點；而NATION表格之NATIONKEY欄位所對應的路徑為 /cust-info/nation@nkey，因此會產生兩組扁平結構的連結關係，我們分別給予編號X_{F1}與X_{F2} (如表格 3.10 所示)。因此於表格 3.11 中，R1 會對應到X_{F1}與X_{F2}，為關聯式連結與扁平結構一對多的關係。

Rname	Aname	XPath	RXPath
CUSTOMER	CUSTKEY	/cust_info/customer@ckey	/cust_info/customer
CUSTOMER	NATIONKEY	/cust_info/customer@nkey	/cust_info/customer
CUSTOMER	NATIONKEY	/cust_info/vip_cust@nkey	/cust_info/vip_cust
CUSTOMER	COMMENT	/cust_info/vip_cust/comment	/cust_info/vip_cust
NATION	NATIONKEY	/cust_info/nation@nkey	/cust_info/nation

表格 3.8：部分 Value Mapping Table 內容

RID	Condition1	Condition2
R1	CUSTOMER.NATIONKEY	NATION.NATIONKEY
R2	NATION.REGIONKEY	REGION.REGIONKEY

表格 3.9：Join Mapping Table

XID	XPath1	XPath2
X _{F1}	/cust_info/customer@nkey	/cust_info/nation@nkey
X _{F2}	/cust_info/vip_cust@nkey	/cust_info/nation@nkey
X _{N1}	/cust_info/nation	/cust_info/nation/region
X _{ND1}	/cust_info/nation	/cust_info/nation/population
X _{N2}	/cust_info/nation	/cust_info/nation/population/region_popu

表格 3.10：Path Mapping Table

RID	XID
R1	X _{F1}
R1	X _{F2}
R2	X _{N1}
R2	X _{ND1}
R2	X _{N2}

表格 3.11：Structure Mapping Table

而 2.3.3 節 D 中所述關聯式連結與巢狀結構一對多的關係，此時，同一個關聯式連結，會對應到不同層次的巢狀結構，也就是說，一個 RID 會對應到不同的 XID。為了區分此情形，我們將 XID 的內容區分為 X_{Ni} 與 X_{NDi}。其中 X_{Ni} 指定給於 PMT 中 XPath1 與 XPath2 分別代表兩個可重覆元素的路徑的 XID 編號。而 X_{NDi} 則指定給對應於 PMT 中 XPath1 與 XPath2 其中一個為空元素的 XID 編號。如圖 2.1 關聯式資料表格 NATION 會對應於圖 2.3 的內部節點 nation，而 REGION 表格由於表格與內部節點一對多的關係，實際會對應到內部節點 region 與 region_popu，另外由於 population 為一個空元素，其對應的表格名稱會與 region_popu 相同，因此如表格 3.11 中之 SMT 可發現，關聯式表格 NATION 與 REGION 的關係 R2，會產生對應於 PMT 編號 X_{N1}、X_{ND1} 與 X_{N2} 三筆資料列（參見表格 3.10）。

至於 2.3.3 節 E 所述關聯式連結與扁平結構多對一的關係，此種情形會發生在其中一個可重覆元素，對應到兩個以上的關聯式表格時發生，同一個扁平結構可經由不同的連結限制式來達成。如圖 2.2 中，內部節點 order 與 customer 之間的扁平關係，若選擇的葉節點為內部節點 customer 下的 name 與 tel，資料的來源皆取自於路徑表示法/order-ship/coustomer 所代表集合，但 name 與 tel 分別對應於圖 2.1 關聯式表格 CUSTOMER 的 NAME 欄位與 CUSTEL 表格的 TELEPHONE 欄位，所以內部節點 customer 實際上會對應到 CUSTOMER 與 CUSTEL 兩個表格，假設目前內部節點 order 是對應到 ORDER 表格，因此 ORDER 表格與 CUSTOMER 及 CUSTEL 之間的關聯，由表 3.7 SMT 中可發現，連結限

制式 R4、R5 都會對應編號為 X_{F1} 的扁平結構。至於 2.3.3 節 F 所述，連結限制式與巢狀結構多對一的關係亦是類似如此。

2.3.3 節 G 點巢狀結構與關係表格的關係，若巢狀結構為一個多元的關係所構成，關係表格中記錄著兩個以上實體表格的鍵值來建立表格間的關聯，我們稱此關係表格為一個多元的關係表格。若這些表格於 DTD 中所對應的可重覆元素形成一組巢狀結構，其中若有實體表格與關係表格對應到同一個內部節點，則其連結限制式可能不會產生任何扁平或巢狀結構對應的轉換，如 ORDER 與 LINEITEM 的連結限制式 $ORDER.ORDERKEY = LINEITEM.ORDERKEY$ ，在 CMT 中我們發現 ORDER 與 LINEITEM 同時皆對應到內部節點 order，因此此連結限制式，沒有記錄結構的對應關係。再觀察 PART 表格與 LINEITEM 表格，分別對應到內部節點 part 與 order，為一個巢狀結構，因此表格 3.5 JMT 中 $PART.PARTKEY = LINEITEM.PARTKEY$ (R3) 會對應到表格 3.6 PMT 的 X_{N3} 這組結構。

注意到在 JMT 的建立中，若連結限制式為關係表格與關係表格之間的連結，則此 RID 編號我們是以 RR_i 來表示，此目的是為了在兩表格對應於 DTD 中為一個巢狀結構時，若同一個 XID 對應到的 RID 編號型式為 R_i 與 RR_i ，進行 XQuery 至 SQL 的結構轉換時，我們將選擇實體表格與實體表格或是實體表格與關係表格間的結構關聯編號 R_i 輸出 (將於 4.6.3 節中說明)。如表格 3.7 中的 X_{N3} 會對應到關聯式連結編號 R3 與 RR_2 ，將會選擇 R3 的關聯限制式做轉換。

此外在表格 3.7 SMT 表格的對應中， RR_1 對應的結構為 $X_{N1} \& X_{N2}$ ，代表此兩結構都會一起轉換出，這是因為 RR_1 所代表的連結限制式 $PARTSUPP.SUPPKEY = LINEITEM.SUPPKEY$ 其中 SUPPKEY 對應的葉節點皆是可重覆元素 supplier 下的屬性節點 skey，而在 CMT 中，PARTSUPP 與 LINEITEM 分別對應到可重覆元素 part 與 order，因此為了符合結構的限制，此連結限制式會轉換出 supplier、part、order 的巢狀結構關係，此範例將於 5.1 節的 C 點中說明。

3.3 對應表格之建立

要建立一份正確、完整的對應表格，使用者必須充分瞭解關聯式綱要與 XML 綱要的對應關係，並進行對應關係的分析才能予以建立。建立的順序依序為集合、欄位（或屬性）還有結構的對應關係。以下針對每個對應表格的建立步驟，與需要注意的事項，再做個統整說明。

● CMT 的建立

1. 以PreOrder順序走訪DTD中的內部節點，考量雙向集合的對應關係，將內部節點的路徑、型態與對應的表格名稱記錄。
2. 內部節點若為Dummy Node，會與其最接近的子孫層元素為可重覆元素的對應表格名稱相同。
3. 內部節點與表格一對多的情形，建立至CMT的時候，實體表格優先於關係表格。
4. 關係表格為建立實體表格之間的關聯，若實體表格之間對應到DTD為巢狀關係，會指定關係表格對應的可重覆元素，與其相關的實體表格對應是最深層的可重覆元素相同，且型態欄位須註明Nested的特性。

● IJT 的建立

1. 由建立好的CMT，挑選出TYPE屬性為Repeatable之對應記錄，然後做以下2、3點之判斷。
2. 若有相同表格名稱，對應到不同可重覆元素的情形，此時IJT表格的Direction會為S2X，代表SQL至XQuery轉換時需要檢查；將可重覆元素之間利用相同屬性建立的內部連結資訊記錄。
3. 若有同一可重覆元素路徑，對應到不同表格名稱的情形，此時IJT表格的Direction會為X2S，代表XQuery至SQL轉換時需要檢查；將表格之間利用相同鍵值建立的內部連結資訊記錄。

- VMT 的建立

1. 依序由 CMT 中的不同表格，由關聯式表格至 DTD 對應的方向，考量表格中的每個欄位與葉節點的關係，將符合的對應關係建立。
2. 其中 XPath 記錄與葉節點最接近的可重覆元素。

- JMT、PMT、SMT 的建立

考量表格與表格之間利用相同鍵值的連結與 DTD 中的對應關係：

1. 若此兩表格對應到相同的可重覆元素，且連結限制式中的兩個鍵值，對應到此可重覆元素之下相同的屬性節點，則此連結限制式不會記錄至 JMT 中。
 - 2.1 若此兩表格對應到不同的可重覆元素，且可重覆元素之間為扁平結構，連結限制式中的兩個鍵值，分別對應到此兩可重覆元素之下的屬性節點，則會給予此連結限制式一個 R_i 編號，記錄於 JMT 中。對應的扁平結構，則分配給與一個 X_{Fi} 編號，記錄於 PMT 中。最後將此組對應關係，記錄於 SMT 中。
 - 2.2 若此兩表格皆為關係表格，對應到不同的可重覆元素，且可重覆元素之間為扁平結構，則會給予此連結限制式的編號為 RR_i ，記錄於 JMT 中。對應的扁平狀結構，則分配給與一個 X_{Fi} 編號，記錄於 PMT 中。並將此組對應關係，記錄於 SMT 中。
- 3.1 若此兩表格（排除關係與關係表格）對應到不同的可重覆元素，且可重覆元素之間為巢狀結構，連結限制式中的兩個鍵值，同時對應到此兩可重覆元素中，屬於在上層可重覆元素之下的屬性節點，則會給予此連結限制是一個 R_i 編號，記錄於 JMT 中。對應的巢狀結構，則分配給與一個 X_{Ni} 編號，記錄於 PMT 中。並將此組對應關係，記錄於 SMT 中。
- 3.2 若此兩表格（排除關係與關係表格）對應到不同的內部節點，且之間為巢狀結構，若其中有一個內部節點為空元素，會給予此連結限制式一個 R_i 編號，記錄於 JMT 中。對應空元素的巢狀結構，則會分配給與一個 X_{NDi} 的編號，記錄於 PMT 中。並將此組對應關係，記錄於 SMT 中。

- 4.1 若此兩表格皆為關係表格，對應到不同的可重覆元素，且可重覆元素之間為巢狀結構，連結限制式中的兩個鍵值，同時對應到此兩可重覆元素中，屬於在上層可重覆元素之下的屬性節點，則會給予此連結限制式的編號為 RR_i ，記錄於 JMT 中。對應的巢狀結構，則分配給與一個 X_{Ni} 編號，記錄於 PMT 中。並將此組對應關係，記錄於 SMT 中。
- 4.2 若此兩表格皆為關係表格，對應到不同的可重覆元素，且可重覆元素之間為巢狀結構，但是連結限制式中的兩個鍵值，同時對應到更上層可重覆元素之下的屬性節點（非此兩表格對應的可重覆元素之屬性節點），此時，需要建立這三個可重覆元素之間的巢狀關聯，一樣會給予此連結限制式的編號為 RR_i ，記錄於 JMT 中。對應的巢狀結構，則是考量關係表格與對應的可重覆元素之間的關聯，分別將三個可重覆元素之間的巢狀關係建立，記錄於 SMT 中。

在建立 JMT 與 PMT 對應時尚需特別注意到，我們不將無結構對應的情況建立至 JMT、PMT 及 SMT 中。接下來，我們將在下一章節，利用所設計的對應表格，詳細的介紹我們查詢句轉換的模組架構與演算法。

第四章 轉換模組與演算法

在本章中，將介紹相關的轉換演算法。在轉換過程中，將由第三章所設計的對應表格，取得轉換時所需要的對應資訊，進一步進行查詢句轉換的處理。配合 2、3 章的介紹，我們將演算法亦區分為解決基本差異與結構差異的演算法來說明，基本差異相關演算法在處理表格與欄位在 DTD 中的元素對應轉換，而結構差異相關的演算法，則是在處理關聯式表格間的連結關係與 DTD 中的結構對應轉換。

4.1 轉換模組架構

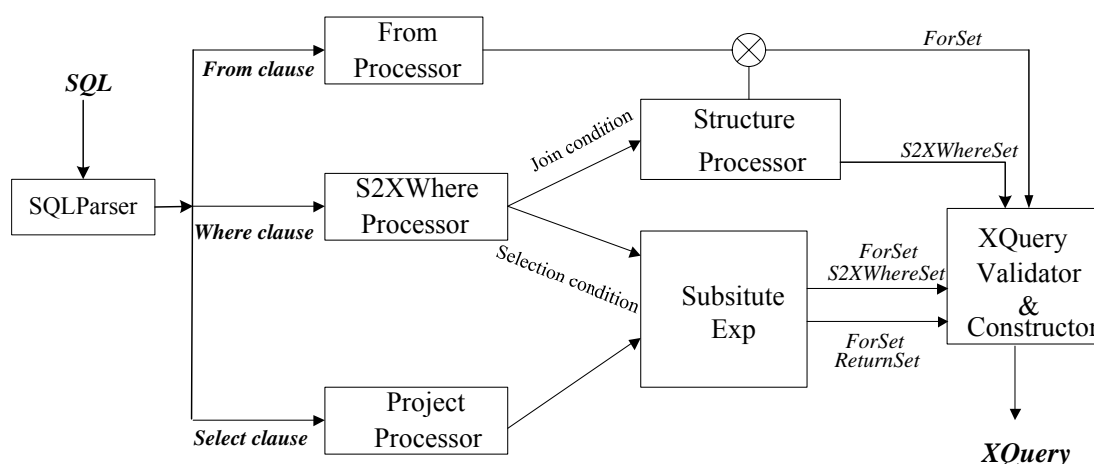


圖 4.1-1 SQL 至 XQuery 轉換模組架構

SQL至XQuery轉換架構以主要的流程來區分如圖4.1-1所示，大致將轉換模組區分為<1>SQL 解析模組 (SQLParser)、<2> 片段查詢句轉換模組 (FromProcessor、S2XWhereProcessor、ProjectProcessor)、<3>結構對應與表示法代換的處理 (StructureProcessor、SubstuteExp)、<4> 結合與驗證模組 (XQueryValidator&Constructor) 四大部分。首先由SQL解析模組將使用者所輸入的SQL查詢句，分別針對SELECT、FROM、WHERE的子句內容，轉換成內部表示法分別傳給片段查詢句轉換模組 ProjectProcessor、FromProcessor、

S2XWhereProcessor等做處理，過程中所需的對應資料皆由對應表格中取得，經由結構對應與表示法的處理模組後，產生的片段結果會記錄於定義4.1的ForSet、S2XWhereSet與ReturnSet之中。我們將此三個集合經過XQueryValidator & Constructor濾除多餘的資料及做合理化的動作後，產生得到最終轉換後的XQuery。

轉換過程中，針對查詢句的Where子句同第二章的說明，我們將之區分為條件限制式 (Selection Condition) 與連結限制式 (Join Condition)。連結限制式會產生如2.3.3節中所述的結構對應差異問題，會由結構轉換模組進行相關的對應處理。SQL的條件限制式與Select 子句內容，所選擇的欄位對應到的葉節點路徑，會經由SubstituteExp模組先與其最接近的可重覆元素路徑所代表的變數結合，轉換為XQuery的表示法。

【定義4.1】 ForSet、S2XWhereSet與ReturnSet

◆ ForSet：記錄SQL的From子句，經過FromProcessor處理後的內容。

ForSet = {(Rname, Var, RXPath, For_Clause, Type, UsedFlag)} 由六個欄位組成：

- Rname：From子句中的表格名稱。
- Var：指定給此表格對應的可重覆元素之變數名稱。
- RXPath：表格對應之可重覆元素的路徑表示法。
- For_Clause：記錄轉換後，XQuery的集合表示法。
- Type：此內部節點的類型。
- UsedFlag：TRUE 或 FALSE，用來判斷多元對應時是否會輸出此對應。

例：(CUSTOMER, \$t0, /cust_info/customer, /cust_info/customer, Repeatable, FALSE)

◆ S2XWhereSet：記錄SQL中的Where子句處理後的結果。

S2XWhereSet = {(S2XWhere_clause)}

- S2XWhere_clause：記錄轉換後 XQuery 的 Where 子句內容。

例：(\$t0/name = '台灣')

◆ ReturnSet：記錄SQL中的Select子句經過ProjectProcessor處理後的結果。

ReturnSet = {(Return_clause)}

■ Return_clause：記錄轉換後所產生的XQuery中Return子句內容。

例：(\$t0/tel)

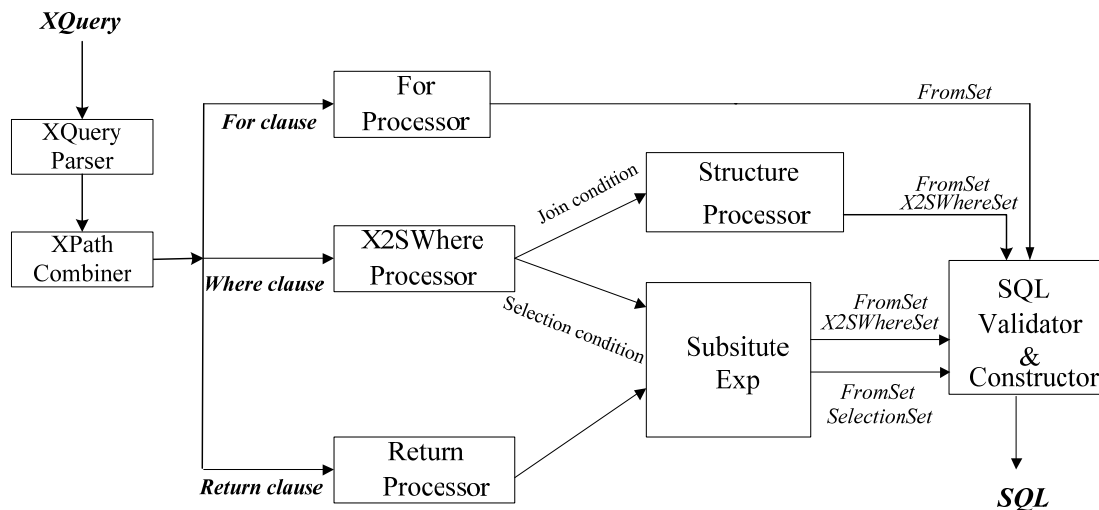


圖 4.1-2 XQuery 至 SQL 轉換模組架構

另外，XQuery至SQL轉換架構以主要的流程來區分如圖4.1-2所示，大致亦將轉換模組區分為<1>XQuery解析與路徑組合模組(XQuery Parser、XPathCombiner)、<2>路徑對應等價SQL轉換模組(ForProcessor、X2SWhereProcessor、ReturnProcessor)、<3>結構對應與表示法代換的處理(StructureProcessor、SubstituteExp)、<4>結合與驗證模組(SQLValidator & Constructor)四大部分。首先由XQuery解析模組將使用者所輸入的XQuery查詢句，分別針對For、Where、Return的子句內容解析，由XPathCombiner分別將變數所對應的路徑結合，傳給查詢句轉換模組ForProcessor、X2SWhereProcessor、ReturnProcessor等做處理，過程中所需的對應資料皆由對應表格中取得，經由結構對應與表示法的處理模組後，產生的片段結果會記錄於定義4.2中的FromSet、X2SWhereSet與SelectionSet之中。我們將此三個集合經過SQLValidator &

Constructor濾除多餘的資料及做合理化的動作後，得到最終轉換後的SQL。

【定義4.2】 FromSet、X2SWhereSet與SelectionSet

- ◆ FromSet：記錄XQuery的For子句，經過ForProcessor處理後的內容。

FromSet = {(XPath, Rname, UsedFlag)} 由三個欄位組成：

- XPath：表示於For子句的路徑
- Rname：此路徑表示的節點所對應的表格名稱
- UsedFlag：TRUE 或 FALSE，用來判斷多元對應時是否輸出此對應

- ◆ X2SWhereSet記錄XQuery中的Where子句處理後的結果

X2SWhereSet = {(X2SWhere_clause)}

- X2SWhere_clause：記錄轉換後所產生 SQL 的 Where 子句內容

- ◆ SelectionSet記錄XQuery中的Return子句經過ReturnProcessor處理後的結果。

SelectionSet = {(Selection_clause)}

- Selection_clause：記錄轉換後所產生SQL的Select子句內容

以下我們開始介紹雙向轉換過程主要的處理<1>Collection對應的轉換、<2>Value對應的轉換、<3>結構對應的轉換、<4>QueryValidator驗證模組<5>QueryConstructor 五大部分相關的轉換演算法。

4.2 解決基本差異的演算法

4.2.1 Collection 的對應處理

為了解決查詢句資料來源指定方式的差異問題，FromProcessor 負責 SQL 的From子句，轉換成相對於XQuery之For子句的表示法，如圖 4.2 FromProcessor 演算法所示。而 XQuery 中的 For 子句內容，將以圖 4.3 ForProcessor 演算法做處理，ForProcessor 演算法會將 For 子句的內容，轉換成對應 SQL 的 From 子句內容。

Algorithm FromProcessor(RelationSet, CMT)	
Input: RelationSet = {r1,r2,...,rn} is the set of Relations in SQL From clause. CMT: Collection Mapping Table	
Output: ForSet	
L01	ForSet={ };
L02	initialize all components in the array UsedFlag[] = FALSE;
L03	for all r _i in RelationSet do
L04	CSet = GetTupleByCMT(r _i , CMT); //find the mapping path and type
L05	for all C _i in CSet do
L06	if C _i .XPath is not NULL and C _i .Type is Repeatable then
L07	For_Clause _i = C _i .XPath;
L08	Var _i = SetVariable(C _i .XPath, ForSet);
	// if different relation map to the same repeatable element, will set the same variable
L09	ForSet = ForSet \cup {(C _i .Rname, Var _i , C _i .XPath, For_Clause _i , C _i .Type, UsedFlag[Var _i])};
L10	else if C _i .Type is Dummy
L11	continue;
L12	end if
L13	end for
L14	end for
L15	return ForSet;

圖 4.2：FromProcessor 演算法

FromProcessor 演算法中，首先會將 From 子句中的表格名稱依序由 L04 比對 CMT 表格，取得表格名稱所對應的所有內部節點路徑。由於相同的表格名稱，會有 2.3.2 節 B 點所述表格與內部節點一對多的關係，對應到的內部節點可能是可重覆元素或是空元素，因為其對應的表格是相同的。因此，我們由 L06 行限定是取得該表格對應於 DTD 中的可重覆元素。L07 讓 For_Clause 的內容此時等於此可重覆路徑；而 L08 SetVariable 函式的作用在於分配給予此可重覆路徑特定的變數編號。當表格與可重覆元素為多對一的關係時，相同的可重覆元素路徑，會分配到相同的變數編號，代表不同表格對應到相同的集合，最後 L09 將此轉換結果記錄於 ForSet 中；UsedFlag 初始值為 FALSE 的目的，是因為在表格與可重覆元素為一對多的情況，最後組合成 XQuery 時，用來判斷並刪除沒有用到的可重覆元素。

Algorithm ForProcessor(CollectionSet, CMT)	
Input: CollectionSet = {c1,c2,...,cn} is the set of Collections (xpath) in XQuery For clause. CMT: Collection Mapping Table	
Output: FromSet	
L01	FromSet={ };
L02	initialize all components in the array UsedFlag[] = FALSE;
L03	for all c _i in CollectionSet do
L04	TSet = GetTupleByCMT(c _i , CMT); //find the mapping path and type
L05	for all T _i in TSet do
L06	if T _i .Rname is not NULL and T _i .Type is Repeatable then
L07	FromSet = FromSet ∪ {(T _i .XPath, T _i .Rname, UsedFlag[T _i .Rname])};
L08	else if T _i .Rname is not NULL and T _i .Type is Dummy then
L09	get dummy node's nearest descendent node which is a repeatable node's xpath (T _i .XPath _{Repeatable}) By CMT;
L10	FromSet = FromSet ∪ {(T _i .XPath _{Dummy} , T _i .Rname, UsedFlag[T _i .Rname])};
L11	FromSet = FromSet ∪ {(T _i .XPath _{Repeatable} , T _i .Rname, UsedFlag[T _i .Rname])};
L12	end if
L13	end for
L14	return FromSet;

圖 4.3：ForProcessor 演算法

而 ForProcessor 演算法會將 XQuery 的 For 子句內容，轉換成對應 SQL 的 From 子句內容，在此我們不考慮 For 子句中，利用不同的兩個變數對應到相同路徑的問題。CollectionSet 中的每個集合 c_i，皆會先經由圖 4.1-2 中的 XPathCombiner 演算法組合成完整路徑，再由 L04 會取得 For 子句中的路徑所對應的表格名稱，並記錄於 FromSet 中。由於 For 子句的路徑有可能為空元素所對應的路徑，此時演算法會將此空元素所對應的表格名稱及其最接近的子孫節點為可重覆元素的路徑，其對應到的表格名稱，皆記錄於 FromSet 中 (L18~L11)，此目的會在後續 ReturnProcess 演算法進行葉節點處理時，再做說明。針對上述的演算法，我們以範例 4.1 的 SQL 查詢句與範例 4.2 的 XQuery 查詢句來做相關的轉換說明。此兩查詢句會對應到圖 2.1 的關聯式綱要與圖 2.3 的 DTD，相關對應表格請參見附錄 B。

【範例 4.1】 重覆範例 2.2 的 SQL 範例

```
SELECT  CUSTOMER.NAME, CUSTEL.TELEPHONE,
        CUSTOMER.COMMENT
FROM    CUSTOMER , CUSTEL, NATION
WHERE   CUSTOMER.NATIONKEY = NATION.NATIONKEY
        AND
        CUSTOMER.CUSTKEY = CUSTEL.CUSTKEY
        AND
        NATION.NAME = '台灣'
```

我們根據範例的 FROM 子句中，CUSTOMER, CUSTEL, NATION 三個表格的轉換來做說明。首先，CUSTOMER 表格先經由圖 4.2 FromProcessor 演算法 L04 比對 CMT 表格，獲知對應的可重覆路徑為/cust_info/customer，L08 會分配一個變數名稱\$t0 給此路徑，此時 L07 For_clause 亦為/cust_info/customer，最後由 L09 將結果記錄至 ForSet 中。而 CUSTEL 在 CMT 中對應的可重覆路徑亦為/cust_info/customer，因此 L08 分配的變數名稱亦為\$t0。另外注意到 CUSTOMER 對應到兩個可重覆元素，所以產生兩筆資料。範例 4.1 經 FromProcessor 演算法的轉換結果表格 4.1 所示。

<pre>{ (CUSTOMER, \$t0, /cust_info/customer, /cust_info/customer, Repeatable, FALSE), (CUSTOMER, \$t1, /cust_info/vip_cust, /cust_info/vip_cust, Repeatable, FALSE), (CUSTEL, \$t0, /cust_info/customer, /cust_info/customer, Repeatable, FALSE), (NATION, \$t2, /cust_info/nation, /cust_info/nation, Repeatable, FALSE) }</pre>

表格 4.1：範例 4.1 經 FromProcessor 演算法產生之 ForSet 內容

【範例 4.2】 重覆範例 2.4 的 XQuery 範例 (對應於圖 2.3 Cust-Info DTD)

```
For $t2 in /cust_info/nation,  
    $t0 in /cust_info/customer,  
    $t1 in /cust_info/vip_cust  
Where $t2/name= “台灣”  
  
    And    $t0@nkey=$t2@nkey  
    And    $t1@nkey=$t2@nkey  
    And    $t0@ckey=$t1@ckey  
  
Return $t0/name, $t0/tel, $t1/comment
```

以範例 4.2 的 For 子句中 \$t2 in /cust_info/nation 來做轉換的說明。首先，\$t2 in /cust_info/nation 的路徑 RXPath 為 /cust_info/nation，經由 L04 至表格 CMT 中比對，取得路徑所對應的關聯式表格名稱為 NATION，由 L07 將此結果記錄於 FromSet 中。另外注意到 CMT 中，/cust_info/customer 同時對應到 CUSTOMER 與 CUSTEL 表格，所以會一併記錄至 FromSet 中，範例 4.2 For 子句的內容轉換結果會如表 4.2 所示。

<pre>{ (/cust_info/nation, NATION, FALSE), (/cust_info/customer, CUSTOMER, FALSE), (/cust_info/customer, CUSTEL, FALSE), (/cust_info/vip_cust, CUSTOMER, FALSE) }</pre>

表格 4.2：範例 4.2 經 ForProcessor 演算法產生之 FromSet 內容

4.2.2 Value 的對應處理

在本小節中，將針對投射的內容與條件限制式的轉換作說明。

A. 投射的內容轉換

SQL 中的 SELECT 子句內容，會分別傳入圖 4.4 ProjectProcessor 演算法做處理，將所選擇的欄位，轉換成相對於 XQuery 的 Return 子句。而 XQuery 的 Return 子句內容，則是分別傳入圖 4.5 ReturnProcessor 演算法做處理，將葉節點的路徑表示法，轉換成對應於 SQL 中 Select 子句的內容。針對表格欄位與葉節點為一對多或是多對一的情形，由於選擇輸出的葉節點或欄位內容相同，可任選一組輸出，在此本論文是選擇取出於 VMT 中比對到的第一組對應。以下將先針對 SQL 中的 SELECT 子句轉換成相對於 XQuery 的 Return 子句的相關演算法做說明，接著再介紹 Return 子句轉換成 SELECT 子句的相關演算法。

轉換 SQL 中的 SELECT 子句的 ProjectProcessor 演算法，所欲輸出的欄位會經由 L03 查詢 VMT 表格，找出欄位所對應的第一筆資料列，再由 L04 將其所對應的 XPath，經由圖 4.5 SubsituteExp 演算法代換為符合 XQuery 的表示法，最後 L05 將結果儲存至 ReturnSet 中，其中 Return_clause 為代換後的路徑表示法。從圖 4.5 SubsituteExp 演算法代換欄位所對應的葉節點路徑的時候，根據至 VMT 中取得的 V_i ，L01 經由圖 4.6 GetVarBy_ForSet 演算法至 ForSet 中取得對應的變數代號，由 L02 將對應的葉節點路徑進行變數代換以符合 XQuery 的表示法。

以下解釋圖 4.6 的 GetVarBy_ForSet 演算法。首先 L01 會以該欄位於 VMT 中所對應的表格名稱和 RXpath 至 ForSet 中比對，找出符合的資料列，取出其所分配的變數代號，並將此變數的 UsedFlag 改設為 TRUE，代表此可重覆元素之下有資料會被輸出 (L02~L04)，此步驟目的在供最後的驗證模組來驗證、濾除表格與可重覆元素一對多對應時，多餘的集合資訊不須輸出，我們將會在後面驗證模組中利用到此資訊。由於在 CMT 中，關係表格對應的可重覆元素，會與其相關的實體表格對應到最深層的可重覆元素相同，因此關係表格中的欄位對應的葉節點，其祖先層最接近的可重覆元素可能不在集合轉換處理後的 ForSet 中，此

時則會另外依 VMT 中取得的表格名稱與對應的 XPath，新增一個代號 $\$s_n$ 代表是在處理欄位轉換時產生的集合（與轉換集合對應所分配的代號 $\$t_n$ 區分），並記錄於 ForSet 中 (L06~L09)。

【範例 4.3】

根據範例 4.1 中 SELECT 子句的內容：

```
SELECT  CUSTOMER.NAME, CUSTEL.TELEPHONE,
        CUSTOMER.COMMENT
```

CUSTOMER 表格的 NAME 欄位經由圖 4.4 ProjectProcessor 演算法的處理，會由 L03 至 VMT 表格取出對應的資料列，而得到欄位所對應路徑 (XPath) 為 /cust_info/customer/name，而可重覆路徑 (XPath) 為 /cust_info/customer。將此可重覆路徑經由圖 4.6 GetVarBy_ForSet 演算法的處理，得到此可重覆路徑所對應的變數名稱為 $\$t0$ ，同時將 UsedFlag[$\$t0$] 設為 TRUE，接著由圖 4.5 SubsituteExp 演算法將欄位所對應路徑 (XPath) 代換為 $\$t0/name$ 。而 CUSTOMER 表格的 COMMENT 欄位，亦是類似的方式處理。整個範例經過 ProjectProcessor 演算法的轉換過程後，表格 4.2 的 ForSet 內容會更動如表格 4.3 所示，而產生的 ReturnSet 會如表格 4.4 所示。

<pre>{ (CUSTOMER, \$t0, /cust_info/customer, /cust_info/customer, Repeatable, TRUE), (CUSTOMER, \$t1, /cust_info/vip_cust, /cust_info/vip_cust, Repeatable, TRUE), (CUSTEL, \$t0, /cust_info/customer, /cust_info/customer, Repeatable, TRUE), (NATION, \$t2, /cust_info/nation, /cust_info/nation, Repeatable, FALSE) }</pre>

表格 4.3：範例 4.1 經 ProjectProcessor 演算法更新之 ForSet 內容

<pre>{ (\$t0/name), (\$t0/tel), (\$t1/comment) }</pre>
--

表格 4.4：範例 4.1 經 ProjectProcessor 演算法產生之 ReturnSet 內容

而 XQuery 的 Return 子句會經由圖 4.7 ReturnProcessor 演算法做處理，將葉節點的路徑表示法，轉換成對應於 SQL 中 Select 子句的內容。演算法首先依序將 Return 子句的內容，由 L03 至 VMT 表格中，取得對應的資料列，由於內部節點與表格一對多的關係，L04 行會由圖 4.8 usedFromCheck 演算法將有被參考到的表格其 UsedFlag 設為 TRUE，供最後的驗證模組來驗證、濾除多餘的表格之用。最後由 L05、L06 將葉節點的路徑表示法轉換成對應的關聯式表格欄位，並加入至 SelectionSet 中。

Algorithm ReturnProcessor(ReturnSet, VMT, FromSet)	
Input: ReturnSet = {r1,r2,...,rn} is the data set in XQuery Return clause.	
Output: SelectionSet	
L01	SelectionSet = { };
L02	for all r_i in ReturnSet do
L03	$v_i = \text{GetTupleByVMT_xpath}(\text{xpath}, \text{VMT});$
L04	usedFromCheck(v_i , FromSet);
L05	selection_clause $_i = v_i.\text{Rname} + " . " + v_i.\text{Aname};$
L06	SelectionSet = SelectionSet $\cup \{(\text{selection_clause}_i)\};$
L07	end for
L08	return SelectionSet;

圖 4.7 : ReturnProcessor 演算法

Algorithm usedFromChcek(V_i , FromSet)	
V_i : tuple from VMT	
Output : FromSet	
L01	Determine if the input $V_i.\text{Rname}$ and $V_i.\text{RXPath}$ is in any tuple of FromSet
L02	if exists such tuple f_i then
L03	set UsedFlag[$f_i.\text{Rname}$] = TRUE;
L04	endif

圖 4.8 : usedFromChcek 演算法

【範例 4.4】

由範例 4.2 中取出 RETURN 子句的內容：

Return \$t0/name, \$t0/tel, \$t1/comment

首先 \$t0/name，經由圖 4.7 ReturnProcessor 演算法處理時，L03 行會以路徑表示法/cust_info/customer/name 至 VMT 表格取出對應的資料列，我們得到對應為 CUSTOMER 表格的 NAME 欄位。L04 行 usedFromChcek 演算法檢查後，會將 FromSet 中，對應 RXPath 為/cust_info/customer 的 CUSTOMER 表格之 UsedFlag 改設為 TRUE，代表 CUSTOMER 表格有被參考到須輸出。最後由 L05 與 L06 行將轉換的結果 CUSTOMER.NAME 加入至 SelectionSet 中。其餘內容節點經過 ReturnProcessor 演算法的轉換過程後，表格 4.2 的 FromSet 內容會更動如表格 4.5 所示，而產生的 SelectionSet 會如表格 4.6 所示。

```
{ ( /cust_info/nation, NATION, FALSE ),
  ( /cust_info/customer, CUSTOMER, TRUE ),
  ( /cust_info/customer, CUSTEL, TRUE ),
  ( /cust_info/vip_cust, CUSTOMER, TRUE ) }
```

表格 4.5：範例 4.2 經 ReturnProcessor 演算法更新之 FromSet 內容

```
{ (CUSTOMER.NAME), (CUSTEL.TELEPHONE), (CUSTOMER.COMMENT) }
```

表格 4.6：範例 4.2 經 ReturnProcessor 演算法產生之 SelectionSet 內容

B. 條件限制式轉換

針對 SQL 至 XQuery 的條件限制式轉換，如圖 4.9 S2XWhereprocessor_sel 演算法所示。Where 子句內容，其中又分為與另外一個欄位間的運算比較 (L06) 或是文數值限制 (L12)。條件限制式中的欄位內容，都會類似先前處理 SQL 中的 SELECT 子句內容的方式，做適當的變數代換以符合 XQuery 表示法 (L04~L05、L07~L08)，進而組合成完整的 Where 條件式(L10、L13)。組合的結果將加入至 S2XWhereSet 中(L15)，其中 S2XWhere_clause 為代換路徑表示法後的條件限制式。特別注意到由於兩個表格對應到同一個可重覆元素時，兩個表格間的連結限制

式，對應到的屬性節點會相同，因此會由 L09 排除產生此限制式。

Algorithm S2XWhereprocessor_sel(W_SelectionSet , VMT , ForSet)	
Input: W_selcond _i : Selection Condition in SQL Where Clause	
Output: S2XWhereSet	
L01	S2XWhereSet = { };
L02	for each W_selcond _i do
L03	split W_selcond _i into (W_selcond _i .PreField , W_selcond _i .operator , W_selcond _i .PostField);
L04	t1 = GetTupleByVMT_attr(W_selcond _i .PreField , VMT);
L05	exp1 = SubstituteExp(t1 , ForSet);
L06	if W_selcond _i .PostField is an attribute
L07	t2 = GetTupleByVMT_attr(W_selcond _i .PostField , VMT);
L08	exp2 = SubstituteExp(t2 , ForSet);
L09	if exp1 != exp2
L10	S2XWhere_clause = exp1 + W_selcond _i .operator + exp2;
L11	end if
L12	else // W_selcond _i .PostField is a value
L13	S2XWhere_clause = exp1 + W_selcond _i .operator + W_selcond _i .PostField;
L14	endif
L15	S2XWhereSet = S2XWhereSet ∪ {S2XWhere_clause} ;
L16	endfor
L17	return S2XWhereSet ;

圖 4.9： S2XWhereprocessor_sel 演算法

【範例 4.5】

範例 4.1 中 WHERE 子句限定表格 NATION 的 NAME 欄位內容為“台灣”，會經由圖 4.9 Whereprocessor_sel 演算法 L03 行拆解，此時 W_selcond_i.PreField 的內容為 NATION.NAME、W_selcond_i.operator 為“=”、W_selcond_i.PostField 的內容為‘台灣’。接著 L04 行比對 VMT 與執行 SubstituteExp 後，會得到 NATION.NAME 所對應的路徑表示法/cust_info/nation/name 被代換為\$t2/name。由於 W_selcond_i.PostField 為一個文數值限制，會經由 L13 將限制式結合成\$t2/name =

“台灣”再經 L15 加入至 S2XWhereSet 之中。此處需特別注意到在經過 SubstituteExp 處理後，會將表格 4.3 的 ForSet 內容中的變數\$t2 資料列的 UsedFlag 改設為 TRUE，更新過後的 ForSet 如表格 4.7 所示。轉換出的 S2XWhereSet 內容，如表格 4.8。

```
{ (CUSTOMER, $t0, /cust_info/customer, /cust_info/customer, Repeatable, TRUE),
  (CUSTOMER, $t1, /cust_info/vip_cust, /cust_info/vip_cust, Repeatable, TRUE),
  (CUSTEL, $t0, /cust_info/customer, /cust_info/customer, Repeatable, TRUE),
  (NATION, $t2, /cust_info/nation, /cust_info/nation, Repeatable, TRUE) }
```

表格 4.7：範例 4.1 經 S2XWhereprocessor_sel 演算法更新之 ForSet 內容

```
{ ($t2/name = “台灣”) }
```

表格 4.8：範例 4.1 經 S2XWhereprocessor_sel 演算法產生之 S2XWhereSet 內容

至於 XQuery 中 Where 子句的條件限制式 (Selection Condition)，處理的過程與圖 4.9 S2XWhereprocessor_sel 演算法觀念相同，主要差別是在於將葉節點的路徑，轉換成對應的欄位，詳細如圖 4.10 X2SWhereprocessor_sel 演算法所示。

【範例 4.6】

範例 5.5 中 WHERE 子句限定\$t2/name= “台灣”，首先會經由圖 4.10 X2SWhereprocessor_sel 演算法 L03 行將 WHERE 子句內容拆解，此時 W_selcondi.PreField 的內容為 \$t2/name、W_selcondi.operator 為 “=”、W_selcondi.PostField 的內容為 “台灣”。接著 L04 行會以所對應的完整路徑 /cust_info/nation/name，至 VMT 表格，取出對應的資料列，我們得到 /cust_info/nation/name 對應為 NATION 表格的 NAME 欄位。因此 L05 行

usedFromCheck 演算法檢查後，會將 FromSet 中，對應 XPath 為/cust_info/nation 的 NATION 表格之 UsedFlag 改設為 TRUE。更新後的 FromSet 內容，如表格 4.9 所示，而 X2SWhereSet 內容會如表格 4.10 所示。

Algorithm X2SWhereprocessor_sel($W_selcond_i$, VMT, FromSet)	
Input: $W_selcond_i$: Selection Condition in SQL Where Clause	
Output: X2SWhereSet	
L01	X2SWhereSet = { };
L02	for all $W_selcond_i$ do
L03	split $W_selcond_i$ into ($W_selcond_i.PreField$, $W_selcond_i.operator$, $W_selcond_i.PostField$);
L04	$t1 = GetTupleByVMT_xpath(W_selcond_i.PreField, VMT)$;
L05	usedFromCheck($t1$, FromSet);
L06	$exp1 = t1.Rname + " . " + t1.Aname$;
L07	if $W_selcond_i.PostField$ is a leaf node's path expression
L08	$t2 = GetTupleByVMT_xpath(W_selcond_i.PostField, VMT)$;
L09	usedFromCheck($t2$, FromSet);
L10	$exp2 = t2.Rname + " . " + t2.Aname$;
L11	if $exp1 \neq exp2$
L12	Where_clause = $exp1 + W_selcond_i.operator + exp2$;
L13	end if
L14	else // $W_selcond_i.PostField$ is a value
L15	X2SWhere_clause = $exp1 + W_selcond_i.operator + W_selcond_i.PostField$;
L16	end if
L17	X2SWhereSet = X2SWhereSet \cup {X2SWhere_clause} ;
L18	end for
L19	return X2SWhereSet ;

圖 4.10： X2SWhereprocessor_sel 演算法

<p>{ (/cust_info/nation, NATION, TRUE), (/cust_info/customer, CUSTOMER, TRUE), (/cust_info/customer, CUSTEL, TRUE), (/cust_info/vip_cust, CUSTOMER, TRUE) }</p>

表格 4.9：範例 4.2 經 X2SWhereprocessor_sel 演算法更新之 FromSet 內容

{ (NATION.NAME = ‘台灣’)}

表格 4.10：範例 4.2 經 X2SWhereprocessor_sel 演算法產生之 X2SWhereSet 內容

4.3 解決結構差異的演算法

針對結構表示法的不同差異，如 3.2.2 節中所述，我們已將關聯式資料的 Join Condition 記錄於 JMT 表格中，DTD 中的結構表示記錄於 PMT 中，而兩者間的關聯則是以 SMT 記錄，轉換過程將以此三個表格所提供的資訊來進行轉換。因為表格與可重覆元素有多元對應的關係，進而造成結構對應也有多元對應的關係，除了結構型態的轉換之外，應該選擇輸出哪個對應的結構，是另外一個重要的問題；本論文的作法，會由 4.2.2 節 Value 的處理，將所對應到的資料集合，將其 UsedFlag 設為 TRUE，代表此集合必須輸出，因此哪些集合之間的關聯必須建立，便是藉此依據來判斷。以下我們將以兩小節分別說明 SQL 至 XQuery 與 XQuery 至 SQL 的結構轉換。

4.3.1 SQL 至 XQuery 的結構轉換處理

SQL 中 Where 子句中的連結限制式 (Join Condition)，為表格與表格間的連結關係，而表格與表格間的關係，對應於 DTD 中為內部節點之間的關係，如 2.3.3 節中所介紹又可分為巢狀或是扁平的結構。在進行結構轉換之前，由於哪些表格所對應的可重覆元素需要被轉換出，已可由 ForSet 中 UsedFlag 被設為 TRUE 者得知，因此本論文作法會先利用該資訊挑選結構表示式。在圖 4.11 PMT_RefPreCheck 演算法中，L02~L06 會將 PMT 中的 xpath1 與 xpath2 其可重覆路徑在 ForSet 中已被設為 TRUE 者，將其 ref_flag 設為 TRUE，接著再由圖 4.12 Whereprocessor_join 演算法進行結構的選擇與轉換。

Algorithm PMT_RefPreCheck (ForSet , PMT)	
Input: ForSet , PMT	
Output:	
L01	for each rxpath _i in ForSet and it's UsedFlag == TRUE do
L02	use rxpath _i to compare with each XID's xpath1 & xpath2 in PMT
L03	if rxpath _i == XID _i .xpath1
L04	set XID _i .xpath1's ref_flag = TRUE;
L05	if rxpath _i == XID _i .xpath2
L06	set XID _i .xpath2's ref_flag = TRUE;
L07	end for

圖 4.11：PMT_RefPreCheck 演算法

圖 4.12 Whereprocessor_join 演算法 L03 行會依此連結條件式至 JMT 中取得對應的 RID 編號，若無取得的對應編號將會交由圖 4.9 S2XWhereprocessor_sel 演算法做處理 (L04、L05)；取得的 RID 編號將會至 SMT 中比對取得該連結限制式可能產生的所有結構對應的 XID 編號，暫存於 s2xtmpStructureList 之中 (L08)，稍後會由 L11 呼叫 S2XSelectStructureList 演算法選擇適當的結構輸出。注意 SQL 中 Where 子句中的連結限制式，若選擇輸出的 XID 編號屬於 X_{Ni} (L13)，代表此結構為一個巢狀結構，會由 L14 行的 ReconstructFor 演算法，進行 For 子句中的結構代換的動作，先說明 L17 行， X_{NDi} 亦是代表一個巢狀結構關係，其中有一個內部節點是空元素，由於空元素對應的表格名稱與最接近的子孫層之可重覆元素相同，我們在處理集合轉換時，是選擇轉換出表格對應到的可重覆元素，因此在 SQL 至 XQuery 轉換時，我們選擇轉換的是可重覆元素與可重覆元素之間的關係 (X_{Ni})，會排除 X_{NDi} 。若選擇輸出的 XID 編號屬於 X_{Fi} (L15)，代表此結構為一個扁平結構，則會由 L16 行的 AddFlatJoinToWhere 演算法，轉換出對應為扁平結構的連結限制式。以下先針對 ReconstructFor 與 AddFlatJoinToWhere 演算法的內容做說明，而 S2XSelectStructureList 選擇適當的結構輸出的演算法，將與 XQuery 轉換成 SQL 的結構對應選擇，一起於後面 4.3.3 節另做說明。

Algorithm S2XWhereProcessor_join(W_joincond , JMT, PMT, SMT, CMT, VMT , ForSet, S2XWhereSet)	
Input: W_joincond _i : Join Condition in SQL Where Clause	
Output: S2XWhereSet	
L01	for each W_joincond _i do
L02	split W_joincond _i into (W_joincond _i .PreField , W_joincond _i .operator , W_joincond _i .PostField);
L03	RID _i = GetRIDByJMT(W_joincond _i .PreField, W_joincond _i .PostField, JMT); // use join condition to get it's RID
L04	if RID _i is NULL
L05	S2XWhereprocessor_sel(W_joincond _i , VMT , ForSet)
L06	end if
L07	if RID _i is not NULL
L08	Get this RID _i 's related XID via SMT and store in s2xtmpStructureList; // s2xtmpStructureList = {(RID _i , XID1, XID2, ..., XIDn)}
L09	end if
L10	end for
L11	OutputStructureList = SelectStructureList(s2xtmpStructureList)
L12	for each XID _i in OutputStructureList //重覆的不會處理
L13	if XID _i ∈ X _{Ni} then
L14	ReconstructFor(XID _i .XPath1, XID _i .XPath2, ForSet , CMT);
L15	else if XID _i ∈ X _{Fi} then
L16	AddFlatJoinToWhere(XID _i .XPath1, XID _i .XPath2, S2XWhereSet , VMT);
L17	else if XID _i ∈ X _{NDi} then
L18	continue;
L19	end if
L20	end for

圖 4.12：S2XWhereProcessor_join 演算法

圖 4.13 ReconstructFor 演算法，處理關聯式連結對應到巢狀結構的情況（此部分轉換範例請參考 4.6.1 節的範例 4.11），基本上會將形成此巢狀結構的兩個內部節點之路徑表示法與其所分別對應的變數名稱，進行表示法代換的動作。首先 L01~L02 將會由 PMT 中取得的 XPath1 與 XPath2 內容，比對 ForSet 中的 RXPath 來取得各自對應的變數名稱，L03 將路徑較深的 XPath2，以 XPath1 的變數名稱來代換成巢狀結構的表示法，最後 L04 更新 ForSet 中 XPath2 相對應的內容。

Algorithm ReconStructFor(XPath1, XPath2, ForSet, CMT)	
Input: XPath1 and XPath2 : Paths got from PMT	
Output: ForSet	
L01	f1 = GetTupleBy_ForSet(XPath1, ForSet);
L02	f2 = GetTupleBy_ForSet(XPath2, ForSet);
L03	For_clause_new = f1.var + substring (XPath1, XPath2); // assume XPath1 < XPath2 in PMT
L04	UpdateForSet(f2.var, For_clause_new, ForSet);

圖 4.13：ReconStructFor 演算法

而圖 4.14 AddFlatJoinToWhere 演算法，則是處理關聯式連結對應到非巢狀結構的情況，演算法 L01、L02 會依輸入的兩個葉節點所對應的路徑，至 VMT 表格中取得其最接近的可重覆元素路徑，進而由 L03、L04 將此葉節點所對應的路徑與其最接近的可重覆元素所分配到的變數名稱，進行路徑表示法代換的動作，最後由 L05 結合此轉換後的連結限制式，將之加入至 S2XWhereSet 中 (L06)。

Algorithm AddFlatJoinToWhere(XPath1, XPath2, ForSet, S2XWhereSet, VMT)	
Input: XPath1 & XPath2 : Paths got from PMT	
Output: S2XWhereSet	
L01	t1 = GetTupleByVMT(XPath1 , VMT);
L02	t2 = GetTupleByVMT(XPath2 , VMT);
L03	exp1 = SubsituteExp (t1 , ForSet);
L04	exp2 = SubsituteExp (t2 , ForSet);
L05	S2XWhere_clause = exp1 + “=” + exp2;
L06	S2XWhereSet = S2XWhereSet \cup {(S2XWhere_clause)};

圖 4.14：AddFlatJoinToWhere 演算法

【範例 4.7】

例如範例 4.1 中 WHERE 子句的第一個連結限制式 CUSTOMER.NATIONKEY=NATION.NATIONKEY 首先會經圖 4.12 S2XWhereProcessor_join 演算法 L02 行將 WHERE 子句內容拆解，此時 W_joincondi.PreField 的內容為 CUSTOMER.NATIONKEY、W_joincondi.operator

為 “=”、W_joincondi.PostField 的內容為 NATION.NATIONKEY。接著由 L03 行至表格 JMT 中，取得對應的 RID 編號為 R1。由 L07~L09 至 SMT 中發現 (參考附錄 B)，R1 對應的 XID 編號為 X_{F1} 與 X_{F2} ，此時 $s2xtmpStructureList = \{(R1, X_{F1}, X_{F2})\}$ ；由於稍後我們才會講解到底該選擇輸出哪個結構，在此我們先假設會同時輸出 X_{F1} 與 X_{F2} 結構為例。由於 X_{F1} 與 X_{F2} 代表扁平結構的連結，因此會由圖 4.14 AddFlatJoinToWhere 演算法做處理。

以 X_{F1} 為例，於 PMT 中 X_{F1} 所對應的 XPath1 為 /cust_info/customer@nkey、XPath2 為 /cust_info/nation@nkey，AddFlatJoinToWhere 演算法中，會先將此兩個葉節點路徑由 L01~L04 行判斷其最接近的可重覆元素路徑是否在 ForSet 中，我們可發現到先前表格 4.7 的 ForSet 中，/cust_info/customer 與 /cust_info/nation 此兩筆資料列，分別的變數名稱為 \$t0 與 \$t2 且 UsedFlag 皆已設為 TRUE，L03 會將 /cust_info/customer@nkey 代換成 \$t0@nkey，L04 會將 /cust_info/nation@nkey 代換成 \$t2@nkey，最後由 L05 結合此非巢狀結構的連結條件式為 \$t0@nkey=\$t2@nkey，並由 L06 加入至 S2XWhereSet 中。至於另外一個編號 X_{F2} 的非巢狀結構 /cust_info/vip_cust@nkey 與 /cust_info/nation@nkey 經由 AddFlatJoinToWhere 演算法處理後，會得到 \$t1@nkey=\$t2@nkey。

而 Where 子句 CUSTOMER.CUSTKEY = CUSTEL.CUSTKEY 在 JMT 中找不到對應的 RID 編號 (因為等式左右的兩個欄位皆對應到相同可重覆元素下的屬性)，所以會由圖 4.9 S2XWhereprocessor_sel 處理，在圖 4.9 演算法的 L09 行會發現等式左右的兩個欄位，轉換後都對應到同一個屬性節點 /cust_info/customer@ckey，因此不會產生任何連結限制式輸出。經由以上說明，此時可得知範例 4.1 經過 S2XWhereProcessor_join 演算法處理後，S2XWhereSet 的內容如表格 4.11 所示。

$\{ (\$t2/name = \text{“台灣”}), (\$t0@nkey=\$t2@nkey), (\$t1@nkey=\$t2@nkey) \}$

表格 4.11：範例 4.1 經 S2XWhereProcessor_join 演算法更新之 S2XWhereSet 內容

4.3.2 XQuery 至 SQL 的結構轉換處理

XQuery 中 Where 子句中的連結限制式 (Join Condition)，表示 DTD 中兩個非巢狀關係的內部節點之間的關係，對應於表格與表格間的連結關係，會經由 X2SWhereprocessor_FlatJoin 演算法進行轉換。如同先前說明，由於一個內部節點與表格的對應可能是一對多對應，因此一個 XID 可能會對要數個關聯式連結限制式，本論文會先由圖 4.15 JMT_RefPreCheck 演算法，針對 JMT 中的 condition1 與 condition2 其表格名稱在 FromSet 中已被設為 TRUE 者，將其 ref_flag 設為 TRUE，接著再進行結構的選擇與轉換動作。

Algorithm JMT_RefPreCheck (FromSet , JMT)	
Input: FromSet , JMT	
Output:	
L01	for each Rname _i in FromSet and it's UsedFlag == TRUE do
L02	use Rname _i to compare with each RID's Relation name in condition1&condition2 by JMT
L03	if Rname _i == RID _i .condition1's Relation
L04	set RID _i .condition1's Relation's ref_flag == TRUE;
L05	if Rname _i == RID _i .condition2's Relation
L06	set RID _i .condition2's Relation's ref_flag == TRUE;
L07	end for

圖 4.15：JMT_RefPreCheck 演算法

接著，XQuery 中 Where 子句中的非巢狀結構的連結限制式會經由圖 4.16 X2SWhereProcessor_FlatJoin 演算法做處理，首先會由 L02 將條件限制式拆解，L03 再以此兩葉節點的路經表示法至 PMT 中，取得對應的 XID。由於內部節點與表格可能會有一對多的關係，相同的連結條件式 (XID) 可能會對應到不同的關聯式連結 (RID)，因此由 L06~L08 依此 XID 至 SMT 中取得所有其對應的 RID 編號，暫存於 x2stmpStructureList 之中，稍後會由 L10 呼叫 X2SSelectStructureList 演算法選擇適當的結構輸出。對應的連結限制式，會由 L13、L14 記錄於 X2SWhereSet 中。

Algorithm X2SWhereProcessor_FlatJoin(W_joincond, VMT, JMT, PMT, SMT, FromSet, WhereSet)	
Input: W_joincond _i : Join Condition in XQuery Where Clause	
Output: X2SWhereSet	
L01	for all W_joincond _i do
L02	split W_joincond _i into (W_joincond _i .PreField , W_joincond _i .operator , W_joincond _i .PostField);
L03	XID _i = GetXIDByPMT(W_joincond _i .PreField, W_joincond _i .PostField , PMT); // XID _i will $\in X_{Fi}$
L04	if XID _i is NULL
L05	X2SWhereprocessor_sel(W_joincond _i , VMT , FromSet)
L06	else if XID _i is not NULL
L07	Get this XID _i 's related RID via SMT and store in x2stmpStructureList; // x2stmpStructureList = {(XID _i , RID1, RID2, ..., RIDn)}
L08	end if
L09	end for
L10	OutputStructureList = SelectStructureList(x2stmpStructureList)
L11	for each RID _i in OutputStructureList //重覆的不會處理
L12	J _i = GetTupleByJMT(RID _i , JMT); //J _i is the tuple with the identified RID
L13	X2SWhere_clause = J _i .Condition1 + "=" + J _i .Condition2;
L14	X2SWhereSet = X2SWhereSet \cup {(X2SWhere_clause)};
L15	end for
L16	return X2SWhereSet;

圖 4.16：X2SWhereProcessor_FlatJoin 演算法

【範例 4.8】

範例 4.2 中，WHERE 子句中的 \$t0@nkey=\$t2@nkey 經過圖 4.16 X2SWhereProcessor_FlatJoin 演算法時，首先會被拆解成 PreField 的內容為 /cust_info/customer@nkey、operator 為 “=”、PostField 為 /cust_info/nation@nkey。接著，L03 與 L04 行會依此兩路徑表示法，比對 PMT 表格，取得對應此關係的 XID 編號 X_{F1} ；接著再由 L06~L08 依此 XID 編號至 SMT 表格中，取出相對應的關聯式連結編號 R1，並記錄至 x2stmpStructureList。另一個連結限制式 \$t1@nkey=\$t2@nkey 在 PMT 中的編號 X_{F1} ，會以相同步驟處理會得到對應 R1 的關聯式連結。此時 $x2stmpStructureList = \{ (X_{F1}, R1), (X_{F2}, R1) \}$ 。由於 XID

對應的 RID 只有一個，所以沒有結構一對多的選擇問題，X2SWhereProcessor_FlatJoin 演算法最後會由 L11~L15 將 R1 對應的關聯式連結，加入至 X2SWhereSet 中。而 \$t0@ckey=\$t1@ckey 在比對 PMT 表格時，無對應的結構編號，因此會由圖 4.10 X2SWhereprocessor_sel 處理，因為等式左右兩邊的葉節點路徑會對應到相同的表格與欄位名稱，因此沒有新的結構產生。X2SWhereSet 的內容如表格 4.12 所示。

```
{ (CUSTOMER.NAME = '台灣'),
```

```
(CUSTOMER.NATIONKEY = NATION.NATIONKEY) }
```

表格 4.12：範例 4.2 經 X2SWhereProcessor_join 演算法更新之 X2SWhereSet 內容

而 For 子句中，表示巢狀結構的內部節點關聯，對應於表格與表格間的連結，將由圖 4.17 NestedToWhereJoin 演算法檢查與轉換。首先會取出表格 PMT 中 XID 編號屬於 X_{Ni} 及 X_{NDi} 的資料列，然後將其 XPath1 與 XPath2 與先前轉換所產生的 FromSet 中比對，若 XPath1 與 XPath2 出現在 FromSet 的 RXPath 中，代表有此巢狀結構關係。L01~L03 取出其 XID 編號，在此注意到 L05 接著依此 XID 至 SMT 中取得所有其對應的 RID 編號，其中 RID 屬於 R_i (原因請參見 4.6.3 節)，暫存於 x2stmpStructureList 之中，稍後再呼叫 X2SSelectStructureList 演算法選擇適當的結構輸出。對應的連結限制式，會由 L12、L13 記錄於 X2SWhereSet 中。

Algorithm NestedToWhereJoin(JMT, PMT, FromSer, WhereSet)	
Input: JMT , PMT , FromSet , WhereSet	
Output: FromSet , WhereSet	
L01	for each tuple P_i that $P_i.XID \in X_{Ni}$ or $P_i.XID \in X_{NEi}$ from PMT do
L02	if $P_i.XPath1$ and $P_i.XPath2$ all found in FromSet's RXpath
L03	$XID_i = \text{GetXIDByPMT}(P_i.XPath1, P_i.XPath2, \text{PMT});$
	// XID_i will $\in X_{Ni}$
L04	if XID_i is not NULL
L05	Get this XID_i 's related $RID \in R_i$ via SMT and store in x2stmpStructureList;
	// $x2stmpStructureList = \{(XID_i, RID1, RID2, \dots, RIDn)\}$
L06	end if
L07	end if
L08	end for
L09	OutputStructureList = SelectStructureList(x2stmpStructureList)
L10	for each RID_i in OutputStructureList //重覆的不會處理
L11	$J_i = \text{GetTupleByJMT}(RID_i, \text{JMT});$ // J_i is the tuple with the identified RID
L12	$X2SWhere_clause = J_i.Condition1 + "=" + J_i.Condition2;$
L13	$X2SWhereSet = X2SWhereSet \cup \{(X2SWhere_clause)\};$
L14	end for
L15	return X2SWhereSet;

圖 4.17：NestedToWhereJoin 演算法

4.3.3 選擇對應結構輸出的方式與演算法

如同之前所討論，在結構轉換時，結構的對應可能為多元的對應關係，因此該轉換出哪個結構，本論文做法會藉由必須輸出的資料集合來判斷與選擇。如 SQL 轉換成 XQuery 的結構處理，首先，圖 4.11 的 PMT_RefPreCheck 演算法，會判斷 PMT 中的個別的集合來源，若在 ForSet 中的 UsedFlag 為 TRUE，代表此集合有被參考到，有可能轉換與其相關的結構對應，因此將 PMT 中此集合的 ref_flag 設為 TRUE。接著，分析被選出來的 RID 所對應到的所有 XID，由圖 4.18 S2XSelectStructureList 演算法選擇適當的結構轉換出來。

為了便於說明，在此設計一個 2*2 的對應關係，具代表性的情形如圖 4.19 中所示，其中 a1 與 a2 為 A 表格所對應的可重覆元素，而 b1 與 b2 為 B 表格所對應的可重覆元素，因此表格 A 與表格 B 之關聯限制式所對應的 XID 編號會有

Algorithm S2XSelectStructureList (s2xtmpStructureList)	
Input: s2xtmpStructureList	
Output: XIDs	
L01	for each tuple s_i in s2xtmpStructureList do
L02	for each $s_i.rid$ and it's related xids
L03	express xids' xpath1&xpath2 into n*m relation graph; //圓代表集合，邊代表兩集合之間的連結，實心圓表示此集合有被參考到須輸出
	//case(1)
L04	if there exists any (($xid_i.xpath1$'s ref_flag == true) &&
L05	($xid_i.xpath2$'s ref_flag == true))
L06	add this xid_i to OutputStructureList;
	//case(2)
L07	else for all xid such that (($xid_i.xpath1$'s ref_flag == true) &&
L08	($xid_i.xpath2$'s ref_flag == false))
L09	select the first xid_i by xpath1 and add to OutputStructureList;
	//case(3)
L10	else for all xid such that (($xid_i.xpath1$'s ref_flag == false) &&
L11	($xid_i.xpath2$'s ref_flag == true))
L12	select the first xid_i by xpath2 and add to OutputStructureList;
	//case(4)
L13	else for all xid such that (($xid_i.xpath1$'s ref_flag == false) &&
L14	($xid_i.xpath2$'s ref_flag == false))
L15	select the first xid_i in s2xtmpStructure and add to OutputStructureList;
L16	end if
L17	end for
L18	end for
L19	return xid;

圖 4.18：S2XSelectStructureList 演算法

四組 XF1~XF4，PMT_RefPreCheck 演算法會先依可重覆元素在 ForSet 中其 UsedFlag 有被設為 TRUE 者，將其 ref_flag 設為 TRUE，在圖中我們以實心圓圈表示，而線條的編號代表此關聯限制式所對應的結構編號，粗實線則是經由圖 4.18 S2XSelectStructureList 演算法則所選擇轉換出的對應結構。在演算法，我們討論四種情況，首先，若 PMT 中的 xpath1 與 xpath2 的 ref_flag 都為 TRUE 時 (L04~L06)，則是優先選擇輸出此結構編號，情形如圖 4.19 (Case1)所示。其次，

L07~L9 則是所對應的 PMT 結構中，其中僅有 xpath1 之 ref_flag 為 TRUE 之情況，此時若對應超過一個 XID 且其 xpath1 相同，可任選一組輸出，我們在此選擇於 PMT 中所找到的第一筆對應，具代表性的情形如圖 4.19 (Case2)所示。而 L10~L12 則是與 Case2 相反，其中僅有一個以上的 XID 其 xpath2 之 ref_flag 為 TRUE，則由 xpath2 來選擇所輸出的對應，如圖 4.19 (Case3)所示。至於 L13~L15 所對應的 XID 中，xpath1 與 xpath2 之 ref_flag 皆為 False 的情形，會選擇第一筆對應的 XID 輸出，如圖 4.19 (Case4)所示，此情況會發生在查詢句參考到多個表格所對應的連結關係，但僅輸出其中一個表格的資料時。

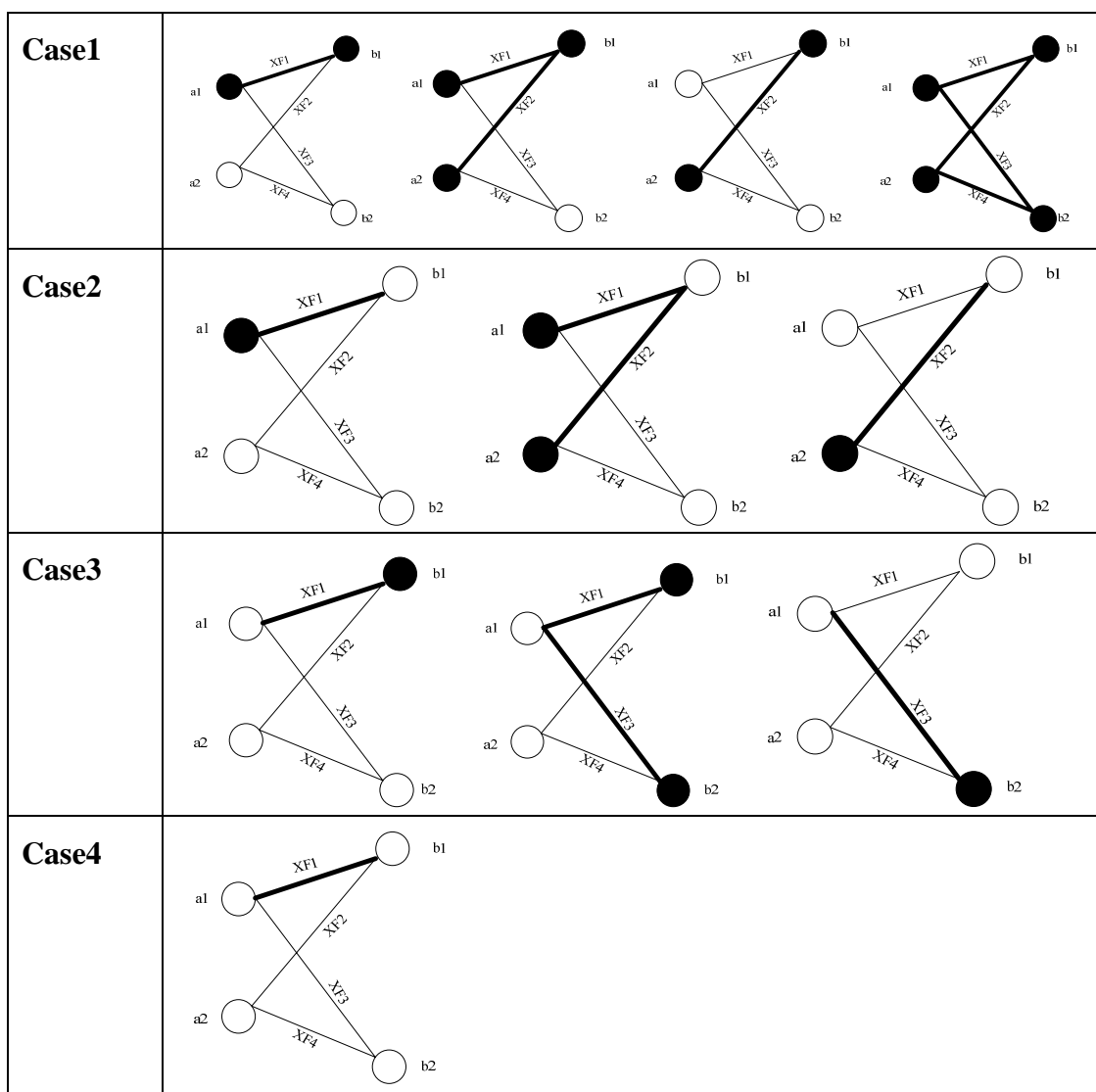


圖 4.19 選擇輸出結構的示意圖

參考範例 4.7 中，連結限制式經由圖 4.12 S2XWhereProcessor_join 演算法處理時，R1 對應的 XID 編號為 X_{F1} 與 X_{F2} ，此時 $s2xtmpStructureList = \{(R1, X_{F1}, X_{F2})\}$ ，而 PMT 中 X_{F1} 與 X_{F2} 所對應的 xpath1 與 xpath2 的資料集合來源 (RXPath)，由表格 4.7ForSet 中得知其 UsedFlag 皆為 TRUE，因此 X_{F1} 與 X_{F2} 所對應的 xpath1 與 xpath2 其集合來源的 ref_flag 皆為 TRUE，因此這兩個結構編號對應的扁平結構都會輸出。至於 XQuery 至 SQL 轉換的結構選擇，如圖 4.20 X2SSelectStructureList 演算法所述，選擇輸出的結構對應與先前說明一樣，只是在此演算法中要判斷的資料集合是關聯式表格。

Algorithm X2SSelectStructureList (x2stmpStructureList)	
Input: x2stmpStructureList	
Output: RIDs	
L01	for each tuple s_i in tmpStructure do
L02	for each $s_i.xid$ and it's related rids
L03	express rids' condition1 & condition1's Relation into n*m relation graph;
	//case(1)
L04	if there exists any ((RID _i .condition1's Relation reflag == true) &&
L05	(RID _i .condition2's Relation reflag == true))
L06	add this RID _i to OutputStructureList;
	//case(2)
L07	else for all RID such that ((RID _i .condition1's Relation reflag == true)
L08	&& (RID _i .condition2's Relation reflag == false))
L09	select the first RID _i by condition1's Relation and add to OutputStructureList;
	//case(3)
L10	else for all RID such that ((RID _i .condition1's Relation reflag == false)
L11	&& (all RID _i .condition2's Relation reflag == true))
L12	select the first RID _i by condition2's Relation and add to OutputStructureList;
	//case(4)
L13	else for all xid such that ((all RID _i .condition1's Relation reflag == false)
L14	&& (all RID _i .condition2's Relation reflag == false))
L15	select the first RID _i in x2stmpStructure and add to OutputStructureList;
L16	end if
L17	end for
L18	end for
L19	return RID;

圖 4.20：X2SSelectStructureList 演算法

4.4 查詢句驗證模組

經由基本差異與結構差異相關的演算法處理後，在驗證模組中，將針對多元對應的關係，將不必要輸出的集合刪除，並將所缺少的集合間，關聯表示式補齊。SQL至XQuery轉換的驗證模組如圖4.21 XQueryValidator演算法所示，ForSet內記錄變數名稱與該可重覆元素對應的路徑，在GetVarBy_ForSet演算法中，若該變數有被使用到，則該變數所對應的集合其UsedFlag會設為TRUE，若都沒被使用到則為FALSE，代表此變數所對應的可重覆路徑是多餘的不須輸出。首先L01~L05行是針對關係表格的轉換作檢查，當選擇關係表格的欄位內容，是在此關係表格對應的可重覆元素的祖先層可重覆元素之下的話，亦需將ForSet中，此關係表格所對應的可重覆元素輸出，再進行表示法的轉換，以符合相關結構限制，由於集合可有一對多的對應關係，我們選擇將ForSet中第一筆對應的UsedFlag設為TRUE，此部分在4.6節中將有範例做說明。接著L06~L10行濾除掉ForSet中沒有用到的集合資訊，並刪除重覆的集合表示法 (L11)。由於XQuery中的For子句中，巢狀結構的表示法會將較長的路徑以較短的路徑所對應的變數代號代換，因此L12會至ForSet中將較短至較長的可重覆元素路徑排序，再進行巢狀結構的變數代換檢查，使得巢狀結構中，集合之間的關聯得以建立，最後輸出合XQuery的表示法。

針對2.3.2節B點表格與內部節點一對多，相同的表格名稱卻對應到不同的可重覆元素路徑的情況，為了表示所取得的資料為關聯式表格中的同一筆資料列，此情況需另外記錄相關的屬性連結。演算法會由L13~L19根據IJT表格內的資料，至ForSet中，檢查該表格式否出現，若有則將此內部連結的關係加入至S2XWhereSet中。由於結構對應也有可能是多對一的關係，最後L20檢查若有多餘相同的連結限制式於S2XWhereSet中，將之刪除。至於XQuery至SQL轉換的驗證模組如圖4.22 SQLValidator演算法所示，類似的L01~L05會至FromSet中濾除掉沒有用到及重覆的表格名稱。針對2.3.2節A點表格與內部節點多對一，相同的內

部節點會對應到不同表格的情形，會由L08、L09判斷，取出IJT表格內的內部連結關係，加入至X2SWhereSet中。

注意到範例4.1經演算法S2XWhereprocessor_sel更新之ForSet內容如表格4.7中所示，圖4.21XQueryValidator的演算法L08~L10檢查時，參考附錄B中的IJT表格，我們發現第二筆資料列/cust_info/customer與/cust_info/vip_cust皆有在表格4.7的ForSet中，因此L10會將此內部連結資訊轉換並加入至S2XWhereSet中。更新

Algorithm XQueryValidator(ForSet, S2XWhereSet, IJT, VMT)	
Input: ForSet, S2XWhereSet, IJT	
Output: S2XWhereSet, ForSet	
	<i>//For check</i>
L01	for all the same (Rname&(Type∈ Repeatable&Nested)) in ForSet do
L02	if their UsedFlag are all FALSE then
L03	choose the first tuple and set it's UsedFlag be TRUE;
L04	end if
L05	end for
L06	for all f_i in ForSet do
L07	if UsedFlag[f_i .var] == FALSE then
L08	Remove f_i from ForSet; <i>//remove unused variable</i>
L09	end if
L10	end for
L11	Remove duplicate collection's path expression(XPath) in ForSet;
L12	Sort Tuples in ForSet by length of Xpath and ReconstructForCheck(ForSet); <i>//為了最後的合併，將\$t1~\$tn 路徑長度小至大先排序，並進行路徑代換檢查</i>
	<i>// Join check</i>
L13	for each tuple I_i in IJT and type∈ S2X do
L14	if I_i .collection1 and I_i .collection2 all appear as the RXpath in ForSet
L15	S2XWhereSet = S2XWhereSet ∪ { (I_i .InternalJoinCondition) };
L16	else
L17	continue;
L18	endif
L19	end for
L20	Remove duplicate join condition in S2XWhereSet;

圖 4.21：XQueryValidator 演算法

後的 S2XWhereSet 如表格 4.13 中所述。

```
{ ($t2/name = “台灣”), ($t0@nkey=$t2@nkey),  
  ($t1@nkey=$t2@nkey), ($t0@ckey=$t1@ckey) }
```

表格 4.13：範例 4.1 經 XQueryValidator 演算法更新之 S2XWhereSet 內容

而範例4.2經演算法X2SWhereprocessor_sel更新之FromSet內容如表格4.9中所示，注意到圖4.22 SQLValidator的演算法L07~L13檢查時，參考附錄B中的IJT表格，我們發現第一筆資料列CUSTOMER與CUSTEL皆有在表格4.9的FromSet中，因此L09會將此內部連結資訊轉換並加入至X2SWhereSet中。更新後的X2SWhereSet如表格4.14中所述。

Algorithm SQLValidator(FromSet, X2SWhereSet, IJT, VMT)	
Input: FromSet, X2SWhereSet, IJT	
Output: FromSet, X2SWhereSet	
	<i>// From check</i>
L01	for all f_i in FromSet do
L02	if UsedFlag[f_i .Rname] == FALSE then
L03	Remove f_i from FromSet; <i>//remove unused Rname</i>
L04	end if
L05	end for
L06	Remove duplicate Relation in FromSet;
	<i>// Join check</i>
L07	for each tuple I_i in IJT and $\text{type} \in \text{X2S}$ do
L08	if I_i .collection1 and I_i .collection2 are all found the same Relation in FromSet's Rname
L09	$\text{X2SWhereSet} = \text{X2SWhereSet} \cup \{(I_i.\text{InternalJoinCondition})\};$
L10	else
L11	continue;
L12	endif
L13	end for
L14	Remove duplicate join condition in X2SWhereSet;

圖 4.22：SQLValidator 演算法

```
{ (CUSTOMER.NAME = '台灣'),
  (CUSTOMER.NATIONKEY = NATION.NATIONKEY),
  (CUSTOMER.CUSTKEY=CUSTEL.CUSTKEY) }
```

表格 4.14：範例 4.2 經 SQLValidator 演算法更新之 X2SWhereSet 內容

4.5 查詢句組合模組

SQL 轉換的結果經過驗證模組處理後，就可由圖 4.23 XQueryConstructor 演算法將轉換的結果組合起來。其中 L02~L05 將轉換後的 ForSet 內容，結合其變數名稱，表示成 XQuery 的 For 子句表示法，最後由 L06~L10 將 For_Clause、WhereSet 與 ReturnSet 的內容組合成完整的 XQuery。而 XQuery 的轉換，經過驗證模組處理後，由圖 4.24 SQLConstructor 演算法將轉換的 FromSet、WhereSet 與 SelectionSet 的內容組合成完整的 SQL。

Algorithm XQueryConstructor(ForSet, WhereSet, ReturnSet)	
Input: ForSet, WhereSet, ReturnSet	
Output: XQuery	
L01	XQuery = { }; For_Clause = { };
L02	for all f_i in ForSet do
L03	For_expression $_i$ = f_i .var + “ in ” + f_i .For_clause $_i$;
L04	For_Clause = For_Clause \cup { (For_expression $_i$)};
L05	end for
L06	FXQuery = “For” + Each For_expression $_i$ from For_Clause;
L07	WXQuery = “Where” + Each Where_clause $_i$ from WhereSet;
L08	RXQuery = “Return” + Each Return_clause $_i$ from ReturnSet;
L09	XQuery = FXQuery + WXQuery + RXQuery ;
L10	return XQuery ;

圖 4.23：XQueryConstructor 演算法

Algorithm SQLConstructor(FromSet, X2SWhereSet, SelectionSet)	
Input: FromSet, X2SWhereSet, SelectionSet	
Output: SQL	
L01	SQL = { };
L02	SQL-S = “Select” + Each Selection_clausei from SelectionSet
L03	SQL -F= “From” + Each Relationi from FromSet avoid duplication;
L04	SQL-W = “Where” + Each X2SWhere_clausei from X2SWhereSet;
L05	SQL = SQL-S + SQL-F + SQL-W ;
L06	return SQL ;

圖 4.24：XQueryConstructor 演算法

綜合以上說明，我們將範例 4.1 經過最後 XQueryValidator 演算法的驗證後，ForSet、S2XWhereSet 與 ReturnSet 的內容，整理如表格 4.15 所示。我們可以看到 ForSet 中第一筆資料列，var 變數名稱為 \$t2，For_clause 內容為 /cust_info/nation，經過圖 4.23 XQueryConstructor 演算法 L01~L05 行，會將正確的表示法轉換成 \$t2 in /cust_info/nation，最後由 L06~L10 組合成的 XQuery 如範例 4.9 所示。

ForSet = { (NATION, \$t2, /cust_info/nation, \$t2 in /cust_info/nation, Repeatable, TRUE), (CUSTOMER, \$t0, /cust_info/customer, \$t0 in /cust_info/customer, Repeatable, TRUE), (CUSTOMER, \$t1, /cust_info/vip_cust, \$t1 in /cust_info/vip_cust, Repeatable, TRUE) }	
S2XWhereSet = { (\$t2/name = “台灣”), (\$t0@nkey=\$t2@nkey), (\$t1@nkey=\$t2@nkey), (\$t0@ckey=\$t1@ckey) }	
ReturnSet = { (\$t0/name), (\$t0/tel), (\$t1/comment)}	

表格 4.15：範例 4.1 經轉換模組處理後產生的內容

另外範例 4.2 再經過最後 SQLValidator 演算法的驗證後，FromSet、X2SWhereSet 與 ReturnSet 的內容，整理如表格 4.16 所示。經過圖 4.12 SQLConstructor 演算法 L02~L05 行組合成的如範例 4.10 所示。

【範例 4.9】

```
For $t2 in /cust_info/nation,  
    $t0 in /cust_info/customer,  
    $t1 in /cust_info/vip_cust  
Where $t2/name="台灣"  
  
    And    $t0@nkey=$t2@nkey  
    And    $t1@nkey=$t2@nkey  
    And    $t0@ckey=$t1@ckey  
  
Return $t0/name, $t0/tel, $t1/comment
```

<pre>FromSet = { (NATION , /Cust-Info/nation , TRUE) , (CUSTOMER, /Cust-Info/customer , TRUE) , (CUSTEL, /Cust-Info/customer , TRUE) } X2SWhereSet = { (CUSTOMER.NAME = '台灣'), (CUSTOMER.NATIONKEY = NATION.NATIONKEY), (CUSTOMER.CUSTKEY=CUSTEL.CUSTKEY) } SelectionSet = { (CUSTOMER.NAME), (CUSTEL.TELEPHONE), (CUSTOMER.COMMENT)}</pre>
--

表格 4.16：範例 4.2 經轉換模組處理後產生的內容

【範例 4.10】

```
SELECT  CUSTOMER.NAME, CUSTEL.TELEPHONE,  
        CUSTOMER.COMMENT  
FROM    NATION , CUSTOMER , CUSTEL,  
WHERE   NATION.NAME = '台灣'  
  
        AND  
        CUSTOMER.NATIONKEY = NATION.NATIONKEY  
        AND  
        CUSTOMER.CUSTKEY = CUSTEL.CUSTKEY
```

4.6 其它轉換範例與特殊情況探討

針對關聯式表格對應於 DTD 中為巢狀結構，以及 XQuery 的 For 子句中的路徑表示法為空元素時，演算法的處理過程，將由 4.6.1 節中的幾個範例來說明。另外我們討論 SQL 至 XQuery 轉換時，某些情況下，進行巢狀結構路徑代換，並不是完全必要的。

4.6.1 其它轉換範例

(A) 關聯式與關聯式表格之間對應於 DTD 中為巢狀結構

【範例 4.11】

```
SELECT  PARTSUPP.SUPPKEY, PARTSUPP.AVAILQTY,  
        LINEITEM.LINENUMBER  
FROM    PARTSUPP, LINEITEM  
WHERE   PARTSUPP.PARTKEY = LINEITEM.PARTKEY  
        AND  
        PARTSUPP.PARTKEY = 'p01'
```

此範例是針對圖 2.1 關聯式資料表格所設計的查詢句，其目的是經由 PARTSUPP 與 LINEITEM 表格，找出零件代號為 'p01' 的供應商編號、產品庫存量與出貨單編號。由於此兩表格皆為關係表格，分別會對應到圖 2.2 Order-Ship DTD Graph 中的 part 與 order 元素，不過由 VMT 中我們可發現，PARTSUPP.SUPPKEY 對應到的元素是可重覆元素 supplier 之下的 skey，因此會需要建立 supplier、part 與 order 此組巢狀結構之間的關聯。

首先轉換模組會先將 FROM 子句中 PARTSUPP 與 LINEITEM 兩個表格經由圖 4.2 FromProcessor 演算法 L04 比對 CMT 表格，獲知對應的可重覆路徑分別為 /order-ship/suppliers/supplier/part 與 /order-ship/suppliers/supplier/part/order，L08 會

分配變數名稱\$*t0* 與\$*t1* 給此兩路徑，最後由 L09 將結果記錄至 ForSet 中。經 FromProcessor 演算法的轉換結果表格 4.17 所示。

```
{ (PARTSUPP, $t0, /order-ship/suppliers/supplier/part,  
  /order-ship/suppliers/supplier/part, Repeatable&Nested, FALSE),  
  (LINEITEM, $t1, /order-ship/suppliers/supplier/part/order,  
    /order-ship/suppliers/supplier/part/order, Repeatable&Nested, FALSE)}
```

表格 4.17：範例 4.11 經 FromProcessor 演算法產生之 ForSet 內容

SELECT 子句中，PARTSUPP 表格的 AVAILQTY 欄位經由圖 4.4 ProjectProcessor 演算法的處理，會由 L03 至 VMT 表格取出對應的資料列，而得到欄位所對應路徑 (XPath) 為/order-ship/suppliers/supplier/part/availqty，可重覆路徑 (XPath) 為/order-ship/suppliers/supplier/part。將此可重覆路徑經由圖 4.6 GetVarBy_ForSet 演算法的處理，得到此可重覆路徑所對應的變數名稱為\$*t0*，同時將 UsedFlag[\$*t0*] 設為 TRUE，接著由圖 4.5 SubsituteExp 演算法將欄位所對應路徑 (XPath) 代換為 \$*t0*/availqty。而 LINEITEM 表格的 LINENUMBER 欄位，亦是類似的方式處理。

在此特別注意到，PARTSUPP 表格的 SUPPKEY 欄位，由圖 4.4 ProjectProcessor 演算法 L03 至 VMT 表格取出對應的資料列，而得到欄位所對應路徑 (XPath) 為 /order-ship/suppliers/supplier@skey，可重覆路徑 (XPath) 為 /order-ship/suppliers/supplier，由於 /order-ship/suppliers/supplier 沒有在先前 FromProcessor 演算法所產生的 ForSet 中，因此會由圖 4.6 GetVarBy_ForSet 演算法 L06~L10 處理，分配一個變數代號\$*s2* 代表此路徑 (\$*s* 表示是由處理欄位轉換才增加的集合)，並記錄於 ForSet 中。接著圖 4.5 SubsituteExp 演算法將欄位所對應的路徑/order-ship/suppliers/supplier@skey 代換成\$*s2*@skey，整個範例經過 ProjectProcessor 演算法的轉換過程後，表格 4.17 的 ForSet 內容會更動如表格 4.18

所示，而產生的 ReturnSet 會如表格 4.19 所示。

```
{ (PARTSUPP, $t0, /order-ship/suppliers/supplier/part,
  /order-ship/suppliers/supplier/part, Repeatable&Nested, TRUE),
  (LINEITEM, $t1, /order-ship/suppliers/supplier/part/order,
  /order-ship/suppliers/supplier/part/order, Repeatable&Nested, TRUE),
  (SUPPLIER, $s2, /order-ship/suppliers/supplier,
  /order-ship/suppliers/supplier, Repeatable, TRUE)}
```

表格 4.18：範例 4.11 經 ProjectProcessor 演算法更新之 ForSet 內容

```
{ ( $s2@skty) , ($t0/availqty) , ($t1@linenum)}
```

表格 4.19：範例 4.11 經 ProjectProcessor 演算法產生之 ReturnSet 內容

WHERE 子句中的條件限制式 PARTSUPP.PARTKEY = 'p01'，會類似之前範例 4.5 的說明，經由圖 4.9 Whereprocessor_sel 演算法處理，換出的結果會是 \$t0@pkey = "p01"，記錄於表 4.20 的 S2XWhereSet 中。

```
{($t0@pkey = "p01" )}
```

表格 4.20：範例 4.11 經 Whereprocessor_sel 演算法產生之 S2XWhereSet 內容

針對結構的處理，連結限制式 PARTSUPP.PARTKEY = LINEITEM.PARTKEY 會經圖 4.12 S2XWhereProcessor_join 演算法 L02 行將 WHERE 子句內容拆解，此時 W_joincond_i.PreField 的內容為 PARTSUPP.PARTKEY、W_joincond_i.operator 為 "="、W_joincond_i.PostField 的內容為 LINEITEM.PARTKEY。接著由 L03 行至表格 JMT 中，取得對應的 RID

編號為 RR2。由 L07~L09 至 SMT 中發現，RR2 對應的 XID 編號為 X_{N3} (參見表格 3.7)，此時 $S2XtmpStructureList = \{(RR2, X_{N3})\}$ ，由於 X_{N3} 是巢狀結構，會由圖 4.13 的 ReconStructFor 演算法做處理 (L14)。ReconStructFor 演算法會進行 XQuery 中的 For 子句路徑代換動作，以表示集合之間為巢狀的結構，L01、L02 行，會取出 PMT 表格中編號 X_{N3} 的兩個可重覆元素路徑 `/order-ship/suppliers/supplier/part` 與 `/order-ship/suppliers/supplier/part/order`，自 ForSet 中取得分別對應的變數代號 \$t0 與 \$t1，此時 \$t1 所對應的路徑，將會由 L03 行代換成 \$t0/order，並且更新 ForSet 中的 For_Clause 資訊 (L04)，整個範例經過 Whereprocessor_sel 演算法與 S2XWhereProcessor_join 演算法的轉換後，表格 4.18 的 ForSet 內容會更動如表格 4.21 所示。

```
{ (PARTSUPP, $t0, /order-ship/suppliers/supplier/part,
  /order-ship/suppliers/supplier/part, Repeatable&Nested, TRUE),
  (LINEITEM, $t1, /order-ship/suppliers/supplier/part/order, $t0/order,
    Repeatable&Nested, TRUE),
  (SUPPLIER, $s2, /order-ship/suppliers/supplier, /order-ship/suppliers/supplier,
    Repeatable, TRUE)}
```

表格 4.21：範例 4.11 經 ReconStructFor 演算法更新之 ForSet 內容

經由以上對應的片段查詢句轉換過程後，將由圖 4.21 的 XQueryValidator 驗證模組做最後的檢查動作，本範例此時會由 L11、L12 行，將 ForSet 中的內容依 XPath 由短至長排序，並進行路徑代換的檢查。因此，`/order-ship/suppliers/supplier` 會移至第一筆，並且將 \$t0 所對的 For_Clauses 內容代換成為 \$s2/part，這表示選擇的資料，必須符合可重覆元素 supplier 之下有 part 元素。經由 XQueryValidator 演算法更新後的 ForSet 內容如表格 4.22 所示。


```
{(SUPPLIER, $s2, /order-ship/suppliers/supplier, /order-ship/suppliers/supplier,
  Repeatable, TRUE),
(PARTSUPP, $t0, /order-ship/suppliers/supplier/part, $s2/part, Repeatable&Nested,
  TRUE),
(LINEITEM, $t1, /order-ship/suppliers/supplier/part/order, $t0/order, Repeatable&Nested,
  TRUE)}
```

表格 4.22：範例 4.11 經 XQueryValidator 演算法更新之 ForSet 內容

綜合以上說明，最後圖 4.23 XQueryConstructor 演算法，會取出表 4.19 ReturnSet、表 4.20 S2XWhereSet 以及表 4.22 ForSet 的內容，組合成轉換後的 XQuery 如範例 4.12 所示。

【範例 4.12】

```
For $s2 in /order-ship/suppliers/supplier,
  $t0 in $s2/part,
  $t1 in $t0/order
Where $t0@pkey= "p01"
Return $s2@skey , $t0/pro , $t1@linenum
```

(B) 選擇轉換關聯式表格的欄位

【範例 4.13】

```
SELECT LINEITEM.SUPPKEY
FROM LINEITEM
```

此範例由 VMT 中我們可發現，LINEITEM.SUPPKEY 對應到的元素是可重覆元素 supplier 之下的 skey，而 LINEITEM 表格對應到的可重覆元素為 order。首先轉換模組會先將 FROM 子句中 LINEITEM 表格經由圖 4.2 FromProcessor 演算法轉換結果表格 4.23 所示。在此注意到，LINEITEM 表格的 SUPPKEY 欄位轉換，會類似範例 4.11 中 PARTSUPP 表格的 SUPPKEY 欄位轉換說明，整個範例經過 ProjectProcessor 演算法的轉換過程後，表格 4.23 的 ForSet 內容會更動如表格 4.24 所示，而產生的 ReturnSet 會如表格 4.25 所示。

```
{(LINEITEM, $t0, /order-ship/suppliers/supplier/part/order,  
/order-ship/suppliers/supplier/part/order, Repeatable&Nested, FALSE)}
```

表格 4.23：範例 4.13 經 FromProcessor 演算法產生之 ForSet 內容

```
{ (LINEITEM, $t0, /order-ship/suppliers/supplier/part/order,  
/order-ship/suppliers/supplier/part/order, Repeatable&Nested, FALSE),  
(SUPPLIER, $s1, /order-ship/suppliers/supplier,  
/order-ship/suppliers/supplier, Repeatable, TRUE)}
```

表格 4.24：範例 4.13 經 ProjectProcessor 演算法更新之 ForSet 內容

```
{ ( $s1@skey) }
```

表格 4.25：範例 4.13 經 ProjectProcessor 演算法產生之 ReturnSet 內容

特別的是，本範例由圖 4.21 的 XQueryValidator 驗證模組做最後的檢查動作時，由 L01~L05 會發現 ForSet 中的 LINEITEM 表格，其 Type 為 Repeatable&Nested，且 UsedFlag 為 FALSE，此時會將其 UsedFlag 改設為 TRUE，以表示此對應的可重覆元素必須要轉換出。L12 行將 ForSet 中的內容依 XPath 由短至長排序，並進行路

徑代換的檢查。因此，/order-ship/suppliers/supplier 會移至第一筆，並且將\$t0 所對的 For_Clauses 內容代換成為\$s1/part/order，這表示選擇的資料，必須符合可重覆元素 supplier 之下的子元素依序為 part 與 order。轉換結果才會符合所找的資料是 LINEITEM 表格的 SUPPKEY。經由 XQueryValidator 演算法更新後的 ForSet 內容如表格 4.26 所示。

```
{ (SUPPLIER, $s1, /order-ship/suppliers/supplier,
  /order-ship/suppliers/supplier, Repeatable, TRUE),
  (LINEITEM, $t0, /order-ship/suppliers/supplier/part/order,
  $s1/part/order, Repeatable&Nested, TRUE) }
```

表格 4.26：範例 4.13 經 XQueryValidator 演算法更新之 ForSet 內容

綜合以上說明，最後圖 4.23 XQueryConstructor 演算法，組合成轉換後的 XQuery 如範例 4.14-1 所示。範例 4.13 中，查詢的資料 LINEITEM.SUPPKEY 於 VMT 中對應到可重覆元素 supplier 之下屬性節點 skey，由於 LINEITEM 關係表格是建立 SUPPLIER、PART 與 ORDER 表格之間的關係，會對應到內部節點 order，而 skey 是在內部節點 supplier 之下的屬性節點，因此範例 4.14-1 需要產生 For 子句中的巢狀結構代換關係，以限定所找出來的 skey 內容，必須是符合內部節點 supplier 有子孫層元素，且依序為 part 與 order。如此一來，所找出對應的 skey 內容才會是 LINEITEM 表格中的 SUPPKEY 欄位內容。

【範例 4.14-1】

```
For $s1 in /order-ship/suppliers/supplier,
  $t0 in $s1/part/order
Return $s1@skey
```

考慮若轉換成範例 4.14-2 的結果時，雖然與範例 4.14-1 一樣是輸出 skey，但 For 子句的內容為/order-ship/suppliers/supplier，沒有限定到符合內部節點 supplier 必須有子孫層元素，且依序為 part 與 order，因此輸出的 skey 會是內部節點 supplier 之下所有的 skey 內容，不是代表 LINEITEM 表格中的 SUPPKEY 欄位內容。因此本論文採取範例 4.14-1 的轉換方式。

【範例 4.14-2】

For \$t0 in /order-ship/suppliers/supplier

Return \$t0@skey

(C) For 子句中的路徑表示法為空元素路徑

【範例 4.15】

For \$t0 in /cust_info/nation/population

Return \$t0/region_popu/comment

範例 4.15 的 For 子句中/cust_info/nation/population 比對附錄 B 中的 CMT 表格，得知其為一個空元素的路徑，經由圖 4.3 ForProcessor 演算法處理時，L09~L11 會將空元素對應表格資訊，以及其子孫元素為可重覆元素的對應一起記錄至 FromSet 之中，轉換結果會如表 4.27 所示。

<pre>{ (/cust_info/nation/population, REGION, FALSE), (/cust_info/nation/population/region_popu, REGION, FALSE) }</pre>

表格 4.27：範例 4.15 經 ForProcessor 演算法產生之 FromSet 內容

接著處理 Return 子句的內容 \$t0/region_popu/comment，經由圖 4.7 ReturnProcessor 演算法處理時，L03 行會以路徑表示法

/cust_info/nation/population/region_popu/comment 至 VMT 表格取出對應的資料列，我們得到對應為 REGION 表格的 COMMENT 欄位。L04 行 usedFromChcek 演算法檢查後，會將 FromSet 中，對應 XPath 為 /cust_info/nation/population/region_popu 的 REGION 表格之 UsedFlag 改設為 TRUE，代表 REGION 表格有被參考到須輸出。最後由 L05 與 L06 行將轉換的結果 REGION.COMMENT 加入至 SelectionSet 中。經過 ReturnProcessor 演算法的轉換過程後，表格 4.23 的 FromSet 內容會更動如表格 4.28 所示，而產生的 SelectionSet 會如表格 4.29 所示。經過最後驗證模組刪除 UsedFlag 為 FALSE 的集合後，轉換組合的結果如範例 4.16。

```
{ (/cust_info/nation/population, REGION, FALSE),  
  (/cust_info/nation/population/region_popu, REGION, TRUE) }
```

表格 4.28：範例 4.15 經 ReturnProcessor 演算法更新之 FromSet 內容

```
{ (REGION.COMMENT) }
```

表格 4.29：範例 4.15 經 ReturnProcessor 演算法產生之 SelectionSet 內容

【範例 4.16】

```
SELECT REGION.COMMENT  
FROM REGION
```

4.6.2 SQL 轉換至 XQuery 可不需路徑代換的情形

在 SQL 轉換至 XQuery 的 ReconStructFor 演算法中，我們將形成巢狀結構的路徑，進行路徑代換的動作。有些情況下，若選擇輸出的欄位，對應到的葉節點是巢狀結構中最深層可重覆元素的子元素（葉節點），實際上不代換的話也是會符合正確的查詢句轉換。參考圖 2.1 關聯式網要與對應圖 2.3 的 DTD，範例 4.17 中的查詢句 (SQ1) 其目的在選出所有國家的地區名稱，經由本論文的演算法，會轉

換出 (XQ1) 的結果，其中 \$t1 in \$t0/region 即是本論文進行巢狀路徑代換的結果。由於 REGION.NAME 對應的葉節點 name 為可重覆元素路徑 /cust_info/nation/region 的子元素，此可重覆路徑已經代表 nation 與 region 元素為巢狀結構關係，因此 (SQ1) 若轉換成 (XQ2)，For 子句為最深層的可重覆元素路徑，或是 (XQ3)，For 子句為該欄位對應的葉節點，皆會符合正確的語意轉換。另外 (SQ1) 若轉換成 (XQ4)，其中 For 子句為重覆元素 nation，而 Return 子句 \$t0/region/name 代表找出 \$t0 所對應的內部節點路徑 /cust_info/nation 之下的子孫節點，是內部節點 region 之下的內容節點 name，亦是符合正確的語意轉換。

【範例 4.17】

(SQ1)

```
SELECT  REGION.NAME  
  
FROM    REGION, NATION  
  
WHERE   NATION.REGIONKEY = REGION.REGIONKEY
```

(XQ1)

```
For $t0 in /cust_info/nation,  
    $t1 in $t0/region
```

```
Return $t1/name
```

(XQ2)

```
For $t0 in /cust_info/nation/region  
Return $t0/name
```

(XQ3)

```
For $t0 in /cust_info/nation/region/name  
Return $t0
```

(XQ4)

For \$t0 in /cust_info/nation

Return \$t0/region/name

但是若選擇輸出的欄位，對應到的葉節點不全是巢狀結構中最深層可重覆元素的子元素，則必須進行路徑代換的動作，以符合集合之間的結構限制關係。如範例 4.18 中，(SQ1) 改成選擇 NATION 表格中的 NAME 欄位，會對應的葉節點 name 為可重覆元素路徑/cust_info/nation 的子元素，由於 NATION 與 REGION 表格分別對應的內部節點 nation 與 region，此兩內部節點之間為巢狀結構，因為本論會將 For 子句中的路徑表示法，代換如範例 4.18 (XQ1)所示。若沒進行代換，而直接輸出可重覆元素路徑/cust_info/nation 之下的葉節點 name 的內容（如範例 4.18 (XQ2)），則查詢出來的結果會類似範例 4.14-2 中的說明，不符合結構的限制關係，此時會輸出所有的name內容。為了符合一般化的轉換方式，我們僅考慮For子句中，為可重覆元素的路徑，並且會進行巢狀結構的路徑表示法代換動作。

【範例 4.18】

(SQ1)

SELECT NATION.NAME

FROM REGION, NATION

WHERE NATION.REGIONKEY = REGION.REGIONKEY

(XQ1)

For \$t0 in /cust_info/nation,

 \$t1 in \$t0/region

Return \$t0/name

(XQ2)

For \$t0 in /cust_info/nation

Return \$t0/name

4.6.3 關係表格與扁平或巢狀轉換結構選擇之說明

關係表格與關係表格之間的連結限制式，我們在建立 JMT 時，給予的編號為 RR_i ，特別區分是為了在 XQuery 巢狀結構轉換成對應的 SQL 時，我們會選擇轉換出實體表格與關係表格之間的關係，使得轉換後的查詢句，可以符合巢狀結構限制，所以在圖 4.17 NestedToWhereJoin 演算法，會選擇的 RID 屬於 R_i 。另外扁平結構不像巢狀結構的路徑表示法可以直接表示出集合與集合之間的關聯，因此圖 4.16 X2SWhereProcessor_FlatJoin 演算法不需特別區分 RID 是屬於 RR_i 或 R_i 。我們用以下範例說明，並探討巢狀結構時，其它可能的轉換方式。

(A) 扁平結構與關係表格

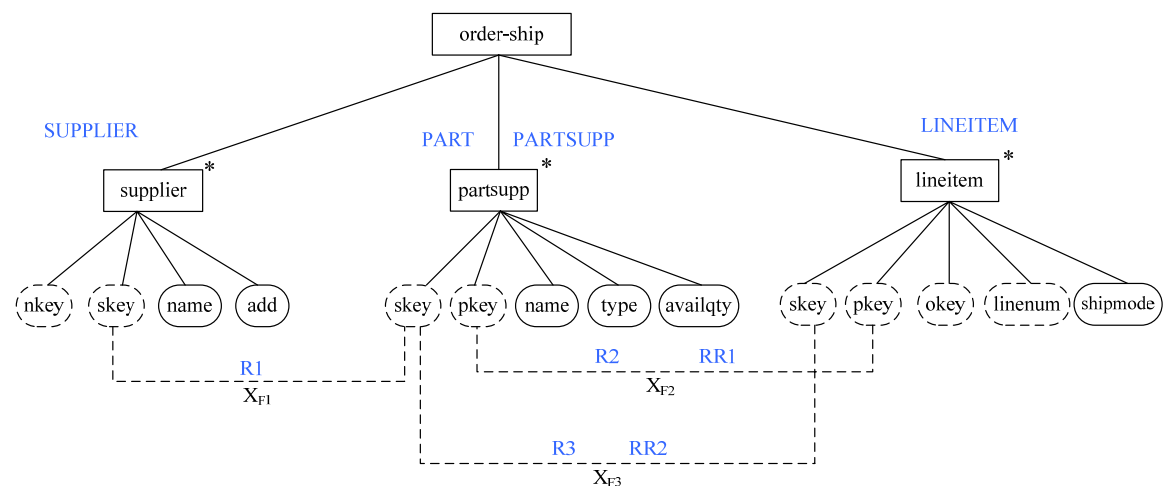


圖 4.25 Flat 結構的 DTD

考慮圖 4.25 的 DTD 範例，此範例會對應到圖 2.1 中的關聯式資料表格，在此圖中，我們將內部節點對應到的表格名稱，以淺色文字標註在旁，另外 XID

會對應的 RID 編號也是以淺色文字標註在旁，如 X_{F2} 會產生對應的 RID 編號為 R2 及 RR2。注意到內部節點 partsupp 會對應到 PART 與 PARTSUPP 兩個表格，而內部節點 lineitem 會對應到 LINEITEM 表格，假設 X_{F2} 為可重覆元素 partsupp 與 lineitem 利用屬性 pkey 連結的扁平結構編號，此時的 X_{F2} 就會對應到關聯式表格間的連結限制式 $PART.PARTKEY = LINEITEM.PARTKEY$ (R2) 或是 $PARTSUPP.PARTKEY = LINEITEM.PARTKEY$ (RR1)。考慮範例 4.19-1 與範例 4.19-2 的兩個 XQuery 查詢句轉換。

範例 4.19-1 查詢句 (XQ1) Return 子句中的 availqty 與 shipmode 會分別是 PARTSUPP 表格中的 AVAILQTY 欄位與 LINEITEM 表格中的 SHIPMODE 欄位，因此經過本論文的演算法轉換 For 子句以及 Return 子句後，會產生對應的 SQL 內容為 FROM PARTSUPP, LINEITEM 與 SELECT PARTSUPP.AVAILQTY, LINEITEM.SHIPMODE。結構對應轉換時，由於 PARTSUPP 及 LINEITEM 表格都必須被輸出，依轉換結構選擇的法則，會轉換出編號為 RR1 的連結限制式 $PARTSUPP.PARTKEY = LINEITEM.PARTKEY$ ，得到的 SQL 如 (SQ1) 所示。

【範例 4.19-1】

(XQ1)

For \$t0 in /order-ship/partsupp,

 \$t1 in /order-ship/lineitem

Where \$t0@pkey = \$t1@pkey (X_{F2})

Return \$t0/availqty, \$t1/shipmode

(SQ1)

SELECT **PARTSUPP.AVAILQTY**, LINEITEM.SHIPMODE

FROM **PARTSUPP**, LINEITEM

WHERE **PARTSUPP.PARTKEY = LINEITEM.PARTKEY** (RR1)

範例 4.19-2 查詢句 (XQ1)，是將範例 4.19-1 查詢句 (XQ1) 的 Return 子句的內容，改為 name 與 shipmode。注意到此時 name 會對應到 PART 表格中的 NAME 欄位，而 shipmode 一樣是對應到 LINEITEM 表格中的 SHIPMODE 欄位，因此經本論文的演算法轉換 For 子句以及 Return 子句後，會產生對應 SQL 的內容為 FROM PART, LINEITEM 與 SELECT PART.NAME, LINEITEM.SHIPMODE。結構對應轉換時，由於是 PART 及 LINEITEM 表格必須被輸出，依轉換結構選擇的法則，會與範例 4.19-1 查詢句 (XQ1) 不同，此範例會轉換出編號為 R2 的連結限制式 PART.PARTKEY = LINEITEM.PARTKEY，結果如以下的 (SQ1) 所示。

【範例 4.19-2】

(XQ1)

For \$t0 in /order-ship/partsupp,

\$t1 in /order-ship/lineitem

Where \$t0@pkey = \$t1@pkey (X_{F2})

Return \$t0/name, \$t1/shipmode

(SQ1)

SELECT **PART.NAME**, LINEITEM.SHIPMODE

FROM **PART**, LINEITEM

WHERE **PART.PARTKEY = LINEITEM.PARTKEY** (R2)

(B) 巢狀結構與關係表格 (符合結構限制的轉換)

考慮圖 2.1 關聯式表格與圖 2.2 的 DTD 對應，範例 4.20 中的 (XQ1) 是重複範例 4.12 的 XQuery 查詢句，會是範例 4.11 關聯式表格與關聯式表格為巢狀結構時，轉換後的結果。範例 4.20 再經由本論文的轉換法則所轉換成對應的 SQL 時，Return 子句中的 skey、availqty、linenum 會轉換比對 VMT 時，找到的第

一筆對應資料，因此會轉換出 SUPPLIER.SUPPKEY、PARTSUPP.AVAILQTY 與 LINEITEM.LINENUMBER，而 WHERE 子句中的 \$t0@pkey=“p01”則是會轉換成 PART.PARTKEY=‘P01’；此時轉換後的 SQL 之 FROM 子句內容會為 FROM SUPPLIER, PART, PARTSUPP, LINEITEM。在此注意到範例 4.20 的 For 子句中，supplier、part 與 order 為一組巢狀結構，本論文會將符合結構對應的 XID 編號皆選出，目的在建立可重覆元素與可重覆元素之間的關聯，挑選出來的編號會是為 X_{N1} 、 X_{N2} 、 X_{N3} （參考表格 3.6 PMT 內容）。但是在此三個 XID 轉換成對應的 RID 編號時，我們是選擇建立實體表格與關係表格之間的關聯 (R_i)，這是因為我們轉換的方式中，葉節點與欄位是一對多狀況時，是選擇 VMT 中所找到的第一筆對應來轉換，又因為不同表格具有相同鍵值的時候，我們會先建立實體表格的鍵值對應，再建立關係表格的鍵值對應，所以找到的第一筆資料對應會是實體表格中的欄位。因此，進行相關的結構轉換時，我們選擇轉換巢狀結構編號是對應實體表格與關係表格之間的關聯，以符合巢狀結構的限制，因此由上述選出的三個 XID，比對結構對應表格 3.7 SMT 內容，會選擇轉換的 RID 為 R1、R2 與 R3。在此為了便於後續的說明，我們將表格之間的關聯以及結構編號對應，以圖 4.26 E-R 及結構對應示意圖表示。

圖 4.26 中，方形代表實體表格（如 SUPPLIER），菱形代表關係表格（如 PARTSUPP）。表格旁的橢圓，是表示此表格對應 DTD 中的可重覆元素名稱，可由表格 3.1 CMT 發現，SUPPLIER 表格對應的可重覆元素為 supplier。另外，表格之間的連接線，代表利用相同鍵值建立的關聯，由表格 3.5 JMT 中，得到 SUPPLIER 表格與 PARTSUPP 表格利用 SUPPKEY 建立的連結限制式編號為 R1，因此在 SUPPLIER 表格與 PARTSUPP 表格之間的連接線上，標示 R1 以及 SUPPKEY；至於 R1 對應 DTD 中的結構編號，會比對表格 3.7 SMT 的內容，得到的 XID 在圖中我們以 (X_{N1}) 標示。注意到 PART 表格與 PARTSUPP 表格之間，由於會對應到相同的可重覆元素 part，因此由 XQuery 轉換成 SQL 時，可能需要經由 IJT（表格 3.2）產生內部連結資訊，PART 表格與 PARTSUPP 表格之間是利

用 PARTKEY 來建立連結，因此我們將 IJT 與 PARTKEY 標示在此兩表格之間的連接線上。

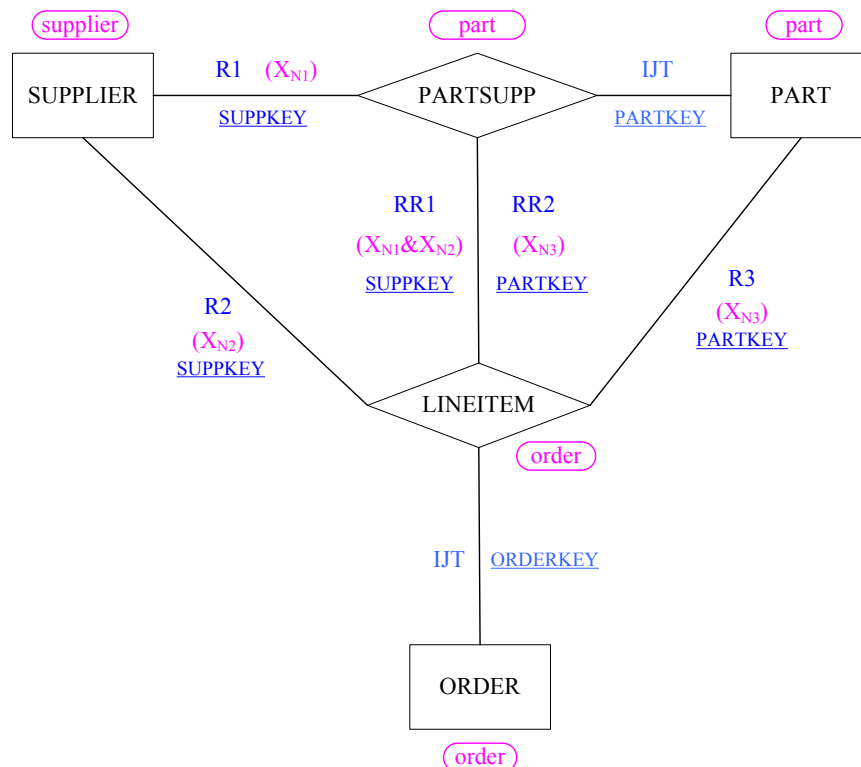


圖 4.26 E-R 及結構對應示意圖

綜合之前說明，範例 4.20 中的 XQ1 我們會轉換實體表格與關係表格之間的連結關係（圖中的 R1、R2、R3），來建立 SUPPLIER、PARTSUPP、PART 與 LINEITEM 之間的關聯，另外在最後驗證模組檢查時，會產生 PART 與 PARTSUPP 表格的內部連結，轉換的結果如範例 4.20 中的 SQ1 所示。

參考 E-R 圖的關係，可重覆元素與可重覆元素之間的關聯，挑選出來的編號會是 X_{N1} 、 X_{N2} 、 X_{N3} 時，我們考慮若選擇轉換對應的結構為 (RR1 及 RR2) 或是 (RR1 及 R3)，對應的查詢句分別如範例 4.21 中的 (SQ1) 與 (SQ2) 所示。我們經由實驗證明，此兩查詢句至資料庫中查詢所得的資料，也會與範例 4.20 中的查詢結果相同。(以下範例的實際查詢結果，請參見附錄 C)

【範例 4.20】 重複範例 4.12

(XQ1)

For \$s2 in /order-ship/suppliers/supplier,

 \$t0 in \$t2/part,

 \$t1 in \$t0/order

Where \$t0@pkey= “p01”

Return \$s2@skey , \$t0/availqty , \$t1@linenum

(SQ1)

SELECT SUPPLIER.SUPPKEY, PARTSUPP.AVAILQTY,

 LINEITEM.LINENUMBER

FROM SUPPLIER, PART, PARTSUPP, LINEITEM

WHERE PART.PARTKEY= ‘P01’

 And SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY (R1)

 And SUPPLIER.SUPPKEY = LINEITEM.SUPPKEY (R2)

 And PART.PARTKEY = LINEITEM.PARTKEY (R3)

 And PART.PARTKEY = PARTSUPP.PARTKEY (IJT)

【範例 4.21】 其它符合結構對應的 SQL 範例

(SQ1) 轉換出的 RID 選擇為 RR1 及 RR2 之情形

SELECT PARTSUPP.SUPPKEY , PARTSUPP.AVAILQTY ,

 LINEITEM.LINENUMBER

FROM PARTSUPP, LINEITEM

WHERE PARTSUPP.PARTKEY= ‘P01’

 And PARTSUPP.SUPPKEY = LINEITEM.SUPPKEY (RR1)

 And PARTSUPP.PARTKEY = LINEITEM.PARTKEY (RR2)

(SQ2) 轉換出的 RID 選擇為 RR1 及 R3 之情形

```
SELECT PARTSUPP.SUPPKEY, PARTSUPP.AVAILQTY,
       LINEITEM.LINENUMBER
FROM PARTSUPP, LINEITEM, PART
WHERE PART.PARTKEY= 'P01'

And PARTSUPP.SUPPKEY = LINEITEM.SUPPKEY (RR1)

And PART.PARTKEY = LINEITEM.PARTKEY (R3)

And PART.PARTKEY = PARTSUPP.PARTKEY (IJT)
```

分析原因，首先注意到在附錄A，圖A-1的VMT中，可發現到可重覆元素supplier下的屬性節點skey，會對應到SUPPLIER表格中的SUPPKEY欄位，以及PARTSUPP表格中的SUPPKEY欄位。另外，可重覆元素part下的屬性節點pkey，會對應到PART表格中的PARTKEY欄位，以及PARTSUPP表格中的PARTKEY欄位。再者，注意到在屬性的轉換選擇，範例4.21中的 (SQ1) 範例與範例4.20 (SQ1) 較不同的是，會選擇的欄位為PARTSUPP的SUPPKEY欄位，且條件限制式為PARTSUPP.PARTKEY= 'P01'。

配合圖4.26 E-R及結構對應示意圖，分析範例4.20 (SQ1) 的結構限制，連結限制式 (R1) 與 (R2) 所產生的關係，利用SUPPKEY間接建立起PARTSUPP與LINEITEM表格間的關聯，會等價於範例4.21 (SQ1) 中的 (RR1)，；因此，範例4.20 (SQ1) 選擇SUPPLIER表格的SUPPKEY內容，會與範例4.21 (SQ1)所選擇PARTSUPP的SUPPKEY內容相同。另外，範例4.20 (SQ1) 的連結限制式 (R3) 與 (IJT) 的所產生的關係，也會等價於範例4.21 (SQ1)中的 (RR2)，因此，範例4.20 (SQ1)中選擇PART表格的PARTKEY內容，會與範例4.21中，所選擇PARTSUPP的PARTKEY內容的範例相同。經由上述說明，可推知範例4.20與範例4.21中的兩個SQL查詢句，互為查詢結果會等價的查詢句。由於本論文並未進行最佳化查詢句的轉換，也就是當屬性對應到不同表格中的欄位時，先判斷並轉換出最少的集

合對應，因此會採取範例4.20的轉換法。

(C) 巢狀結構與關係表格 (不符合結構限制的轉換)

另外我們考慮另一種做法，巢狀結構時，僅會分別建立可重覆元素父子之間的關聯，也就是範例 4.20 (XQ1)中，僅分別轉換 supplier 與 part 的結構，還有 part 與 order 的結構 (R1 及 R3) 來比較，結果如範例 4.22 所示。經由實驗發現，在有考量關係表格與關係表格間的對應為巢狀結構時，這樣的轉換方式，查詢的結果可能會與原先 XQuery 不同。配合圖 4.26，範例 4.22 轉換出 R1、R3 還有 IJT 的內容，會建立 SUPPLIER、PARTSUPP、PART 與 LINEITEM 之間的關聯，但是會缺少了 SUPPLIER 表格與 LINEITEM 表格或是 PARTSUPP 與 LINEITEM 表格利用 SUPPKEY 來連結的關係 (R2 或 RR1)，因此轉換出來的結果，會不完全符合範例 4.20 中的 XQ1 的巢狀結構限制。

【範例 4.22】 轉換出的 RID 為 R1 及 R3

```
SELECT  SUPPLIER.SUPPKEY , PARTSUPP.AVAILQTY ,  
        LINEITEM.LINENUMBER  
FROM    SUPPLIER, PARTSUPP, LINEITEM, PART  
WHERE   PART.PARTKEY= 'P01'  
  
And     SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY    (R1)  
  
And     PART.PARTKEY = LINEITEM.PARTKEY        (R3)  
  
And     PART.PARTKEY = PARTSUPP.PARTKEY        (IJT)
```

我們再考量範例 4.23 中的 (XQ1)，此範例在找出可重覆元素 part 之下的內容節點 availqty 以及 order 元素之下的屬性節點 linenum，比對 VMT 時，會轉換出 PARTSUPP.AVAILQTY 與 LINEITEM.LINENUMBER。而由 For 子句中，可發現到可重覆元素 part 與 order 為巢狀結構，比對 PMT 表格 (表格 3.6) 對應結構

編號為 X_{N3} ，配合表格 3.7 SMT 與 圖 4.26 E-R 及結構對應示意圖，經由本論文的轉換法則，會轉換出的 SQL 如 (SQ1) 所示，會轉換出對應的 RID 編號為 R3，並建立 PART 與 PARTSUPP 表格之間的內部連結。另外與範例 4.21 的 (SQ1) 相仿，假設轉換出 X_{N3} 對應的 RID 為關係表格與關係表格之間的連結限制式 (RR2) 時，查詢句會如範例 4.23 的 (SQ2) 所示。但經由實驗證明，範例 4.23 中的 (SQ1) 與 (SQ2) 查詢的結果相同，但因為不完全符合 (XQ1) 巢狀結構的關係，所查詢出的結果與 (XQ1) 不同。

【範例 4.23】

(XQ1)

For \$t0 in /order-ship/supplier/part,

\$t1 in \$t0/order

Return \$t0/availqty, \$t1@linenum

(SQ1)

SELECT PARTSUPP.AVAILQTY, LINEITEM.LINENUMBER

FROM PARTSUPP , LINEITEM , PART

WHERE PART.PARTKEY = LINEITEM.PARTKEY (R3)

AND

PART.PARTKEY = PARTSUPP.PARTKEY (IJT)

(SQ2)

SELECT PARTSUPP.AVAILQTY, LINEITEM.LINENUMBER

FROM PARTSUPP , LINEITEM

WHERE PARTSUPP.PARTKEY = LINEITEM.PARTKEY (RR2)

配合圖 4.26 E-R 及結構對應示意圖，分析 (SQ1) 查詢句的連結限制式 (R3) 與 (IJT) 的所產生的關係，會利用 PARTKEY 間接建立起 PARTSUPP 與 LINEITEM 表格之間的關聯，因此會等價於範例 (SQ2) 中的連結限制式 (RR2)，所以查詢所得的 PARTSUPP.AVAILQTY 與 LINEITEM.LINENUMBER 結果會是相同的。但注意到 (XQ1) 查詢句中，選擇的葉節點會對應到 PARTSUPP 與 LINEITEM 表格，而此兩表格之間必須利用組合鍵 (SUPPKEY 及 PARTKEY) 來連結此兩表格的關係，才能完全符合對應 part 與 order 的巢狀結構限制 (參考圖 4.26)。因此 (SQ1) 與 (SQ2) 僅用 PARTKEY 建立表格之間的連結，查詢的結果會與 (XQ1) 不同。

綜合以上說明，若是轉換關係表格與關係表格之間的關聯，且此兩關係表格是利用複合鍵 (Composite Key) 來建立連結，當此兩關係表格對應於 DTD 中的結構為巢狀時，不同的鍵值對應到巢狀結構中不同階層可重覆元素之下的屬性節點，此時，若僅使用其中的一個鍵值來建立兩關係表格間的關聯，此鍵值是對應到較下層的可重覆元素時，則 SQL 轉換至 XQuery 時，因為另一個鍵值所對應的集合未被參考到，本論文目前的做法在此情況下，轉換出的查詢句不一定會完全符合巢狀結構的關係。

第五章 正確性分析與轉換效率評估

在本章中，我們將以不同類型的查詢句，來驗證轉換系統的正確性，評估輸入的查詢句與轉換出的查詢句是否具相同的語義，以及相關性質的探討比較。另外，我們將真實的資料分別以關聯式表格與 XML 格式儲存於資料庫中，再分別將輸入轉換系統的查詢句與轉換出來的對等查詢句，至資料庫中進行查詢，比較資料回傳的結果並分析。轉換效率部份，主要在評估本論文所設計的 Value、Collection 與 Structure 對應為多元對應時，對轉換效率的影響；其次評估針對雙向轉換輸出的 Structure 個數、DTD 為階層式的深度與扁平式的寬度對轉換效率的影響。實驗測試的電腦配備中央處理器為 P4-2.4 Ghz、512MB DDR 記憶體，搭配 Windows 2000 Advance Server 中文版作業系統，使用的資料庫為 SQL 2005。

5.1 正確性分析

在本論文中，對應表格記錄了關聯式資料與 XML Schema 的對應關係，我們將設計一系列的實驗證明所提出的轉換系統在多元對應的情況下，經由對應表格所提供的資訊，可轉換出等價的查詢句。在本節中，我們將以圖 2.1、圖 2.2 及圖 2.3 的關聯式與 XML 綱要，分別針對 Value、Collection 與 Structure 對應的區分來設計與分析不同類型的查詢句，驗證轉換系統的正確性。在實驗中，我們將主要以 SQL 為輸入的查詢句，作為 SQL 至 XQuery 的轉換範例，而轉換出的 XQuery，會再經由轉換系統轉換成對應的 SQL'（第二次轉換稱為反向轉換）；若輸入的查詢句為 XQuery，亦會類似上述進行兩次的轉換，藉此來分析雙向轉換的正確性與特性。

A. Value 的對應轉換

我們考慮圖 2.3 的 Cust-Info DTD，由附錄 B 的 VMT 表格看出 CUSTOMER 表格的 CUSTKEY 欄位，對應到 DTD 中的屬性節點/cust_info/customer@ckey 與

/cust_info/vip_cust@ckey，為一個欄位與葉節點 1 對 2 的對應關係。範例 5.1-1 的 SQL 查詢句目的在找出所有顧客的編號。首先，演算法在處理 Collection 對應時，會記錄 CUSTOMER 表格 (參見附錄 B 的 CMT) 對應到可重覆元素路徑 /cust_info/customer 與 /cust_info/vip_cust；而處理 Value 的對應時，會轉換出的 Attribute 為 VMT 中所找到的第一筆，也就是 /cust_info/customer@ckey，所以演算法會選擇輸出可重覆元素為 /cust_info/customer。由結果發現本範例雙向轉換皆會符合第二章正確性的定義一。轉換後的 XQuery 再經由轉換系統轉換時，可重覆元素路徑 /cust_info/customer 比對 CMT 可發現它對應到 CUSTOMER 與 CUSTEL 表格，而由 return \$t0@ckey 將葉節點的完整路徑比對 VMT 表格亦會取得所找到的第一筆 CUSTOMER.CUSTKEY，因此轉換的結果如 SQL' 所示，我們可觀察 SQL' 與剛開始輸入的 SQL 會相同。

【範例 5.1-1】

輸入的 SQL	→	轉換後的 XQuery	→	由 XQuery 再轉換回的 SQL'
SELECT CUSTOMER.CUSTKEY		for \$t0 in /cust_info/customer		SELECT CUSTOMER.CUSTKEY
FROM CUSTOMER		return \$t0@ckey		FROM CUSTOMER

另外，由 VMT 表格可發現到關聯式表格 CUSTOMER 與 CUSTEL 表格的 CUSTKEY 欄位，同時對應到屬性節點 /cust-info/customer@ckey，為一個欄位與葉節點 2 對 1 的對應關係，我們探討此時輸入為 XQuery 的雙向轉換的結果如範例 5.1-2 所示。For 子句中 /cust_info/customer 由 CMT 表格中可發現到其對應的表格名稱為 CUSTOMER 與 CUSTEL；而此查詢句 Return 的葉節點為 /cust-info/customer 下的屬性節點 ckey，比對 VMT 表格會發現其對應的第一筆資料為 CUSTOMER 表格的 CUSTKEY 欄位，因此由演算法會轉換出的 SQL 如範例中所示。此時再由轉換出的 SQL 轉換成 XQuery' 時，SELECT 子句選擇 CUSTOMER 表格的 CUSTKEY 欄位，經由比對 VMT 得到的第一筆對應亦為 /cust-info/customer@ckey，因此轉換後的 XQuery' 會與輸入的 XQuery 相同，本範例雙向轉換皆會符合第二章正確性的定義一。

【範例 5.1-2】

輸入的 XQuery	→	轉換後的 SQL	→	由 SQL 再轉換回的 XQuery'
for \$t0 in /cust_info/customer		SELECT CUSTOMER.CUSTKEY		for \$t0 in /cust_info/customer
return \$t0@ckey		FROM CUSTOMER		return \$t0@ckey

注意到範例 5.2 的情況，輸入的 SQL 改成選擇 CUSTEL 的 CUSTKEY 欄位。於演算法處理 Collection 對應時，CUSTEL 表格對應到可重覆元素路徑 /cust_info/customer，而處理 Value 的對應時，可發現 CUSTOMER.CUSTKEY 與 CUSTEL.CUSTKEY 在 VMT 中所對應的屬性節點皆為 /cust-info/customer@ckey，為一個欄位與葉節點 2 對 1 的對應關係，因此當選擇轉換 CUSTEL 的 CUSTKEY 欄位，轉換後的 XQuery 會與範例 5.1-1 的 XQuery 相同；但是轉換後的 XQuery 再經由轉換系統轉換成 SQL' 時，由於可重覆元素路徑 /cust-info/customer 比對 CMT 可發現它對應到 CUSTOMER 與 CUSTEL 表格，而由 return \$t0@ckey 將葉節點的完整路徑比對 VMT 表格亦會取得所找到的第一筆 CUSTOMER.CUSTKEY，因此轉換結果如 SQL' 所示，我們可觀察到此轉換後的 SQL' 會與輸入的 SQL 不相同，此情況是因為本轉換系統選擇輸出的 Attribute 為在 VMT 中找到的第一筆對應所致，但雙向轉換結果皆符合定義一的對應關係，亦為一個正確的轉換。

【範例 5.2】

輸入的 SQL	→	轉換後的 XQuery	→	由 XQuery 再轉換回的 SQL'
SELECT CUSTEL.CUSTKEY		for \$t0 in /cust_info/customer		SELECT CUSTOMER.CUSTKEY
FROM CUSTEL		return \$t0@ckey		FROM CUSTOMER

範例 5.3 是針對圖 2.1 與圖 2.2 的 Order-Ship DTD 所設計的查詢句，其目的在找出有顧客訂單的廠商編號，直接從 LINEITEM 表格中選擇 SUPPKEY 來位輸出。在我們建立 CMT 的規則中，若關係表格與其他表格之間的關係在 DTD 中為一個巢狀結構，此時關係表格對應的可重覆元素會與對應到最深層可重覆元

素的實體表格相同，所以 LINEITEM 表格會對應到 order，詳細的對應表格請參見附錄 A。但是，LINEITEM 的 SUPPKEY 對應到的屬性節點路徑為 /order-ship/supplier/suppliers@skey，並非定義在最深層的可重覆元素上，本論文所轉換出的 XQuery 會如範例中所示，其中 \$t0 in \$s1/part/order 是限定輸出的 skey 內容，必需滿足可重覆元素 suppliers 其下兩層的子孫節點分別為 part 與 order，所以 Collection 的個數比原來 SQL 中的多。另外，注意到轉換後的 XQuery 再經由轉換模組轉成對應的 SQL'時，For 子句中的巢狀結構符合 PMT 中的結構編號 (X_{N2})，所以此結構會轉換其對應的 RID 編號 (R2) 內容為 SUPPLIER.SUPPKEY = LINEITEM.SUPPKEY。屬性節點 skey 比對 VMT 表格時，找到的第一筆資料會選擇輸出 SUPPLIER 的 SUPPKEY，因此轉換結果如 SQL' 中所示。我們發現到 SQL' 會與原先輸入的 SQL 不同，但其具有適當的連結關係符合查詢語意，因此，本範例由 SQL 轉換至 XQuery 符合定義二的對應關係，亦為一個正確的轉換；而 XQuery 轉換至 SQL' 則是符合定義一。

【範例 5.3】

輸入的 SQL	→	轉換後的 XQuery	→	由 XQuery 再轉換回的 SQL'
SELECT LINEITEM.SUPPKEY		for \$s1 in		SELECT SUPPLIER.SUPPKEY
		/order-ship/suppliers/supplier,		FROM SUPPLIER, LINEITEM
		\$t0 in \$s1/part/order		WHERE SUPPLIER.SUPPKEY=
FROM LINEITEM		return \$s1@skey		LINEITEM.SUPPKEY

B. Collection 的對應轉換

本小節中我們將探討 Collection 對應與轉換的正確性分析。參考範例 5.4 中的 SQL，其目的在從 CUSTOMER 表格中找出顧客編號為 'c03' 顧客的 NAME 與 COMMENT；同範例 5.1-1 的討論，當資料表格的來源為 CUSTOMER 時集合的對應轉換會產生對應的 /cust_info/customer 與 /cust_info/vip_cust 路徑，而由於選擇 CUSTOMER 表格的 NAME 與 COMMENT 欄位，分別是上述可重覆元素下的葉節點 name 與 comment，因此上述兩個可重覆路徑都會輸出；由於表

格與內部節點 1 對 2 對應關係，於檢查內部連結資訊時，會產生兩集合間的連結限制式 \$t0@ckey = \$t1@ckey 來建立兩個內部節點間的關聯，這樣一來轉換 WHERE 子句中的 CUSTOMER.CUSTKEY = 'c03' 時，在此可發現到 SQL 與 XQuery 之間的 Collection 個數不同，這是因為表格與內部節點 1 對 2 對應關係所照成，但是轉換出的 XQuery 集合之間會具有適當的連結關係，符合第二章中定義二的對應關係，轉換後的 XQuery 如範例 5.4 中所示。此時再由轉換後的 XQuery 進行轉換成相對的 SQL' 時，因為 CMT 中 /cust_info/customer 與 /cust_info/vip_cust 都會對應到 CUSTOMER 表格，且選擇輸出的葉節點 name 與 comment 在 VMT 中的對應分別為 CUSTOMER 表格中的 NAME 與 COMMENT 欄位，而 XQuery 中的 where 限制式 \$t0@ckey = \$t1@ckey，等式兩邊的屬性節點都會對應到 CUSTOMER 表格的 CUSTKEY 欄位，因此不會有連結限制式輸出，轉換後的 SQL' 如範例中所示，會與輸入的 SQL 相同。

【範例 5.4】

輸入的 SQL	→	轉換後的 XQuery	→	由 XQuery 再轉換回的 SQL'
SELECT CUSTOMERNAME, CUSTOMER.COMMENT		for \$t0 in /cust_info/customer, \$t1 in /cust_info/vip_cust where \$t0@ckey = "c03"		SELECT CUSTOMERNAME, CUSTOMER.COMMENT
FROM CUSTOMER		and t0@ckey = \$t1@ckey		FROM CUSTOMER
WHERE CUSTOMER.CUSTKEY='c03'		return \$t0/name, \$t1/comment		WHERE CUSTOMER.CUSTKEY=c03'

接下來範例 5.5 類似範例 5.4，只是情況為內部節點與表格為 1 對 2 的對應關係，輸入的 XQuery 查詢句是要輸出可重覆元素路徑 /cust_info/customer 之下的葉節點 ckey 編號為 c05 的 name 與 tel 元素內容，至於轉換後的 SQL 再轉換成相對應的 XQuery' 時，如範例 5.5 中所示，會與輸入的 XQuery 相同。如範例 5.4 的討論，XQuery 與 SQL 之間的 Collection 個數也不同，但轉換出的 SQL 表格之間會具有適當的連結關係，因此符合第二章中定義二的對應關係，屬於一個正確的轉換。

【範例 5.5】

輸入的 XQuery	→	轉換後的 SQL	→	由 SQL 再轉換回的 XQuery'
for \$t0 in /cust_info/customer		SELECT CUSTOMER.NAME,		for \$t0 in /cust_info/customer
where \$t0@ckey = "c05"		CUSTEL.TELEPHONE		where \$t0@ckey = "c05"
		FROM CUSTOMER , CUSTEL		
return \$t0/name , \$t0/tel		WHERE CUSTOMERCUSTIKEY= 'c05'		return \$t0/name, \$t0/tel
		And		
		CUSTOMERCUSTIKEY=CUSTEL.CUSTIKEY		

再來注意到輸入的查詢句若為範例 5.6 的 XQuery，其目的在找出所有葉節點路徑為/cust_info/nation/population/region_popu/comment 的內容，For 子句中/cust_info/nation/population 由附錄 B 的 CMT 表格中可發現到其為一個空元素所對應的路徑，對應的表格名稱為 REGION；而此查詢句輸出的葉節點為/cust_info/nation/population/region_popu 下的 comment，比對 VMT 表格會發現其對應的欄位為 REGION 表格的 COMMENT 欄位，因此由演算法會轉換出的 SQL 如範例中所示。在此須特別注意到由於 REGION 表格與內部節點為 1 對 2 的對應關係，也就是內部節點為空元素的/cust_info/nation/population 與可重覆元素/cust_info/nation/population/region_popu 兩個，本論文 SQL 轉換成 XQuery 的演算法，會挑選可重覆元素所對應的路徑來進行轉換，因此由轉換出的 SQL 轉換成的 XQuery'雖然會與原先所輸入的 XQuery 不同，但此雙向轉換結果亦皆符合定義一的對應關係，為一個正確的轉換。

【範例 5.6】

輸入的 XQuery	→	轉換後的 SQL	→	由 SQL 再轉換回的 XQuery'
for \$t0 in		SELECT REGION.COMMENT		for \$t0 in
/cust_info/nation/population		FROM REGION		/cust_info/nation/population/region_popu
return \$t0/region_popu/comment				return \$t0/comment

C. Structure 的對應轉換

本小節中我們將探討 Structure 對應與轉換的正確性分析，由於結構對應較複雜，也是本論文的核心所在，以下將細分不同結構特性的對應轉換，分析轉換

的正確性，其中類似先前說明的集合對應與選擇輸出的屬性轉換，在此就不做說明。

● 關聯式連結與扁平結構的轉換

範例 5.7 是針對圖 2.1 與圖 2.2 的 Order-Ship DTD 所設計的查詢句，其目的在找出顧客編號“c01”的顧客名稱及所訂購的商品處理狀態；針對結構部分，我們可由表格 3.6 SMT 中，發現 CUSTOMER 表格與 ORDER 表格間的連結關係 (R4) 是利用 CUSTKEY 來做為連結，對應於 DTD 中為可重覆元素 order 與 customer 之間利用屬性 nkey 來連結的一個扁平結構 (X_{F1})，因此轉換後的 XQuery 如範例中所示。轉換後的 XQuery 再轉換成相對應的 SQL’ 時，由於 X_{F1} 與 R4 為一對一對應，因此會與原先輸入的 SQL 相同。此轉換符合定義一的對應關係，為一個正確的轉換。

【範例 5.7】

輸入的 SQL	→	轉換後的 XQuery	→	由 XQuery 再轉換回的 SQL’
<pre>SELECT CUSTOMERNAME, ORDER.ORDERSTATUS FROM CUSTOMER,ORDER WHERE CUSTOMER.CUSTKEY= ORDER.CUSTKEY AND CUSTOMER.CUSTKEY='c01'</pre>		<pre>for \$t0 in /cust_info/customer, \$t1 in /order-ship/supplier/suppliers/part/order where \$t0@ckey="c01" and \$t0@ckey=\$t1@ckey return \$t0/name, \$t1/status</pre>		<pre>SELECT CUSTOMER.NAME, ORDER.ORDERSTATUS FROM CUSTOMER , ORDER WHERE CUSTOMER.CUSTKEY = 'c01' AND CUSTOMER.CUSTKEY = ORDER.CUSTKEY</pre>

● 關聯式連結與巢狀結構的轉換

範例 5.8 是針對圖 2.1 與圖 2.2 的 Order-Ship DTD 所設計的另個查詢句，其目的在找出供應商名稱及其所生產的產品名稱；由圖 2.1 可知道此兩表格之間的關聯是經由關係表格 PARTSUPP 分別利用 SUPPKEY 與 PARTKEY 來建立連結，由於 PART 表格與 PARTSUPP 表格的 PARTKEY 皆對應到可重覆元素 part 之下的屬性 pkey，因此連結限制是不會轉換出任何結構輸出。而我們可由表格 3.6 SMT 中，發現 SUPPLIER 表格與 PARTSUPP 表格間的連結關係 (R1)，對應於圖

2.2 DTD 中為可重覆元素 supplier 與 part 的一個巢狀結構 (X_{N1})，因此轉換後的 XQuery 如範例中所示，由於集合個數不同，此轉換結果符合定義二的對應關係。轉換後的 XQuery 再轉換成相對應的 SQL' 時，由於 supplier 與 part 的巢狀結構 X_{N1} 會對應到 R1 結構，因此會產生 SUPPLIER.SUPPKEY=PARTSUPP.SUPPKEY 此組連結限制式，並將 PARTSUPP 加入至 FROM 子句中；處理結構轉換之前，葉節點的轉換已將 PART 表格也加入至 FROM 中，所以最後檢查內部連結關係時，由表格 3.3 會發現需要增加 PART.PARTKEY=PARTSUPP.PARTKEY 至 WHERE 子句中，最後我們得到轉換後的 SQL' 會與原先輸入的 SQL 相同，此轉換可由對應表格所提供的對應資訊正確的進行轉換，符合定義二的對應關係。

【範例 5.8】

輸入的 SQL	→	轉換後的 XQuery	→	由 XQuery 再轉換回的 SQL'
SELECT SUPPLIER.NAME, PART.NAME FROM SUPPLIER, PARTSUPP, PART WHERE SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY AND PART.PARTKEY= PARTSUPP.PARTKEY		For \$t0 in /order-ship/suppliers/supplier, \$t1 in \$t0/part Return \$t0/name, \$t1/name		SELECT SUPPLIER.NAME, PART.NAME FROM SUPPLIER, PARTSUPP, PART WHERE SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY AND PART.PARTKEY= PARTSUPP.PARTKEY

● 關聯式連結與多元結構的對應轉換

此對應又可分為關聯式連結與巢狀或扁平狀結構一對多及多對一的雙向轉換關係，由於結構多元對應的轉換，是經由第四章中的 S2XSelectStructureList 演算法或 X2SSelectStructureList 演算法，選擇適當的集合輸出，並建立之間的結構關聯，選擇方法類似，只是 SQL 至 XQuery 的轉換過程中需要另外處理路徑代換的動作，在此以關聯式連結與扁平結構一對多的關係做說明。參考圖 2.3 的 Cust-Info DTD 與附錄 B 的對應表格，由 SMT 表格看出 R1 對應的 Xid 有 X_{F1} 與 X_{F2} ，為一個 1 對 2 的對應。範例 5.9 查詢句在查詢國家名稱為台灣的顧客名稱，由 SMT 表格發現到 CUSTOMER.NATIONKEY = NATION.NATIONKEY(R1)對

應的結構可能為 X_{F1} 或 X_{F2} ，由於輸出的顧客名稱在 DTD 中對應的內部節點為 customer，因此依結構轉換法則會選擇轉換出編號為 X_{F1} ：
 $/cust_info/customer@nkey = /cust_info/nation@nkey$ 的結構。經過路徑代換後，所得到的轉換結果如範例中的 XQuery 所示。而此 XQuery 再經由轉換系統轉換時，結構限制式 X_{F1} 轉換成對應的 RID 僅為 R1，所以 SQL' 會與原先輸入的 SQL 相同，此雙向轉換關係皆符合定義一的對應關係，為一個正確的轉換。

【範例 5.9】

輸入的 SQL	→	轉換後的 XQuery	→	由 XQuery 再轉換回的 SQL'
<pre>SELECT CUSTOMER.NAME FROM CUSTOMER, NATION WHERE CUSTOMER.NATIONKEY = NATION.NATIONKEY AND NATION.NAME = '台灣'</pre>		<pre>for \$t1 in /cust_info/nation, \$t0 in /cust_info/customer where \$t1/name = "台灣" And \$t0@nkey=\$t1@nkey return \$t0/name</pre>		<pre>SELECT CUSTOMER.NAME FROM NATION, CUSTOMER WHERE NATION.NAME = '台灣' And CUSTOMER.NATIONKEY= NATION.NATIONKEY</pre>

● 巢狀結構與關係表格之間的對應轉換

由於關係表格代表實體表格之間的連結關係，在建立對應表格 CMT 時，若實體表格在 DTD 中對應到巢狀結構，我們將令關係表格對應的可重覆元素為實體表格對應到最深層的元素，而若巢狀結構為關係表格與關係表格之間的關係，則結構轉換有可能更加的複雜，這兩種情況將由下述的範例做說明。

(1) 實體表格與關係表格

範例 5.10-1 是針對圖 2.1 與圖 2.2 的 Order-Ship DTD 所設計的查詢句，其目的在找出實體表格 SUPPLIER 中供應商編號為 's01' 的供應商名稱及關係表格 LINEITEM 中顧客訂購的訂購單編號。由圖 2.1 可知道此兩表格間的連結是經由 SUPPKEY 來建立連結，針對結構部分，我們可由表格 3.6 SMT 中，發現此連結關係 (R2)，對應於圖 2.2 DTD 中為可重覆元素 supplier 與 order 的一個巢狀結構 (X_{N2})，所以可重覆元素 order 的路徑表示法在此會被代換成 $\$t1$ in $\$t0/part/order$ ，轉換後的 XQuery 如範例中所示。轉換後的 XQuery 再轉換成相對應的 SQL' 時，由於 supplier 與 order 的巢狀結構 X_{N2} 只會對應到 R2 結構，因此會產生

SUPPLIER.SUPPKEY=LINEITEM.SUPPKEY 此組連結限制式，轉換的 SQL' 會與原先輸入的 SQL 相同，此雙向轉換關係皆符合定義一的對應關係，為一個正確的轉換。

【範例 5.10-1】

輸入的 SQL	→	轉換後的 XQuery	→	由 XQuery 再轉換回的 SQL'
SELECT SUPPLIER.NAME, LINEITEM.LINENUMBER FROM SUPPLIER, LINEITEM WHERE SUPPLIER.SUPPKEY = LINEITEM.SUPPKEY AND SUPPLIER.SUPPKEY = "s01"		for \$t0 in /order-ship/suppliers/supplier, \$t1 in \$t0/part/order where \$t0@skey = 's01' return \$t0/name, \$t1/linenum		SELECT SUPPLIER.NAME , LINEITEM.LINENUMBER FROM SUPPLIER , LINEITEM WHERE SUPPLIER.SUPPKEY = "s01" AND SUPPLIER.SUPPKEY= LINEITEM.SUPPKEY

範例 5.10-2 則是在找出 ORDER 表格中，訂單編號為 o01 的總金額，以及 LINEITEM 表格中記錄的運送方式，此兩個表格之間是以 ORDERKEY 做連結。由於 CMT 中，ORDER 與 LINEITEM 表格皆對應到可重覆元素 order，且兩表格的 ORDERKEY 由 VMT 中可發現皆對應到可重覆元素 order 的子節點 okey，因此轉換結果如範例中所示。轉換後的 XQuery 再轉換成相對應的 SQL' 時，因為葉節點 price 與 shipmode 分別屬於 ORDER 與 LINEITEM 表格中的欄位，轉換出此兩個表格後，最後驗證模組會將此兩表格中的內部連結限制式 ORDER.ORDERKEY 與 LINEITEM.ORDERKEY 加入至 WHERE 子句中。由於集合的個數不同，此雙向轉換為符合定義二的轉換關係。

【範例 5.10-2】

輸入的 SQL	→	轉換後的 XQuery	→	由 XQuery 再轉換回的 SQL'
SELECT ORDER.TOTALPRICE, LINEITEM.SHIPMODE FROM ORDER, LINEITEM WHERE ORDER.ORDERKEY = LINEITEM.ORDERKEY AND ORDER.ORDERKEY = "o01"		for \$t0 in /order-ship/suppliers/supplier/part/order where \$t0@okey = 'o01' return \$t0/price, \$t0/shipmode		SELECT ORDER.TOTALPRICE, LINEITEM.SHIPMODE FROM ORDER, LINEITEM WHERE ORDER.ORDERKEY = "o01" AND ORDER.ORDERKEY = LINEITEM.ORDERKEY

(2)關係表格與關係表格

範例 5.11 同樣是針對圖 2.1 與圖 2.2 的 Order-Ship DTD 所設計的查詢句，其目的在找出 PARTSUPP 表格中，零件編號為 'p01' 的供應商編號以及 LINEITEM 表格中顧客訂購的訂購單編號。查詢句中兩表格間的連結是經由 PARTKEY 來建立連結，針對結構部分，我們可由表格 3.6 SMT 中，發現此連結關係 (RR2)，對應於圖 2.2 DTD 中為可重覆元素 part 與 order 的一個巢狀結構 (X_{N3})，此 SQL 轉換後的 XQuery 如範例中所示。而轉換後的 XQuery 再轉換成相對應的 SQL' 時，由於演算法的轉換的結構選擇，巢狀結構 X_{N3} 不會轉換出 RR2 而是分析 For 子句可重覆元素 supplier、part 與 order 之間的巢狀結構的關係，本論文的做法會將集合之間的關聯性皆建立，因此會取得表格 3.5 PMT 表格中所符合的結構編號 (X_{N1} 、 X_{N2} 、 X_{N3})，而此三個 XID 編號於表格 3.6 中，演算法會選擇轉換出其對應的 RID 編號 (R1、R2、R3) 之內容，因此轉換結果 SQL' 會與原先輸入的 SQL 不同，但此關係符合定義二的對應關係，集合之間會具有適當的連結，為一個正確的轉換。

【範例 5.11】

輸入的 SQL	→	轉換後的 XQuery	→	由 XQuery 再轉換回的 SQL'
SELECT PARTSUPP.SUPPKEY, PARTSUPP.AVAILQTY, LINEITEM.LINENUMBER FROM PARTSUPP, LINEITEM WHERE PARTSUPP.PARTKEY = LINEITEM.PARTKEY AND PARTSUPP.PARTKEY = 'p01'		for \$s2 in /order-ship/suppliers/supplier \$t0 in \$s2/part, \$t1 in \$t0/order where \$t0@pkey="p01" return \$s2@skey, \$t0/availqty, \$t1@linenum		SELECT SUPPLIER.SUPPKEY, PARTSUPP.AVAILQTY, LINEITEM.LINENUMBER FROM SUPPLIER, PART, PARTSUPP, LINEITEM WHERE PART.PARTKEY='p01' And SUPPLIER.SUPPKEY= PARTSUPP.SUPPKEY And SUPPLIER.SUPPKEY= LINEITEM.SUPPKEY And PART.PARTKEY= LINEITEM.PARTKEY And PART.PARTKEY= PARTSUPP.PARTKEY

- 多組關聯式連結與多元結構的對應轉換

由於前述結構對應僅在說明一組 RID 與 XID 為一對多或是多對一的對應關係，本實驗討論同時處理多組的情況，在此以圖 5.A 的關聯式表格與圖 5.B 的 DTD 做說明。關聯式表格中有 A、B、C、D 四個表格，此四個表格各分別對應到 DTD 中 3 個可重覆元素，對應關係為 A 表格會對應到 a1~a3、B 表格會對應到 b1~b3、C 表格會對應到 c1~c3，而 D 表格會對應到 DTD 中的 d1~d3。由於關聯式表格 A 與 B、B 與 C、C 與 D 表格之間有連結關係，因此對應於 DTD 中的結構關係更加的複雜。我們利用不同的查詢句，驗證結構選擇與雙向轉換的正確性。

A (AKEY, AE1, AE2, AE3)
 B (BKEY , AKEY , BE1, BE2, BE3)
 C (CKEY , BKEY , CE1, CE2, CE3)
 D (DKEY , CKEY , DE1, DE2, DE3)

圖 5.A： 關聯式表格架構

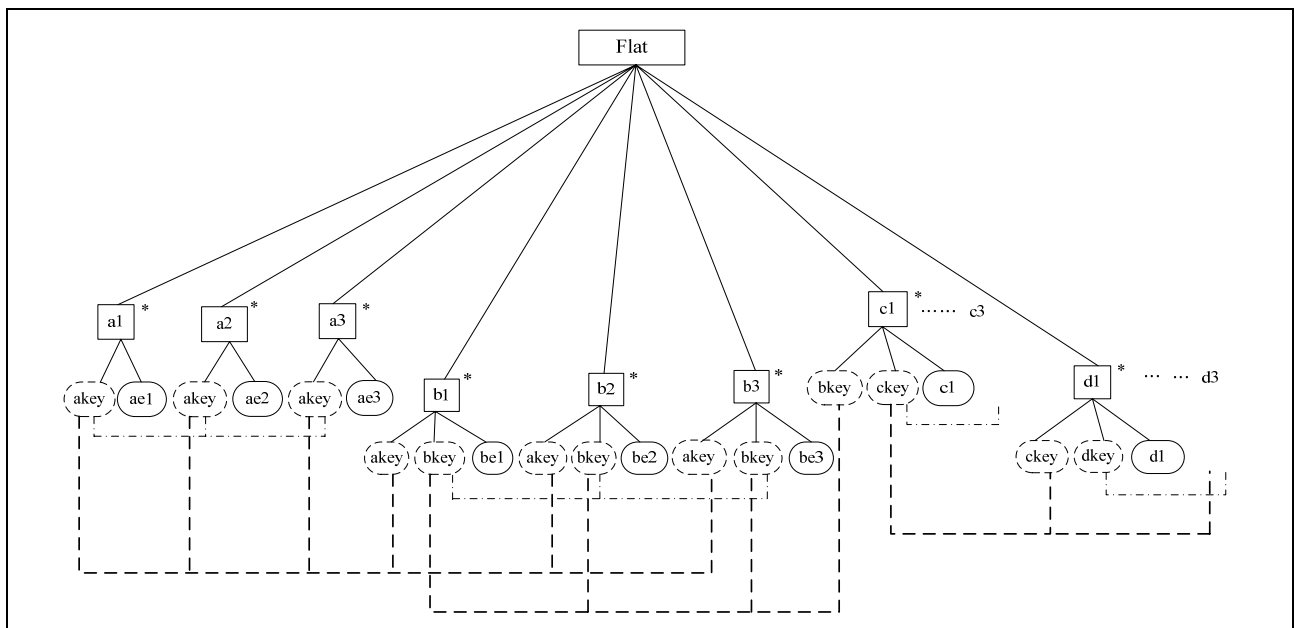


圖 5.B： 對應 Flat 結構的 DTD

範例 5.12-1 的 SQL 查詢句與範例 5.12-2 中的 Q1 查詢句相同，SQL 查詢句選擇輸出欄位 A.AE1, C.CE2, D.DE3，資料的來源為 A、B、C、D 四個表格，且 A 與 B、B 與 C、C 與 D 表格之間有連結關係，此時，由於選擇輸出的欄位 A.AE1、C.CE2、D.DE3 對應到 DTD 中的內容節點分別在可重覆元素 a1、c2、e3 之下，因此第四章圖 4.11 PMT_RefPreCheck 演算法，會將結構對應為可重覆元素 a1、c2、e3 的 ref_flag 設為 TRUE，如範例 5.12-2 查詢句 Q1 轉換結構選擇的示意圖的實心圓所示；而轉換出的對應的結構選擇，如粗黑實線所示。因此會產生 \$t0@akey=\$t3@akey、\$t3@bkey=\$t7@bkey、\$t7@ckey=\$t11@ckey 三組扁平的結構連結。我們可由轉換結果發現，此雙向轉換結果皆符合定義一的對應關係，為一個正確的轉換。

【範例 5.12-1】

輸入的 SQL	→	轉換後的 XQuery	→	由 XQuery 再轉換回的 SQL
SELECT A.AE1, C.CE2, D.DE3 FROM A, B, C, D WHERE A.AKEY=B.AKEY AND B.BKEY=C.BKEY AND C.CKEY=D.CKEY		For \$t0 in /f/a1, \$t3 in /f/b1, \$t7 in /f/c2, \$t11 in /f/d3 Where \$t0@akey=\$t3@akey And \$t3@bkey=\$t7@bkey And \$t7@ckey=\$t11@ckey Return \$t0/ae1, \$t7/ce2, \$t11/de3		SELECT A.AE1, C.CE2, D.DE3 FROM A, B, C, D WHERE A.AKEY=B.AKEY And B.BKEY=C.BKEY And C.CKEY=D.CKEY

【範例 5.12-2】

編號	輸入的 SQL	轉換結構選擇的示意圖	轉換出來的 XQuery
Q1	SELECT A.AE1, C.CE2, D.DE3 FROM A, B, C, D WHERE A.AKEY=B.AKEY AND B.BKEY=C.BKEY AND C.CKEY=D.CKEY		For \$t0 in /flat/a1, \$t3 in /flat/b1, \$t7 in /flat/c2, \$t11 in /flat/d3 Where \$t0@akey=\$t3@akey And \$t3@bkey=\$t7@bkey And \$t7@ckey=\$t11@ckey Return \$t0/ae1, \$t7/ce2, \$t11/de3

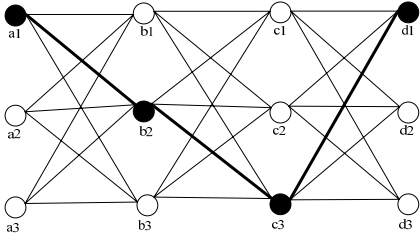
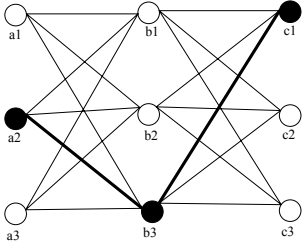
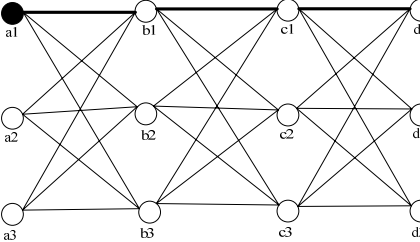
Q2	SELECT A.AE1, B.BE2, C.CE3, D.DE1 FROM A, B, C, D WHERE A.AKEY=B.AKEY AND B.BKEY=C.BKEY AND C.CKEY=D.CKEY		For \$t0 in /flat/a1, \$t4 in /flat/b2, \$t8 in /flat/c3, \$t9 in /flat/d1 Where \$t0@akey=\$t4@akey And \$t4@bkey=\$t8@bkey And \$t8@ckey=\$t9@ckey Return \$t0/ae1, \$t4/be2, \$t8/ce3, \$t9/de1
Q3	SELECT A.AE2, B.BE3, C.CE1 FROM A, B, C WHERE A.AKEY=B.AKEY AND B.BKEY=C.BKEY		For \$t0 in /flat/a1, \$t5 in /flat/b3, \$t8 in /flat/c3 Where \$t0@akey=\$t5@akey And \$t5@bkey=\$t8@bkey Return \$t0/ae1, \$t5/be3, \$t8/ce3
Q4	SELECT A.AE1 FROM A, B, C, D WHERE A.AKEY=B.AKEY AND B.BKEY=C.BKEY AND C.CKEY=D.CKEY		For \$t0 in /flat/a1, \$t3 in /flat/b1, \$t6 in /flat/c1, \$t9 in /flat/d1 Where \$t0@akey=\$t3@akey And \$t3@bkey=\$t6@bkey And \$t6@ckey=\$t9@ckey Return \$t0/ae1

圖 5.C：結構選擇最佳化 範例與示意圖

進一步解釋範例 5.12-2 中的四個查詢句，在 Q2 中，被標示的集合為 a1、b2、c3、d1，符合演算法中的 (case1) 會選擇雙邊都有被標記到的結構輸出，因此 WHERE 子句中的三個連結限制式，依序會轉換出 (a1,b2)、(b2,c3)、(c3,d1) 此三個對應的結構。而 Q3 查詢句來源的表格只有 A、B、C 三個表格，如同 Q2 的說明，因此轉換出的結構如示意圖所示為 (a2,b3)、(b3,c1)。注意到 Q1 查詢句，資料表格來源為 A、B、C、D 四個表格，被標記的集合僅有 a1、c2、d3，因此 WHERE 子句中的連結限制式 A.AKEY = B.AKEY 在經過 S2XSelectStructureList 演算法處理時，符合 (case2) 僅有左邊的 a1 集合有被標記，會選擇 a1 所對應到

的三個結構中的第一組 (a1,b1)，而連結限制式 B.BKEY = C.BKEY 則是符合演算法中 (case3) 僅有右邊的 c2 集合有被標記，會由 c2 選擇所對應到的三個結構中的第一組 (b1,c2)。另外，Q4 查詢句中，資料表格來源為 A、B、C、D 四個表格，但是被標記的集合僅有 a1，WHERE 子句中的連結限制式 A.AKEY = B.AKEY 在經過 S2XSelectStructureList 演算法處理時，會選擇 a1 所對應到的三個結構中的第一組 (a1,b1)，而另外兩個連結條件式由於相對應的集合皆沒有被標記，符合演算法中的 (case4) 情形，會挑選關聯式連結對應到的第一組結構，因此會選擇(b1,c1)、(c1,d1)結構輸出。

綜合以上說明，結構選擇的演算法會先選擇兩個集合都有被標記到的結構輸出，若結構中僅有單一個集合有被標記，或是都沒有被標記，演算法都會選擇第一組對應，而不需要產生內部連結 (InternalJoin)，因此所轉換出對應的結構數會是最精簡的。經由以上各小節分析，本轉換系統所轉換的查詢句皆會符合第二章中的正確性定義一與定義二。另外，綜合本章的範例，在此將輸出的查詢句兩次轉換前後查詢句不同的原因，分析歸納如性質 1 中所述。

【性質 1】 轉換系統轉換出的 Query，再經由轉換系統轉換一次，結果可能與原先輸入的查詢句不同

產生此情況的情形有以下 3 種可能：

1. 輸入的查詢句中有一 Collection 為多對一對應時，如範例 5.3。
2. 輸入的 SQL 查詢句，WHERE 子句中，為關係表格與關係表格之間的連結限制式 (結構編號為 RRi)，也就是結構對應為多對一時，如範例 5.11。
3. 輸入的 XQuery，其 For 子句中有一路徑對應到空元素，如範例 5.6。

5.2 查詢結果分析

針對第二章中定義三所述，輸出資料答案相同但個數不同的情形，以範例 5.13 來舉例說明。由於關聯式資料表格間的連結關係，相同的資料可能會被重覆輸出。參考範例中是要找出顧客編號為 c01 的顧客及國家名稱，利用 NATIONKEY 來連結 CUSTOMER 與 NATION 表格，由 CUSTOMER 表格中，顧客編號為 c01 的顧客，其 NATIONKEY 為 n01，但是於 NATION 表格中，NATIONKEY 為 Composite Key 中的其中一個鍵值，值為 n01 的有 3 筆資料，因此利用 CUSTOMER.NATIONKEY = NATION.NATIONKEY 做為連結，此 SQL 範例會輸出 3 筆相同的結果。參考圖 2.3 的 DTD，實際的 XML 資料中，顧客編號 ckey 為 c01 的顧客，其 nkey 已知是 n01，此時 Where 限制式中 \$t0@nkey=\$t1@nkey，將會以 nkey = “n01”來做為連結，因為 XML 格式的關係，可重覆元素 nation 下的相同的 nkey 與 name 僅有一筆資料列，因此 XQuery 僅有一筆結果輸出。此情形已在第二章中描述可用 distinct 指令，使得重覆的資訊不輸出。

【範例 5.13】 重覆範例 5.9 的查詢句

輸入的 SQL

```
SELECT  CUSTOMER.NAME ,
        NATION.NAME
FROM    CUSTOMER , NATION
WHERE  CUSTOMER.NATIONKEY =
        NATION.NATIONKEY
AND
CUSTOMER.CUSTKEY = 'c01'
```

→

轉換後的 XQuery

```
for $t1 in /cust_info/nation,
   $t0 in /cust_info/customer
where $t0@ckey="C01"
   And
      $t0@nkey=$t1@nkey
return $t0/name, $t1/name
```



NAME	NAME
邦鑑	台灣
邦鑑	台灣
邦鑑	台灣





```
<name>邦鑑</name>
<name>台灣</name>
```

傳統的資料庫對應轉換，是以個別的查詢句至資料庫中查詢，比較輸出結果的異同，若輸出結果相同，則視為是正確的查詢句轉換。大部分的查詢句經由本論文所設計的轉換系統，分別至該類型資料庫所查詢出的資料，皆會相同；但由

於 XML 可具有階層式結構的關係，因此，當查詢句的語意恰巧沒符合階層結構限制的時候，會有查詢結果不同的問題，如範例 5.14 所示。

【範例 5.14】 重覆範例 5.11 的查詢句

輸入的 SQL	→	轉換後的 XQuery															
SELECT PARTSUPP.SUPPKEY, PARTSUPP.PRODUCTION, LINEITEM.LINENUMBER FROM PARTSUPP, LINEITEM WHERE PARTSUPP.PARTKEY = LINEITEM.PARTKEY AND PARTSUPP.PARTKEY = 'p01'		for \$t2 in /order-ship/suppliers/supplier \$t0 in \$t2/part, \$t1 in \$t0/order where \$t0@pkey="p01" return \$t2@skey , \$t0/pro , \$t1/linenum															
																	
<table><thead><tr><th>SUPPKEY</th><th>PRODUCTION</th><th>LINENUMBER</th></tr></thead><tbody><tr><td>s01</td><td>10000</td><td>tw-l-03</td></tr><tr><td>s02</td><td>11000</td><td>tw-l-03</td></tr><tr><td>s08</td><td>12000</td><td>tw-l-03</td></tr><tr><td>s05</td><td>13000</td><td>tw-l-03</td></tr></tbody></table>	SUPPKEY	PRODUCTION	LINENUMBER	s01	10000	tw-l-03	s02	11000	tw-l-03	s08	12000	tw-l-03	s05	13000	tw-l-03		<skey>s01</skey> <pro>10000</pro> <linenum>tw-l-03</linenum>
SUPPKEY	PRODUCTION	LINENUMBER															
s01	10000	tw-l-03															
s02	11000	tw-l-03															
s08	12000	tw-l-03															
s05	13000	tw-l-03															

範例 5.14 的 SQL 選擇的表格來源為關係表格 PARTSUPP 與 LINEITEM 表格，此兩表格間的關聯，應以複合鍵 (SUPPKEY , PARTKEY) 作為連結，但在此範例中僅限定兩表格的 PARTKEY 必須相同，且 PARTKEY = 'p01'。原始資料中，PARTKEY = 'p01' 的資料在 LINEITEM 中原本就只有一筆，但 PARTKEY = 'p01' 在 PARTSUPP 表格中的資料有四筆（請參考附錄 C），因此 PARTSUPP.PARTKEY=LINEITEM.PARTKEY 的連結會使查詢結果共產生了四筆資料列；比較轉換出 XQuery 對 XML 資料所查詢出來的結果卻只有一筆；探究原因是因為 XML 階層結構的關係，For 子句中有限定 part 元素下必須要有 order 元素，也就是該零件有人購買，由實際對應的 XML 資料得知，有人購買 PARTKEY = 'p01' 的零件，是廠商編號 s01 所生產的，而廠商編號 s02、s08、s05 生產的該零件並無人訂購，以致於 SQL 與轉換出的 XQuery 查詢句輸出的結果不同，這是因為 XML 資料可以具有階層式的結構所影響，我們將此特性，以如下之性質 2 表示。

【性質 2】 輸入的查詢句與轉換後的查詢句，分別至資料庫中查詢所得的資料可能不同

會發生在 SQL 的查詢句恰巧沒符合 XML 階層結構的特性時。

綜合以上實驗結果分析與第四章中不同的範例，除了查詢句中的屬性、集合與結構對應皆為 1 對 1 的對應關係之外，我們將符合不同特性的類別及相關的範例區分歸納如後所述，其中(s2x)代表該範例中 SQL 轉換至 XQuery 的雙向轉換，(x2s)代表該範例中 XQuery 轉換至 SQL 的雙向轉換，無特別標示代表兩個方向的條件皆符合。表 5.1 為每個類別符合的定義與特性歸納。

【類別 1-1】 查詢句中的集合為 1 對多對應關係，且選擇相同集合的屬性分別在對應的多個集合之下，轉換後的集合會變多。

如範例：5.4(s2x)、5.5(x2s)、5.10-2(s2x)

【類別 1-2】 查詢句中的集合為多對 1 對應關係，且選擇不同集合的屬性分別在對應的相同一個集合之下，轉換後的集合會變少。

如範例：5.4(x2s)、5.5(s2x)、5.10-2(s2x)

【類別 2-1】 輸入的屬性為多元對應關係，且非 VMT 中的第一筆對應記錄。

反向轉換時，會選擇 VMT 中的第一筆對應，因此輸入的查詢句與再經過反向轉換的查詢句可能不同。

如範例：5.2(s2x)

【類別 2-2】 輸入的屬性為多元對應關係，且為 VMT 中的第一筆對應記錄。因此輸入的查詢句與再經過反向轉換的查詢句會相同。

如範例：5.1-1、5.1-2、5.2(x2s)

【類別 3】查詢句中的結構為多元對應關係，但是雙向轉換結構的選擇，會是同組結構對應，因此反向轉換結果會與原先輸入的查詢句相同。

如範例：4.1、4.2、4.19-1、4.19-1、5.8、5.11(x2s)、5.12-1、5.12-2

【類別 4】查詢句中的屬性、集合與結構對應其中有多元的對應關係，但是皆為對應表格中的第一筆資訊，因此反向轉換結果會與原先輸入的查詢句相同。

如範例：5.3(x2s)、5.6(s2x)、5.7、5.9、5.10-1

【類別 5】輸入之 XQuery 的 For 子句中，含有 Dummy Node Path，反向轉換時，會選擇轉換出 Repeatable Node Path，因此會與原先輸入的查詢句不同。

如範例：5.6(x2s)

【類別 6】輸入之 SQL 的 From 子句中為關係表格，且選擇的屬性，是該組巢狀結構中，關係表格所對應的可重覆元素，更上層可重覆元素的子元素，轉換過程會轉換出此兩可重覆元素路徑的結構關係，且集合的個數會增加。

如範例：4.11(s2x)、4.13(s2x)、5.3(s2x)

【類別 7】兩個關係表格，對應於 DTD 中若是巢狀結構，由於此巢狀結構會是相關實體表格之間的關係，當兩個關係表格之間的關聯是以複合鍵作為連結時，若僅以其中的一個鍵值來限定相關條件的話，會不完全符合巢狀結構的意含，因此查詢出來的結果可能會不同。

如範例：4.23、5.11(s2x)

定義 1: 集合個數相同 定義 2: 集合個數不同但具適當連結 定義 3: Distinct 後相同 性質 1: Query 與 Query' 不相同 性質 2: Output Data 不同

區分	定義一	定義二	定義三	性質一	性質二
類別 1-1		○	○		
類別 1-2		○	○		
類別 2-1	○		○	○	
類別 2-2	○		○		
類別 3		○	○		
類別 4	○		○		
類別 5	○		○	○	
類別 6		○	○	○	
類別 7		○		○	○
表 5.1：性質歸納					

5.3 轉換效率評估

轉換所需的時間主要是由(1)Value 的處理、(2)Collection 處理、(3)Structure 的處理、(4)Valid 驗證模組及(5)Compose 模組輸出轉換後的查詢句五大部分所組成。在本節中，我們將以不同的控制因子，來探討其對轉換的效率影響，針對不同的區分，我們設計適當的 Schema 來實驗，列出主要的影響因子來探討。以下實驗圖表中的時間為轉換執行一千次的時間總和。詳細的 Schema 設計與查詢句內容，請參照附錄 D。

5.3.1 控制相同的查詢句輸入，改變 Mapping 的對應個數

● 控制 Collection 及 Structure 一對多的 Mapping 個數

針對 Structure 的處理效率分析，我們設計同一組關聯式表格，對應到的 DTD 分別為 Flat 與 Nested 結構，比較 Structure 一對多對應與轉換效率的影響。首先參考附錄 D(I) 中 Flat 結構的 DTD，例如表格 A 會對應到元素 a1，表格 B 會對應到元素 b1 與 b2，因此表格 A 與表格 B 之間的連結限制式 A.AKEY=B.AKEY 會對應到 DTD 中的連結限制式 /flat/a1@akey=/flat/b1@akey 與 /flat/a1@akey=/flat/b2@akey，此為一個 1 對 2 的對應關係；依此類推關聯式表格

A 與關聯式表格 E、B、C、D 之間連結限制式，對應於 Flat 與 Nested 的 DTD 結構分別為 1 對 1 至 1 對 4。

由實驗結果圖 5.1(a)與圖 5.1(b)看出主要影響轉換時間最多的 Structure 模組發現，結構一對多的個數，並不至於造成轉換的時間大幅差異，主要是因為雖為一對多的對應關係，但是在此實驗中，皆僅會輸出一個結構對應。而轉換成 Flat 結構所需的時間比 Nested 結構平均多出 0.67 秒，分析原因主要是因為在處理 Nested 結構時，會根據 PMT 中的巢狀結構對應處理 ForSet 中的資訊，進行變數代換；而處理 Flat 結構時，除了亦須進行變數代換的動作，且需另外加入至 S2XWhereSet 中，因此所需時間會些微的多一些。

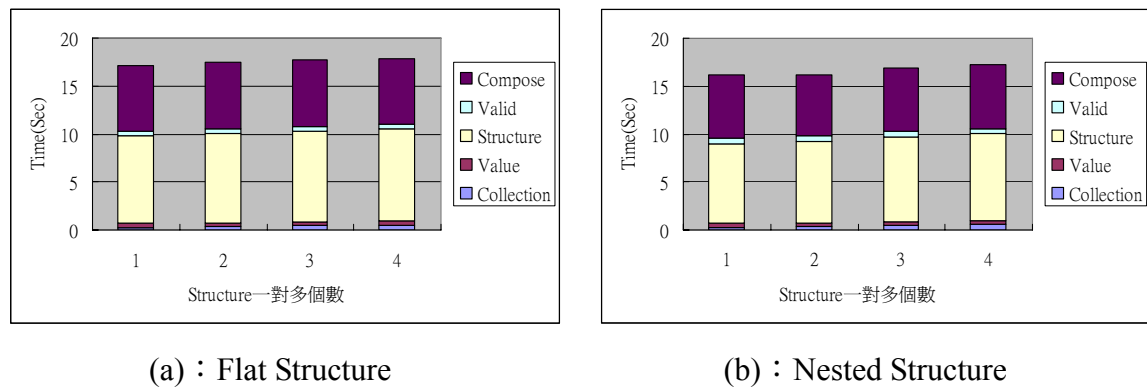


圖 5.1：(S2X) Structure 一對多個數與轉換時間的影響

● 控制 Value 一對多的 Mapping 個數

Value 一對多的測試，我們固定查詢句中選擇的表格與欄位僅有一個，參考附錄 D(II)，可發現到關聯式表格 A、B、C、D 中的鍵值 AKEY、BKEY、CKEY、DKEY，其對應於 DTD 中的 akey、bkey、ckey、dkey 分別為 1 對 1、1 對 2、1 對 3 與 1 對 4 的對應關係，由於在演算法針對 Value 一對多的處理時，皆是取得 VMT 表格中的第 1 筆對應，可由圖 5.2 發現到轉換所需的時間幾乎相同，一對多的對應關係對整體轉換所需時間無特別之影響，處理時間的些微差距是於 VMT 比對時，搜尋到較後面的資料列會些微的增加一點時間。

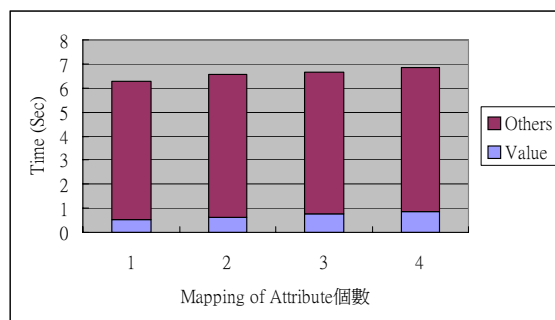


圖 5.2：Mapping of Attribute 個數與轉換時間的影響

我們可由以上兩個實驗發現，當結構或屬性為一對多對應時，若選擇的輸出僅為其中一組，則對應表格中的對應關係與轉換效率的影響很小。

5.3.2 針對個別的 Schema Mapping，控制輸出(改變 Input)的個數

- **Collection 對應為 1 對 1 情況下，控制輸出的 Attribute 個數**

在本小節中，我們設計實驗，來探討 SQL 轉換至 XQuery 時，所選擇輸入的 Attribute 個數對轉換的效率影響，結果如圖 5.3 所述。對應的 Schema 設計與查詢句內容，請參照附錄 D(III)。由實驗中發現，選擇的資料來源為同一個 Collection 時，轉換的時間會隨所選擇的 Attribute 個數呈線性遞增，多增加一個 Attribute 大約增加 0.5 秒的處理時間。

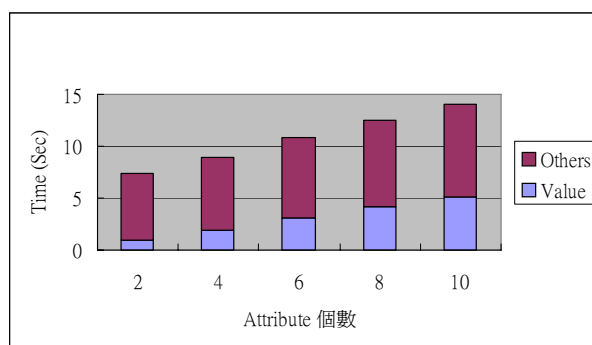


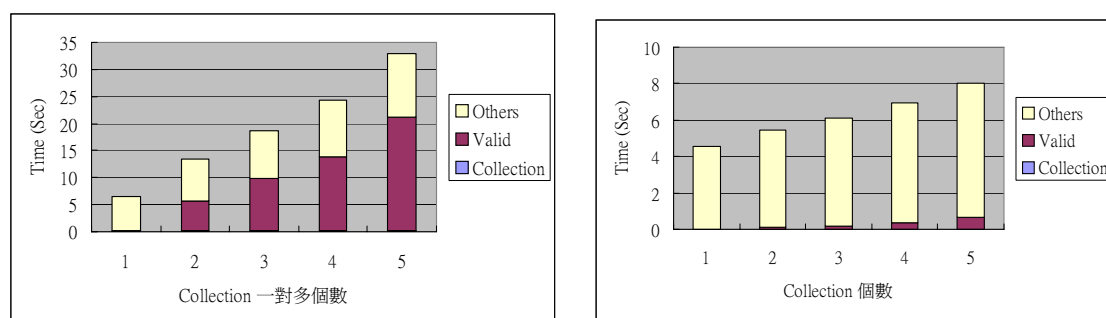
圖 5.3：輸出 Attribute 個數與轉換時間的影響

- **Collection 對應為一對多的情況下，控制輸出的 Collection 個數**

參考附錄 D(IV-1)中，關聯式表格 A 對應到 DTD 中的內部節點 a1 至 a5，我

們利用 SQL 查詢句控制參考到關聯式表格與內部節點的對應個數關係，以比較對應個數的增長與 SQL 至 XQuery 轉換效率的影響，結果如圖 5.4(a)所示，當輸出的 Collection 為 2 時，產生的內部連結有 1 組；若為 3 時，產生的內部連結有 3 組；若為 4 時，產生的內部連結有 6 組；也就是若有 n 個 Collection，則產生 $C_{(n,2)}$ 組，所以時間增長較快。另外參考附錄 D(IV-2)，我們比較內部節點與表格的對應關係，DTD 中內部節點 a 對應關聯式表格 A1 至 A5，在此利用 XQuery 查詢句控制參考到內部節點與關聯式表格的產生個數關係，結果如圖 5.4(b)所示。

比較圖 5.4(a)與圖 5.4(b)可發現針對 Collection 一對多時的處理，主要的影響因子會是於最後 Valid 時檢查是否產生 Internal Join。另外於 Valid 處理時，SQL 至 XQuery 的轉換較耗時，分析原因差別在於轉換成 XQuery 時，取出 IJT 表格



(a) : SQL to XQuery 轉換

(b) : XQuery to SQL 轉換

圖 5.4 : Collection 對應個數與轉換時間的影響

對應的 Internal Join 資訊，需要額外將路徑代換成符合 XQuery 的表示法。

● Collection 及 Structure 一對多的情況下，控制結構的輸出個數

此處的實驗與 5.3.1 小節 (控制 Collection 及 Structure 一對多的 Mapping 個數) 不同處在於先前只會輸出一組結構對應中的其中一個結構，而此處是在比較一組 1 對 4 的結構對應，輸出的結構數與轉換時間的關係。參考附錄 D(V-1)，關聯式表格 A 對應元素 a，而關聯式表格 B 對應元素 b1、b2、b3、b4，因此表格 A 與 B 的連結限制式對應 Flat 與 Nested 的 DTD，結構對應皆為一對 4。我們比較 SQL 轉 XQuery 時，輸出的結構數增長對轉換效率的影響，針對 Flat 與 Nested

結構的對應實驗結果如圖 5.5(a)與 5.5(b)所示。挑選主要影響轉換時間的模組來比較，我們觀察到於 Structure 處理時，Flat 結構所花的處理時間稍多，原因是在處理 Nested 結構時，會取得 PMT 中的巢狀結構對應與 ForSet 中的資訊，將較長的重覆元素路徑，結合分配給較短可重覆元素路徑的變數名稱，代換為巢狀的路徑表示法；而處理 Flat 結構時，由 PMT 中取得的兩個葉節點路徑，皆須結合其最接近的可重覆元素於 ForSet 中所分配的變數名稱，進行變數代換的動作，且需另外加入至 S2XWhereSet 中，因此所需時間會些微的多一些。在本實驗的 Flat 結構中，每增加輸出一個結構對應，結構處理時間平均線性遞增 3.904 秒的時間，而 Nested 結構每增加輸出一個結構對應，結構處理時間平均遞增 2.678 秒。

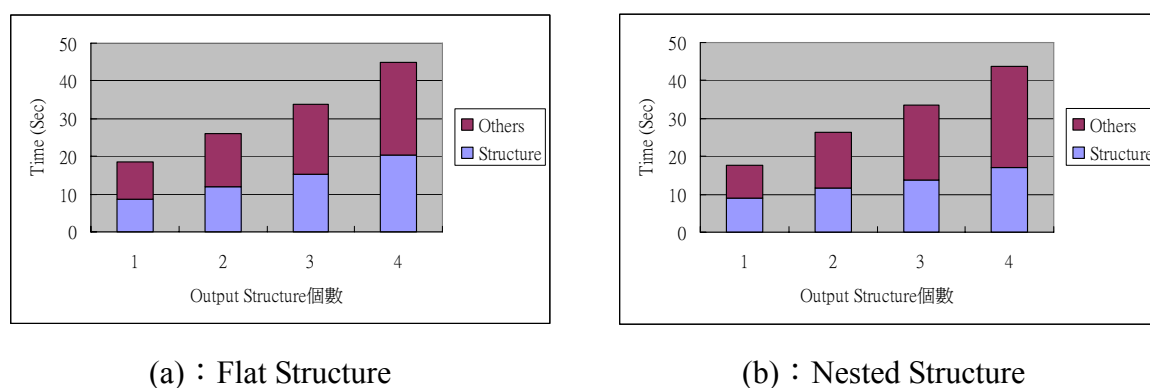
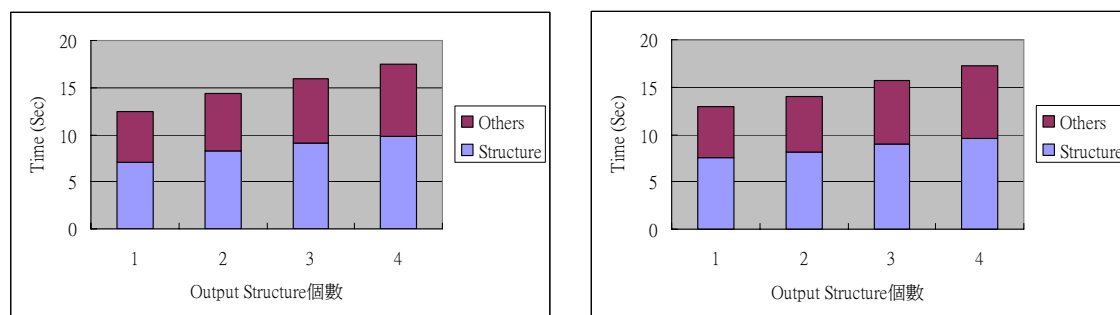


圖 5.5：(SQL to XQuery) Structure 輸出個數與轉換時間的影響

另外參考附錄 D(V-2)，與前述 Schema 對應相反，是由 XML DTD 對應到關聯式資料表格，元素 a 會對應到關聯式表格的 A1，而元素 b 會對應到關聯式表格 B1、B2、B3、B4，因此，元素 a 與元素 b 之間的結構關聯，對應到關聯式資料表格的連結限制式為 1 對 4 的關係，我們在此比較 Flat 結構與 Nested 結構的 DTD，XQuery 轉換成對應的 SQL 時，輸出的結構數增長對轉換效率的影響。Flat 結構與 Nested 結構對應的實驗結果如圖 5.6(a)與 5.6(b)所示。我們由實驗發現到 Flat 結構或 Nested 結構的 XQuery 查詢句轉換對應到相同的 SQL 時，所花的時間幾乎是相當的，這是為關聯式資料皆為扁平結構的關係，且沒有 XQuery 需要路徑代換的問題，由實驗發現平均多輸出一組對應的結構，結構處理時間平均會線性遞增 0.8 秒。

再觀察圖 5.5(a)與圖 5.6(a)，雖然 DTD 皆為 Flat 結構，同先前 Collection 一對多的實驗結論，我們可發現到 SQL 轉 XQuery 比 XQuery 轉 SQL 還要費時，因為 SQL 轉 XQuery 結構處理時需要另外做路徑代換的處理，我們比較 Output Structure 的個數同為 4 筆時，SQL 轉 XQuery 比 XQuery 轉 SQL 大約多了 31 秒。而圖 5.5(b)與圖 5.6(b)的 DTD 同為 Nested 結構，我們比較 Output Structure 的個數同為 4 筆時，SQL 轉 XQuery 比 XQuery 轉 SQL 大約多了 26 秒。

由以上三個實驗的結果，我們可發現結構的處理，對轉換效率的影響最大，除了 Collection 一對多，產生的 IJT 內容非線性遞增外，Value 與輸出 Structure 個數的增加，對轉換時間的影響皆為線性遞增。



(a) : Flat Structure

(b) : Nested Structure

圖 5.6 : (XQuery to SQL) Structure 輸出個數與轉換時間的影響

5.3.3 改變 Collection 與 Join 的個數

本處的實驗 Schema 與 Query 對應請參考附錄附錄 D(VI)，此處的實驗目的，在比較關聯式連結的個數，對應 DTD 的結構全為 Flat 或 Nested 時，Join 個數的增長與轉換時間之影響。我們設計 15 個表格 A、B、...至 O，其對應到 DTD 中的元素分別為 a、b、...至 o；前後兩表格間皆有關聯，如 A join B、B join C、...、N join O，且此結構關聯對應於 DTD 中時，分別為元素 a 與 b、b 與 c、...、n 與 o 之間的結構對應。針對 DTD 為 Flat 結構與 Nested 結構，我們控制關聯式連結的個數，來測試 SQL 至 XQuery 轉換時間的影響。實驗結果分別如圖 5.7(a)與 5.7(b)所示。我們由本實驗結果發現，Join 個數的成長主要會影響 Structure 的處理時

間，於 Flat 結構時，增加 3 個 join 個數，平均轉換時間會增加 38 秒，而於 Nested 結構時，增加 3 個 join 個數，平均轉換時間會增加 35 秒。

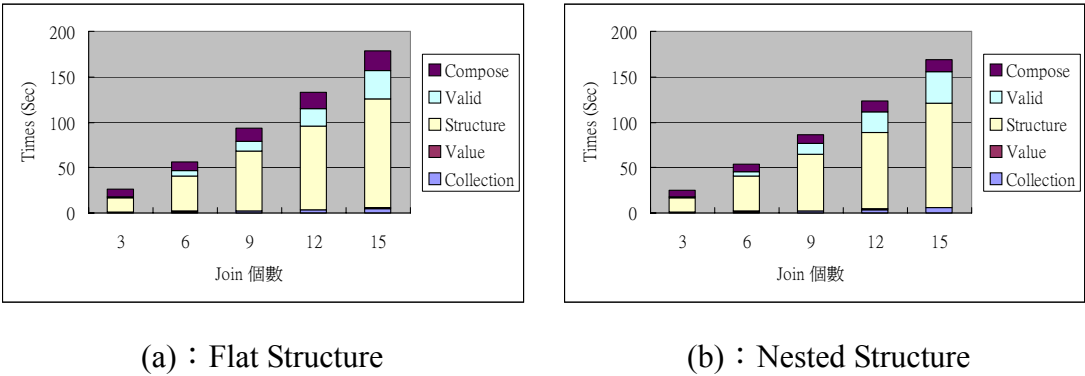


圖 5.7：Join 個數與轉換時間的影響

綜合以上小節說明，可發現到多元的對應關係，若僅輸出其中一組對應，則對應關係對整體的轉換效率無特別明顯的影響；而針對屬性、集合與結構多元對應關係的處理，以結構對應的處理影響轉換效能最大。

第六章結論與未來展望

本論文提出了一個介於關聯式資料庫及XML資料之間的雙向查詢句轉換系統，透過查詢語言的轉換，使得使用者可方便的依不同類型需求來下達查詢句，再進一步進行資料的交換與使用。我們設計了數個對應表格來記錄關聯式綱要和XML綱要之間的屬性、集合與結構的多元對應關係，而轉換系統則是依據對應表格所提供的資訊來進行查詢句轉換，因此集合之間的結構關聯性亦會予以保留。經由實驗證明，經由本轉換系統可正確轉換出對應的查詢句，由於XML資料的表示法較有彈性，可以具有階層式的結構，經由對應查詢句至資料庫中查詢出的資料筆數可能會不同，我們以Distinct指令將重覆的資料濾除；關聯式連結限制式之間，若無完全符合XML中的巢狀結構關係，則相對應的查詢句所查詢得到的資料有可能會是不同的。效率方面，由SQL轉換成對應的XQuery由於要做路徑代換的處理，會比XQuery轉換至SQL花費的時間稍多；另外，當有複雜的結構對應時，會花費較多的時間轉換。

在未來研究規劃了三個方向，第一是設計對應表格的建立方式，可由更親和簡便的視窗介面，由使用者拖拉 Schema 之間的對應關係，即可自動化建立完成相關的對應表格。第二是針對轉換後的查詢句進行最佳化的動作，在集合為 1 對 n 對應關係時，目前轉換系統經由 IJT 表格所產生的內部連結限制式為 $C_{(n,2)}$ 個，但是會有多餘的連結，可另外設計檢查的演算法將多餘的連結濾除，達到最佳化只產生(n-1)個內部連結限制式；另外，由於集合一對多的關係，屬性轉換的處理時，我們是選擇於 VMT 中所找到的第一筆資訊，但實際上有可能這些屬性皆在一個集合之中，目前轉換系統有可能轉換出兩個以上的集合，再建立集合之間的關聯，可以挑戰轉換出集合個數最少、最佳化查詢句（如範例 4.21 的 SQ1）。第三就是研討與設計更完善的對應表格與轉換機制，使得可以處理更複雜的 DTD 結構對應，支援更多類型的查詢句轉換。

參考文獻

- [AL05] Marcelo Arenas, Leonid Libkin, “XML Data Exchange: Consistency and Query Answering”, Proceedings of the 24th PODS Conference, pages 13-24, Baltimore, Maryland, USA, 2005.
- [BCF+02] Michael Benedikt, Chee Yong Chan, Wenfei Fan, Rajeev Rastogi, Shihui Zheng, Aoying Zhou, “DTD-Directed Publishing with Attribute Translation Grammars”, Proceedings of the 28th VLDB Conference, pages 814-825, Hong Kong, China, 2002.
- [BDH04] Vanessa P. Braganholo, Susan B. Davidson, Carlos A. Heuser, “From XML view updates to relational view updates: old solutions to a new problem”, Proceedings of the 30th VLDB Conference, pages 276-287, Toronto, Canada. 2004.
- [CLIO] <http://www.cs.toronto.edu/db/clio/>
- [CR03] Kajal T. Claypool, Elke A. Rundensteiner, “Sangam: A Transformation Modeling Framework”, Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA), pages 219-224, Kyoto, Japan, 2003.
- [DTC+03] David DeHaan, David Toman, Mariano P. Consens, and M. Tamer Ozsu, “A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding”, Proceedings of the SIGMOD International Conference on Management of Data, pages 623-634, San Diego, California, 2003.
- [FFM05] Ariel Fuxman, Elham Fazli, Ren´ee J. Miller, “ConQuer: Efficient Management of Inconsistent Databases”, Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 155-166, Baltimore, Maryland, USA, 2005.
- [FYL+05] Wenfei Fan, Jeffrey Xu Yu, Hongjun Lu, Jianhua Lu and Rajeev Rastogi, “Query Translation from XPath to SQL in the Presence of Recursive DTDs ”, Proceedings of the 31th VLDB Conference, pages 337-348., Trondheim, Norway. 2005.
- [FKS+02] Mary F. Fernandez, Yana Kadiyska, Dan Suciu, Atsuyuki Morishima, Wang Chiew Tan, “SilkRoute: A Framework For Publishing Relational Data in XML”, ACM Transactions on Database Systems (TODS), 27(4) pages 438-493, 2002.
- [JB04] Ole G. Jensen and Michael H. Böhlen, “Lossless Conditional Schema Evolution”, Proceedings of the 23rd International Conference on Conceptual Modeling (ER2004), pages 610-623, Shanghai, China, 2004.

- [JLS+04] H. V. Jagadish, Laks V.S. Lakshmanan, Monica Scannapieco, Divesh Srivastava, Nuwee Wiwatwattana, “Colorful XML: One Hierarchy Isn’t Enough”, Proceedings of the SIGMOD International Conference on Management of Data, pages 251-262, Paris, France, 2004.
- [KCK+04] Rajasekar Krishnamurthy, Venkatesan T. Chakaravarthy, Raghav Kaushik, Jeffrey F. Naughton, “Recursive XML Schemas, Recursive XML Queries, and Relational Storage: XML-to-SQL Query Translation”, Proceedings of the 20th International Conference on Data Engineering, pages 42-53, Boston, Massachusetts, USA, 2004.
- [KKN04] Rajasekar Krishnamurthy · Raghav Kaushik · Jerey F. Naughton, “Efficient XML-to-SQL Query Translation : Where to Add the Intelligence? (Extended Abstract)”, Proceedings of the 30th VLDB Conference, pages 144-155, Toronto, Canada, 2004.
- [LYY02] Mong Li Lee, Liang Huai Yang, Wynne Hsu, Xia Yang, “Xclust: Clustering XML Schemas for Effective Integration.”, Proceedings of the 7th international conference on Information and knowledge management, pages 292-299, McLean, Virginia, USA , 2002.
- [MH03] Jayant Madhavan and Alon Halevy, “Composing Mappings among Data Sources”, Proceedings of the 29th VLDB Conference, pages 572-583 Berlin, Germany, 2003.
- [ML02] Murali Mani, Dongwon Lee, “XML to Relational Conversion using Theory of Regular Tree Grammars”, Proceedings of the VLDB Workshop on Efficiency and Effectiveness of XML Tools, and Techniques (EEXTT), pages 81-103, Hong Kong, China, 2002.
- [PBM04] Sandeep Prakash, Sourav S. Bhowmick, and Sanjay Madria, “SUCXENT: An Efficient Path-Based Approach to Store and Query XML Documents”, Proceedings of DEXA 2004, pages 285-295, Springer-Verlag , Berlin , Heidelberg, 2004.
- [PC03] Feng Peng and Sudarshan S. Chawathe, “Xpath Queries On Streaming Data“, Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 431-442, San Diego, California ,2003.
- [SKZ+99] Jayavel Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David DeWitt and Jeffrey Naughton, “Relational Databases for Querying XML Documents: Limitations and Opportunities”, Proceedings of the 25th VLDB Conference, pages 302-314, Edinburgh, Scotland, 1999.
- [SYU99] T. Shimura, M. Yoshikawa and S. Uemura, “Storage and Retrieval of XML Documents Using Object-Relational Databases”, Proceedings of DEXA 1999, pages 206-217, Florence, Italy, 1999.

- [TPCH] <http://www.tpc.org/tpch/>
- [WXD04] David W. Embley, Li Xu, Yihong Ding, “Automatic direct and indirect schema mapping: experiences and lessons learned”, SIGMOD Record, Vol. 33, No. 4, pp.14-19, 2004.
- [XE03] Li Xu and D.W. Embley, “Discovering Direct and Indirect Matches for Schema Elements”, Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA), pages 39-46, 2003.
- [YAS+01] M. Yoshikawa and T. Amagasa, T. Shimura and S. Uemura, “XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases”, ACM Transactions on Internet Technology , Vol1, No.1, pages110-114, 2001.
- [YLL03] Xia Yang, Mong Li Lee, Tok Wang Ling, “Resolving Structural Conflicts in the Integration of XML Schemas: A Semantic Approach”, Proceedings of the 22nd International Conference on Conceptual Modeling (ER2003), pages 520-533, Chicago, Illinois.
- [邱03] 邱豐傑, “轉換 SQL 為 XQuery 之研究與實作”, 碩士論文, 國立台灣海洋大學資訊科學系, 2003。
- [呂 04] 呂信賢, “以資料定義為基礎轉換 XQuery 查詢句至 SQL 查詢句”, 碩士論文, 國立台灣海洋大學資訊工程學系, 2004。

附錄 A：TPC-H RDB 與對應 Order-Ship DTD 之 VMT 內容

Rname	Aname	XPath	RXPath
SUPPLIER	SUPPKEY	/order-ship/suppliers/supplier@skey	/order-ship/suppliers/supplier
SUPPLIER	NATIONKEY	/order-ship/suppliers/supplier@nkey	/order-ship/suppliers/supplier
SUPPLIER	NAME	/order-ship/suppliers/supplier/name	/order-ship/suppliers/supplier
SUPPLIER	ADDRESS	/order-ship/suppliers/supplier/add	/order-ship/suppliers/supplier
PART	PARTKEY	/order-ship/suppliers/supplier/part@pkey	/order-ship/suppliers/supplier/part
PART	NAME	/order-ship/suppliers/supplier/part/name	/order-ship/suppliers/supplier/part
PART	TYPE	/order-ship/suppliers/supplier/part/type	/order-ship/suppliers/supplier/part
PARTSUPP	PARTKEY	/order-ship/suppliers/supplier/part@pkey	/order-ship/suppliers/supplier/part
PARTSUPP	SUPPKEY	/order-ship/suppliers/supplier@skey	/order-ship/suppliers/supplier
PARTSUPP	AVAILQTY	/order-ship/suppliers/supplier/part/availqty	/order-ship/suppliers/supplier/part
ORDER	ORDERKEY	/order-ship/suppliers/supplier/part/order@okey	/order-ship/suppliers/supplier/part/order
ORDER	CUSTKEY	/order-ship/suppliers/supplier/part/order@ckey	/order-ship/suppliers/supplier/part/order
ORDER	ORDERSTATUS	/order-ship/suppliers/supplier/part/order/status	/order-ship/suppliers/supplier/part/order
ORDER	TOTALPRICE	/order-ship/suppliers/supplier/part/order/price	/order-ship/suppliers/supplier/part/order
LINEITEM	ORDERKEY	/order-ship/suppliers/supplier/part/order@okey	/order-ship/suppliers/supplier/part/order
LINEITEM	LINENUMBER	/order-ship/suppliers/supplier/part/order@linenum	/order-ship/suppliers/supplier/part/order
LINEITEM	SUPPKEY	/order-ship/suppliers/supplier@skey	/order-ship/suppliers/supplier
LINEITEM	PARTKEY	/order-ship/suppliers/supplier/part@pkey	/order-ship/suppliers/supplier/part
LINEITEM	SHIPMODE	/order-ship/suppliers/supplier/part/order/shipmode	/order-ship/suppliers/supplier/part/order
CUSTOMER	CUSTKEY	/order-ship/customer@ckey	/order-ship/customer
CUSTOMER	NATIONKEY	/order-ship/customer@nkey	/order-ship/customer
CUSTOMER	NAME	/order-ship/customer/name	/order-ship/customer
CUSTOMER	COMMENT	/order-ship/customer/comment	/order-ship/customer
CUSTEL	CUSTKEY	/order-ship/customer@ckey	/order-ship/customer
CUSTEL	TELEPHONE	/order-ship/customer/tel	/order-ship/customer

圖 A-1：Value Mapping Table

附錄 B：TPC-H RDB 與對應 Cust-Info DTD 之對應表格內容

Rname	Aname	XPath	RXPath
CUSTOMER	CUSTKEY	/cust_info/customer@ckey	/cust_info/customer
CUSTOMER	NATIONKEY	/cust_info/customer@nkey	/cust_info/customer
CUSTOMER	NAME	/cust_info/customer/name	/cust_info/customer
CUSTOMER	CUSTKEY	/cust_info/vip_cust@ckey	/cust_info/vip_cust
CUSTOMER	NATIONKEY	/cust_info/vip_cust@nkey	/cust_info/vip_cust
CUSTOMER	COMMENT	/cust_info/vip_cust/comment	/cust_info/vip_cust
CUSTEL	CUSTKEY	/cust_info/customer@ckey	/cust_info/customer
CUSTEL	TELEPHONE	/cust_info/customer/tel	/cust_info/customer
NATION	NATIONKEY	/cust_info/nation@nkey	/cust_info/nation
NATION	REGIONKEY	/cust_info/nation/region@rkey	/cust_info/nation/region
NATION	NAME	/cust_info/nation/name	/cust_info/nation
NATION	REGIONKEY	/cust_info/nation/population/region_popu@rkey	/cust_info/nation/population/region_popu
REGION	REGIONKEY	/cust_info/nation/region@rkey	/cust_info/nation/region
REGION	REGIONKEY	/cust_info/nation/population/region_popu@rkey	/cust_info/nation/population/region_popu
REGION	NAME	/cust_info/nation/region/name	/cust_info/nation/region
REGION	COMMENT	/cust_info/nation/population/region_popu/comment	/cust_info/nation/population/region_popu

圖 B-2：Value Mapping Table

Rname	XPath	Type
CUSTOMER	/cust_info/customer	Repeatable
CUSTEL	/cust_info/customer	Repeatable
CUSTOMER	/cust_info/vip_cust	Repeatable
NATION	/cust_info/nation	Repeatable
REGION	/cust_info/nation/region	Repeatable
REGION	/cust_info/nation/population	Dummy
REGION	/cust_info/nation/population/region_popu	Repeatable

圖 B-3：Collection Mapping Table

Direction	Collection1	Collection2	Internal Join Condition
X2S	CUSTOMER	CUSTEL	CUSTOMER.CUSTKEY= CUSTEL.CUSTKEY
S2X	/cust_info/customer	/cust_info/vip_cust	/cust_info/customer@ckey= /cust_info/vip_cust@ckey
S2X	/cust_info/nation/region	/cust_info/nation/population/region_popu	/cust_info/nation/region@rkey= /cust_info/nation/population/region_popu@rkey

圖 B-4：Internal Join Table

RID	Condition1	Condition2
R1	CUSTOMER.NATIONKEY	NATION.NATIONKEY
R2	NATION.REGIONKEY	REGION.REGIONKEY

圖 B-5 : Join Mapping Table

XID	XPath1	XPath2
X _{F1}	/cust_info/customer@nkey	/cust_info/nation@nkey
X _{F2}	/cust_info/vip_cust@nkey	/cust_info/nation@nkey
X _{N1}	/cust_info/nation	/cust_info/nation/region
X _{ND1}	/cust_info/nation	/cust_info/nation/population
X _{N2}	/cust_info/nation	/cust_info/nation/population/region_popu

圖 B-6 : Path Mapping Table

RID	XID
R1	X _{F1}
R1	X _{F2}
R2	X _{N1}
R2	X _{ND1}
R2	X _{N2}

圖 B-7 : Structure Mapping Table

附錄 C：TPC-H RDB 與對應 Order-Ship XML 部分資料內容

PARTKEY	SUPPKEY	AVAILQTY
p01	s01	10000
p01	s02	11000
p01	s08	12000
p01	s05	13000
p02	s07	14000
p03	s05	15000
p03	s06	16000

圖 C-1：PARTSUPP 表格 部分資料內容

ORDERKEY	SUPPKEY	PARTKEY	LINENUMBER	SHIPMODE
o01	s03	p04	tw-l-01	陸運
o02	s07	p02	jp-l-01	航運
o03	s07	p09	jp-l-02	水運
o04	s05	p06	tw-l-02	陸運
o05	s09	p06	us-l-01	航運
o06	s01	p01	tw-l-03	自取
o07	s07	p09	jp-l-03	水運
o08	s03	p04	tw-l-04	陸運
o09	s13	p03	cn-l-01	水運
o10	s10	p05	us-l-02	航運
o11	s07	p09	jp-l-04	航運
o12	s06	p06	jp-l-05	水運

圖 C-2：LINEITEM 表格資料內容

```

<order-ship>
  <suppliers>
    <supplier skey="s01" nkey="n01">
      <name>tsmc</name>
      <add>力行路 1 號</add>
      <part pkey="p01">
        <name>dram</name>
        <type>ddr2-733</type>
        <pro>10000</pro>
        <order ckey="c06" okey="o06">
          <status>uncharge</status>
          <price>10000</price>
          <linenum>tw-l-03</linenum>
          <shipmode>自取</shipmode>
        </order>
      </part>
    </supplier>

    <supplier skey="s02" nkey="n01">
      <name>psc</name>
      <add>力行路 2 號</add>
      <part pkey="p01">
        <name>dram</name>
        <type>ddr2-733</type>
        <pro>11000</pro>
      </part>
    </supplier>

    <supplier skey="s05" nkey="n01">
      <name>asus</name>
      <add>南京東路 1 號</add>
      <part pkey="p01">
        <name>dram</name>
        <type>ddr2-733</type>
        <pro>13000</pro>
      </part>
      <part pkey="p03">
        <name>dvd</name>
        <type>16x</type>
        <pro>15000</pro>
      </part>
      <part pkey="p06">
        <name>notebook</name>
        <type>17"</type>
        <pro>25000</pro>
        <order ckey="c04" okey="o04">
          <status>paid</status>
          <price>8000</price>
          <linenum>tw-l-02</linenum>
          <shipmode>陸運</shipmode>
        </order>
      </part>
      <part pkey="p11">
        <name>keyboard</name>
        <type>104</type>
        <pro>36000</pro>
      </part>
    </supplier>

    <supplier skey="s08" nkey="n02">
      <name>nec</name>
      <add>名古屋 1 號</add>
      <part pkey="p01">
        <name>dram</name>
        <type>ddr2-733</type>
        <pro>12000</pro>
      </part>
    </supplier>
  </suppliers>
  .....
  .....

```

圖 C-3：部分 Order-Ship 的 XML 資料內容

4.6.3 節中的查詢句 實際查詢結果

範例 4.20 - (XQ1)

```

<xmlresult1.xml>
  <result>
    <skey>s01</skey>
    <pro>10000</pro>
    <linenum>tw-1-03</linenum>
  </result>

```

範例 4.20 - (SQ1)及範例 4.21-(SQ1),(SQ2)

	SUPPKEY	PRODUCTION	LINENUMBER
1	s01	10000	tw-l-03

範例 4.22 - (XQ1)

	SUPPKEY	PRODUCTION	LINENUMBER
1	s01	10000	tw-l-03
2	s02	11000	tw-l-03
3	s08	12000	tw-l-03
4	s05	13000	tw-l-03

範例 4.23 - (XQ1)

```

<xmlresult2.xml>
  <result>
    <pro>10000</pro>
    <linenum>tw-1-03</linenum>
  </result>
  <result>
    <pro>19000</pro>
    <linenum>tw-1-01</linenum>
  </result>
  <result>
    <pro>19000</pro>
    <linenum>tw-1-04</linenum>
  </result>
  <result>
    <pro>25000</pro>
    <linenum>tw-1-02</linenum>
  </result>
  ...

```

共 12 筆

範例 4.23 - (SQ1), (SQ2)

	PRODUCTION	LINENUMBER
1	10000	tw-l-03
2	11000	tw-l-03
3	12000	tw-l-03
4	13000	tw-l-03
5	14000	jp-l-01
6	15000	cn-l-01
7	16000	cn-l-01
8	17000	cn-l-01
9	18000	cn-l-01
10	19000	tw-l-01
11	19000	tw-l-04
12	20000	tw-l-01

共 42 筆

附錄 D：

(I) Structure 一對多轉換效率影響的實驗

說明： 設計同一組關聯式表格，對應到的 DTD 分別為 Flat 與 Nested 結構，比較 Structure 一對多對應與轉換效率的影響。其中表格 A 會對應到元素 a1，表格 B 會對應到元素 b1 與 b2，因此表格 A 與表格 B 之間的連結限制式為一個 1 對 2 的對應關係；依此類推關連式表格 A 與關連式表格 E、B、C、D 之間連結限制式，對應於 Flat 與 Nested 的 DTD 結構分別為 1 對 1 至 1 對 4。查詢句中，選擇的欄位皆為 A.AE1，控制連結限制式為表格 A 與 B、A 與 C 或是 A 與 D 之間的連結。

A (AKEY , AE1)

E (EKEY , AKEY , EE1)

B (BKEY , AKEY , BE1, BE2)

C (CKEY , AKEY , CE1, CE2, CE3)

D (DKEY , AKEY , DE1, DE2, DE3, DE4)

圖 D(I)-1： 關連式表格架構

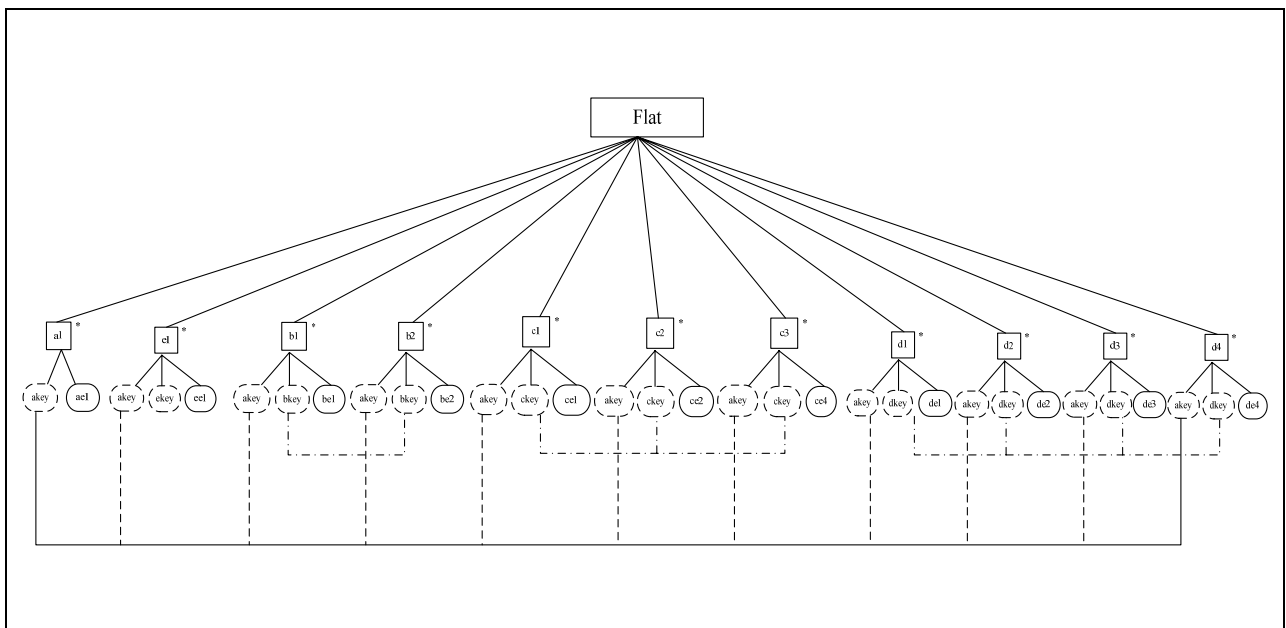


圖 D(I)-2： 對應 Flat 結構的 DTD

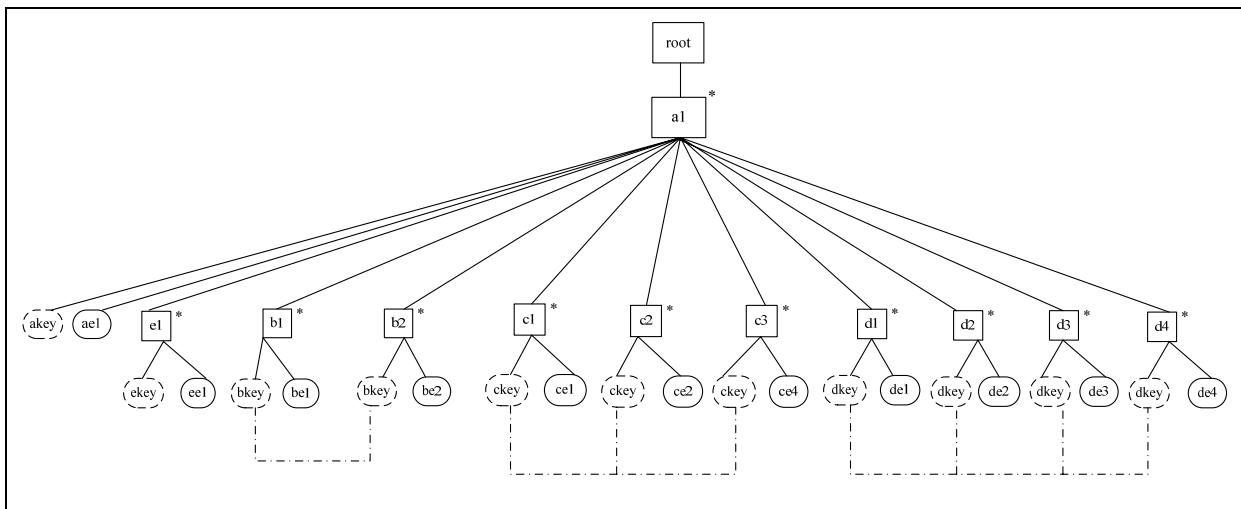


圖 D(I)-3：對應 Nested 結構的 DTD

實驗使用的 SQL 查詢句與針對 Flat 及 Nested 結構個別轉換出的 XQuery：

Structure 對應	輸入的 SQL	轉換出來的 XQuery (Flat 結構)	轉換出來的 XQuery (Nested 結構)
1 對 1	SELECT A.AE1 FROM A, E WHERE A.AKEY=E.AKEY	For \$t0 in /flat/a1,\$t1 in /flat/e1 Where \$t0@akey=\$t1@akey Return \$t0/ae1	For \$t0 in /root/a1,\$t1 in \$t0/e1 Return \$t0/ae1
1 對 2	SELECT A.AE1 FROM A, B WHERE A.AKEY=B.AKEY	For \$t0 in /flat/a1,\$t1 in /flat/b1 Where \$t0@akey=\$t1@akey Return \$t0/ae1	For \$t0 in /root/a1, \$t1 in \$t0/b1 Return \$t0/ae1
1 對 3	SELECT A.AE1 FROM A, C WHERE A.AKEY=C.AKEY	For \$t0 in /flat/a1,\$t1 in /flat/c1 Where \$t0@akey=\$t1@akey Return \$t0/ae1	For \$t0 in /root/a1,\$t1 in \$t0/c1 Return \$t0/ae1
1 對 4	SELECT A.AE1 FROM A, D WHERE A.AKEY=D.AKEY	For \$t0 in /flat/a1,\$t1 in /f/d1 Where \$t0@akey=\$t1@akey Return \$t0/ae1	For \$t0 in /root/a1, \$t1 in \$t0/d1 Return \$t0/ae1

(II)：Mapping of Attribute 個數對轉換效率影響的實驗

說明：關聯式表格 A、B、C、D 中的鍵值 AKEY、BKEY、CKEY、DKEY，其對應於 DTD 中的 akey、bkey、ckey、dkey 分別為 1 對 1、1 對 2、1 對 3 與 1 對 4 的對應關係；查詢句中，選擇的欄位皆為個別表格的鍵值，測試屬性一對多對轉換效率的影響。

A (AKEY , AE1)

B (BKEY , BE1 , BE2)

C (CKEY , CE1 , CE2 , CE3)

D (DKEY , DE1 , DE2 , DE3 , DE4)

圖 D(II)-1：關連式表格架構

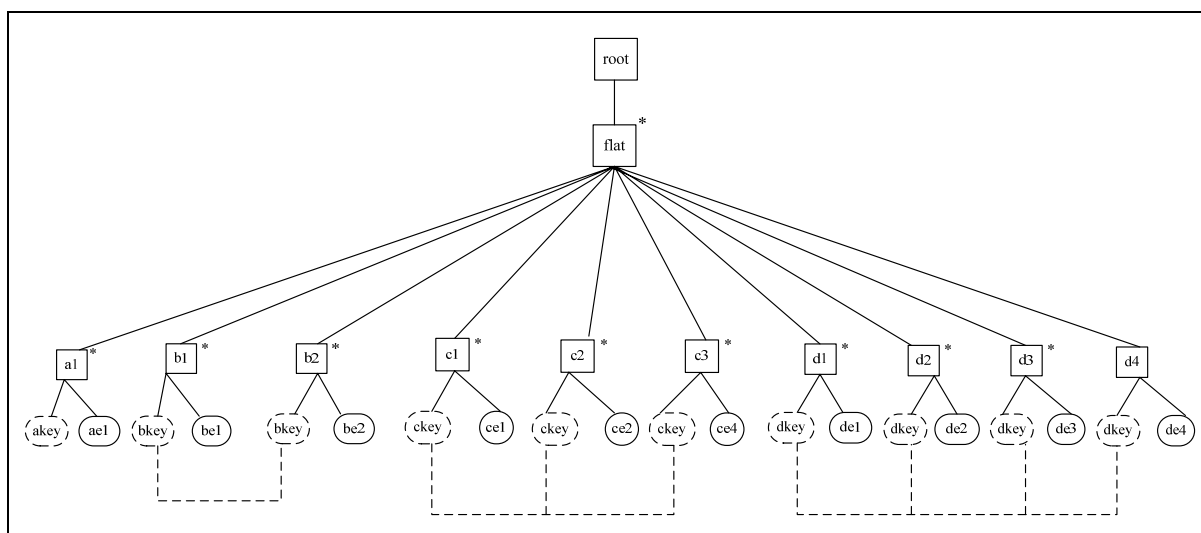


圖 D(II)-2：對應的 DTD

實驗使用的查詢句：

Mapping of Attribute 個數	輸入的 SQL	轉換出來的 XQuery
1	SELECT A.AKEY FROM A	For \$t0 in /root/flat/a1 Return \$t0@akey
2	SELECT B.BKEY FROM B	For \$t0 in /root/flat/b1 Return \$t0@bkey
3	SELECT C.CKEY FROM C	For \$t0 in /root/flat/c1 Return \$t0@ckey
4	SELECT D.DKEY FROM D	For \$t0 in /root/flat/d1 Return \$t0@dkey

(III)：輸出 Attribute 個數對轉換效率影響的實驗

說明：查詢句中，控制選擇的 Attribute 個數，比較其對轉換效率的影響。

A (AKEY, AE1, AE2, AE3, AE4, AE5, AE6, AE7, AE8, AE9, AE10)

圖 D(III)-1：關連式表格架構

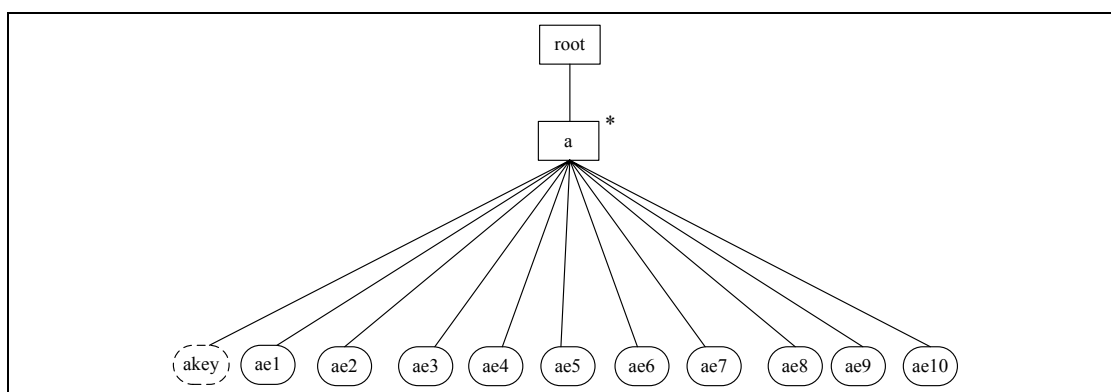


圖 D(III)-2：對應的 DTD

實驗使用的查詢句：

Attribute 個數	輸入的 SQL	轉換出來的 XQuery
2	SELECT A.AE1, A.AE2 FROM A	For \$t0 in /root/a Return \$t0/ae1, \$t0/ae2
4	SELECT A.AE1, A.AE2, A.AE3, A.AE4 FROM A	For \$t0 in /root/a Return \$t0/ae1, \$t0/ae2, \$t0/ae3, \$t0/ae4
6	SELECT A.AE1, A.AE2, A.AE3, A.AE4, A.AE5, A.AE6 FROM A	For \$t0 in /root/a Return \$t0/ae1, \$t0/ae2, \$t0/ae3, \$t0/ae4,\$t0/ae5, \$t0/ae6
8	SELECT A.AE1, A.AE2, A.AE3, A.AE4, A.AE5, A.AE6, A.AE7, A.AE8 FROM A	For \$t0 in /root/a Return \$t0/ae1, \$t0/ae2, \$t0/ae3, \$t0/ae4,\$t0/ae5, \$t0/ae6 , \$t0/ae7, \$t0/ae8
10	SELECT A.AE1, A.AE2, A.AE3, A.AE4, A.AE5, A.AE6, A.AE7, A.AE8, A.AE9, A.AE10 FROM A	For \$t0 in /root/a Return \$t0/ae1, \$t0/ae2, \$t0/ae3, \$t0/ae4,\$t0/ae5, \$t0/ae6 , \$t0/ae7, \$t0/ae8,\$t0/ae9, \$t0/ae10

(IV-1) : (S2X) Collection 一對多對轉換效率影響的實驗

說明: 查詢句中, 控制關聯式表格對DID 中可重覆元素一對多對應的個數, 比較其對轉換效率的影響。

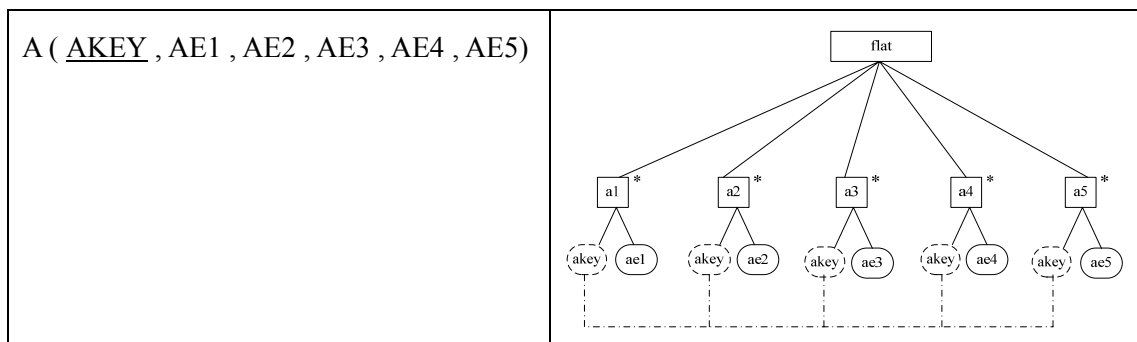


圖 D(IV-1)-1： 關連式表格架構

圖 D(IV-1)-2： 對應的 DTD

實驗使用的查詢句：

Collection 對應	輸入的 SQL	轉換出來的 XQuery
1 對 1	SELECT A.AE1 FROM A	For \$t0 in /flat/a1 Return \$t0/ae1
1 對 2	SELECT A.AE1, A.AE2 FROM A	For \$t0 in /flat/a1,\$t1 in /flat/a2 Where \$t0@akey=\$t1@akey Return \$t0/ae1, \$t1/ae2

1 對 3	SELECT A.AE1, A.AE2, A.AE3 FROM A	For \$t0 in /flat/a1,\$t1 in /flat/a2,\$t2 in /flat/a3 Where \$t0@akey=\$t1@akey And \$t0@akey=\$t2@akey And \$t1@akey=\$t2@akey Return \$t0/ae1, \$t1/ae2, \$t2/ae3
1 對 4	SELECT A.AE1, A.AE2, A.AE3, A.AE4 FROM A	For \$t0 in /flat/a1,\$t1 in /flat/a2, \$t2 in /flat/a3,\$t3 in /flat/a4 Where \$t0@akey=\$t1@akey And \$t0@akey=\$t2@akey And \$t0@akey=\$t3@akey And \$t1@akey=\$t2@akey And \$t1@akey=\$t3@akey And \$t2@akey=\$t3@akey Return \$t0/ae1, \$t1/ae2, \$t2/ae3, \$t3/ae4
1 對 5	SELECT A.AE1,A.AE2, A.AE3, A.AE4, A.AE5 FROM A	For \$t0 in /flat/a1,\$t1 in /flat/a2, \$t2 in /flat/a3,\$t3 in /flat/a4, \$t4 in /flat/a5 Where \$t0@akey=\$t1@akey And \$t0@akey=\$t2@akey And \$t0@akey=\$t3@akey And \$t0@akey=\$t4@akey And \$t1@akey=\$t2@akey And \$t1@akey=\$t3@akey And \$t1@akey=\$t4@akey And \$t2@akey=\$t3@akey And \$t2@akey=\$t4@akey And \$t3@akey=\$t4@akey Return \$t0/ae1, \$t1/ae2, \$t2/ae3, \$t3/ae4, \$t4/ae5

(IV-2) : (X2S) Collection 一對多對轉換效率影響的實驗

說明: 查詢句中, 控制可重覆元素與關聯式表格一對多對應的個數, 比較其對轉換效率的影響。

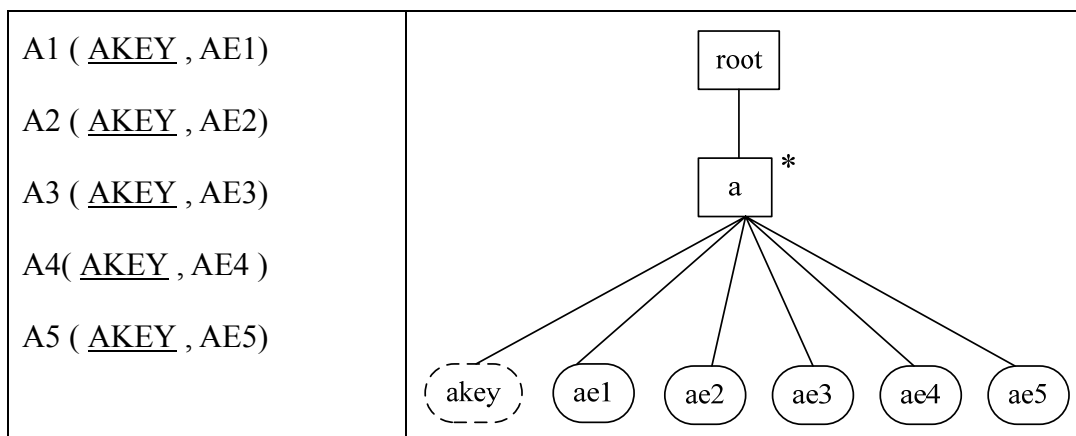


圖 D(IV-2)-1：關連式表格架構

圖 D(IV-2)-2：對應的 DTD

實驗使用的查詢句：

Collection 對應	輸入的 XQuery	轉換出來的 SQL
1 對 1	For \$t0 in /root/a Return \$t0/ae1	SELECT A1.AE1 FROM A1
1 對 2	For \$t0 in /root/a Return \$t0/ae1, \$t0/ae2	SELECT A1.AE1 , A2.AE2 FROM A1 , A2 WHERE A1.AKEY=A2.AKEY
1 對 3	For \$t0 in /root/a Return \$t0/ae1, \$t0/ae2, \$t0/ae3	SELECT A1.AE1 , A2.AE2 , A3.AE3 FROM A1 , A2 , A3 WHERE A1.AKEY=A2.AKEY And A1.AKEY=A3.AKEY And A2.AKEY=A3.AKEY
1 對 4	For \$t0 in /root/a Return \$t0/ae1, \$t0/ae2, \$t0/ae3, \$t0/ae4	SELECT A1.AE1 , A2.AE2 , A3.AE3 , A4.AE4 FROM A1 , A2 , A3 , A4 WHERE A1.AKEY=A2.AKEY And A1.AKEY=A3.AKEY And A1.AKEY=A4.AKEY And A2.AKEY=A3.AKEY And A2.AKEY=A4.AKEY And A3.AKEY=A4.AKEY
1 對 5	For \$t0 in /root/a Return \$t0/ae1, \$t0/ae2, \$t0/ae3, \$t0/ae4, \$t0/ae5	SELECT A1.AE1 , A2.AE2 , A3.AE3 , A4.AE4 , A5.AE5 FROM A1 , A2 , A3 , A4 , A5 WHERE A1.AKEY=A2.AKEY And A1.AKEY=A3.AKEY And A1.AKEY=A4.AKEY And A1.AKEY=A5.AKEY And A2.AKEY=A3.AKEY And A2.AKEY=A4.AKEY And A2.AKEY=A5.AKEY And A3.AKEY=A4.AKEY And A3.AKEY=A5.AKEY And A4.AKEY=A5.AKEY

D(V-1)：(S2X) Output Structure 個數對轉換效率影響的實驗

說明： 關連式表格 A 對應元素 a，而關連式表格 B 對應元素 b1、b2、b3、b4，

表格 A 與 B 的連結限制式對應 Flat 與 Nested 的 DTD，結構對應皆為一對

4。比較 SQL 轉 XQuery 時，輸出的結構數增長對轉換效率的影響。

A (<u>AKEY</u> , AE1) B (<u>BKEY</u> , <u>AKEY</u> , BE1, BE2, BE3, BE4)

圖 D(V-1)-1： 關連式表格架構

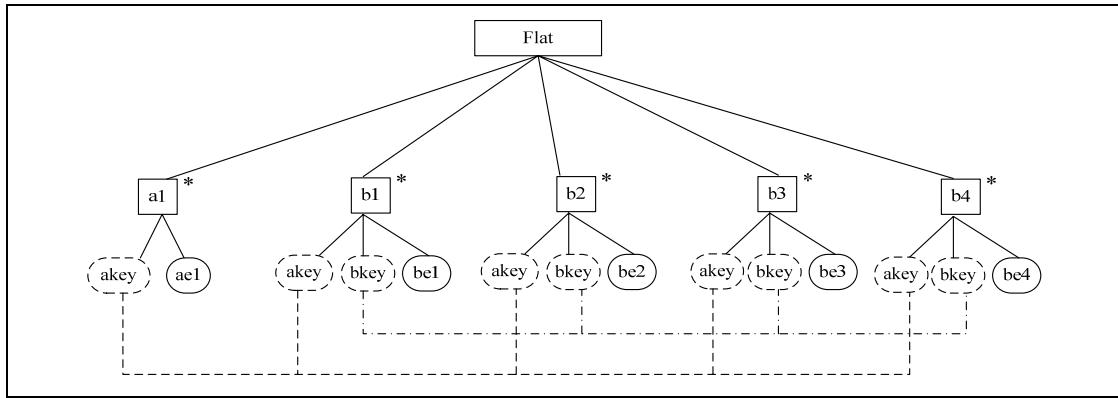


圖 D(V-1)-2：對應 Flat 結構的 DTD

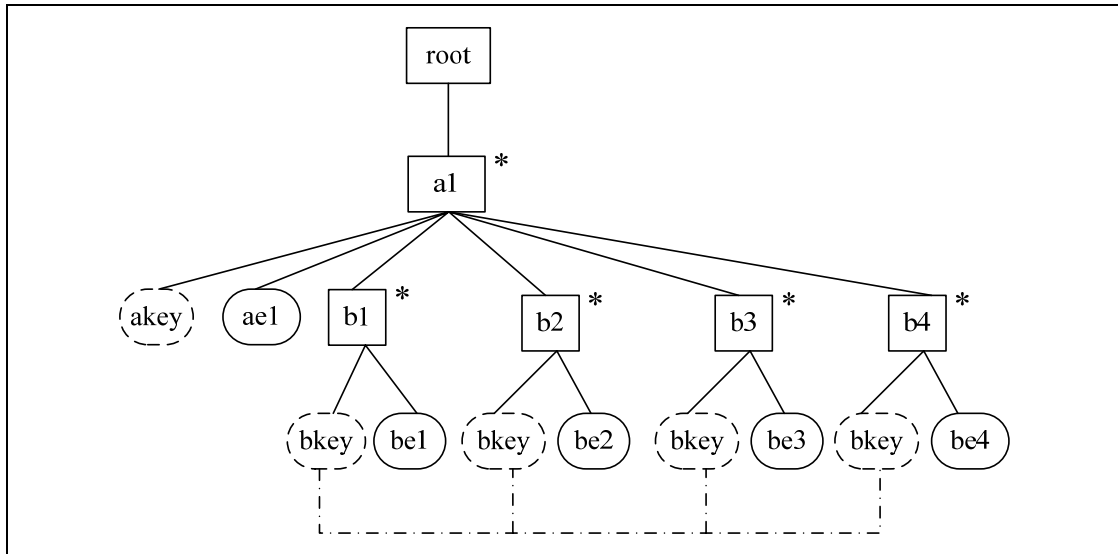


圖 D(V-1)-3：對應 Nested 結構的 DTD

實驗使用的 SQL 查詢句與針對 Flat 及 Nested 結構個別轉換出的 XQuery：

Output Structure 個數	輸入的 SQL	轉換出來的 XQuery Flat 結構	轉換出來的 XQuery Nested 結構
1	SELECT A.AE1, B.BE1 FROM A, B WHERE A.AKEY=B.AKEY	For \$t0 in /flat/a1,\$t1 in /flat/b1 Where \$t0@akey=\$t1@akey Return \$t0/ae1, \$t1/be1	For \$t0 in /root/a1,\$t1 in \$t0/b1 Return \$t0/ae1, \$t1/be1
2	SELECT A.AE1, B.BE1, B.BE2 FROM A, B WHERE A.AKEY=B.AKEY	For \$t0 in /flat/a1,\$t1 in /flat/b1, \$t2 in /f/b2 Where \$t0@akey=\$t1@akey And \$t0@akey=\$t2@akey And \$t1@bkey=\$t2@bkey Return \$t0/ae1, \$t1/be1, \$t2/be2	For \$t0 in /root/a1,\$t1 in \$t0/b1, \$t2 in \$t0/b2 Where \$t1@bkey=\$t2@bkey Return \$t0/ae1, \$t1/be1, \$t2/be2
3	SELECT A.AE1, B.BE1, B.BE2, B.BE3 FROM A, B WHERE A.AKEY=B.AKEY	For \$t0 in /flat/a1,\$t1 in /flat/b1, \$t2 in /flat/b2,\$t3 in /flat/b3 Where \$t0@akey=\$t1@akey And \$t0@akey=\$t2@akey And \$t0@akey=\$t3@akey And \$t1@bkey=\$t2@bkey And \$t1@bkey=\$t3@bkey And \$t2@bkey=\$t3@bkey Return \$t0/ae1, \$t1/be1, \$t2/be2, \$t3/be3	For \$t0 in /root/a1,\$t1 in \$t0/b1, \$t2 in \$t0/b2,\$t3 in \$t0/b3 Where \$t1@bkey=\$t2@bkey And \$t1@bkey=\$t3@bkey And \$t2@bkey=\$t3@bkey Return \$t0/ae1, \$t1/be1, \$t2/be2, \$t3/be3

4	SELECT A.AE1, B.BE1, B.BE2, B.BE3, B.BE4 FROM A, B WHERE A.AKEY=B.AKEY	For \$t0 in /flat/a,\$t1 in /flat/b1, \$t2 in /flat/b2,\$t3 in /flat/b3, \$t4 in /flat/b4 Where \$t0@akey=\$t1@akey And \$t0@akey=\$t2@akey And \$t0@akey=\$t3@akey And \$t0@akey=\$t4@akey And \$t1@bkey=\$t2@bkey And \$t1@bkey=\$t3@bkey And \$t1@bkey=\$t4@bkey And \$t2@bkey=\$t3@bkey And \$t2@bkey=\$t4@bkey And \$t3@bkey=\$t4@bkey Return \$t0/ae1, \$t1/be1, \$t2/be2, \$t3/be3, \$t4/be4	For \$t0 in /root/a,\$t1 in \$t0/b1, \$t2 in \$t0/b2,\$t3 in \$t0/b3, \$t4 in \$t0/b4 Where \$t1@bkey=\$t2@bkey And \$t1@bkey=\$t3@bkey And \$t1@bkey=\$t4@bkey And \$t2@bkey=\$t3@bkey And \$t2@bkey=\$t4@bkey And \$t3@bkey=\$t4@bkey Return \$t0/ae1, \$t1/be1, \$t2/be2, \$t3/be3, \$t4/be4
---	---	---	---

D(V-2) : (X2S) Output Structure 個數對轉換效率影響的實驗

說明：元素 a 會對應到關連式表格的 A1，而元素 b 會對應到關連式表格 B1、B2、B3、B4，因此，元素 a 與元素 b 之間的結構關連，對應到關連式資料表格的連結限制式為 1 對 4 的關係。比較 XQuery 轉 SQL 時，輸出的結構數增長對轉換效率的影響。

A1 (AKEY , AE1)

B1 (BKEY , AKEY , BE1)

B2 (BKEY , AKEY , BE2)

B3 (BKEY , AKEY , BE3)

B4 (BKEY , AKEY , BE4)

圖 D(V-2)-1： 關連式表格架構

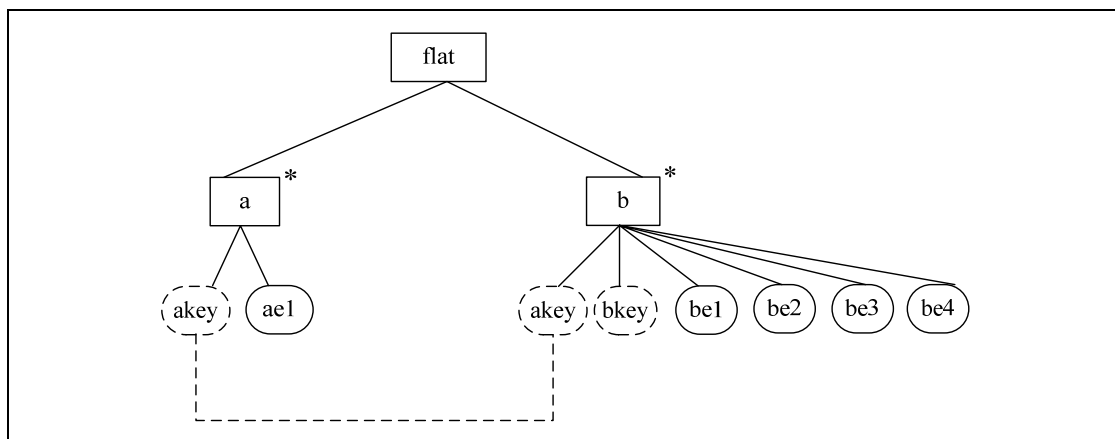


圖 D(V-2)-2： 對應 Flat 結構的 DTD

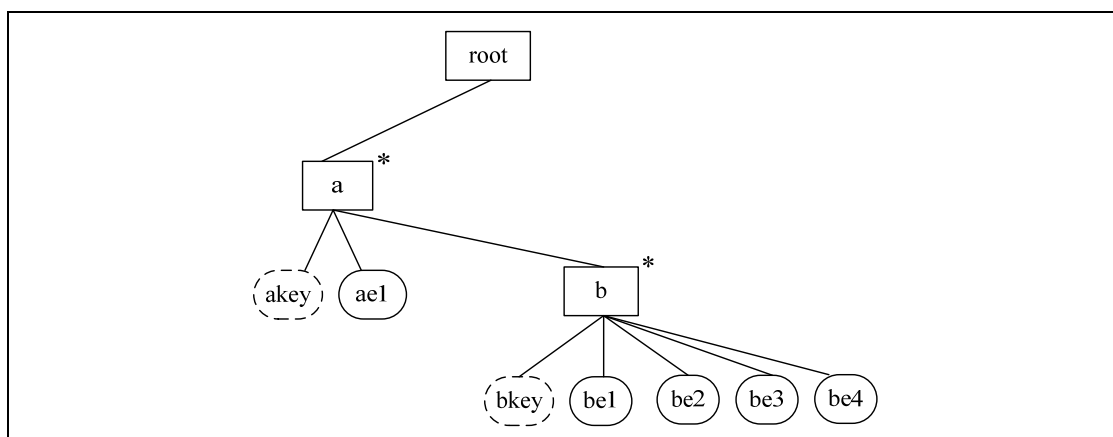


圖 D(V-2)-3：對應 Nested 結構的 DTD

實驗針對 Flat 及 Nested 結構使用的 XQuery 查詢句與轉換出的相同 SQL：

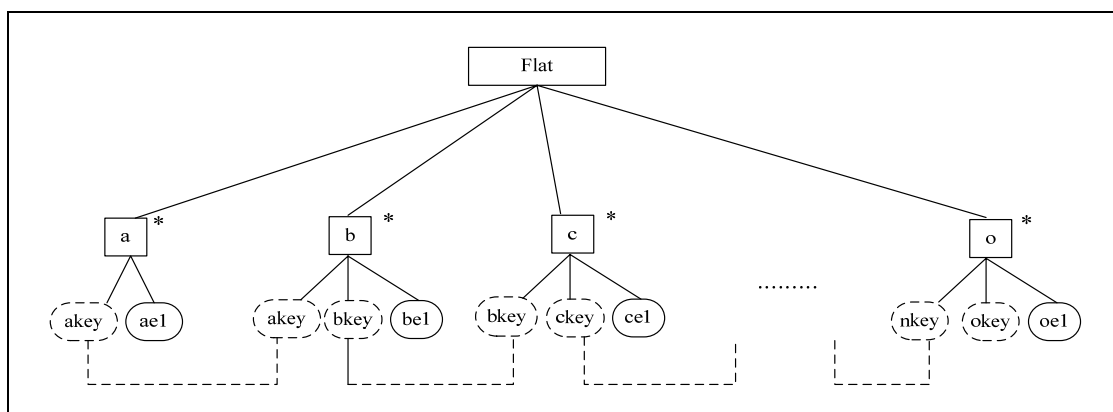
Output Structure 個數	輸入的 XQuery Flat 結構	輸入的 XQuery Nested 結構	轉換出來的相同 SQL
1	For \$t0 in /flat/a,\$t1 in /flat/b Where \$t0@akey=\$t1@akey Return \$t0/ae1, \$t1/be1	For \$t0 in /root/a, \$t1 in \$t0/b Return \$t0/ae1, \$t1/be1	SELECT A1.AE1 , B1.BE1 FROM A1 , B1 WHERE A1.AKEY=B1.AKEY
2	For \$t0 in /flat/a, \$t1 in /flat/b Where \$t0@akey=\$t1@akey Return \$t0/ae1, \$t1/be1, \$t1/be2	For \$t0 in /root/a, \$t1 in \$t0/b Return \$t0/ae1, \$t1/be1, \$t1/be2	SELECT A1.AE1 , B1.BE1 , B2.BE2 FROM A1 , B1 , B2 WHERE A1.AKEY=B1.AKEY And A1.AKEY=B2.AKEY And B1.BKEY=B2.BKEY
3	For \$t0 in /flat/a, \$t1 in /flat/b Where \$t0@akey=\$t1@akey Return \$t0/ae1, \$t1/be1, \$t1/be2, \$t1/be3	For \$t0 in /root/a, \$t1 in \$t0/b Return \$t0/ae1, \$t1/be1, \$t1/be2, \$t1/be3	SELECT A1.AE1 , B1.BE1 , B2.BE2 , B3.BE3 FROM A1 , B1 , B2 , B3 WHERE A1.AKEY=B1.AKEY And A1.AKEY=B2.AKEY And A1.AKEY=B3.AKEY And B1.BKEY=B2.BKEY And B1.BKEY=B3.BKEY And B2.BKEY=B3.BKEY
4	For \$t0 in /flat/a,\$t1 in /flat/b Where \$t0@akey=\$t1@akey Return \$t0/ae1, \$t1/be1, \$t1/be2, \$t1/be3, \$t1/be4	For \$t0 in /root/a, \$t1 in \$t0/b Return \$t0/ae1, \$t1/be1, \$t1/be2,\$t1/be3 , \$t1/be4	SELECT A1.AE1 , B1.BE1 , B2.BE2 ,B3.BE3 , B4.BE4 FROM A1 , B1 , B2 , B3 , B4 WHERE A1.AKEY=B1.AKEY And A1.AKEY=B2.AKEY And A1.AKEY=B3.AKEY And A1.AKEY=B4.AKEY And B1.BKEY=B2.BKEY And B1.BKEY=B3.BKEY And B1.BKEY=B4.BKEY And B2.BKEY=B3.BKEY And B2.BKEY=B4.BKEY And B3.BKEY=B4.BKEY

(VI)：關連式連結限制式對應 DTD 寬度與深度對轉換效率影響的實驗

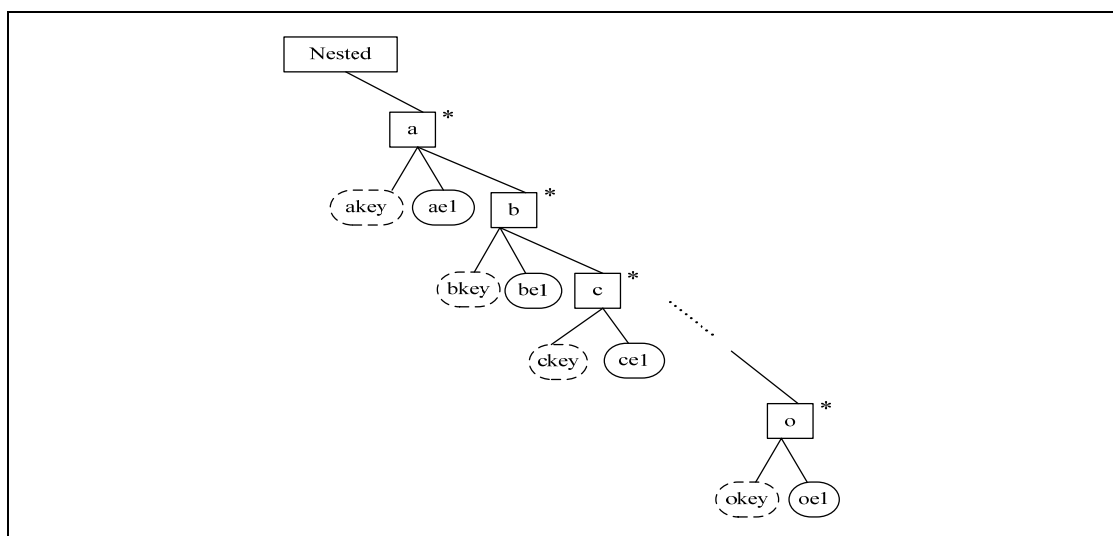
說明：設計 15 個表格 A、B、...至 O，其對應到 DTD 中的元素分別為 a、b、...至 o；前後兩表格間皆有關連，如 A join B、B join C、...、N join O，且此結構關連對應於 DTD 中時，分別為元素 a 與 b、b 與 c、...、n 與 o 之間的結構對應。針對 DTD 為 Flat 結構與 Nested 結構，我們僅選擇輸出 A 表格的 AE1 欄位，控制關聯式連結的個數，來測試 SQL 至 XQuery 轉換時間的影響。



圖(VI)-1： 關連式表格架構



圖(VI)-2： 對應 Flat 結構的 DTD



圖(VI)-3： 對應 Nested 結構的 DTD

實驗針對 Flat 及 Nested 結構所輸入的 SQL：(轉換的 XQuery 結果因篇幅關係 略)
輸入的 SQL：

[RDB Join 個數為 3]

```
SELECT A.AE1
FROM A, B, C
WHERE A.AID=B.BID
      AND
      B.BID=C.CID
```

[RDB Join 個數為 6]

```
SELECT A.AE1
FROM A, B, C, D, E, F
WHERE  A.AID=B.BID
      AND
      B.BID=C.CID
      AND
      C.CID=D.DID
      AND
      D.DID=E.EID
      AND
      E.EID=F.FID
```

[RDB Join 個數為 9]

```
SELECT A.AE1
FROM A, B, C, D, E, F, G, H, I
WHERE A.AID=B.BID
      AND
      B.BID=C.CID
      AND
      C.CID=D.DID
      AND
      D.DID=E.EID
      AND
      E.EID=F.FID
      AND
      F.FID=G.GID
      AND
      G.GID=H.HID
      AND
      H.HID=I.IID
```

[RDB Join 個數為 12]

```
SELECT A.AE1
FROM A, B, C, D, E, F, G, H, I, J, K, L
WHERE A.AID=B.BID
      AND
      B.BID=C.CID
      AND
      C.CID=D.DID
      AND
      D.DID=E.EID
      AND
      E.EID=F.FID
      AND
      F.FID=G.GID
      AND
      G.GID=H.HID
      AND
      H.HID=I.IID
      AND
      I.IID=J.JID
      AND
      J.JID=K.KID
      AND
      K.KID=L.LID
```

[RDB Join 個數為 15]

```
SELECT A.AE1
FROM A, B, C, D, E, F, G, H, I, J, K, L, M, N, O
WHERE A.AID=B.BID
      AND
      B.BID=C.CID
      AND
      C.CID=D.DID
      AND
      D.DID=E.EID
      AND
      E.EID=F.FID
      AND
      F.FID=G.GID
      AND
      G.GID=H.HID
      AND
      H.HID=I.IID
      AND
      I.IID=J.JID
      AND
      J.JID=K.KID
      AND
      K.KID=L.LID
      AND
      L.LID=M.MID
      AND
      M.MID=N.NID
      AND
      N.NID=O.OID
```