國立臺灣海洋大學

資訊工程學系

碩士學位論文

指導教授：張雅惠 博士

巢狀 XQuery 與等價 SQL 查詢句雙向轉換之研究

Bidirectional Query Translation between Nested XQuery and Equivalent SQL

研究生：李佳臻　撰

中華民國 99 年 1 月

巢狀 XQuery 與等價 SQL 查詢句雙向轉換之研究

Bidirectional Query Translation between Nested XQuery and Equivalent SQL

研 究 生：李佳臻　　Student：Chia-Zhen Lee
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 (Dissertation)
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
Jan 2010
Keelung, Taiwan, Republic of China

中華民國 99 年 1 月

摘要

如何提供關聯式與XML格式的資料共享，已經成為重要的研究議題。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。在本論文中，我們提出一個雙向查詢句轉換系統，使用者可方便地依不同類型定義來下達SQL或XQuery的巢狀查詢句，而系統可將該查詢句轉換成合適的表示式。

我們設計了數個對應函式來記錄關聯式綱要和 XML 綱要之間的資料內容、集合與結構的對應關係，為了處理多元對應的情況，會額外記錄優先權。我們實作一個查詢句雙向轉換系統，該系統會先剖析使用者輸入的查詢句，對查詢句中不同的集合建立集合樹，至於資料內容則建立值樹。之後集合樹和值樹會依據對應函式取得對應資料，然後再依樹的階層有順序性地處理巢狀結構，並且收集樹轉換後的集合和資料內容。同時，針對查詢句中結構的多元對應，我們會利用最小生成樹做最佳化處理。

我們會正式證明轉換的正確性，同時利用實驗顯示本系統可以有效率地進行查詢句轉換。

Abstract

How to provide the sharing of data between relational formats and XML formats, has become an important research topic. The standard query language for relational database is SQL, and the standard query language for XML data is XQuery. In this thesis, we propose a bi-directional query translation system. Users can easily query different types of schemas, and system can convert nested SQL or XQuery queries into the appropriate expressions.

We have designed a set of mapping functions to record the complex mappings among attributes, collections and structural constructs. We also record the priorities of attributes and collections to process multiple mappings. The proposed bidirectional translation system will first parse the input query, construct the Collection Tree and Value Tree. Next, the Collection Tree and Value Tree will get the necessary information from the mapping tables. It then systematically processes the nested structure by its tree level and combines the transformed collection and value constructs. It will also perform optimization by constructing minimum spanning trees when choosing the appropriate structural constructs.

We will formally prove the correctness of the translation system, and perform a set of experiments to show that our translation system can produce queries efficiently.

章節目錄

圖目錄

表格目錄

第 1 章　序論

1.1. 背景與研究動機

W3C 組織制訂的可擴充式標註語言（eXtensible Markup Language；XML）是一種描述資料的標註語言，使用者可自由定義標籤以將資料結構化。XML 的結構可由 DTD（Document Type Definition）或 XML Schema 的定義所限制，其表示法有較多的彈性，特別是可利用巢狀結構直接表示出集合之間的關係，由於 XML 為純文字格式，非常有利做為資料傳遞和交換的格式，所以許多企業、學術單位和機關團體都已將 XML 做為資料交換的統一格式。

然而，大多資料仍儲存在非 XML 資料庫系統中，特別是具有高穩定性和高成熟度特性的關聯式資料庫系統（Relational Database Management System；RDBMS）。由於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），因此當資料分別以關聯式格式和 XML 格式儲存時，會因使用的資料庫格式不同，增加了資料管理的複雜度以及資料整合的困難，所以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，並解決兩類型資料的不一致性，以達到資料共享。

在處理綱要間的對應關係時，DTD 與關聯式綱要之間可能具有「多元對應」。當 DTD 的節點與關聯式綱要表格欄位有一對多或多對一的對應關係時，表示有值的多元對應，而當 DTD 的節點與關聯式綱要表格有一對多或多對一的對應關係時，則代表有集合的多元對應。表格的正規化或 DTD 中存在多個意義相同的節點會造成值的多元對應。而關係表格對應到 XML 文件中的巢狀結構、同一路徑上的空節點和其父節點或子節點對應到同樣的表格、存在多個意義相同的節點會造成集合的多元對應，除此之外，值的多元對應也會造成集合的多元對應。

在查詢 XML 資料方面，W3C 制定了 XQuery 查詢語言，XQuery 依據 DTD 定義下達合法的路徑表示法（XPath Expression），XQuery 敘述句包含 FOR、LET、WHERE、RETURN 子句。LET 子句和 RETURN 子句可包含子查詢句。另一方面，關聯式資料則是根據關聯式綱要使用結構化查詢語言（Structure Query Language；SQL）。SQL 資料查詢句可分為 SELECT、FROM、WHERE 三個子句，FROM 子句和 WHERE 子句中可包含子查詢句，其中 FROM 子句中的子查詢句不可直接使用外部查詢句的來源表格。

以 XQuery 與 SQL 相比較，XQuery 的 FOR 子句大致相當於 SQL 查詢語言中的 FROM 子句，兩者皆是定義資料來源；XQuery 的 RETURN 子句則相當於 SQL 查詢語言中的 SELECT 子句，兩者皆為輸出資料；XQuery 的 WHERE 子句則對應到 SQL 查詢語言中的 WHERE 子句，其中 WHERE 子句又可分為文數值限制和連結限制，文數值限制限定了屬性內容，連結限制則限定結構關係，在 XQuery 可以用巢狀結構直接限制節點之間的連結，而在 SQL 則是利用主鍵和外來鍵限制表格間的連結。針對子查詢句，XQuery 的子查詢句為 Let 子句或 Return 子句中的 XQuery，而在關聯式資料庫中，子查詢句為 FROM 子句或 WHERE 子句中的 SQL 查詢句。

綜合以上，我們發現轉換查詢句時，綱要間會有多元對應，而子查詢句的轉換也需要特別處理，所以論文針對以上的問題探討，設計出一個轉換系統轉換具有巢狀結構和子查詢句的複雜查詢句。

1.2. 研究目的與貢獻

本研究的目的是建立一個介於關聯式資料庫與 XML 資料之間的轉換系統，以方便使用者轉換查詢語言。論文中亦針對具有巢狀結構和子查詢句的複雜查詢句探討如何轉換出正確的查詢句。轉換系統可將具巢狀結構的 XQuery 查詢語言

轉換成等價 SQL 查詢語言。

針對多元對應的情況，對應函數會依表格和節點的相似度設定權重，相似度越高則權重越高，每個對應集合和對應值都會有自己的權重，權重越高代表輸出此表格或節點的可能性越高。在多元對應的情況下會優先輸出權重高的資料。

本論文特別考量子查詢句作深入的探討。轉換系統首先剖析輸入的查詢句，查詢句中的每個集合和回傳值都會分別建立樹，WHERE子句中的值則會成為樹的葉節點。樹中會記錄查詢句中的扁平連結以及巢狀關係，每棵樹具有不同的階層碼，利用階層碼可以清楚得知樹之間的階層關係。

接著利用前面建立的樹和對應函數建立正規樹，正規樹會保留樹中的所有連結關係並記錄對應資訊。根據階層碼，正規樹會分成多個正規樹林並有順序性的轉換。而在建立正規樹時，不必要的節點會被刪除，若原始路徑包含跳層的話會加入節點。最後依對應函數加入對應資料到節點中，在多元對應的情況下會優先選擇權重高的資料。最後，從正規樹中可以得到所有的對應集合和值，利用最小生成樹演算法可以為集合產生合理且個數最少的連結條件。

本論文的貢獻條列如下：

1、以對應函數建立值、集合、結構的對應關係。針對多元對應，給予每個對應集合和對應值權重，並依權重由高到低排序資料。在多元對應的情況下會優先輸出權重高的資料。

2、利用樹結構以及階層碼處理查詢句。根據查詢句產生樹，樹中的階層碼用來區分外部查詢句和子查詢句產生的樹。而正規樹會保留樹中的所有連結關係並記錄對應資訊。在多元對應的情況下會優先選擇權重高的資料。

3、利用最小生成樹演算法，為正規樹中的集合產生合理且個數最少的連結

條件。

4、 實作查詢句轉換系統。實際開發此轉換系統，並證明轉換系統可以正確及有效率的轉換查詢句。

1.3. 相關研究

建立本論文之查詢句轉換系統，所牽涉的相關研究可分為不同格式資料間的對應、如何處理 XQuery 中的巢狀表示式、跳層與遞迴路徑以及等價查詢句。

針對資料間的對應問題，[ABJ+07]的目標是使用邏輯表示法（logical expressions）表示來源綱要（Schema）和目標綱要之間的對應關係。其作法是先用 conceptual model（CM）來表示每個表格的語意（semantics），並將 CM 轉換成圖形。接著利用這篇論文提出的演算法找尋圖形的子圖形（subgraph）。如果找到一組語意類似的來源（source）和目標（target）子圖形 ，則這組子圖形就是 conceptual mapping 的正規者（Canonical），得到的正規者最後會被轉變成資料庫層級（database level）的表示法。

而有時我們會利用關聯式資料庫產生 XML View，所以兩者之間的對應也必須處理，其中[BGW+07]主要探討 XML view 進行更新（update）時，如何將其對應的更新結果回傳給 relation view。由於 XML view 允許遞迴定義，所以該論文利用 ATG（Attribute Translation Grammars）將 relation 的資料轉成 XML 文件格式，同時再將該文件轉換成 DAG，讓可以被分享的子樹只會出現一次。該論文並利用表格記錄元素的父子祖孫關係，所以可以根據表格的記錄進行 XML view 的更新。更新時所產生的副作用則利用拓撲順序表（topological order list）和可相連矩陣（reachability matrix）這兩個輔助結構解決，拓撲順序表會拓撲排序

（topological sort）包含 DAG 中所有非重複（distinct）的元素，而可相連矩陣會記錄 DAG 中的祖孫關係。

另外，目前 XML 文件的資料交換在將資料從來源綱要（source schema）轉到目標綱要(target schema)時，來源綱要和目標綱要都是事先定義好的。[MPW08]提出了資料交換時 data-metadata translation 的問題還有解決方法，並實作出 data-metadata translations 系統，系統從現有的綱要對應語言延伸處理 data-metadata，並延伸現有演算法來產生綱要對應。[ALM09]則探討 XML 綱要間的對應，他們提出表示對應的語言，並探討相關的理論問題，包含兩者之間是否存在對應、時間複雜度，以及對應資料能否組合。至於[ATV08]則建立 STMark 評估所有對應系統的基本功能，其主要是根據操作的複雜度評估對應系統的使用介面是否友善，此外，STMark 可以根據輸入的綱要建立複雜的對應和產生大量資料（instance）。

以下則列舉和查詢句轉換相關的論文。[DPX04]提出 XQuery 的查詢最小化技術，XQuery 中可以包含巢狀結構、多變的 join 以及混合語意(mix semantics)，論文的最小化演算法避免了巢狀子查詢句中多餘的造訪(navigation)，偵測 group-by 是最小化技術的重要應用。[FCB07]探討如何處理轉換查詢句（transform query）。查詢句會在依據原始 XML 文件複製出來的 XML tree 上執行，而不會影響到原先的 XML tree。此論文利用自動機技術將轉換查詢句的 XPath 用 selecting NFA 標記，再由自動機告知何時要做 update，所以不用處理 XML 文件中的所有路徑。而[ABM+07]在 ULoad 上用具體化 view 重寫查詢句，其中查詢句和 views 都用 pattern laguage 描述，同時我們以結構化 summary 來限制 pattern 的內容以及判斷重寫是否正確。在重寫查詢句時， 查詢句會先用一個 tree pattern 或多個 tree pattern 的連結表示式表示。接著每個 tree pattern 會分別被重寫成代數表示法（algebraic expressions），最後 cost-based query optimizer 會用連結運算元

結合一些重寫的查詢句，並輸出重寫結果。

針對 XPath 中的遞迴路徑，[FYL+05]利用 recursive with 語法解決 XPath 中的遞迴路徑，論文中會將 XPath 轉換成包含最小不動點（least fixpoint）的 SQL 查詢句。在轉換查詢句的過程時，一開始先把 XPath 改寫成正規表示式，正規表示法可以表示 XPath 中遞迴路徑，接著用最小不動點將正規表示式改寫成 SQL，最後最佳化會減少最小不動點的個數。[FYL+09]延續[FYL+05]的研究，在轉換查詢句時， XPath 會改寫成延伸 XPath 表示法（extended XPath expressions），延伸 XPath 表示法是一個包含所有符合路徑的 XPath 表示法，延伸 XPath 表示法以動態規劃（dynamic programming）為基礎描繪符合路徑的特性。最佳化則會為最小不動點的運算效能加速，並減少不必要的結構連結。[K04]將 XQuery 轉換成單一的 SQL 並最佳化。除此之外，透過事先計算 XML 的資料限制，可以產生有效率的 SQL。

在關聯式資料系統中，將資料集合分散成多個部分可以有效改善 scalability，而 XML 資料需要以不同的分散式技術達到 scalability。[KOD09]以在地化（localization）和刪減（pruning）處理分散式 Qurey 並最佳化 XML 資料。在地化會將查詢句轉換成分散式查詢句 plan，並將分散式查詢句 plan 放入擁有相關 XML 資料的分散式系統執行，而不需要的分散式查詢句則會被刪減。

對於查詢句是否等價， [D09]針對混合了巢狀結構和未排序集合（unorder collection）的查詢句決定等價，論文中首先用編碼的方式將巢狀物件變成扁平化 relation，將問題轉變成決定關聯式連結查詢句的等價，接著整合之前的等價研究，論文以正規化型式（normal form）決定編碼查詢句（encoding queries）的等價。至於[CG09]則處理具有嵌入式相依（embedded dependencies）的 SQL CQ（conjunctive queries），並將其最小化。論文中用公式定義嵌入式相依 SQL 的正確性，並提出演算法重寫查詢句（Query-Reformulation），將具有嵌入式相依

的 SQL 最小化。

1.4. 論文架構

本篇論文架構如下：在第二章針對關聯式資料與 XML 格式資料的表示法做介紹與比較，並定義處理的語法範圍以及何謂正確的轉換。第三章說明對應資料表示法，以及相關的定義與內容。第四章定義查詢樹以及介紹樹的建立演算法。第五章介紹轉換系統模組和查詢轉換處理。第六章說明 SQL 轉換 XQuery 的處理方法，第七章的實驗評估轉換系統的正確性以及轉換效率，並證明轉換的正確性。最後在第八章提出本篇論文的結論與未來展望。

第 2 章　相關定義

本章介紹關聯式綱要與 XML 綱要，並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，最後定義處理的綱要範圍、語法範圍以及何謂等價的查詢句。

2.1. 關聯式綱要與 XML 綱要

首先我們介紹關聯式綱要。圖 2.1 是依據 TPC-H Benchmark[TPCH] 所適度修改與簡化欄位個數的關聯式資料範例，TPC-H Benchmark 為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。論文中以矩形表示關聯式表格，每個矩形從上到下分成三個部分，最上方標示表格名稱，中間標示主鍵，最下面則記錄表格的其餘屬性，而外來鍵會以箭頭指向參考的表格主鍵。例如 PARTSUPP 表格的主鍵為 PARTKEY 和 SUPPKEY，另有一個屬性為 AVAILQTY，而 PARTKEY 為外來鍵會指向 PART 表格的主鍵 PARTKEY。

圖 2.1 中共有九個表格，其中表格 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及顧客類別；而 CUSTEL 記錄了顧客的電話資料，此表格是正規化後從顧客資料中所獨立出來的表格；ORDER 表格記錄訂單的內容，包含了訂單的處理狀態與總金額；此外， 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 SHIPEODE 為運送方式；NATION 與 REGION 表格則是記錄了供應商與顧客的地理位置。INTRODUCE 表格則記錄了顧客間的引薦關係，顧客 CUSTKEY1 介紹了顧客 CUSTKEY2 到供應系統中。

表格中的主鍵（Primary Key）具有唯一性，是尋找或查詢資料的依據，主鍵又可分為單鍵（Simple Key）和組合鍵（Composite Key）。單鍵只包含一個屬性，而組合鍵包含多個屬性；外來鍵（Foreign Key）則會參照其它表格的主鍵，外來鍵中記錄的資料必定存在於參照的主鍵中，不同的表格可以利用相同意義的鍵值連結。在關聯式資料庫中，表格可區分成「實體表格」和「關係表格」。實體表格由多組相同型態的實體所組成，利用主鍵可以區分不同的實體，如 SUPPLIER 表格用 SUPPKEY 主鍵區分不同的實體。關係表格則記錄多個實體表格間的關係，並利用外來鍵連結實體表格，例如 PARTSUPP 表格連結 SUPPLIER 表格和 PART 表格。

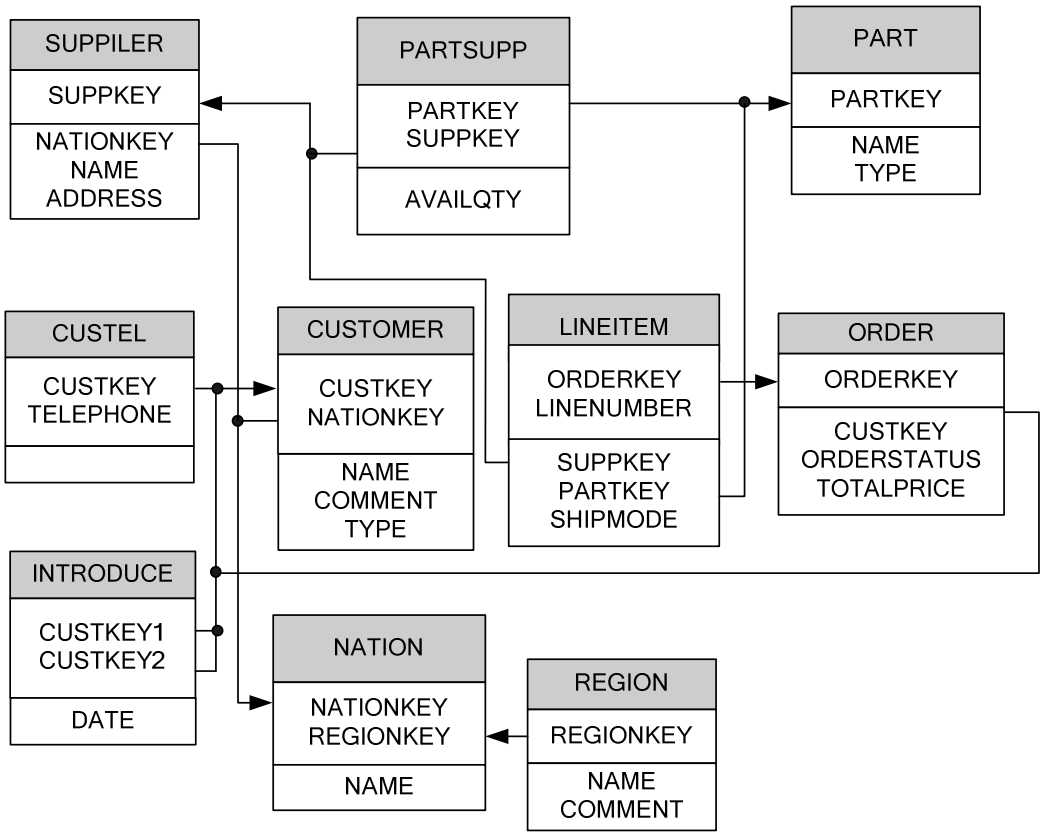


圖 2.1 關聯式資料範例

我們接著介紹 XML 資料表示法。XML （eXtensible Markup Language）文件是一種簡單、彈性很大的純文字格式檔案。XML 多用來做為資料交換的格式，亦可當作資料庫儲存資料。XML 屬於半結構化資料（Semi-Structured Data），使用者可以自行定義節點標籤以表達內容資料之意義，因此 XML 資料的表示法有較多的彈性；相較之下關聯式資料為結構化資料（Structured Data），具有固定的格式結構。

為了表示某一個 XML 文件合法的結構， W3C 組織（World Wide Web Consortium）頒訂文件型別定義 DTD（Document Type Definition）以定義與規範每一個節點的子節點以及屬性。為了解說方便，在論文中我們用 DTD Graph 表示 DTD Schema。DTD Graph 中的節點等同 DTD Schema 中的元素，DTD Graph 如圖 2.2、圖 2.3 所示。圖 2.2、與圖 2.3 是針對加拿大多倫多大學 Clio 計畫中的 TPC-H Nested XML Schema [CLIO]範例修改而來，圖 2.2 記錄顧客與供應商間的貨品訂購資訊，以及顧客間的引薦關係。圖 2.3 則是記錄顧客的資料與地理資訊，其中 VIP 顧客和一般顧客位於不同的 customer 節點下，右子樹的 region 和 region_popu 元素記錄地理位置資訊，nation 與 region、region_popu 元素分別對應到關聯式表格 NATION 與 REGION。

在 DTD Graph 中，我們以矩形代表元素，根節點以外的所有的矩形皆為「內部節點」。若矩形的右上方有 “*” 或 “+” 代表為「可重覆節點」，星號 “ * ” 代表元素可出現零次以上；“+” 代表元素可出現一次以上。橢圓代表「葉節點」，葉節點分為「值節點」（Value Node）和「屬性節點」（Attribute Node），其中虛線橢圓代表屬性，實線橢圓代表 PCDATA 型態的元素，圖中以虛線連結相同意義的葉節點。值節點右上方亦可用“ * ”或 “+” 表示該值節點為可重覆節點。而兩元素若在圖中為父子或祖孫關係，則稱兩元素具有「巢狀結構」。

根節點可以沒有對應的關聯式表格，而每個可重覆節點都有對應的關聯式表

格。「遞迴節點」則是迴圈中被箭頭指向的節點。例如圖 2.2 中有箭頭指向 customer 節點，因此 customer 為遞迴節點。「空節點」（Dummy Node）在 XML 文件中，針對同一個父節點，只能出現一次。空節點可出現在可重覆節點的上方或下方。若空節點在節點上方，由於此空節點的功能為包覆下方節點資料，但其非可重覆節點，所以不能與子節點對應到相同表格，如圖 2.2 的 suppliers 節點。若空節點在節點下方，由於空節點的功能為包覆下方節點的資料，其意義與父節點相同，因此空節點與父節點的對應表格相同，如圖 2.3 的 popu_data 節點。若空節點的上方和下方節點皆為可重覆節點，則與其父節點的對應表格相同。

另外需注意的是，DTD 中的“or”關係會在 DTD Graph 中展開，在下查詢句的時候，若查詢句中同時出現 “or”關係中的兩個節點，則系統會告知使用者查詢句不合理。參考[KOD09]的表示式，具有“or”關係的節點會在節點會在右上方標示 “or”字串。

以圖 2.2 為例，此範例在文件根節點（order-ship）下分成了兩個子樹，左子樹記錄了所有供應商（supplier）、零件（part）與訂單（order）相關的資料，其中左子樹的 suppliers 節點是「空節點」。supplier 為「可重覆節點」，其子節點 name 為「值節點」。Supplier 和 part 節點為父子關係、supplier 和 order 節點為祖孫關係，所以為巢狀結構。order-ship 的右子樹記錄了顧客（customer）的資料，customer 節點與 introduce 節點形成迴圈，其中被箭頭指向的 customer 節點為「遞迴節點」。右子樹與左子樹以虛線連結屬性 ckey 以及 nkey。

而圖 2.3 的 vip 節點和 normal 節點亦為「空節點」，這兩個空節點將顧客分為重要顧客和一般顧客，兩個意義相同的 customer 節點之間以虛線連結屬性 ckey，而空節點 popu_data 的上方節點為可重覆節點，因此 popu_data 和父節點 region_popu 節點對應到相同的表格。

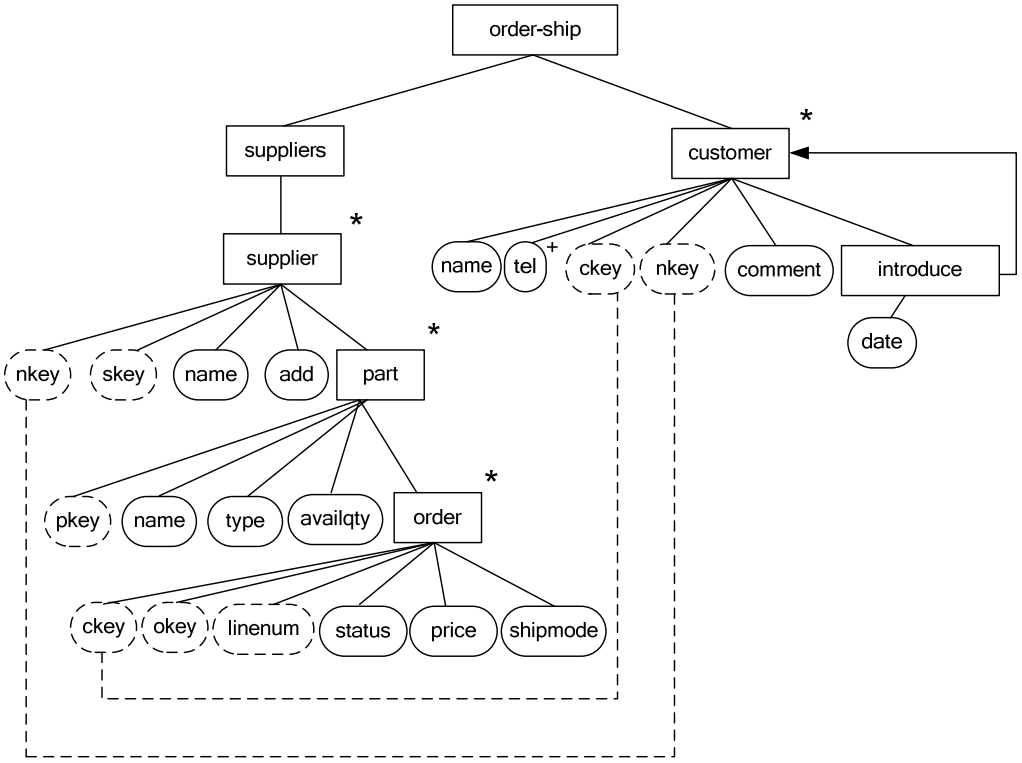


圖 2.2 Order-Ship DTD Graph 範例

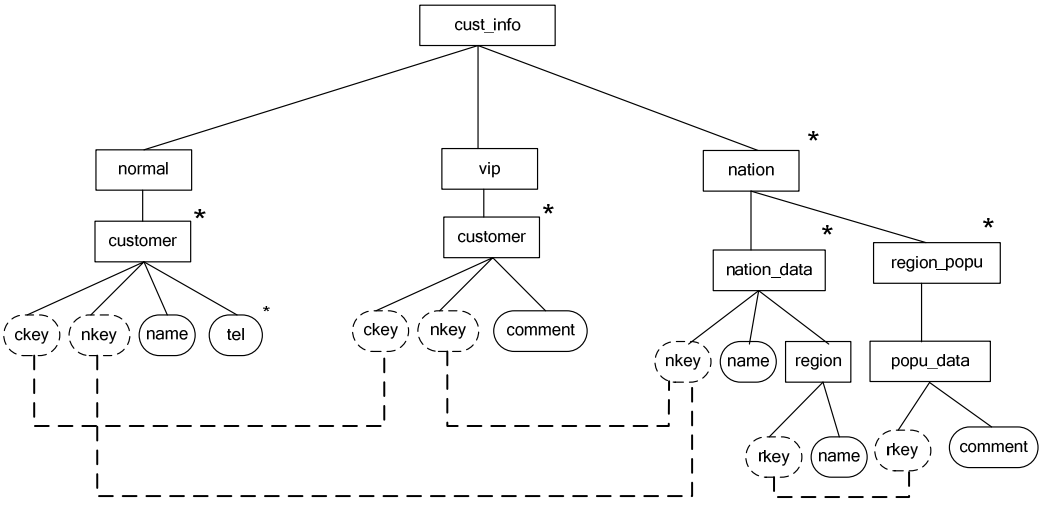


圖 2.3 Cust-Info DTD Graph 範例

以下說明關聯式綱要與 XML 綱要之間「值」和「集合」的多元對應情形。在關聯式綱要中以表格欄位為值、表格為集合，而在 XML 綱要中，葉節點為值、內部節點為集合。內部節點對應到表格，如 supplier 節點對應到 SUPPLIER 表格。葉節點對應至表格欄位，而可重覆的值節點會同時也對應到表格。例如圖 2.2 中 supplier 底下的葉節點對應到 SUPPLIER.NAME，tel 葉節點右上方的 “*”代表此節點為可重覆的值節點，所以 tel 節點會對應到 CUSTEL 表格以及 CUSTEL.TELEPHONE 欄位。

針對值的多元對應，值的多元對應的狀況如下：

1．欄位與葉節點多對一：以圖 2.1 為例， CUSTOMER 表格和 CUSTEL 表格的 CUSTKEY 欄位皆對應到圖 2.2 customer 節點底下的 ckey 葉節點。這是因為 CUSTOMER 表格經過正規化後，顧客的電話資料會記錄在 CUSTEL 表格，而 CUSTOMER 表格和 CUSTEL 表格透過 CUSTKEY 鍵值連結。所以表格正規化會造成鍵值的多元對應。

2．欄位與葉節點一對多：如圖 2.1 的 CUSTOMER.CUSTKEY 欄位會對應到圖 2.3 中兩個 customer 節點底下的屬性節點 ckey。這是因為存在多個意義相同的節點，所以表格欄位會對應到多個葉節點。

針對集合的多元對應，集合的多元對應的狀況如下：

1、 表格與節點多對一：以圖 2.2 的 customer 節點與圖 2.1 的 CUSTOMER 和 CUSTEL 表格為例，CUSTOMER.CUSTKEY 以及 CUSTEL.CUSTKEY 皆對應到 customer@ckey，因此 CUSTOMER 和 CUSTEL 兩個表格對應到 customer 節點，這是由於鍵值的多元對應造成表格與節點多對一。

2、 表格與節點一對多：以圖 2.1 的 PARTSUPP 表格與圖 2.2 的 supplier 節點及 part 節點為例，PARTSUPP.SUPPKEY 對應到 supplier 節點的 skey；而

PARTSUPP.PARTKEY 對應到 part 節點的 pkey，因此 PARTSUPP 表格對應到 supplier 和 part 兩個節點。這是因為關係表格對應到 XML 文件中的巢狀結構。

而同一路徑上的空節點和其父節點或子節點也會對應到同樣的表格，例如圖 2.3 圖 2.2 的 region_popu 節點和 popu_data 節點皆對應到圖 2.1 的 REGION 表格。除此之外，不同路徑上的節點可能會對應到相同的表格名稱。如圖 2.1 的 CUSTOMER 表格對應到圖 2.3 中的兩個 customer 節點，這是因為存在多個意義相同的節點，所以表格會對應到多個節點。

以下用數學表示法說明本論文處理的 RDB 和 DTD 範圍。參考[KOD09]的表示法，以（N,E,R,Q）代表 DTD 文件以及 RDB。當代表 DTD 文件時，N 為樹的所有元素集合，E 為元素之間的所有 Edge 集合，$E \subseteq N \times N$，R 為樹的根節點，$R \in N$。對每一個元素 n ，$n \in N$，Q（n）∈ {ONCE,*,+,?,null}， “ONCE”代表只出現一次，“＊”代表元素可出現零次以上； “+” 表示代表元素可出現一次以上，“?”代表出現零次或一次，null 代表為空節點。而當代表 RDB 時，N 為所有的表格集合以及所有的表格欄位集合，E 為表格與表格欄位的父子關係以及表格之間的連結，由於 RDB 非樹狀結構，因此 R 為 null，表格以及表格欄位在 RDB 皆只出現一次，因此 Q（n）∈ { “ONCE” }代表出現次數為 1。

2.2. XQuery 與 SQL

我們在此節說明關聯式資料和 XML 資料對應的查詢語言。首先，SQL 是關聯式資料庫管理系統的標準查詢語言， SQL 資料查詢句可分為 SELECT、FROM、WHERE 三個子句，SELECT 子句表示輸出的表格欄位名稱，FROM 子句表示提供資料的來源表格，WHERE 子句對表格欄位給定限制條件。在關聯式資料庫中， FROM 子句和 WHERE 子句中可包含子查詢句，FROM 子句中的子查詢句不可直接使用外

部查詢句的來源表格。

範例 2.1 是 FROM 子句有子查詢句的 SQL 查詢句。參考圖 2.1 的關聯式資料範例，此範例的目的為列出每一家製造商中單筆訂單金額大於一萬元的顧客。在範例 2.1 的外部查詢句裡，SELECT 子句取出供應商名字以及子查詢句所取得的顧客名字，FROM 子句限定資料來源為 SUPPLIER、LINEITEM 表格以及名稱為 a 的子查詢句，子查詢句會取出訂單金額大於 1 萬元的顧客姓名。在子查詢句裡， SELECT 子句取出顧客名字和訂單編號，訂單編號會用來連結子查詢句和外部查詢句。From 子句限定資料來源為 ORDER 和 CUSTOMER 表格； L05~L06 的 Where 子句利用 CUSTKEY 建立表格間的關聯，並限定訂單金額大於一萬。L08~L09 則在外部查詢句中用 ORDERKEY 和 SUPPKEY 建立表格間的關聯。

範例 2.1

```
L01  Select SUPPLIER.NAME,a.NAME
L02  From SUPPLIER, LINEITEM（
L03       Select CUSTOMER.NAME, ORDER.ORDERKEY
L04       From ORDER,CUSTOMER
L05       Where     ORDER.CUSTKEY = CUSTOMER.CUSTKEY
L06                 AND ORDER.TOTALPRICE>10000
L07       ）as a
L08  Where    a.ORDERKEY= LINEITEM.ORDERKEY
L09          AND SUPPLIER.SUPPKEY = LINEITEM.SUPPKEY
```

在查詢 XML 資料方面，W3C 制定了 XQuery 查詢語言，XQuery 依據 DTD 定義下達合法的路徑表示法（XPath Expression）。路徑中以 “/” 符號代表父子關係， “//” 符號為祖孫關係。 “//” 符號會用來取出所有符合條件的節點。以圖 2.2 為例， /order-ship/suppliers/supplier//name 會取出 supplier 節點之下所有的 name 葉節點，也就是會取出 /order-ship/suppliers/supplier/name 以及

/order-ship/suppliers/supplier/part/name。

XQuery 敘述句可視為一組 FLOWR 表示法，FLOWR 為 FOR-LET-ORDERBY-WHERE-RETURN 的縮寫，本論文不處理 ORDERBY 子句。在 FLOWR 表示法中，FOR 子句利用路徑表示法依序取得對應的元素。LET 則對應到資料集合。LET 子句可由變數、XPath 或子查詢句組成。RETURN 子句不僅可回傳一般的 XPath 路徑，也可包含子查詢句。WHERE 子句則根據限制條件取出符合條件的資料。

若 XQuery 中不包含子查詢句，整個查詢句稱為外部查詢句。子查詢句為一完整的 XQuery 查詢句，並可使用外部查詢句的變數。範例 2.2 是 let 子句中具有子查詢句的 XQuery。範例 2.2 參考圖 2.2 列出每一家製造商中單筆訂單金額大於一萬元的顧客。在範例 2.2 的 XQuery 中， L01 的 For 子句以變數$s 取出 /order-ship/suppliers/supplier 路徑所代表的節點內容集合。L02~L06 的 Let 子句包含了子查詢句，變數 a 會連結此子查詢句，此子查詢句會取出訂單金額大於 1 萬元的顧客姓名。L02 建立$o 與$s 的巢狀關係， $o 為$s 的孫節點 order 的內容集合，L03 取出顧客的資料。L04 的 WHERE 條件句利用 ckey 和 nkey 建立連結關係。L05 則限定只取出訂單金額大於一萬的資料。L06 回傳顧客姓名。處理完子查詢句，L07 回傳製造商名稱及子查詢句取得的顧客名稱。

範例 2.2

```
L01  FOR    $s in /order-ship/suppliers/supplier
L02  Let $a:=  For        $o in $s//order,
L03                       $c in /order-ship/customer
L04            Where      $o@ckey=$c@ckey
L05                       And $o/price >10000
L06            Return $c/name
L07  RETURN $s/name,$a/name
```

以 XQuery 與 SQL 相比較，XQuery 的 FOR 子句大致相當於 SQL 查詢語言

中的 FROM 子句，兩者皆是定義資料來源；XQuery 的 RETURN 子句則相當於 SQL 查詢語言中的 SELECT 子句，兩者皆為輸出資料；XQuery 的 WHERE 子句則對應到 SQL 查詢語言中的 WHERE 子句，其中 where 子句又可分為「文數值限制」和「連結限制」,「文數值限制」限定了屬性內容,「連結限制」則限定結構關係,在 XQuery 可以用巢狀結構直接限制可重覆節點之間的關係,而在 SQL 則是利用主鍵和外來鍵限制表格間的連結。例如範例 2.1 L05 的連結限制限定 ORDER 表格和 CUSTOMER 表格以 CUSTKEY 連結，L06 的文數值限制限制 ORDER 表格的 TOTALPRICE 要大於 10000，而範例 2.2 的 L02 的巢狀結構限定 $o 為$s 的孫節點。針對子查詢句，XQuery 的子查詢句為 Let 子句或 Return 子句中的 FLOWR 表示式；而在關聯式資料庫中，子查詢句為 FROM 子句或 where 子句中的 SQL 查詢句。

2.3. 查詢句轉換的文法範圍與定義

本論文可轉換的 XQuery 與轉換出來 SQL 之精簡 Backus-Naur Form(BNF) 文法範圍如圖 2.5 與圖 2.4 所示。從圖 2.4 的 XQuery 文法範圍看出在 Let 子句子句中可具有 FLOWR 表示式，本論文不討論 return 子句中有子查詢句的狀況。XQuery 可處理的 XPath 除了基本的 PC（Parent-Child）路徑外，亦可處理 AD（Ancestor-Descendant)路徑。而圖 2.5 的 SQL 文法範圍的 FROM 子句和 WHERE 子句可具有 SQL 表示法。

```
FLOWR::=ForClause+ LetClause?+ WhereClause?+ ReturnClause
ForClause ::= “For ”（Var “in” Expr “,”?）+
LetClause::= “LET”（Var “::=” Ei “,”?）+
WhereClause ::= “WHERE"（Wi LogicalOp?）+
ReturnClause::= “Return”（Expr “,”?）+
Expr::=pathExpr
      |Var pathExpr
Ei::=FLOWR|Expr
```


```
Wi::= Expr CompOP Literal
        | Expr CompOp Expr
Var::= “$”Literal
Literal::=String| Digits….
pathExpr::= Axis RegularExpr
RegularExpr ::= RegularExpr Axis Step| Axis Step
Axis ::= “/” | “//”
Step::=”@”?Name|”*”
CompOp::= “=” | “!=” | “<” | “<=” | “>” | “>=”
LogicalOp::= “and ” | “or”
String ::= [A-Za-z]（[A-Za-z0-9._] | '-'）
Digits ::= [0-9]+
```

圖 2.4 XQuery 文法範圍

```
SQL ::= SelectClause + FromClause + WhereClause?
SelectClause :: = “SELECT”（Ei “,”?）+
FromClause :: = “FROM“（Ri “,”?）+
WhereClause :: = “WHERE“（Wi LogicalOp?）+
Ri::= Relation | “（” SQL  “）as” Var
Ei ::= Relation.Attribute
Wi ::=  Ei CompOP Value
        | Ei CompOp Ei
        | Ei “in” SQL
Var ::=String
CompOp::= “=” | “!=” | “<” | “<=” | “>” | “>=”
LogicalOp::= “and ” | “or”
```

圖 2.5 轉換出來的 SQL 文法範圍

本論文設計 XQuery 轉換成 SQL 的系統，為了驗證其是否能轉換出正確的查詢句，我們提出查詢句的等價定義。若轉換前後有正確對應且個數相同的值，同時有對應的集合和合理結構，則稱為等價。等價中不要求相同個數的集合，這是因為 XQuery 的巢狀關係會對應到 SQL 的扁平連結，且對應的集合個數不定。

在本論文中，若查詢句中不包含子查詢句，整個查詢句稱為外部查詢句。在以下的定義中，定義 2.1、定義 2.2 為外部查詢句的等價定義，定義 2.3 為子查詢句的等價定義。定義 2.4 則整合定義 2.1 到定義 2.3。

- 【定義 2.1】轉換前與轉換後的查詢句 Q 和 Q'，其值、集合個數相同，且分別有正確的對應關係，而轉換後的集合有適當的結構關係，則 Q 和 Q’等價。

- 【定義 2.2】轉換後的查詢句 Q’，其集合個數與轉換前的查詢句 Q 集合個數不同，但轉換後的集合有適當的結構關係，且轉換前後輸出的值個數相同且等價，則 Q 和 Q’等價。

若集合之間可形成樹結構則稱為具有適當的結構關係。當轉換後的查詢句具有所有集合之間的結構關係時，其查詢結果必定正確，而拿掉會形成迴圈的結構後，所得到的查詢結果會和未拿掉迴圈前相同，因此集合之間的結構若形成樹結構則具有適當的結構關係。

以下以圖 2.6 說明樹結構為何是適當的結構關係。圖 2.6 為圖 2.1 中部分表格形成的 ER 圖，圖中以矩形代表實體表格，如 SUPPLIER。而菱形代表關係表格，如 PARTSUPP。圖中的實線代表格間的連結，線上記錄用來連結的表格欄位，如 SUPPLIER 和 PARTSUPP 表格透過 SUPPKEY 欄位連結。

若使用者輸入 XQuery 查詢句，且轉換後輸出的集合為圖 2.6 的 5 個表格，從圖中可發現表格間共有六筆連結資料，若輸出所有的連結表示式則查詢結果必定正確。而在去掉 SUPPLIER、LINEITEM，以及 PART、LINEITEM 之間的連結之後，輸出的連結資料為 4 筆且集合之間會形成樹結構，查詢句查詢結果會與具有六筆連結資料時相同，故集合之間的結構若形成樹結構則具有適當的結構關係。

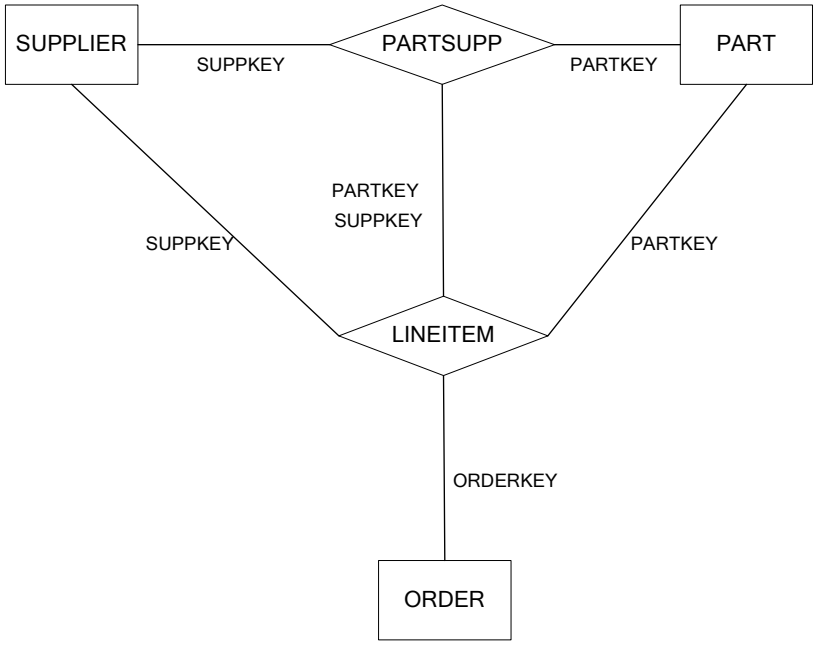


圖 2.6 圖 2.1 中部分表格的 ER 圖

定義 2.3 為子查詢句的等價定義。內容如下所示。

◆ 【定義 2.3-1】子查詢句轉換後需符合定義 2.1 或定義 2.2，集合個數為子查詢句中宣告的變數個數。除此之外，子查詢句會輸出連結條件句中和外部查詢句相關的表格欄位。輸出此表格欄位是為了連結轉換後的子查詢句與外部查詢句。而子查詢句的文數值限制轉換後，文數值限制出現在外部查詢句或子查詢句皆正確。

◆ 【定義 2.3-2】外部查詢句轉換後會輸出連結外部查詢句和子查詢句的連結條件句。

綜合以上的定義，我們提出以下完整的等價定義。

◆ 【定義 2.4】

假設一個查詢句 Q，其外部查詢句為 MQ，而其子查詢句集合為{SubQ}，轉換後的查詢句 Q'包含 MQ'和{SubQ'}，MQ'與 MQ 等價（符合定義 2.1 或定義 2.2），每一個 SubQ 和 SubQ'等價（符合定義 2.3-1），且 MQ'和 SubQ'有連接關係（符合定義 2.3-2），則稱 Q 和 Q'等價。

在等價定義中，我們認為轉換前後若有相同個數且對應的值，且有對應的集合及結構，則稱為等價，對應的集合個數

以範例 2.2 轉換成範例 2.1 為例，範例 2.2 的子查詢句集合$o、$c 轉換後對應到範例 2.1 的 ORDER、CUSTOMER 表格，值 $c/name 轉換後為 CUSTOMER.NAME，where 子句中的 $o/price >10000 轉換成 ORDER.TOTALPRICE>10000，子查詢句中以 ORDER.CUSTKEY = CUSTOMER.CUSTKEY 建立集合間的結構連結，綜合以上得知轉換前後的查詢句，其值、集合個數相同，且分別有正確的對應關係，而轉換後的集合有適當的結構關係，因此符合定義 2.1 的等價定義。除此之外，子查詢句會回傳連結條件中的 ORDER.ORDERKEY 以連結外部查詢句。以上對應到定義 2.3-1。

而子查詢句轉換後需輸出連結外部查詢句和子查詢句的連結條件句。由於 SUPPLIER 和 a 之間透過 LINEITEM 表格連結，因此會輸出連結條件 a.ORDERKEY= LINEITEM.ORDERKEY，以上對應到定義 2.3-2。

範例 2.2 的外部查詢句，其值 $s/name、$a/name 轉換後分別為 SUPPLIER.NAME、a.NAME，集合$s、$a 轉換後對應到 SUPPLIER、a。由於 SUPPLIER 表格和子查詢句中的 ORDER 表格需透過 LINEITEM 表格連結，所以會輸出 LINEITEM。綜合以上得知轉換前與轉換後的查詢句，其集合個數與轉換前的查詢句集合個數不同，但轉換後的集合有適當的結構關係，且轉換前後輸出的屬性個數相同且等價，因此符合定義 2.2 的等價定義。

由以上可知，範例 2.2 的外部查詢句轉換後符合定義 2.2，範例 2.2 的子查詢句轉換後符合定義 2.3-1，且外部查詢句和子查詢句符合定義 2.3-2 有連接關係，因此範例 2.2 和範例 2.1 等價。

注意到，定義中並沒有限制文數值限制式出現的位置，以下用範例 2.3 的 XQuery 以及範例 2.4 的 SQL 說明子查詢句中的文數值限制轉換後出現在外部查詢句的情形，範例 2.3 轉換後會得到範例 2.4。範例 2.3 參考圖 2.2，查詢句會列出製造商單筆訂單金額大於一萬元的顧客，供應商名稱為“ABC”。在範例 2.3 的子查詢句中，where 子句限制訂單金額大於一萬且供應商的名稱要為 ABC。而在轉換後的範例 2.4 SQL 中，文數值限制不是出現在子查詢句，而是出現在 L08 的外部查詢句。因為會發生此種情況，所以在定義 2.3-1 特別說明子查詢句的文數值限制轉換後，不管是出現在外部查詢句或子查詢句皆正確。

【範例 2.3】

```
L01  FOR    $s in /order-ship/suppliers/supplier
L02  Let $a:=  For        $o in $s//order,
L03                       $c in /order-ship/customer
L04            Where      $o@ckey=$c@ckey
L05                       And $o/price >10000
L06                       And $s/name= “ABC”
L07            Return $c/name
L08  RETURN $s/name,$a/name
```

【範例 2.4】

```
L01  Select SUPPLIER.NAME,a.NAME
L02  From SUPPLIER, LINEITEM（
L03       Select CUSTOMER.NAME, ORDER.ORDERKEY
L04       From ORDER,CUSTOMER
L05       Where     ORDER.CUSTKEY = CUSTOMER.CUSTKEY
L06                 AND ORDER.TOTALPRICE>10000
```

L07	) as a
L08	Where SUPPLIER.NAME= “ABC”
L09	AND a.ORDERKEY= LINEITEM.ORDERKEY
L09	AND SUPPLIER.SUPPKEY = LINEITEM.SUPPKEY

第 3 章　對應資料表示法

在本章中將說明綱要間對應資料的表示法。我們設計了數個對應函數來表示關聯式綱要與 XML 綱要的值、集合與結構的差異對應資訊，轉換系統會利用對應表示法來轉換查詢句。

3.1. 權重（Weight）及成本（Cost）

為了處理多元對應，我們給予每一個對應集合和對應值一個權重，權重高的資料會優先輸出。而集合間的結構連結則給予成本（Cost），成本低的會優先輸出。以下分別說明如何取得或計算集合、值的權重以及結構的成本。

針對集合的權重，每一個對應集合的權重依表格與節點的相似度而決定，當相似度越高則權重越高，在多元對應的情況下會優先選擇權重最高的對應集合。權重的計算公式為（葉節點對應的表格欄位個數/ 此表格欄位個數）。

以圖 2.1 與圖 2.2 為例，part 節點對應到 PART 表格、PARTSUPP 表格、LINEITEM 表格，其中 PART 表格有三個欄位 PARTKEY、NAME 以及 TYPE，這三個欄位分別對應到 part 節點的葉節點 pkey、name 以及 type，故權重為 3/3=1；PARTSUPP 表格則有三個欄位，其中 PARTKEY 和 AVAILQTY 欄位對應到 part 節點的 pkey 和 availqty，所以權重為 2/3；LINEITEM 表格的五個欄位中，只有 PARTKEY 在 part 節點下有對應的葉節點，故權重為 1/5。

針對值的權重，在[劉 06]中以值的型態做為權重的依據，例如表格欄位的權重為主鍵>外來鍵>其他。而本論文則將值的權重設成和其所在的集合權重相同，這是因為表格與節點的相似度越高，取出此表格的可能越高。而當節點同時為可重覆節點和遞迴節點時，其下葉節點的權重將會依據可重覆節點，這是因為我們將葉節點當作僅位於可重覆節點底下。

例如在圖 2.1 和圖 2.2 中， part@pkey 對應到 PART.PARTKEY 的權重和 part 節點對應到 PART 表格的權重同樣為 3/3。

針對結構的成本，在 DTD 中，由於沒有一組結構會同時是巢狀結構以及扁平結構，因此巢狀結構與扁平結構的成本相同，不需要記錄 DTD 的結構成本。而在關聯式資料庫中，結構的成本依表格的連結情況而定，若表格之間的連結條件越少則成本越低，為了用最少的連結條件連結集合，在結構有多元對應的情況下會優先選擇成本最低的結構。所以對應到同一個可重覆節點的兩個表格，其成本為 1，而不需透過關係表格的表格連結其成本為 2，其他情況則成本為 4。

以圖 2.1 與圖 2.2 為例，CUSTOMER 和 CUSTEL 表格對應到 customer 節點，因此其連結式為 “CUSTOMER.CUSTKEY=CUSTEL.CUSTKEY”，成本為 1。而 SUPPLIER 和 PARTSUPP 不需透過關係表格即可直接連結，因此連結式為 “SUPPLIER.SUPPKEY=PARTSUPP.SUPPKEY”，成本為 2。而 SUPPLIER 和 PART 因為需透過關係表格 PARTSUPP 連結，因此其連結式為{（SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY），（PARTSUPP.PARTKEY = PART.PARTSUPP）}，成本為 4。

3.2. 值對應表示法

我們以定義 3.1 的 Value Mapping 函數記錄關聯式資料庫表格欄位與 XML 端葉節點路徑的對應關係。

【定義3.1】：Value Mapping （簡稱:VM）

根據來源綱要給與值vi，VM（vi）會回傳目標綱要中與vi對應的值。

- vi=（XPath），VM（vi）= [（Rname, Aname, Weight）]。其中，XPath為與此欄位對應的DTD葉節點的路徑，Rname為表格名稱，Aname為表格的欄位名稱，

Weight則記錄此筆對應資料的權重，若VM（vi）有多筆對應資料時，資料會依Weight由大到小排序。各變數的意義同上，所以不再重覆。

VM 會針對每一個葉節點與表格欄位之間的關係建立對應資料。以下用表格說明建立的 VM，由圖 2.1 與圖 2.2 所建立的部份 VM 內容如表格 3.1 所示。完整的 VM 內容請參考附錄 A。為方便區分所指的資料列，表格 3.1、3.2 中新增欄位「列」，但實際上此欄位不存在 VM 中。

值的多元對應關係會建立至 VM 中，如表格 3.1 中的 11~14 列，/order-ship/customer@ckey 對應到 CUSTOMER.CUSTKEY、CUSTEL.CUSTKEY、INTROUDUCE.CUSTKEY1 以及 INTROUDUCE.CUSTKEY2，所以在表格 3.1 中會建立四筆資料記錄多元對應關係。customer 節點同時為可重覆節點和遞迴節點，由於我們將葉節點當作僅位於可重覆節點底下，因此不會記錄//customer@ckey 的資料。

列	XPath	Rname	Aname	RXPath	Weight
1	/order-ship/suppliers/supplier@nkey	SUPPLIER	NATIONKEY	/order-ship/suppliers/supplier	4/4
2	/order-ship/suppliers/supplier@skey	SUPPLIER	SUPPKEY	/order-ship/suppliers/supplier	4/4
3	/order-ship/suppliers/supplier@skey	PARTSUPP	SUPPKEY	/order-ship/suppliers/supplier	1/3
4	/order-ship/suppliers/supplier@skey	LINEITEM	SUPPKEY	/order-ship/suppliers/supplier	1/5
5	/order-ship/suppliers/supplier/name	SUPPLIER	NAME	/order-ship/suppliers/supplier	4/4
6	/order-ship/suppliers/supplier/add	SUPPLIER	ADDRESS	/order-ship/suppliers/supplier	4/4
7	/order-ship/suppliers/supplier/part@pkey	PART	PARTKEY	/order-ship/suppliers/supplier/part	3/3
8	/order-ship/suppliers/supplier/part/name	PART	NAME	/order-ship/suppliers/supplier/part	3/3
9	/order-ship/suppliers/supplier/part/type	PART	TYPE	/order-ship/suppliers/supplier/part	3/3
10	/order-ship/suppliers/supplier/part/order/shipmode	LINEITEM	SHIPMODE	/order-ship/suppliers/supplier/part/order	3/5

11	/order-ship/customer@ckey	CUSTOMER	CUSTKEY	/order-ship/customer	4/5
12	/order-ship/customer@ckey	CUSTEL	CUSTKEY	/order-ship/customer	2/2
13	/order-ship/customer@ckey	INTRODUCE	CUSTKEY1	/order-ship/customer	2/3
14	/order-ship/customer@ckey	INTRODUCE	CUSTKEY2	/order-ship/customer	2/3
15	/order-ship/customer/tel	CUSTEL	TELEPHONE	/order-ship/customer	2/2

表格 3.1 圖 2.1 與圖 2.2 的部分 Value Mapping 內容

3.3. 集合對應表示法

定義 3.2 的 Collection Mapping 記錄關聯式資料庫表格與 XML 端內部節點的路徑對應關係。

【定義 3.2】: Collection Mapping（簡稱:CM）

根據來源綱要給與集合ci，CM（ci）會回傳目標綱要中與ci對應的集合。

- ci={ xpath }，CM(ci)=[(Rname，Type，Weight，Condition ，LoopPaths)]。若節點為空節點，則CM不記錄。其中，XPath為表格對應的XML節點路徑，Rname為關聯式資料庫的表格名稱，Type為節點類型，Type分為Repeatable、Recursive，其中Repeatable為可重覆節點，Recursive為遞迴節點。轉換查詢句時會依不同的type而有不同的處理方式，而當type為Recursive 時，會以Recursive With的語法處理對應資料。Weight為記錄表格與節點的相似度，相似度越高Weight的值越大。若CM（ci）回傳多筆資料，資料列會依Weight由大到小排序。Condition記錄節點對應到表格時的條件限制。Condition記錄表格對應到節點時的條件限制，在轉換查詢句後會將條件限制加入where子句中。LoopPaths為當節點為遞迴節點時迴圈的路徑，用來判斷輸入的XPath是否合法。

由圖 2.1 與圖 2.2 所建立的 CM 如表格 3.2 所示。為方便區分所指的資料列，

表格 3.2 新增欄位「列」，但實際上此欄位不存在 CM 中。在 DTD Graph 中，葉節點若為可重覆的葉節點，則此葉節點亦為集合，VM、CM 皆會記錄。例如圖 2.2 中的 tel 葉節點為可重覆的葉節點，因此 tel 葉節點會在表格 3.1 VM 的第 15 列記錄/order-ship/customer/tel 對應 CUSTEL.TELEPHONE，並在表格 3.2 CM 的第 15 列記錄/order-ship/customer/tel 對應 CUSTEL 表格。

由於利用 XPath 表示法中的 “//”表示式可取出遞迴節點所有對應到的元素，因此遞迴節點的路徑為 “//” 加上節點名稱， LoopPaths 中會記錄迴圈路徑。以圖 2.2 為例， customer 節點同時為可重覆節點和遞迴節點，當 customer 節點為可重覆節點時，其 XPath 為 “/order-ship/customer “ ，如表格 3.3 的 9~11 列所示。而當 customer 節點視為遞迴節點時，XPath 為 “//customer “，對應表格和權重會與可重覆節點記錄的資料相同，如表格 3.3 的 12~14 列所示，但遞迴節點會在 LoopPaths 記錄迴圈路徑 “/introuduce/customer”。

再來以圖 2.1 與圖 2.3 為例說明 Condition 欄位，圖 2.1 中的 CUSTOMER 表格以 TYPE 欄位區分顧客的等級，CUSTOMER 表格中 TYPE 欄位內容為 “VIP” 的資料列會對應 vip 節點底下的 customer 節點，TYPE 欄位內容為 “Normal”的資料列對應Normal 節點底下的customer 節點。因此在表 3.3 中，列 1、列 2 的 XPath 為 cust_info/Normal/customer 的資料列，其 Condition 為 CUSTOMER.TYPE = “Normal” ，而列 3、列 4 的 XPath 為 cust_info/vip/customer，其 Condition 為 CUSTOMER.TYPE = “VIP”。

而在圖 2.3 中，由於空節點 popu_data 的上方節點為可重覆節點，因此 popu_data 和父節點 region_popu 節點會對應到相同的表格。表格 3.3 的列 8、列 9 中兩者皆對應到 REGION 表格，權重同為 2/3。

列	XPath	Rname	Type	Weight	Condition	LoopPaths
1	/order-ship/suppliers/supplier	SUPPLIER	Repeatable	4/4		
2	/order-ship/suppliers/supplier	PARTSUPP	Repeatable	1/3		
3	/order-ship/suppliers/supplier	LINEITEM	Repeatable	1/5		
4	/order-ship/suppliers/supplier/part	PART	Repeatable	3/3		
5	/order-ship/suppliers/supplier/part	PARTSUPP	Repeatable	2/3		
6	/order-ship/suppliers/supplier/part	LINEITEM	Repeatable	1/5		
7	/order-ship/suppliers/supplier/part/order	ORDER	Repeatable	4/4		
8	/order-ship/suppliers/supplier/part/order	LINEITEM	Repeatable	3/5		
9	/order-ship/customer	CUSTOMER	Repeatable	4/5		
10	/order-ship/customer	CUSTEL	Repeatable	2/2		
11	/order-ship/customer	INTRODUCE	Repeatable	2/3		
12	//customer	INTRODUCE	recursive	4/5		/introduce/customer
13	//customer	INTRODUCE	recursive	2/2		/introduce/customer
14	//customer	INTRODUCE	recursive	2/3		/introduce/customer
15	/ order-ship/customer/tel	CUSTEL	Repeatable	2/3		
16	/order-ship/customer/introduce	INTROUUCE	Repeatable	1/3		

表格 3.2 圖 2.1 與圖 2.2 所建立的 Collection Mapping

列	XPath	Rname	Type	Weight	Condition	LoopPaths
1	/cust_info/ Normal/customer	CUSTOMER	Repeatable	3/5	CUSTOMER.TYPE = “Normal”	
2	/cust_info/ Normal/customer	CUSTEL	Repeatable	2/2	CUSTOMER.TYPE = “Normal”	
3	/cust_info/VIP/customer	CUSTOMER	Repeatable	4/5	CUSTOMER.TYPE = “VIP”	
4	/cust_info/VIP/customer	CUSTEL	Repeatable	1/2	CUSTOMER.TYPE = “VIP”	
5	/cust_info/nation	NATION	Repeatable	2/3		
6	/cust_info/nation/region	REGION	Repeatable	2/3		
7	/cust_info/nation/population	REGION	Repeatable	2/3		
8	/cust_info/nation/region_popu	REGION	Repeatable	2/3		
9	/cust_info/nation/region_popu/popu_data/	REGION	DummyInLower	2/3		

表格 3.3 圖 2.1 與圖 2.3 所建立 Collection Mapping

3.4. 結構表示法

關聯式資料庫中，表格之間以相同意義的鍵值連結，論文中以 JE 記錄關聯式表格間的連結關係。

【定義3.3】：Join Expression （簡稱:JE）

根據關聯式綱要給與ji，JE（ji）會回傳綱要中與ji對應的資料。ji=（Table1, Table2），JE（ji）= [（RID,RelationShip,Weight,Conditions ）]。其中Table1和Table2為具連結關係的兩個表格之名稱。RID為連結限制式的編號，不同的編號代表不同的意義。編號若為EE_i則代表兩個實體表格間的連結，表格間透過關係表格連結，權重為4。編號若為RR_i則代表關係表格與關係表格的連結，表格間透過組合鍵連結，權重為2。編號若為IJ_i則代表當兩個實體表格對應到同一個可重覆元素的表格連結，權重為1。編號若為REC_i則代表遞迴節點的結構關聯對應到的表格連結，其中若為兩個實體表格間的連結，且表格間透過關係表格連結，則權重為4，其他

狀況則權重為2。編號若為R_i則代表除了上面分類以外的表格連結，權重為2。RelationShip為連結Table1和Table2的關係表格。RelationShip的內容值可能為空或為一個以上關係表格。Weight記錄多元對應時的權重，當JE（ji）取得多筆資料時會依權重由小到大排序。Conditions記錄兩個表格間的所有連結條件。

由圖 2.1 關聯式表格中可得知，SUPPLIER 與 PART 表格可經由 PARTSUPP 關係表格或 LINEITEM 關係表格連結，因此表 3.4 建立了兩筆資料，編號 EE3 透過 PARTSUPP 關係表格連結兩表格，而 EE4 透過 LINEITEM 關係表格連結。而 CUSTOMER 表格和 CUSTEL 表格皆對應到 customer 節點，因此表 3.4 JE 中以編號 IJ12 的資料列記錄此連結。而 PARTSUPP 和 LINEITEM 這兩個關係表格可以透過 SUPPKEY 以及 PARTKEY 欄位連結，因此編號 RR9 的資料列記錄此連結時會在 Conditions 欄位記錄兩筆資料。至於圖 2.2 中的遞迴節點 customer 則為特殊情形， JE 中會記錄遞迴節點的結構關聯對應到的表格連結，customer 節點對應到 CUSTOER、CUSTEL、INTRODUCE 三個表格，當遞迴節點對應到 CUSTOMER 表格時，兩個 CUSTOMER 表格會透過 INTRODUCE 表格連結，權重為 4，以編號 REC13 的資料列記錄。而當對應到 CUSTEL、INTRODUCE 表格時，也同樣為其在 JE 建立資料。

RID	Table1	Table2	RelationShip	Weight	Conditions
R1	SUPPLIER	PARTSUPP		2	SUPPLIER.SUPPKEY=PARTSUPP.SUPPKEY
R2	SUPPLIER	LINEITEM		2	SUPPLIER.SUPPKEY=LINEITEM.SUPPKEY
EE3	SUPPLIER	PART	PARTSUPP	4	SUPPLIER.SUPPKEY=PARTSUPP.SUPPKEY, PARTSUPP.PARTKEY=PART.PARTKEY
EE4	SUPPLIER	PART	LINEITEM	4	SUPPLIER.SUPPKEY=LINEITEM.SUPPKEY, LINEITEM.PARTKEY=PART.PARTKEY
EE5	SUPPLIER	ORDER	LINEITEM	4	SUPPLIER.SUPPKEY=LINEITEM.SUPPKEY ORDER.ORDERKEY=LINEITEM.ORDERKEY
R6	ORDER	CUSTOMER		2	ORDER.CUSTKEY=CUSTOMER.CUSTKEY
R7	ORDER	CUSTEL		2	ORDER.CUSTKEY=CUSTEL.CUSTKEY
R8	ORDER	LINEITEM		2	ORDER.ORDERKEY=LINEITEM.ORDERKEY
R8	ORDER	PART	LINEITEM	4	PART.PARTKEY =LINEITEM.PARTKEY ORDER.ORDERKEY=LINEITEM.ORDERKEY
RR9	PARTSUPP	LINEITEM		2	PARTSUPP.SUPPKEY=LINEITEM.SUPPKEY, PARTSUPP.PARTKEY=LINEITEM.PARTKEY
R10	PARTSUPP	PART		2	PARTSUPP.PARTKEY=PART.PARTKEY
R11	PART	LINEITEM		2	PART.PARTKEY=LINEITEM.PARTKEY
IJ12	CUSTOMER	CUSTEL		1	CUSTOMER.CUSTKEY=CUSTEL.CUSTKEY
REC13	INTRODUCE	INTRODUCE		2	INTRODUCE.CUSTKEY1= INTRODUCE.CUSTKEY2
REC13	CUSTOMER	CUSTOMER	INTRODUCE	4	CUSTOMER .CUSTKEY= INTRODUCE.CUSTKEY1, INTRODUCE.CUSTKEY2=CUSTOMER .CUSTKEY
REC14	CUSTEL	CUSTEL	INTRODUCE	4	CUSTEL.CUSTKEY= INTRODUCE.CUSTKEY1, INTRODUCE.CUSTKEY2=CUSTEL.CUSTKEY

表格 3.4 圖 2.1 產生的 Join Expression

第 4 章　查詢樹

在本論文中，我們設計 CVForest 和 CanForest 兩種查詢樹的結構，以便處理子查詢句的複雜結構與相關語意。我們會先提出查詢樹的範例，給予正式的定義，再介紹對應的演算法。

4.1. 範例

本論文會以範例4.1以及範例4.2說明建立CVForest以及CanForest的過程。範例4.1是由範例2.2產生而成的CVForest，範例4.2則是根據範例4.1產生的CanForest。範例2.2轉換後的SQL查詢句如圖5.10所示，圖5.10與範例2.1為相同意義的查詢句，但是每個表格會有自己的變數。為了方便對照，我們在此把範例2.2的XQuery再列舉一遍：

範例 2.2

L01	FOR $s in /order-ship/suppliers/supplier
L02	Let $a:= For $o in $s//order,
L03	$c in /order-ship/customer
L04	Where $o@ckey=$c@ckey
L05	And $o/price >10000
L06	Return $c/name
L07	RETURN $s/name,$a/name

論文中以「樹 “變數名稱”」代表依 “變數名稱”產生的樹根節點。例如集合樹 “s”代表變數名稱 “s”產生的集合樹根節點。並以「簡單路徑」代表不含跳層和變數的路徑，「絕對路徑」為查詢句產生的路徑，絕對路徑可包含跳層或變數，「相對路徑」為從非根節點開始的路徑，路徑中可包含跳層。例如/order-ship/supplier/part/order為簡單路徑。/order-ship/customer@ckey、$s/name和/order-ship/supplier//order為絕對路徑，@ckey為相對路徑。

以下簡單說明範例4.1。範例4.1上方為集合樹林（ColForest），下方為值樹林（ValForest）。由於我們會為每一個For和Let變數產生集合樹，因此範例2.2會產生集合樹“s”、集合樹“a”、集合樹“o”、集合樹“c”。注意到集合樹“a”對應到子查詢句，由於我們以路徑null代表子查詢句，因此集合樹“a”的路徑記錄null。

ColForest的根節點依序標明節點變數名稱、路徑以及階層碼，為了簡化起見，圖中僅標示路徑的最後一個節點名稱，實際儲存的資料則為絕對路徑。階層碼的目的為記錄樹的巢狀階層關係，在4.3節中會詳細說明階層碼。以變數$s產生的集合樹為例，其根節點變數為“s”、絕對路徑為/order-ship/supppliers/supplier，在範例4.1以“suppiler”代表絕對路徑，階層碼則為“1”。

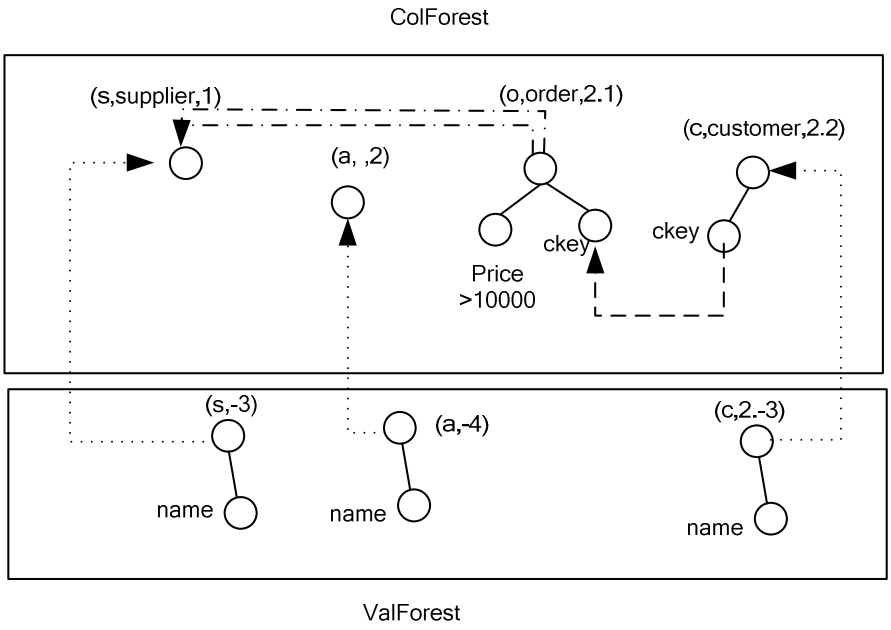
Where子句中的路徑會在集合樹中建立對應的節點，因此範例2.2 L04的$o@ckey=$c@ckey會分別在o、c兩棵集合樹下建立葉節點ckey，而L05 $o/price >10000則在集合樹“o”底下建立葉節點price，price的路徑為$o/price >10000。注意到，樹中的實線代表父子關係，例如集合樹“o”和其葉節點ckey為父子關係。而當線為雙線時則代表祖孫關係，例如範例4.1的集合樹“o”指向集合樹“s”的線為雙線，這代表集合樹“o”為集合樹“s”的子孫。

接下來，return子句的每一個變數會產生值樹，因此L06的$c/name產生值樹“c”，值樹“c”底下建立葉節點name，L08的$s/name,$a/name產生s、a兩棵值樹。值樹“s”底下建立葉節點name，值樹“a”底下建立葉節點name。ValForest的根節點依序標明節點名稱、階層碼。至於ColForest以及ValForest的所有葉節點僅標明絕對路徑，同樣路徑只以最後一個節點名稱代替。節點上的箭頭指向宣告節點。例如變數$s產生的值樹，其根節點變數名稱為“s”，階層碼為“-3”。根節點指向ColForest中由$s產生的集合樹。

樹中的虛線分為三類，「– · – · –」代表ColForest中根節點之間的連結，箭頭指向路徑中所包含變數，如集合樹“o”的路徑為$s//order，所以集合樹“o”指向集

合樹“s”。「----」代表葉節點之間的連結，箭頭指向扁平連結中的另一個葉節點，例如扁平連結為$o@ckey=$c@ckey，則集合樹“c”的葉節點ckey指向集合樹“o”的葉節點ckey。「·····」則代表ValForest中的根節點與ColForest中的根節點的連結，箭頭指向ColForest中的根節點，例如值樹“s”指向集合樹“s”。

【範例4.1】



範例4.2的CanForests是由範例4.1的CVForest產生。CanForests中包含多個CanForest，相同父階層的樹會放入相同的CanForest。在CanForests會將CVForest中所有的節點路徑由絕對路徑改寫成簡單路徑，並在節點的CVMs欄位記錄取得的對應資料，CVMs中包含多筆 [（輸出變數，對應集合或值，是否輸出）]。若非根節點的路徑包含跳層，則CanForests會在根節點與葉節點中間加入非葉節點，若是根節點的路徑包含跳層，則會在根節點以及指到的根節點之間加入節點，例如集合樹“o”的路徑包含跳層，因此集合樹“o”和集合樹“s”之間會加入part節點。

除此之外CanForest還會刪除空節點。

以集合樹 “o”為例，集合樹 “o”的路徑為$s//order，由於路徑包含跳層，因此會在集合樹 “o”與集合樹 “s”之間加入part節點，且集合樹 “o”的路徑會改寫成簡單路徑 /order-ship/suppliers/supplier/part/order。

CanForests會依階層分成多個CanForest。範例4.1中集合樹 “s”、集合樹 “a”以及值樹 “s”、值樹 “a”的父階層（PLevel）皆為0，因此隸屬於父階層為 “0”的CanForest，而集合樹 “c”、集合樹 “o”以及值樹 “c”隸屬父階層為 “2” 的CanForest。PLevel為 “2”的CanForest是由子查詢句產生，Let子句的變數a連結此子查詢句，因此CanForest的子查詢句變數（NestName）為“a_0”， “a_0”為集合樹“a”的CVMs欄位中第一筆資料的變數。

範例4.2中，每個實線方框代表一個CanForest，而虛線方框代表正規集合樹（CanColForest）以及正規值樹（CalValForest）。CanForest由上往下依序記錄PLevel、NestName、CanColForest以及CanValForest。CanForest中集合樹所有節點的CVMs欄位內容如表格4.1所示，CanForest中值樹的CVMs欄位內容如表格4.2所示。例如在 PLevel 為 “0” 的 CanForest 中 ， 正 規 集 合 樹 “s” 的 路 徑 為 “/order-ship/suppliers/supplier”，由CM可找到三筆對應表格SUPPLIER、PARTSUPP以及LINEITEM，這三筆對應表格的輸出變數分別為s_1、s_2以及s_3，usedflag的預設值皆為false。而表格4.1中SUPPLIER表格的usedflag會為true，這是因為在建立正規值樹葉節點 name時，VM找到對應資料SUPPLIER.NAME，因此得知必需輸出SUPPLIER表格，故SUPPLIER表格的usedflag為true。

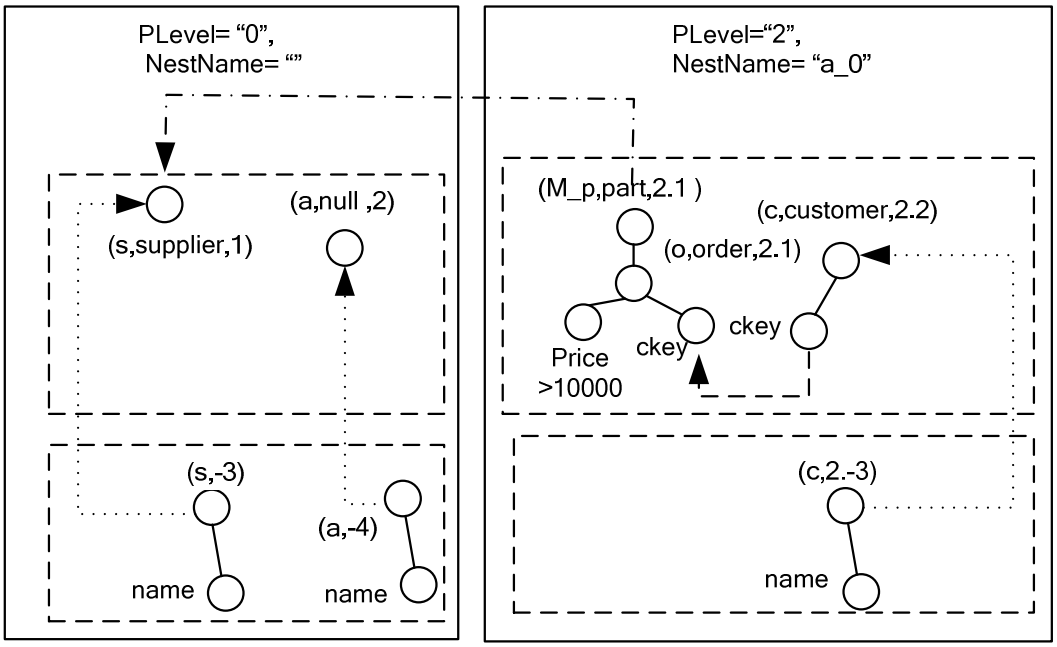
正規集合樹 “a”由於代表子查詢句， CM中找不到a代表的資料，因此在CVM欄位中，變數為a_0，對應集合為null，又因為查詢句轉換後會輸出變數宣告，所以usedflag為真。集合樹 “o”的葉節點 price的對應欄位為ORDER.TOTALPRICE，在父節點order得知ORDER表格的變數為o_1，由於葉節點 price是由where子句的條

件限制產生，所以路徑中必需記錄條件限制，因此price節點的CVMs欄位會記錄[（o_1,TOTALPRICE>10000,TRUE）]。

正規值樹的根節點由於是以指標指向正規集合樹根節點，因此其記錄的資料會相同，當其中一方改變則另一方也會改變。例如正規值樹 “s”和正規集合樹 “s”的 TagName、path、CVMs 欄位都一樣。正規值樹 “s”的葉節點 name，其對應欄位為 SUPPLIER.NAME，因此會到正規值樹 “s”尋找 SUPPLIER 表格的變數 s_1，並將值樹 “s”中 SUPPLIER 的 usedflag 設為真，而葉節點 name 的 CVMs 欄位會記錄{（s_1,NAME,TRUE）}，此時正規集合樹 “s”的資料也會跟著正規值樹 “s”改變。

而正規值樹 “a”的葉節點 name，由於 “a” 為代表子查詢句，因此會到正規集合樹 “a”取出 CVMs 欄位第一筆資料的變數，取得 a_0 後在葉節點 name 的 CVMs 欄位記錄[（a_0,NAME,TRUE）]。

【範例4.2】



PLevel	NESTNAME	TagName	path	CVM= {Var,ColOrVal,UsedFlag}
0		s	/order-ship/suppliers/supplier	s_1,SUPPLIER,True s_2,PARTSUPP,FALSE s_3,LINEITEM,FALSE
		a	null	a_0,null,TRUE
2	a_0	part	/order-ship/suppliers/supplier/part	M_p_1,PART,FALSE M_p_2,PARTSUPP,FALSE
		o	/order-ship/suppliers/supplier/part/order	o_1,ORDER,TRUE o_2,LINEITEM,FALSE
		price	/order-ship/suppliers/supplier/part/order/price>10000	o_1,TOTALPRICE>1000,TRUE
		ckey	/order-ship/suppliers/supplier/part/order@ckey	o_2,CUSTKEY,TRUE
		c	/order-ship/customer	c_1, CUSTEL, FALSE c_2, CUSTOMER, TRUE c_3, INTRODUCE,FALSE
		ckey	/order-ship/customer@ckey	c_1,CUSTKEY,TRUE

表格 4.1 範例 4.2 的正規集合樹 CVM 欄位資料

PLevel	TagName	NESTNAME	path	CVM= {Var,ColOrVal,UsedFlag}
0	s		/order-ship/suppliers/supplier	s_1,SUPPLIER,True s_2,PARTSUPP,FALSE s_3,LINEITEM,FALSE
	name		/order-ship/suppliers/supplier/name	s_1,NAME,TRUE
	a		null	a_0,null,TRUE
	name		$a/name	a_0,NAME,TRUE
2	c	a_0	/order-ship/customer	c_1, CUSTEL, FALSE c_2, CUSTOMER, TRUE c_3, INTRODUCE,FALSE
	name		/order-ship/customer/name	c_2,NAME,TRUE

表格 4.2 範例 4.2 的正規值樹 CVM 欄位資料

4.2. 所有定義

參考[呂04]的定義，本節會先定義階層碼及集合樹、值樹，以及正規集合樹和正規值樹。

4.2.1. 階層碼（Level Number）

利用階層碼可以清楚得知樹之間的巢狀階層關係，也可以清楚區分集合樹和值樹。另外，利用階層碼，我們可以有順序性的收集不同階層樹的轉換結果。以下為階層碼的定義：

【定義4.1】：階層碼（Level Number）

階層碼 l 由一個以上的階層元件（Level Component）lc 組成，lc 為整數且由1 開始，每個lc 之間以“.”隔開，因此l = lc1.lc2.....lcn。階層碼具有下列性質：

- LastComponenet（l）：取出l 的最後一個階層元件。
- LevelCount（l）：計算階層元件個數。
- PLevel（l）：傳回l的父階層，PLevel（l）=l- LastComponenet（l）。若只有一個階層元件則會回傳0，例如PLevel（2）=0;
- 任兩個階層碼 lm 和ln，
  - 若LevelCount（lm）– LevelCount（ln）= 1，且lm ⊇ ln，則稱ln 為lm 的父階層（Parent Level）。
  - 若LevelCount（lm）– LevelCount（ln）= 0，且lm和ln有相同的的父階層，則ln 和lm 同層。
  - 若LevelCount（lm）– LevelCount（ln）> 1，且lm ⊇ ln，則稱ln 為lm 的祖先階層（Ancestor Level）。

◆ 若LevelCount（lm）– LevelCount（ln）> 1，則稱lm的階層比ln低。

以範例4.1中由變數 “$o”產生的集合樹為例，階層碼為2.1，而LastComponenet（2.1）=1，LevelCount（2.1）=2，PLevel（2.1）=2。而變數“$c”產生的集合樹其階層碼為2.2，由於PLevel（2.1）＝PLevel（2.2），因此可得知2.1和2.2的父階層相同，父階層相同的樹會放在同一個CanForest中。而集合樹 “s”的的階層碼為1，集合樹 “o”的的階層碼為2.1，LevelCount（2.1）>LevelCount（1），因此集合樹 “o”的階層比集合樹 “s”低。

在建立集合樹和值樹時，一開始的階層碼為1，每建立一棵樹LastComponenet（l）會加1。進入巢狀子句後，階層元件個數會加1，且LastComponenet（l）由1開始，每建立一棵樹LastComponenet（l）會加1。為了清楚地分辨集合樹和值樹，值樹的LastComponent（l）為負數。

在範例4.1中，依範例2.2逐步建立集合樹時，supplier節點建立的集合樹階層碼為1，let子句的變數$a其階層碼為2，由於$a連結子查詢句，進入巢狀子句後，階層元件個數會加1，且LastComponenet（l）由1開始，所以order節點代表的集合樹階層碼為2.1，customer節點代表的集合樹階層碼為2.2，子查詢句中的return子句的$c/name產生的值樹階層碼則為2.-3。處理完子查詢句，外部查詢句的return子句接續階層碼2，$s/name產生的值樹階層碼為-3，$a/name的值樹階層碼則為-4。

4.2.2. 集合樹和值樹定義

查詢句中包含資料連結、資料限制和資料輸出，在XQuery中，FOR、LET 子句為資料連結，WHERE 子句為資料限制，RETURN 子句則是資料輸出；而在SQL中，FROM子句為資料連結，WHERE 子句為資料限制，SELECT子句則是資料輸出。本論文將資料連結和資料限制的內容轉成集合樹，而資料輸出則是轉成值樹。值

樹的根節點會包含資料連結的資訊並指向對應的集合樹根節，這樣可確保來源和輸出的值一致。

在介紹集合樹和值樹之前，我們必須先定義樹中的節點內容。

【定義4.2】：樹節點與樹邊

節點={TagName,ln,path}。

- TagName: 節點名稱。節點名稱為變數宣告、元素或屬性。當節點為根節點時，節點名稱為變數宣告。
- ln: 節點的階層碼。
- path：節點的絕對路徑。需注意的是，路徑若為null則節點代表子查詢句。

邊={Axis, FlatJoin, NestedJoin, RootJoin}。

- Axis:依父節點取得的相對路徑之軸。若為根節點則Axis內容為空。Axis由父節點指向子節點。
- FlatJoin：葉節點之間的連結。若兩個葉節點的階層不同，則階層低的葉節點指向階層高的葉節點，若兩者的階層相同，則後建立的葉節點指向另一個葉節點。
- NestedJoin：集合樹根節點之間的連結。當集合樹根節點之間有「巢狀關係」時會有NestedJoin。當兩個節點為父子查詢句，或兩者之間具有父子祖孫關係，稱為具有巢狀關係。NestedJoin由子孫節點指向祖父節點，或由子查詢句指向父查詢句。
- RootJoin：值樹根節點與對應集合樹根節點的連結。RootJoin由值樹根節點指向對應集合樹根節點。

利用上面所定義的樹節點及樹邊，我們可以定義集合樹和值樹，集合樹和值樹的定義如定義4.3和定義4.4所示。此處定義的節點適用在集合樹和值樹。當節

點為集合樹節點時，RootJoin為空，而當節點為值樹節點時， FlatJoin和NestedJoin為空。在範例4.1和範例4.2中，Axis以實線表示，FlatJoin對應到「－－－－」，NestedJoin對應「－‧－‧－」，RootJoin對應到「‧‧‧‧‧‧‧‧」。

【定義4.3】：集合樹

每個集合樹（Collection Tree）=（CN, E），其中 CN 為集合樹中的節點集合，E 為邊集合。集合樹中的節點可分為根節點（root node）、非葉節點（non-leaf node）以及葉節點（leaf node），每種節點分別具有下列性質：

- 根節點：每棵樹的根節點階層碼都是唯一的。根節點的Tagname為FOR 子句、LET 子句中的連結變數名稱。
- 非葉節點： Where子句的限制式路徑分解成多組{軸-節點名稱}後，每一組{軸-節點名稱}會有順序性地加在根節點下。樹中根節點與葉節點以外的節點稱為非葉節點。非葉節點的階層碼和根節點的階層碼相同。
- 葉節點： Where子句的限制式路徑分解成多組{軸-節點名稱}後，最後一組{軸-節點名稱}所產生的節點稱為葉節點。葉節點的階層碼和根節點的階層碼相同。

以範例4.1為例，集合樹 “s”由集合樹由範例2.2For子句中的 $s in /order-ship/suppliers/supplier 產生，其TagName為 “s”，ln為1，path為/order-ship/suppliers/supplier。此節點沒有可連結的葉節點，也未與其他集合樹有巢狀關係，所以FlatJoin和NestedJoin為null。

“o”是由範例2.2For子句中的 $o in $s//order產生，其TagName為 “o”，ln為2.1，path為$s//order。由路徑$s//order可看出與集合樹 “s”為巢狀關係，因此NestedJoin指向集合樹“s”。

where子句中的$o@ckey=$c@ckey產生集合樹 “o”和集合樹 “c”的葉節點

ckey，其中集合樹“c”的葉節點ckey的TagName為ckey，ln與父節點customer相同皆為2.2，path為$c@ckey。集合樹“c”的葉節點ckey與父節點customer的相對路徑為@ckey，所以Axis為@，又因為此節點與集合樹“o”的葉節點ckey為扁平連結，所以FlatJoin指向集合樹“o”的葉節點ckey。

【定義4.4】：值樹

對每個值樹（Value Tree）=（VN, E），其中VN 為節點集合，E 為邊集合。值樹中的節點可分為根節點（root node）、非葉節點（non-leaf node）以及葉節點（leaf node），每種節點具有下列性質：

- 根節點：每個根節點的階層碼都是唯一的。根節點的名稱為變數宣告或表格名稱，根節點的RootJoin指向名稱相同的集合樹根節點。
- 非葉節點：非葉節點的階層碼和根節點的階層碼相同。在XQuery的Return子句中，回傳值路徑可能會分解成多組{軸-節點名稱}，每一組{軸-節點名稱}會有順序地加在根節點下。其中根節點與葉節點以外的節點稱為非葉節點。注意到，集合樹的非葉節點由where子句產生，值樹的非葉節點由return子句產生。
- 葉節點：葉節點的階層碼和根節點的階層碼相同。在XQuery的Return子句中，回傳值路徑會分解成多組{軸-節點名稱}，最後一組{軸-節點名稱}所產生的節點稱為葉節點。注意到，集合樹的葉節點由where子句產生，值樹的葉節點由return子句產生。

以範例4.1為例，值樹“c”由範例2.2 return子句中的$c/name產生，根節點的TagName為“c”，ln為2.-3，path為/order-ship/customer。由於值樹“c”與集合樹“c”連結，所以RootJoin指向集合樹“c”。$c/name在產生值樹根節點後，接著會產生TagName為name的葉節點，此葉節點的ln和父節點一樣是2.-3，path為$c/name。

由於葉節點name與父節點的相對路徑為 /name，所以Axis為“/”。

4.2.3. 正規集合樹和正規值樹

利用集合樹和值樹的資料可以建立正規集合樹和正規值樹，正規樹中的節點路徑會改寫為簡單路徑，若樹中葉節點的路徑包含跳層，則正規樹會在根節點與葉節點中間加入非葉節點。而若是根節點的路徑包含跳層，則會在根節點以及NestJoin指到的根節點之間加入節點。除此之外正規樹還會刪除空節點，並在正規樹節點的CVMs欄位記錄取得的對應資料。經過處理的集合樹和值樹分別稱為正規集合樹和正規值樹。

在介紹正規集合樹和正規值樹之前，我們必須先定義正規樹中的節點內容，此處定義的節點適用在正規集合樹和正規值樹。

【定義4.5】：正規樹節點與正規樹邊

- 正規節點={ NodeData,CVMs }。
- NodeData: NodeData ={TagName,ln,path}。NodeData的欄位與定義4.1的樹節點相同，但是定義4.1的路徑為絕對路徑，正規樹節點的Path為簡單路徑。
- CVMs: CVMs為多筆CVM的集合。CVM為利用CM或VM取得的對應資料，CVM={Var,ColOrVal,UsedFlag}。
  - Var：CVM資料列的變數，用以代表每一筆CVM不同的編號。Var根據NodeData的TagName產生。
  - ColOrVal：對應集合或對應值。
  - UsedFlag：記錄此對應資料列是否要被輸出，若為True則輸出。

- 正規樹邊={Axis, FlatJoin, NestedJoin, RootJoin}。由於正規樹中的樹邊和定

義4.2中的樹邊定義相同，因此不再重覆敘述。當節點為正規集合樹節點時，RootJoin為空，而當節點為正規值樹節點時，FlatJoin和 NestedJoin為空。

正規集合樹的結構和集合樹相同，正規值樹的結構和值樹相同，但樹中節點為定義4.5的正規樹節點而不是定義4.2的樹節點。正規集合樹請參考定義4.3的集合樹定義，正規值樹請參考定義4.4的值樹定義。

以範例4.1為例，正規集合樹 “s”的NodeData、樹邊和集合樹 “s”一樣，所以NodeData是{ “s”,1, /order-ship/suppliers/supplier }，樹邊為{null, null, null }。由於/order-ship/suppliers/supplier在CM對應到SUPPLIER、PARTSUPP和LINEITEM表格，所以CVMs欄位中會有三筆資料{（s_1,SUPPLIER,FALSE）,（s_2,PARTSUPP,FALSE）,（s_3,LINEITEM,FALSE）}，其中的s_1、s_2、s_3是根據TagName “s”產生的變數。

集合樹 “o”由於路徑$s//order包含跳層，所以會以$s//order的正規表示法/order-ship/suppliers/supplier[/a-z+]*/order 在 CM 中取得簡單路徑/order-ship/suppliers/supplier/part/order，並在根節點和葉節點中間加上集合節點part、order。詳細的建立過程會在4.4節敘述。

4.3. CVForest 建立演算法

在建立CVForest時，程式會先剖析使用者輸入的查詢句，如果查詢句在圖2.4的文法範圍，則為FOR和LET子句中的每一個變數宣告分別產生集合樹，集合樹的根節點名稱與變數相同，而對RETURN子句中的每一個變數宣告分別產生值樹的根節點，並將回傳的屬性或葉節點加入根節點底下。

在建立集合樹和值樹時，每棵不同的樹都會有自己的階層碼，從階層碼可以得知樹之間的巢狀關係。集合樹的集合結果稱為「集合樹林」(Collection Forest)，

值樹林的集合結果稱為「值樹林」(Value Forest)，集合樹林和值樹林則合稱為CVForest。以下為CVForest的定義。

【定義4.5】：CVForest

CVForest=（ColForest,ValForest)，其中ColForest為集合樹林，ValForest為值樹林。

集合樹林與值樹林皆以湊雜表（HashTable）記錄資料，湊雜表以樹根節點的節點名稱為鍵值，以樹根節點為資料內容。集合樹林的湊雜表稱為 HTColForest，值樹林的湊雜表稱為 HTValForest。在建立 NestJoin、RootJon、FlatJoin 時，利用 HTColForest 可以馬上指到需要的集合樹。除此之外，where 子句在加入節點到集合樹時，HTColForest 可以快速取得 where 子句對應的集合樹。而當同一個變數回傳多個值時，HTValForest 可以快速取得變數代表的值樹。

圖4.1的BuildCVForest演算法中， FLOWR子句的每個集合和回傳值會建立集合樹與值樹，演算法會以遞迴的型式處理子查詢句。輸入的查詢句子句依序為FOR、LET、WHERE以及RETURN子句，其中LET及RETURN子句中可以有子查詢句。演算法開始時的階層碼ln設為1，因為演算法是採遞迴型式，遇到子查詢句時亦需傳入子查詢句的階層碼，所以演算法中宣告nest_ln記錄子查詢句的階層碼。

圖4.1的L02~L05處理For子句，L03會新增一個集合樹節點到HTColForest，ColForest.Add（v,ln,Expression）函式如圖4.2所示。L06~L17處理Let子句，若Let子句的變數宣告連結到一般路徑而非連結至子查詢句，則此變數宣告和For子句中的變宣告相同，會為變數宣告產生集合樹根節點。若Let子句的變數宣告連結至子查詢句，則為變數宣告產生一個路徑為空的集合樹根節點，在之後的處理過程中，將會把路徑為空的變數認定為連結至子查詢句，接著遞迴呼叫BuildCVForest函式處理子查詢句。

L18~L35處理where子句。若where子句為文數值限制，則在對應的集合樹中加入相關的子孫節點，L22的AddChildNode函式如圖4.2所示。where子句若為連結限制式，則在連結式兩端路徑對應的集合樹下分別加入相關的節點，並為階層碼較低的葉節點的FlatJoin指向另一端葉節點，這是因為外部查詢句的變數可以用在子查詢句中，而子查詢句的變數無法用在外部查詢句中。若兩者階層一樣，則在後建立的葉節點的FlatJoin加入另一端葉節點。L36~L48處理Return子句，return子句中的每個回傳值會建立值樹，L40行的CVForest.HTValForest.Add（Expression,ln）函式會根據Expression產生值樹節點，並加入此值樹節點到HTValForest，函式內容如圖4.4所示。若回傳的內容為子查詢句，則遞迴呼叫BuildCVForest處理子查詢句。剖析完查詢句後會回傳CVForest。

```
演算法名稱：BuildCVForest（Query, ln, CVForest）
輸入：Query      /*輸入的查詢句*/
      ln     /*階層碼*/
      CVForest      /*集合樹林和值樹林*/
輸出：CVForest      /*集合樹林和值樹林*/
變數說明：nest_ln /*巢狀查詢句中的階層碼*/
L01 nest_ln = “”;
L02 For（each “$v IN Expression”in Query.ForClause）{//processing FOR
L03       ColForest.Add（v,ln,Expression）;
L04       LastComponent（ln）++;
L05 }
L06 For（each “$v ::= Expression”in Query.LetClause）{    //processing LET
L07       if（Expression != FLOWR）{
L08             ColForest.Add（v, ln, Expression）;
L09             LastComponent（ln）++;
L10       }
L11       if（Expression == FLOWR）{
L12             CVForest.ColForest.Add（v, ln, Null）;
L13             nest_ln = ln + “.1”;
L14             CVForest= BuildCVForest（Expression, nest_ln, CVForest）;
```


```
L15             LastComponent（ln）++;
L16         }
L17 }
L18 For（each Expression “Expression1 CompOp Expression2” in
L19 Query.WhereClause）{ //processing WHERE
L20         if（Expression2 is Literal）{ // Selection Condition
L21             Var=getvar（Expression1）;
L22             leafNode=HTColForest [Var].AddChildNode（Expression）;
L23             leafNode.path=operator+expression2;
L24         }
L25         else{ // Join Condition
L26             Var1=getvar（Expression1）;
L27             Var2=getvar（Expression2）;
L28             LeafNode1=HTColForest [Var1].AddChildNode（Expression1）;
L29             LeafNode2=HTColForest [Var2].AddChildNode（Expression2）;
L30
L31             If（LeafNode1.getLn（）< LeafNode2.getLn（））{
L32                 LeafNode1.setFlatJoin（LeafNode2）;
L33             }
L34             Else { LeafNode2. setFlatJoin（LeafNode1）; }
L35         }
L36 }
L37 If（Query.ReturnClause）{
L38         ln=-ln
L39         For（each Expression in Query.ReturnClause）{
L40             if（Expression!=FLOWR）{
L41                 HTValForest.Add（Expression,ln）;
L42             }
L43             else {
L44                 nest_ln=ln+.1;
L45                 CVForest= BuildCVForest（Expression, nest_ln, CVForest）;
L46             }
L47         }
L48 }
L49 Return CVForest;
```

圖 4.1 BuildCVForest 演算法

圖4.2的演算法ColForest.Add(var,ln,Expression)會在HTColForest新增集合樹。L01新增一個集合樹節點並將其指派給RootNode，L02將輸入參數中的Var、ln以及Expression 指派給 RootNode 。 L03 的 HTColForest.Add （ var,RootNode ） 會在HTColForest中建立一筆鍵值為Var，內容為RootNode的資料。L4取出Expression中的變數，例如Expression為 $s//order，則變數為$s。如果存在變數，則L05~L07 會將變數對應的集合樹根節點指派給RootNode的NestJoin。

```
演算法名稱：ColForest.Add（var,ln,Expression）
輸入：  var          /*變數*/
        ln           /*階層碼*/
        Expression   /*路徑表示式*/
輸出：  HTColForest  /*集合樹林的湊雜表*/
變數說明：  RootNode      /*新增的集合樹根節點*/
            JoinVar       /* Expression 中包含的變數*/
            NestJoinNode  /*TagName 為 JoinVar 的集合樹根節點*/
L01 RootNode=new ColTreeNode;
L02 RootNode's{TagName,ln,path} ={var,ln, Expression}
L03 HTColForest.Add（var,RootNode）;
L04 JoinVar=getJoinVar（Expression）;
L05 if（JoinVar!=null）{//process nested structure
L06       NestJoinNode= HTColForest[JoinVar];
L07       RootNode->setNestJoin（NestJoinNode）;
L08 }
L09 Return HTColForest;
```

圖 4.2 ColForest.Add（var,ln,Expression）演算法

以範例 2.2 為例，階層碼的預設值為 1，BuildCVForest 演算法的 L02~L05 處理 for 子句的每一個變數宣告，變數$s 會進入 ColForest.Add（ s,1, /order-ship/suppliers/supplier）。L01~L02 在 ColForest.Add 演算法中會建立一個集合樹根節點，節點的 TagName 為 s，ln 為 1，path 為/order-ship/suppliers/supplier。L03 以 s 為鍵值，將此節點加入 HTColForest。L05 由於變數$s 不是其他變數的子孫節點，因此 JoinVar 為空，最後回傳 HTColForest。

而 BuildCVForest 演算法的 L06~L17 處理 Let 子句，在處理變數宣告 $a 時，由於$a 為子查詢句，因此進入 L11~L16，L12 HTColForest.Add 函式會產生一個集合樹根節點，其 TagName 為 a，ln 為 2，path 為空。並以 a 為鍵值加入 HTColForest。L14 會以 a 的子查詢句為參數呼叫 BuildCVForest 演算法。處理子查詢句時，同樣的會為變數宣告$o、$c 產生集合樹根節點。

圖 4.3 的演算法 HTColForest[Var].AddChildNode（Expression）中，HTColForest[Var]會取出HTColForest中以Var為鍵值的集合樹根節點。此演算法會解析Expression路徑，然後將結果逐一建立到HTColForest[Var]節點底下，若有相同的節點則不再重複建立。例如集合樹根節點路徑為/a，Expression為/a/b/c，則會在集合樹根底下加入名稱為b的節點，接著在節點b底下加入名稱為c的節點。

L01將HTColForest[Var]代表的集合樹根節點指派給CurrentNode，L02取出Expression的相對路徑，例如Expression為$o/price，對應的集合樹 “o”的路徑為/order-ship/suppliers/supplier/part/order，由於 $o/price 會展開成/order-ship/suppliers/supplier/part/order/price，所以會取得相對路徑/price。L03將相對路徑分解成多組{軸-節點名稱}，並循序處理每一組{軸-節點名稱}。L05行在Current節點底下尋找符合{軸-節點名稱}的子節點，並將取得的子節點指定給ChildNode。若不存在符合{軸-節點名稱}的子節點，L06行到L08行會在CurrentNode底下依{軸-節點名稱}建立子節點，並將子節點指派給變數ChildNode。之後在L09行向下移到ChildNode，並處理下一組{軸-節點名稱}。處理完所有的{軸-節點名稱}後，程式會回傳最後一組{軸-節點名稱}所對應的的節點。

```
演算法名稱：HTColForest [Var].AddChildNode（Expression）
輸入：Expression          /*路徑表示式*/
輸出：CurrentNode        /*Expression 中最後一組{軸-節點名稱}代表的節點*/
變數說明：  CurrentNode        /*目前走到的查詢樹節點*/
            Axis_TagName       /*軸與節點名稱*/
            ChildNode          /* CurrentNode 的子節點*/
```


```
L01 CurrentNode=HTColForest[var];
L02 RelativePath= ExpandPath（Expression）- HTColForest[var].getpath（）;
L03 for each Axis_TagName in RelativePath
L04 {
L05       ChildNode=CurrentNode.Search（Axis,tagName）;
L06       If（ChildNode==NULL）{
L07             ChildNode = CurrentNode.add（Axis,tagName）;
L08       }
L09       CurrentNode= ChildNode;
L10 }
L11 Return CurrentNode;
```

圖 4.3 HTColForest [Var].AddChildNode（Expression）演算法

繼續以範例 2.2 為例，BuildCVForest 演算法的 L18~L36 會處理 where 子句，在處理範例 2.2 子查詢句的$o@ckey=$c@ckey 時，由於$c@ckey 不是文數值限制，因此進入 L25~L35，L28 進入 HTColForest [o].AddChildNode（$o@ckey）函式後，L02 取得相對路徑@ckey，L03~L10 為@ckey 產生集合樹節點，並將其加入集合樹 “o”底下。

接下來，圖4.4的演算法HTValForest.Add（Expression,In）會在HTValForest新增值樹。L01取得Expression中的變數，例如$s/name的變數為$s。 若HTValForest中不存在節點名稱為Var的值樹根節點，則新增一個值樹節點並將其指派給RootNode， L05的HTValForest.Add（var, RootNode）會在HTValForest中建立一筆鍵值為Var，內容為RootNode的資料，L06將Var對應的集合樹根節點指派給RootNode的RootJoin。L08的HTValForest[Var].AddChildNode（Expression）作法與圖4.3相同，同樣都是把Expression解析出的路徑逐一的建立在RootNode底下，若有相同的節點則不再重複建立。最後回傳HTValForest。

演算法名稱：HTValForest.Add（Expression,ln）	
輸入： Expression /*路徑表示式*/	
ln /*階層碼*/	
輸出： HTValForest /*值樹林*/	
變數說明： RootNode /*新增的集合樹根節點*/	
Var /* Expression 中包含的變數*/	
JoinRootNode /*TagName 為 JoinVar 的集合樹根節點*/	
L01	Var=getVar（Expression）
L02	If（HTValForest ->containskey（Var）==false）{
L03	RootNode=new ValTreeNode;
L04	RootNode's{var,ln}={var,ln};
L05	HTValForest.Add（var, RootNode）;
L06	RootNode ->setRootJoin（HTColForest[Var]）;
L07	}
L08	HTValForest[Var]. AddChildNode（Expression）;
L09	Return HTValForest

圖 4.4 HTValForest.Add（Expression）演算法

繼續以範例 2.2 為例，在處理子查詢句的 Return $c/name 時，HTValForest.Add（$c/name,2.-3）在 HTValForest 產生節點名稱為 c、階層碼為 2.-3 的值樹根節點，並在根節點下加入 name 子節點。L06 則加入集合樹 "c"到 RootJoin 中。範例 2.2 外部查詢句的 return 子句分別產生值樹 "a" 及值樹 "c"。

4.4. Canonical CVForest 建立演算法

以下為 Canonical CVForest 的定義。

【定義 4.6】：Canonical CVForest（簡稱 CanForest）

CanForest = { PLevel,CanColForest,CanValForest, NestName, OutQuery }。

- PLevel：樹的父階層。PLevel相同的樹隸屬同一個 CanForest中 。這是因為我們會將外部查詢句與子查詢句產生的集合樹放在不同的CanForest，所

有外部查詢句產生的集合樹，其父階層會相同。而所有子查詢句產生的所有集合樹，其父階層也會相同。

- CanColForest：正規集合樹林。
- CanValForest：正規值樹林。
- NestName：子查詢句的名稱，由宣告變數命名。若CanForest不是由子查詢句建立，則NestName為空。
- OutQuery={ Collections, Values, JoinConditions, ValueConditions }，Collections記錄輸出的集合，Values記錄輸出的回傳值，JoinConditions記錄連結限制，若為連結父子查詢句的連結限制，則記錄在父查詢句的JoinConditions。ValueConditions記錄文數值限制。

以階層做區分，不同階層的樹隸屬於不同的 CanForest ，而所有的 CanForest 集合成 CanForests。CanForests 以湊雜表記錄資料，我們以 HtCanForest 稱呼。HtCanForest 的鍵值為 PLevel，內容值為 CanForest。在有多個子查詢句時，以湊雜表分層記錄 CanForest，可以依 PLevel 快速的取得父子查詢句。論文中以 BuildCanColForest 演算法建立正規集合樹林。BuildCanColForest 演算法如圖 4.5 所示。建立正規值樹林的邏輯精神與 BuildCanColForest 演算法類似，故不敘述。

在建立節點的對應資料時，論文中以雜湊表 HTCM 記錄 CM 資料，以雜湊表 HTVM 記錄 VM 資料，以雜湊表記錄 CM 和 VM 可以快速的依鍵值取得對應資料。HTCM 的鍵值為節點路徑，內容值為節點路徑對應的 CM 資料列，而 HTVM 的鍵值為葉節點路徑，內容值為葉節點路徑對應的 VM 資料列。

圖 4.5 的 BuildCanColForest 演算法根據 HTColForest 以及對應表格建立正規集合樹。L01 的 CanRootNode 是個 null 節點。L02 依序處理每一棵集合樹的根節點，L03 呼叫 CreateCanNode 函式，此函式利用 CVRootNode 新增正規集合樹根節點。新建立的正規集合樹根節點會指派給 CanRootNode，CreateCanNode 演算

法如圖 4.6 所示。

L04 取得 CanRootNode 的 PLevel，L05~L07 把 CanRootNode 放入 HTCanForest[PLevel]的 CanColForest 中。若不存在鍵值為 PLevel 的 HTCanForest，則 L08~L12 會在 HTCanForest 新增一個鍵值為 PLevel 的 CanForest，並將 CanRootNode 放入 HTCanForest[PLevel]的 CanColForest 中。為了保持節點的邊關係，L13~L17 處理和根節點具子查詢句巢狀關係的節點，L15 取出 CanColForest 中具子查詢句巢狀關係的節點，並將此節點指派給 CanNestJoinNode。L16 建立 CanRootNode 與 CanNestJoinNode 的子查詢句巢狀連結關係。

L18~L25 處理 let 子句的變數。L18 的 IsSubQVar（CanRootNode）會以路徑是否為空判斷 CanRootNode 是否連結到子查詢句，這是因為在 BuildCVForest 的 L12，連結子查詢句的變數宣告所產生的節點，其路徑為空。L18~L25 中，L19 以 CanRootNode 的 TagName 取得 LetName，L20~L22 則新增 CVM 到 CanRootNode.CVMs，新增的 CVM 其內容為（LetName, “”,TRUE），CVM.usedflag 設為真是因為子查詢句轉換後會利用 LetName 包覆子查詢句，所以需要輸出。L23 則將 CVM 加入 OutQuery.Collections。L24 則以 LetName 設定子查詢句對應的 HTCanForest 的 LetName 欄位。L26 的 CanRootNode->AddChildNode 函式依 CVRootNode 的子節點產生正規節點並將其加入 CanRootNode 底下成為子節點。CanRootNode->AddChildNode 函式如圖 4.7 所示。最後則回傳建立好的 HTCanForest。

演算法名稱：BuildCanColForest（CVForest, HTCanForest）
輸入：　CVForest　　/*集合樹林和值樹林*/
　　HTCanForest　/*CanForest 的集合，記錄正規集合樹的湊雜表，鍵值為樹根的節點名稱*/
輸出：HTCanForest　　/* CanForest 的集合*/
變數說明：　CVRootNode　/*集合樹的根節點*/
　　CanRootNode /*依 CVRootNode 新增的正規集合樹根節點*/
　　CanNestJoinNode /*與 CanRootNode 有巢狀關係的正規節點*/

```
L01 CanRootNode=null CanColNode;
L02 for each CVRootNode in HTColForest {
L03     CanRootNode =CreateCanNode（CanRootNode, CVRootNode, CM,VM）;
L04     PLevel=GetPLevel（CanColNodeRoot->getLn（））;
L05     if（HTCanForest.ContainsKey（PLevel））{
L06         HTCanForest[PLevel]->CanColForest.Add（CanRootNodes）;
L07     }
L08     else{
L09         NewCanForest =new CanForest（）;
L10         NewCanForest ->CanColForest.Add（CanRootNodes）;
L11         HTCanForest.Add（PLevel, CanForest）;
L12     }
L13     NestJoinNode= CVRootNode ->GetNestJoinNode（）;
L14     If（NestJoinNode!=Null）{
L15         CanNestJoinNode= GetCanNode（NestJoinNode）;
L16         CanRootNode ->setJoinNode（CanNestJoinNode）;
L17     }
L18     If（IsSubQVar（CanRootNode）==true）{
L19         LetName=GetLetName（CanRootNode->getTagName（））;
L20         CVMs= CanRootNode->getCVMs（）;
L21         NewCVM's {TagName,Path,Usedflag}={LetName, "",true};
L22         CVMs->Add（NewCVM）;
L23         CanForest[PLevel].OutQuery.Collection.Add（CVM）;
L24         HTCanForest[CanColNodeRoot->getLn( )]->setLetName( LetName );
L25     }
L26 CanRootNode->AddChildNode（RootVar , CVRootNode, CM ,VM）;
L27 }
L28 Return HTCanForest;
```


圖 4.5 BuildCanColForest 演算法

圖 4.6 的 CreateCanNode 函式依 InCVChildNode 產生對應的正規節點，並將其加入 InCanNode 底下成為子節點。L01 的 Inpath 為 InCVChildNode 的節點路徑，L02~L05 則取出路徑中的選擇條件式，例如 “//order/price>10000”會取出“>1000”。L06~L12 處理路徑不含有跳層的情況，L13 到 L17 則處理路徑含跳層的情況。

在路徑不含有跳層的情況下，L07~L09 會新增正規節點到 InCanNode 底下成為子節點，新增的正規節點會根據 InCVChildNode 設定節點欄位，L10 呼叫 DealWithPCPath 函式，此函式會設定節點的 CVMs 欄位，DealWithPCPath 函式如圖 4.8 所示。而在路徑中包含跳層的情況下，L14 取得節點路徑的正規表示法，路徑中的“//”會改寫為[/[A-Za-z]+]*/，這代表允許出現任意長度的標籤名稱，標籤名稱的前面有 “/”字元，且可出現 0 次以上的標籤名稱，例如 a//b 會改寫成 a[/[A-Za-z]+]*/b，a/b、a/b/b 和 a/cc/de/b 皆符合此正規表示式。L15 呼叫 DealWithADPath（）函式，若節點為集合節點，則以正規表示法到 HTCM 逐筆比對資料，若正規表示法符合資料的鍵值，則依取得的資料建立正規節點並設定 CVMs 欄位，若為值節點則利用 HTVM。 DealWithADPath（）函式如圖 4.12 所示。L18 回傳根據 InCVChildNode 所建立的正規節點。

演算法名稱：CreateCanNode（InCanNode, InCVChildNode）	
輸入： InCanNode	
InCVChildNode /* 用來產生正規節點的 CVNode，產生的正規節點會成為 InCanNodes 的子節點*/	
輸出： CanChildNode /*依 InCVChildNode 新增的正規節點*/	
變數說明：InPath /* InCVChildNode 的路徑*/	
RegularPath/* InCVChildNode 的路徑正規表示式*/	
L01	InPath= InCVChildNode ->getpath（）;
L02	If（IsValCondition（InPath））{
L03	SelConditon=GetValCondition（InPath）;

```
L04       InPath=InPath-ValCon;
L05 }
L06 If（InPath ->find（“//”）==false）{
L07       CanChildNode=new CanNode;
L08       CanChildNode->SetNodeData（InCVChildNode）;
L09       InCanNode->AddChild（CanChildNode）;
L10       CanChildNode =DealWithPCPath（CanChildNode, InPath,
L11 SelConditon,CM,HTVM）;
L12 }
L13 Else if（InPath ->find（“//”）==true）{
L14       RegularPath= InCVChildNode ->getRegularPath（）;
L15       CanChildNode =DealWithADPath（InCanNode,
L16       InCVChildNode ,RegularPath ,SelConditon）;
L17 }
L18 Return CanChildNode;
```

圖 4.6 CreateCanNode 函式

圖 4.7 的 CanNode->AddChildNode（CVNode）演算法依序處理 CVNode 的每一個子節點 CVChildNode，L03 會利用 CreateCanNode 演算法為 CVChildNode 產生正規節點，並將產生的正規節點加入 CanNode 底下成為子節點。L05 則遞迴處理 CVChildNode 的子節點。最後會回傳依 CVChildNode 產生的正規節點。

```
演算法名稱：CanNode->AddChildNode（ CVNode）
輸入：   CanNode /*輸入的 CanNode*/
         CVNode
輸出：   CanChildNode /* 依 CVChildNodee 新增的正規節點*/
變數說明：
L01 For each CVChildNode in CVNode
L02 {
L03       CanChildNode =CreateCanNode（CanNode, CVChildNode）;
L04       CanChildNode=CanCVChildNode->AddChildNode（ CVChildNode）;
L05 }
L06 Return CanChildNode;
```

圖 4.7 CanNode->AddChildNode 演算法

4.4.1. 處理路徑不含跳層的情況

圖 4.8 的 DealWithPCPath 演算法處理路徑不含跳層的情況。L01~L03 處理路徑為集合節點的情況，L04~L06 處理路徑為值節點的情況，至於 L08~L11 則處理路徑為遞迴節點的情況。最後回傳內容更新後的 InCanNode。L01 的 HTCM.ContainsKey（InPath）函式會判斷 HTCM 中是否擁有鍵值為 InPath 的資料，若有則代表輸入的正規節點為集合節點，L02 的 SetCMs 演算法會根據 HTCM[InPath]設定正規節點的 CVMs 欄位，SetCMs 演算法如圖 4.9 所示。而當 HTCM 找不到鍵值為 InPath 的資料時，會接著判斷 HTVM 中是否擁有鍵值為 InPath 的資料，若有則代表輸入的正規節點為值節點，L05 的 SetVMs 演算法會根據 HTVM[InPath]以及 SelCondition 設定正規節點的 CVMs 欄位，SetVMs 演算法如圖 4.10 所示。最後當 Inpath 不為集合節點也不是值節點時，則會判斷 Inpath 是否為遞迴節點。L08 的 IsRecNode 函式會用 "//"加上 Inpath 的最後一個節點名稱判斷是否可以在 CM 找到路徑，若找到則代表 InPath 為遞迴節點，例如 /order-ship/customer/introuduce/customer 在 CM 和 VM 找不到對應路徑，而用 //customer 可以在 CM 找到對應路徑。若節點為遞迴節點則呼叫 DealWithPCPathRecNode 函式，DealWithPCPathRecNode 如圖 4.14 所示。若輸入的 InCanNode 不是集合節點、值節點、遞迴節點，則將此節點移除。

演算法名稱：DealWithPCPath（ InCanNode ,InPath,SelCondition,HTCM,HTVM） 輸入： InCanNode InPath SelCondition /*選擇條件式*/ HTCM/*記錄 CM 資料的雜湊表，雜湊表的鍵值為節點路徑，內容值為節點路徑對應的 CM 資料列*/ HTVM/*記錄 VM 資料的雜湊表，雜湊表的鍵值為葉節點路徑，內容值為葉節點路徑對應的 VM 資料列*/ 輸出： InCanNode /*內容更新後的 InCanNode*/	
L01	if（HTCM.ContainsKey（InPath））{
L02	InCanNode =SetCMs（InCanNode , HTCM[InPath]）;

```
L03 }
L04 else if（HTVM.ContainsKey（InPath））{
L05       InCanNode = SetVMs（InCanNode , HTVM[InPath],SelConditions）;
L06 }
L07 Else {
L08 if（IsRecNode（InCanNode）{
L09       InCanNode = DealWithPCPathRecNode（InCanNode s,
L10 childColTreeNode ,HTCM）;
L11 }
L12 Else remove（InCanNode）;
L13 Return InCanNode;
```

圖 4.8 DealWithPCPath 演算法

圖 4.9 的 SetCMs 演算法根據輸入參數 CMs 設定正規節點 InCanNode 的 CVMs 欄位。L03 取出 InCanNode 的 CVMs 欄位，取得的 CVMs 稱為 CanNodeCVMs。由於不確定要輸出哪一個對應表格，因此 L04~L09 會依序處理 CMs 的每一筆 CM，在 CanNodeCVMs 加入由 CM 產生的 CVM 資料列。最後回傳 CVM 欄位更新後的 InCanNode。在 SetCMs 演算法中由於不知道會輸出哪一個對應表格，因此 CVMs 欄位會加入所有的 CM 資料，而查詢句中每一個回傳值都只會輸出一個對應值，因此 SetVMs 演算法會取出 VMs 中權重最高的資料列，並用取得的資料設定 CVMs 欄位。

```
演算法名稱：
SetCMs（InCanNode,CMs）
輸入：  InCanNode   /*正規節點*/
        CMs /*用來設定 CVM 欄位的 CM 資料列集合*/
輸出：  InCanNode /*CVM 欄位更新後的 InCanNode*/
變數說明：VarNum /*區分 CVs 每一資料列的數字*/
L01 VarNum=1;
L02 ColName= InCanNode->getTagName（）;
L03 CanNodeCVMs= InCanNode->GetCVMs（）;
```


L04	For each CM in CanNodeCVMs
L05	{
L06	OutVar= ColName+"_"+ VarNum;
L07	VarNum ++;
L08	CanNodeCVMs->Add（OutVar, CM->GetColName（）,false）;
L09	}
L10	Return InCanNode;

圖 4.9 SetCMs 演算法

SetVMs 演算法利用 VMs 設定正規節點的 CVM 欄位，並記錄要最後要輸出的值以及集合。L01 的 TopVM 取出 VMs 中權重最高的資料列，並用取得的資料設定 CVMs 欄位。由於 VMs 依權重由高到低排序，因此會取出 VMs 中的第一筆資料。L02 以 ColName 記錄 TopVM 的集合名稱， L04~L15 到 InCanNode 的父節點中，尋找其 CVMs 欄位中有相同集合名稱的資料列，L10 的 OutVar 記錄取得的資料列中的變數。由於查詢句轉換後會輸出轉換值以及轉換值所位於的集合，因此 L11 將父節點中對應的 CVM 資料列其 usedflag 設為真。

L12 在對應 HTCanForest 的 OutQuery.Collections 欄位記錄此 CVM 資料列。L16~L18 以父節點對應 CVM 資料列的變數、TopVM 取得的欄位名稱以及 SelCondition 設定 InCanNode 的 CVM 欄位。L19 判斷 InCanNode 位於 HTColForest 或是 HTValForest，若是在 HTValForest，則在對應 HTCanForest 的 OutQuery->Values 欄位記錄此 CVM 資料列，而如果是在 HTColForest，則代表這是 where 子句產生的選擇條件式，因此會在對應 HTCanForest 的 OutQuery-> ValueConditions 欄位記錄此 CVM 資料列。最後回傳 CVM 欄位更新後的 InCanNode。

演算法名稱：
SetVMs（InCanNode,VMs,SelCondition）
輸入：　InCanNode /*正規節點*/
　　VMs /*用來設定 CVM 欄位的 VM 資料列集合*/
　　SelCondition /*選擇條件式*/
輸出：　InCanNode /*CVM 欄位更新後的 InCanNode*/
變數說明：　TopVM/*VMs 中權重最高資料列 */
　　ColName /*TopVM 中的集合名稱*/
　　PLevel /*InCanNode 所在的集合*/
　　ParentCanNode /*InCanNode 的父節點*/
　　PCVMs /*ParentCanNode 的 CVMs 欄位*/
　　OutVar /* PCVMs 中取得的 CVM 資料列的變數*/

```
L01 TopVM=VMs[0];
L02 ColName=TopVM->GetColName（）;
L03 PLevel= Getleve（CanNode->getLn（））;
L04 ParentCanNode= InCanNode ->Parent（）;
L05 PCVMs= ParentCanNode->getCVMs（）;
L06 For each PCM in PCVMs
L07 {
L08       If（PCM->getColName（）==ColName）
L09       {
L10             OutVar=PCM->GetVar（）;
L11             PCM->setUsedflag（true）;
L12             HTCanForest[PLevel]->OutQuery->Collection->Add
L13 （OutVar+PCM->GetColName（））;
L14       }
L15 }
L16 CVMs= CanNode->getCVMs（）;
L17 Column= TopVM->getColumn（）;
L18 CVMs->Add（OutVar,Column,true）;
L19 If（IsInValForest）
L20 {
L21 HTCanForest[PLevel]->OutQuery->Values->Add（OutVar+ “.”+Column）;
L22 }
L23 Else
L24 {
L25 HTCanForest[PLevel]->OutQuery->ValueConditions->Add（OutVar+
```


L26	“.”+Column+selCondition）;
L27	}
L28	Return InCanNode;

圖 4.10 SetVMs 演算法

以下以範例4.1的CVForest說明建立CanForest的建立過程，建立的CanForest如範例4.2所示。表4.1記錄CanForests正規集合樹中的CVM欄位，表4.2記錄CanForests正規值樹中的CVM欄位。BuildCanColForest演算法會依序處理每一個集合樹根節點，以處理集合樹“s”的根節點supplier為例，L04會進入CreateCanNode函式，在CreateCanNode演算法中，由於節點路徑不含跳層，因此L07~L08會根據CVForest中的集合樹“s”根節點新增一個正規節點，L10會進入DealWithPCPath函式。在DealWithPCPath演算法中，由於CM的鍵值存在“/order-ship/suppliers/supplier”，因此會進入L02的SetCMs函式。在SetCMs演算法中，supplier節點的CVMs欄位會加入{（S_1, SUPPLIER,FALSE），（S_2, PARTSUPP,FALSE）,（S_1, LINEITEM,FALSE）}。

而BuildCanColForest演算法在處理集合樹“a”的根節點時。L18的IsSubQVar會得知此節點為子查詢句的變數宣告，L19設定LetName為 a_0，並將（a_0,null,TRUE）加入a節點的CVMs欄位，L23在PLevel為 “0”的OutQuery->Collections加入（a_0,null），L24將PLevel為“2”的HTCanForest的LetName設定為a_0。

而在處理值樹“s”的子節點name時，CreateCanNode演算法中，由於節點路徑不含跳層，因此L07~L08會根據CVForest中的集合樹“s”的子節點name新增一個正規節點，L10會進入DealWithPCPath函式。在DealWithPCPath演算法中，由於VM的鍵值存在“/order-ship/suppliers/supplier/name”，因此會進入L05的SetVMs函式。在SetVMs演算法中取得的對應表格欄位為SUPPLIER.NAME，L04~L15會到父節點supplier中將SUPPLIER表格的usedflag設為真，並將（S_1,SUPPLIER）資料放

入OutQuery.Collections。最後OutQuery.Collections會加入（S_1.NAME）。

以下為所有路徑不含跳層的節點產生的OutQuery欄位。

PLevel	Values	Collections	ValueConditons
0	（S_1.NAME）， （a_0.NAME）	（S_1,SUPPILER）， （a_0, “”）	
2	（C_2.NAME）	（C_2,CUSTOMER）	

而根據目前的OutQuery欄位可以產生如下的SQL。

```
L01  Select S_1.NAME,a_0.NAME
L02  From SUPPLIER S_1,（
L03        Select C_2.NAME
L04        From CUSTOMER C_2
L05        ）as a_0
```

4.4.2. 處理路徑包含跳層的情況

圖 4.11 的 DealWithADPath 演算法處理路徑包含跳層的情況，L01~L05 利用路徑正規表示法逐筆比對 HTCM 中的資料，並將 HTCM 中符合的路徑加入 MatchPaths 中。若 MatchPaths 中的資料個數大於零，代表利用路徑的正規表示法可產生集合節點，L08 會呼叫 DealWithCanADColNode 函式，此函式會依 MatchPaths 以及輸入節點新增正規節點和設定 CVMs 欄位，並將新增的正規節點加到輸入的正規節點底下。SetADNode 函式如圖 4.13 所示。若路徑正規表示法不為集合節點，則 L12~L16 利用正規表示法逐筆比對 HTVM 中的資料，HTVM 中符合的資料會加入MatchPaths 中。若MatchPaths 中的資料個數大於零，代表利用路徑正規表示法可產生值節點，L19 和 L08 一樣是呼叫 SetADNode 函式。若為遞迴節點則由 L23~L25 處理，L24 的 DealWithADPathRecNode 如圖 4.15 所示。

演算法名稱：
DealWithADPath（ InCanNode ,
InCVChildNode,SelCondition,RegularPath ,HTCM,HTVM）
輸入： InCanNode
InCVChildNode /* 用來產生正規節點的 CVNode，產生的正規節點會成為 InCanNode 的子節點*/
SelCondition /* 選擇條件件*/
RegularPath /*路徑的正規表示式*/
HTCM/*記錄 CM 資料的雜湊表，雜湊表的鍵值為節點路徑，內容值為節點路徑對應的 CM 資料列*/
HTVM/*記錄 VM 資料的雜湊表，雜湊表的鍵值為葉節點路徑，內容值為葉節點路徑對應的 VM 資料列*/
輸出： CanChildNode /*依 InCVChildNode 新增的正規節點*/
變數說明：matchPaths /*依 RegularPath 在 HTCM 或 HTVM 中找到的符合路徑集合 */

```
L01 for each HTCM in CM{
L02     if（RegularPath ->IsMatch（HTCM.Key））{
L03         matchPaths->Add（HTCM.Key）;
L04     }
L05 }
L06 If（matchPaths->count>0）{
L07     IsColNode=true;
L08     CanChildNode= SetADNode（matchPaths ,InCanNode, InCVChildNode,
L09 SelCondition ,HTCM,HTVM）;
L10 }
L11 If（IsColNode==false）{
L12     for each VM in HTVM {
L13         if（RegularPath ->IsMatch（VM.Key））{
L14             matchPaths->Add（VM.Key）;
L15         }
L16     }
L17     If（matchPaths->count>0）{
L18         IsValNode=true;
L19         CanChildNode= SetADNode（matchPaths ,InCanNode, InCVChildNode,
L20 SelCondition ,HTCM,HTVM）;
L21     }
L22 }//end If（IsColNode==false）
```


```
L23 If（IsRecursiveNode（InCVChildNode））{
L24       DealWithADPathRecNode（InCVChildNode）;
L25 }
L26 Return CanChildNode;
```

圖 4.11 DealWithADPath 演算法

圖 4.12 SetADNode 演算法的輸入參數 MatchPaths 記錄所有符合路徑正規表示式的路徑，演算法會依照 MatchPaths 產生正規節點，並將產生的正規節點加到輸入參數 InCanNode 底下。若輸入節點為根節點，L03 會取得輸入節點的 RootJoin，並以此 RootJoin 到 CanForests 取得對應的 RootJoin，取得的 RootJoin 稱為 CanRootJoin，L04 則將 CanRootJoin 的路徑指派給 InPath。若輸入的節點不為根節點，則以輸入節點的路徑為 InPath。

L11 依序處理 MatchPaths 中的每一條路徑，L13 的 RelativePath 記錄輸入的節路徑與 InPath 之間的相對路徑，L14~L37 的處理方式和圖 4.3 的演算法 HTColForest[Var].AddChildNode 相似，同樣解析 RelativePath 路徑，然後將結果逐一建立正規節點，若有相同的節點則不再重複建立。L15 將 RelativePath 分解成多組{軸-節點名稱}，並循序處理每一組{軸-節點名稱}。L18 行在 CurrentNode 底下尋找符合{軸-節點名稱}的子節點，並將取得的子節點指定給 ChildNode。L19~L35 處理不存在符合{軸-節點名稱}的子節點的情況。L21 依據輸入節點建立正規節點，並將其加入到 CurrentNode 底下。當 ChildNode 為集合節點，L24 會用 SetCM 函數設定 CVMs 欄位。注意到 L25~L29，當輸入節點為根節點，第一組{軸-節點名稱}建立的正規節點的 RootJoin 會指向 CanRootJoinNode。當 ChildNode 為值節點時，L32 會用 SetVM 函數設定 CVMs 欄位。

演算法名稱：
SetADNode（InCanNode, InCVChildNode,
MatchPaths,SelCondition,HTCM,HTVM）
輸入： InCanNode /*輸入的正規節點*/
InCVChildNode /*CVForest 中的節點*/
matchPaths /*所有符合條件的路徑*/
SelCondition /*選擇條件式*/
HTCM/*記錄 CM 資料的雜湊表，雜湊表的鍵值為節點路徑，內容值為節點路徑對應的 CM 資料列*/
HTVM/*記錄 VM 資料的雜湊表，雜湊表的鍵值為葉節點路徑，內容值為葉節點路徑對應的 VM 資料列*/
輸出： InPath /*InCanNode 的路徑點*/
CanRootJoinNode/* InCVChildNode 對應於 CanForests 的 RootJoin*/
CurrentPath/*目前的節點路徑*/
RelativePath/* InPath 與 matchPaths 中路徑的相對路徑*/
CurrentNode /*目前走到的查詢樹節點*/
Axis_TagName /*軸與節點名稱*/
ChildNode /* CurrentNode 的子節點*/

```
L01 If（IsRootNode（InCVChildNode））
L02 {
L03     CanRootJoinNode= InCVChildNode->GetCanRootJoinNode（）；
L04     InPath = CanRootJoinNode ->getPath（）；
L05 }
L06 Else
L07 {
L08     InPath= InCanNode->getPath（）；
L09 }
L10 CurrentPath= InCanNode->getPath（）；
L11 For each path in MatchPaths
L12 {
L13     RelativePath= path- InPath;
L14     CurrentNode= InCanNode;
L15     For each Axis-TagName pair in RelativePath
L16     {
L17         CurrentPath+= Axis+TagName;
L18         ChildNode=CurrentNode->Search（Axis,tagName）；
L19         If（ChildNode==NULL）
```


```
L20             {
L21                     ChildNode = CurrentNode.AddChild
L22 （InCVChildNode ,CurrenPath）;
L23                     If（IsColNode（ChildNode））{
L24                         SetCM（ChildNode,HTCM[CurrentPath]）;
L25                         If(IsFirstPair(Axis-TagName) && IsRootNode
L26 （InCVChildNode））
L27                         {
L28                             ChildNode->SetRootJoin(CanRootJoinNode);
L29                         }
L30                     }
L31                     Else if（IsValNode（ChildNode））{
L32                         SetVM（ChildNode,HTVM[CurrentPath],selCondition）;
L33                     }
L34             }
L35         }
L36         CurrentNode= ChildNode;
L37 }
L38 Return InCanNode;
L39
```

圖 4.12 SetADNode 演算法

延續範例4.1，在處理集合樹 “o”時，輸入參數RegularPath 為/order-ship/suppliers/supplier[/[A-Za-z]+]*/order，DealWithADPath的L01~L05會在HTCM中找到符合的資料/order-ship/suppliers/supplier/part/order。此路徑會加入matchPaths。接著進入L08的SetADNode函式。SetADNode演算法中， L03中，根據集合樹 “o”的RootJoin集合樹 “s”，我們在CanForests取得正規集合樹 “s”。並將正規集合樹 “s”的路徑 /order-ship/suppliers/supplier設為InPath。

由於/order-ship/suppliers/supplier/part/order與/order-ship/suppliers/supplier的相對路徑為 /part/order，L13的RelativePath會記錄/part/order。L14~L37會將相對路徑分解成{（/,part），（/,order），建立了part節

點之後，part節點底下會再加入order節點。Part和order節點會用HTCM設定CVMs欄位，為方便說明，CVMs欄位中給予開頭字元 “M_”代表由跳層路徑產生的中間集合。例如我們會將part節點的變數設為M_p。L25~L29 會讓part節點的RootJoin指向正規集合樹 “s”。

處理完CVForest後，CanForests產生的OutQuery欄位如表格4.3所示。

PLevel	Values	Collections	ValueConditons
0	（S_1.NAME）， （a_0.NAME）	（S_1,SUPPILER）， （a_0, “”）	
2	（C_2.NAME）	（C_2,CUSTOMER）， （O_1,ORDER）	（O_1.TOTALPRICE>10000）

表格 4.3 CanForests 的 OutQuery 欄位

而根據目前的OutQuery欄位可以產生如下的SQL。

```
L01  Select S_1.NAME,a_0.NAME
L02  From SUPPLIER S_1,（
L03        Select C_2.NAME
L04        From CUSTOMER C_2, ORDER O_1
L05        WHERE O_1.TOTALPRICE>10000
L06        ）as a_0
```

4.4.3. 遞迴節點的處理方法

當要處理的遞迴節點其路徑不包含跳層時，會先判斷路徑是否為有效的遞迴路徑，若是有效路徑，則利用CM中的LoopPath欄位將路徑中所有符合LoopPath欄位的路徑移除，也就是會移除迴圈。例如路徑為/order-ship/customer/introuduce/customer/introduce/customer，LoopPath為/introduce/customer，在移除兩個LoopPath後路徑會變成/order-ship/customer。接著以移除掉迴圈的路徑在HTCM或HTVM中尋找符合的資料列，最後設立CVMs欄

位。

圖 4.14的DealWithPCPathRecNode函式處理路徑中沒有跳層的遞迴節點。L02會判斷子節點的路徑是否為有效的遞迴路徑，OmitCyclePath函式會移除掉InPath中所有的迴圈，若移除掉迴圈的路徑在CM或VM中有符合的資料列，則回傳迴圈移除後的路徑。若回傳路徑不為空，代表輸入的Inpath為有效路徑，NoCyclePath記錄OmitCyclePath函式得到的結果。若NoCyclePath 不為空，L04~L07依據InCVChildNode新增正規節點並將其加入InCanNode底下，新增的正規節點其路徑為NoCyclePath。L08的SetADNode函式會取出RecPCPath 與NoCyclePath 的相對路徑，然後依路徑逐一正規節點。

演算法名稱： DealWithPCPathRecNode（InCanNode, InCVChildNode） 輸入： InCanNode InCVChildNode 輸出： ReturnNode 變數說明： RecPCPath /* InCVChildNode 的路徑，路徑中包含迴圈*/ NoCyclePath /*移除掉迴圈的 RecPCPath */	
L01	RecPCPath= InCVChildNode->getpath（）;
L02	NoCyclePath=OmitCyclePath（RecPCPath）;
L03	If（NoCyclePath）{
L04	NewCanNode=new CanNode（）;
L05	NewCanNode->SetNodeData（InCVChildNode）;
L06	NewCanNode ->setPath（ NoCyclePath）;
L07	InCanNode ->AddChild（NewCanNode）;
L08	ReturnNode=SetADNode（NewCanNode,
L09	InCVChildNode,RecPCPath,SelCondition,CM,VM）
L10	}
L11	Return ReturnNode;

圖 4.13 DealWithPCPathRecNode 演算法

而當遞迴節點的路徑包含跳層時，本論文將此視為特殊情況處理，目前論文僅處理查詢句中具有兩個變數，一個變數為可重覆節點，另一個變數為遞迴節點，

且兩者在 DTD 為同一個節點的情況。本節以範例 4.3 說明遞迴節點路徑包含跳層的情況，範例 4.3 會列出顧客以及引薦的顧客，引薦的顧客會包含直接和間接引薦的顧客，例如 a 引薦 b，b 引薦 c，則稱 a 直接引薦 b，a 間接引薦 c。查詢句中的 $c1//customer 會遞迴取出所有的 customer 節點。輸入及輸出的查詢句如範例 4.3 所示。

【範例 4.3】

輸入的 XQuery	輸出的 SQL
For $c1 in /order-ship/customer $c2 in $c1//customer Return $c1@name,$c2@name	with recursive Rec_CUSTOMER（CUSTKEY1, CUSTKEY2）as （ select C1.CUSTKEY, *C2*.CUSTKEY from CUSTOMER C1, CUSTOMER C2, INTRODUCE I where C1.CUSTKEY=.I.CUSTKEY1 AND C2.CUSTKEY=.I.CUSTKEY2 union select C1.CUSTKEY, Rec_CUSTOMER.CUSTKEY2 from CUSTOMER C1, Rec_CUSTOMER where C1.CUSTKEY= Rec_CUSTOMER.CUSTKEY2 ） select * from Rec_CUSTOMER

以下簡單說明遞迴節點的處理方法。當遞迴節點的路徑包含跳層時，首先用"//"切開節點路徑， "//"之前的路徑稱為 StartPath，接著取出遞迴節點 NestJoin 的節點，判斷其路徑和 StartPath 是否相同，若相同則代表會形成迴圈。例如$c in //customer，$c2 in $c//customer。$c 會改寫成/order-ship/customer，$c//customer 等於/order-ship/customer//customer，StartPath 為/order-ship/customer，StartPath 與 $c 的路徑相同，兩者相同代表會形成迴圈。最後利用 SQL 的 Recsive With 語法記錄並輸出遞迴節點轉換後的結果並結束轉換查詢句。對應的演算法為 DealWithADPathRecNode 演算法。

圖 4.15 的 DealWithADPathRecNode 演算法中，L02 以 StartPath 記錄路徑中“//”之前的路徑。L03~L11 處理路徑中僅有一個 TagName 的遞迴節點，例如“//customer”。L05~L07 新增正規節點，並將其加入到 InCanNode 底下，L08 取出此遞迴節點視為可重覆節點時的路徑，L09 以取得的資料設定正規節點的 CVMs 欄位。

L11~L12 取出 InCVChildNode 的 RootJoin 連結節點以及 RootJoin 連接節點的路徑。若連結節點的路徑和 StartPath 相同，則 L13~L15 判斷 StartPath 和連接節點的路徑是否相同，若相同則代表會形成迴圈。L17 的 TransAndOutPutRecNode 函式會利用 SQL 的 Recsive With 語法記錄並輸出遞迴節點轉換後的結果並結束轉換查詢句。

演算法名稱： DealWithADPathRecNode（InCanNode, InCVChildNode,CM） 輸入：　InCanNode 　　InCVChildNode 　　CM 輸出：　CanChildNode 變數說明：　Recpath /* InCVChildNode 的路徑*/ 　　StartPath　/* Recpath 中第一個 “//”字樣前的路徑*/	
L01	Recpath= InCanNode ->getpath（）;
L02	StartPath=Recpath ->substring（0, Recpath->indexof（“//”））;
L03	If（StartPath ==Null）
L04	{
L05	CanChildNode=new CanNode;
L06	CanChildNode->SetNodeData（InCVChildNode）;
L07	InCanNode->AddChild（CanChildNode）;
L08	RepeatableNodePath=GetRepeatableNodePath（Recpath）;
L09	CanChildNode= SetCM（CanChildNode,CM[RepeatableNodePath]）;
L10	}
L11	JoinNode =InCVChildNode->getJoinNode（）;
L12	JoinPath= JoinNode->getPath（）;
L13	If（JoinPath== StartPath）{

L14	TransAndOutPutRecNode（ “//”+LastTagName]）;
L15	}
L16	Return CanChildNode;

圖 4.14 DealWithADPathRecNode 演算法

以範例 4.3 為例，經過 CVForestBuilder 處理後，建立的 CVForest 如圖 4.15 所示。CanForestBuilder 模組維持樹的結構並加入對應資料。在 DealWithADPathRecNode 演算法中，L02 以 StartPath 記錄$c2 路徑中“//”之前的路徑。由於$c2 包含$c 變數，$c 會改寫成/order-ship/customer，所以$c2 的路徑 $c//customer 等於 /order-ship/customer//customer，最後 StartPath 為 /order-ship/customer。L11~L12 取得 RootJoin 連結節點$c，由於$c 的路徑和 StartPath 一樣是/order-ship/customer，這代表兩個節點會形成迴圈，所以會進入 L17 的 TransAndOutPutRecNode 函式。

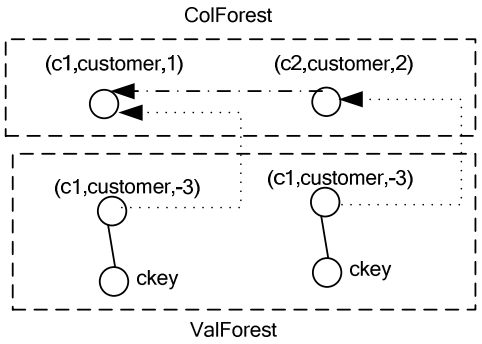


圖 4.15 範例 4.3 的 CVForest

圖 4.16 的 TransAndOutPutRecNode 演算法會利用 SQL 的 Recursive With 語法記錄並輸出遞迴節點轉換後的結果。轉換後會產生基本子句和遞迴子句，Recursive 查詢句用”union” 關鍵字連結兩者。基本子句是由兩個表格產生，遞迴子句是由 Recursive 查詢句的查詢結果以及表格產生。因此 L01~L03 中由 OuterQ 記錄要輸出的變數名稱、欄位名稱，BasicQ、RecursiveQ 則分別代表基本子句和遞迴子句。L04 的 GetValNode 會取出正規節點在正規值樹中的葉節點，取得的

葉節點會記錄在 CanValTreeLeafNode，L05 的 ColName 記錄 CanValTreeLeafNode 在父節點中的對應集合，L05 給予對應表格變數、L06 以對應表格產生 Recursive With 查詢句的別名。L08~L09 則取得 JE 中的連結欄位。L11~L20 則分別在 OuterQ、BasicQ、RecursiveQ 中加入要輸出資料。L21~L31 將 OuterQ、BasicQ、RecursiveQ 的資料組合成完整的 Recursive With 查詢句。最後輸出轉換後的結果。

演算法名稱： TransAndOutPutRecNode（InCVChildNode） 輸入：　InCVChildNode /*正規節點 */ 輸出：　RecursiveWithStr /*Recursive With 查詢句*/ 變數說明：　OuterQ　/*記錄 Recursive With 查詢句的別名和輸出值*/ BasicQ、RecursiveQ/*記錄 Recursive With 查詢句中子查詢句的資料*/ CanValTreeLeafNode /**/ ColName/*遞迴節點的對應表格*/ ColNameVar*遞迴節點的對應變數*/ JoinColumn1、JoinColumn2 /*JE 中的對應連結*/ BasicQStr /* 組合 basicQ 的資料而成的查詢句*/	
L01	OuterQ=new OutQuery（）;
L02	BasicQ =new OutQuery（）;
L03	RecursiveQ=new OutQuery（）;
L04	CanValTreeLeafNode=GetValNode（InCVChildNode）;
L05	ColName=CanValTreeLeafNode-> getColName（）;
L06	ColNameVar= ColName->getVar（）;
L07	OutVar= “Rec_”+ ColName;
L08	JoinCondition=getJoinCondition（JE[ColName+ “_”+ ColName]）;
L09	OuterQ->Collections->AddTable（JoinCondition）;
L10	OuterQ->Collections->Add（OutVar）;
L11	OuterQ->Values->Add（ JoinColumn1）;
L12	OuterQ->Values->Add（ JoinColumn2）;
L13	BasicQ ->Collections->Add（ColName）;
L14	BasicQ ->Values->Add（ColNameVar + JoinColumn1）;
L15	BasicQ ->Values->Add（ColNameVar + JoinColumn2）;
L16	RecursiveQ->Collections->Add（OutVar）;
L17	RecursiveQ->Collections->Add（ColName）;

```
L18 BasicQ ->Values->Add（ColNameVar + JoinColumn1）;
L19 BasicQ ->Values->Add（OutVar + JoinColumn2）;
L20 BasicQ ->JoinConditons->Add（ColName + JoinColumn1+ OutVar
L21 +JoinColumn2）;
L22 BasicQ Str= BasicQ ->GetOutPut（）;
L23 RecursiveQStr= RecursiveQ->GetOutPut（）;
L24 RecursiveWithStr+=”Recursive With”
L25 + OutVar+“（”+ JoinColumn1+ JoinColumn1+”）”+”as（”
L26 + basicQStr
L27 +”union”
L28 + basicQStr
L29 +”）”
L30 + “select *”
L31 +”from ”
L32 + OutVar;
    Return RecursiveWithStr;
```

圖 4.16 TransAndOutPutRecNode 演算法

延續範例 4.3，在 TransAndOutPutRecNode 演算法中，在處理集合樹 “c2” 時，其根節點路徑為 “/order-ship/customer//customer”，L04 會取出回傳的值樹葉節點/order-ship/customer//customer/name，L05 的 ColName 記錄集合樹 “c2”中的對應表格 CUSTOMER，L07 的 OutVar 為 Rec_CUSTOMER，L08 的 JoinCondition 會在 JE 取得{（CUSTOMER.CUSTKEY=INTRODUCE.CUSTKEY1），（INTRODUCE.CUSTKEY2=CUSTOMER.CUSTKEY）}。L09 發現 JoinCondition 中有兩個 CUSTOMER 表格和一個 INTRODEUCE 表格，分別給予 CUSTOMER 表格變數 C1、C2，INTRODEUCE 表格給予變數 I 後，OuterQ->Collections->加入這三個表格。L11~L22 則加入對應資料到 OuterQ、BasicQ、RecursiveQ 中，處理後的 OuterQ、BasicQ、RecursiveQ 如下所示。最後轉換出的查詢句如範例 4.3 所示。

Query	Values	Collections	JoinConditions
OuterQ	（CUSTKEY1） （CUSTKEY2）	（Rec_CUSTOMER, “”）	
BasicQ	（C1.CUSTKEY）， （C2.CUSTKEY）	（C1,CUSTOMER）， （ C2,CUSTOMER）， （ I ,INTRODUCE）	（C1.CUSTKEY=.I.CUSTKEY1）， （C2.CUSTKEY1=.I.CUSTKEY2）
RecursiveQ	（C1.CUSTKEY）， （Rec_CUSTOMER.CUSTKEY2）	（C1,CUSTOMER）， （Rec_CUSTOMER , “”）	（ C1.CUSTKEY = Rec_CUSTOMER.CUSTKEY2）

第 5 章　轉換處理

在本章中，我們說明完整的查詢句轉換過程。首先我們會提出整體架構，接著說明相關的演算法。

5.1. 轉換系統架構

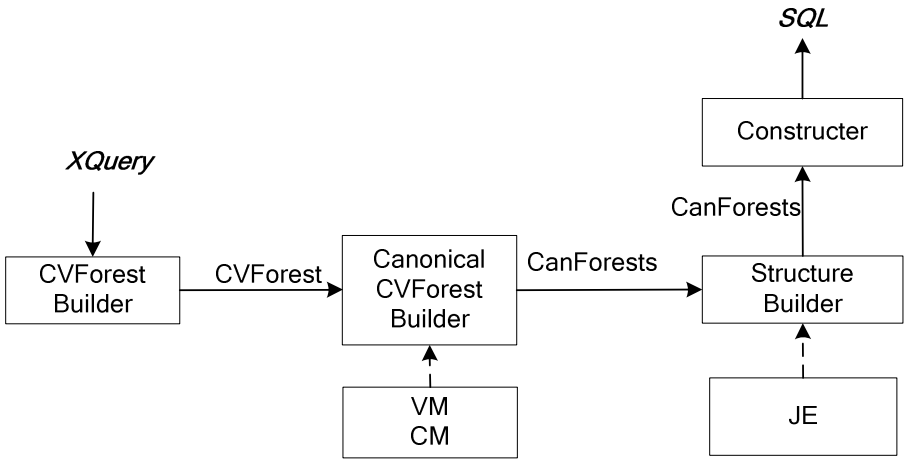


圖 5.1 轉換系統架構

查詢語言轉換系統的架構如圖 5.1 所示，使用者下達查詢句後，CVForest Builder 會剖析查詢句，查詢句中的每個集合會建立集合樹，每個回傳的值會建立值樹，所有集合樹和值樹的集合稱為 ColValForest，簡稱 CVForest。Canonical CVForest Builder 會利用 VM、CM 的資訊刪減不必要的節點，加入對應的轉換資料到節點，若葉節點的路徑包含跳層，則在根節點與葉節點中間加入非葉節點。CVForest 經過處理後會產生多個 Canonical CVForests，簡稱 CanForests，輸出的對應轉換資料會記錄在 CanForests 的 OutQuery 欄位。接著 StructureBuilder 利用最小生成樹演算法及對應函數 JE 處理集合之間的結構連結最小生成樹演算法會確保集合之間具有合理的結構，最小生成樹中的連結關係即為要輸出的結構資料，該連結關係會記錄在 CanForests 中。而 Constructer 會合併 CanForests 的資料並輸出轉換後的查詢句。

在上一章中，我們已經解釋了 CVForest Builder 和 Canonical CVForest Builder。在本章中，我們將解釋 Structure Builder 和 Constructer。

5.2. StructureBuilder

由於集合之間只要形成樹結構即為合理的連結關係，因此StructureBuilder利用最小生成樹（Minimum Spanning Tree）及對應函數JE輸出集合間必要的結構，最小生成樹可以為集合產生個數最少且合理的連結。CanForest中所有的非葉節點皆為集合節點，因此會以非葉節點的對應集合(記錄於CVMs欄位中)為輸入的點。利用Prim演算法，點和邊會產生最小生成樹。接著最佳化最小生成樹，演算法會刪除不輸出值且連結線為1的節點，這是因為刪除該節點不會破壞樹狀連結的關係，也就不會違反等價定義。

[WS]為最小生成樹所下的定義如下所示。在一給定的無向圖 G =(V, E)中，V 為點的集合，E 為邊的集合。(u, v) 代表連接點 u 與點 v 的邊，其中邊存在於 E，即 (u,v) ∈ E。而 w (u,v) 代表 (u,v) 的成本 (Cost)。若存在 T 為 E 的子集且 G2 = (V,T) 為無循環圖表，同時所有邊的成本總和最小，也就是 $w(T) = \sum_{(u,v) \in T} w(u,v)$ 最小，則此 G2 為 G 的最小生成樹。在最小生成樹中，若點的個數為 n，則最小生成樹恰好使用 n-1 個邊。邊不可形成迴圈，且最小生成樹不必是唯一的。

以下我們先定義最小生成樹中的點及邊，最後介紹建立最小生成樹的演算法。

【定義5.1】：MSTNode與MSTEdge

MSTNode = { var, CVM, traveled, EdgeNum } 。

- var: MSTNode的名稱，var與CVM的var欄位內容相同。
- CVM: 與此點對應的正規集合節點CVM欄位。CVM欄位記錄的資料為[（輸出變數，對應表格，是否輸出）]。當正規集合節點的CVMs欄位中具有usedflag為真的資料，則僅為正規集合節點usedflag為真的CVM建立MSTNode。而當正規集合節點的CVMs欄位中不具有usedflag為真的資料，則為正規集合節點所有的CVM資料建立MSTNode，這是因為其他的集合可能透過此筆CVM連結。
- traveled:記錄是否已造訪此點。已造訪則記錄TRUE。
- EdgeNum: 在最小生成樹中連結此點的邊個數。

MSTEdge = { Table1, Table2, Weight, JoinConditions} 。

當兩個集合之間可以連結，則會為其產生MSTEdge。以下為MSTEdge的欄位內容。

- Table1、Table2: MSTNode中CVM欄位中的對應表格。
- Cost: 以Table1、Table2為鍵值到JE中取得的Cost。若Table1或Table2其中之一未輸出值（usedflag為false），則Cost會加3。若2者皆未輸出值，則Cost加6。沒有輸出值的集合會增加Cost，這是因為當表格同時與多個表格連結時，應該要優先選擇有輸出值的集合。
- JoinConditions: Table1、Table2的連結條件式。

以範例 4.2 的 CanForests 為例，產生的所有 MSTNode 與 MSTEdge 如圖 5.2 所示，MSTNode 的 CVM 欄位如表 5.1 所示。圖 5.2 中以不同的虛線方框區分不同的 CanForest。圖中每個 MSTNode 以圓圈代表，若為沒有值輸出的集合則以雙圓圈代表，MSTNode 標明 var。圖中的線為輸入的線，虛線代表最小生成樹沒有使用此連結，實線代表最小生成樹使用此連結。線上標示 JoinConditions 以及 Cost。

每個 MSTNode 連結的實線個數為 EdgeNum，MSTNode 若為實心圓形代表已造訪。

以圖 5.2 的 PLevel 為 2 說明，由於 order、customer 節點的 CVMs 欄位都有輸出值的對應表格，因此 order 節點為 ORDER 表格建立 MSTNode，customer 節點為 CUTOMER 表格建立 MSTNode。而 part 節點由於沒有輸入的對應表格，因此會為 PART 表格和 PARTSUPP 表格建立 MSTNode，其中 PART 表格和 PARTSUPP 之間連結的 Cost 原本為 2，但因為 2 個表格皆未輸出，因此 Cost 會加 6 變為 8。而 PARTSUPP 表格和外部查詢句的 SUPPLIER 表格之間的連結 Cost 為 2，由於 PARTSUPP 未輸出，因此 Cost 會加 3 變成 5。ORDER 表格和 CUSTOMER 表格可以連結，因此兩點之間的線上會標示連結表示式 ORDER.CUSTKEY=CUSTOMER. CUSTKEY。從圖中可看出所有 MSTNode 的初始狀態，所有的 MSTNode 皆未造訪，因此皆為空心圓形，由於尚未建立最小生成樹，因此樹中的線皆為虛線，而所有的 MSTNode 皆沒有連結實線，其 EdgeNum 皆為 0。

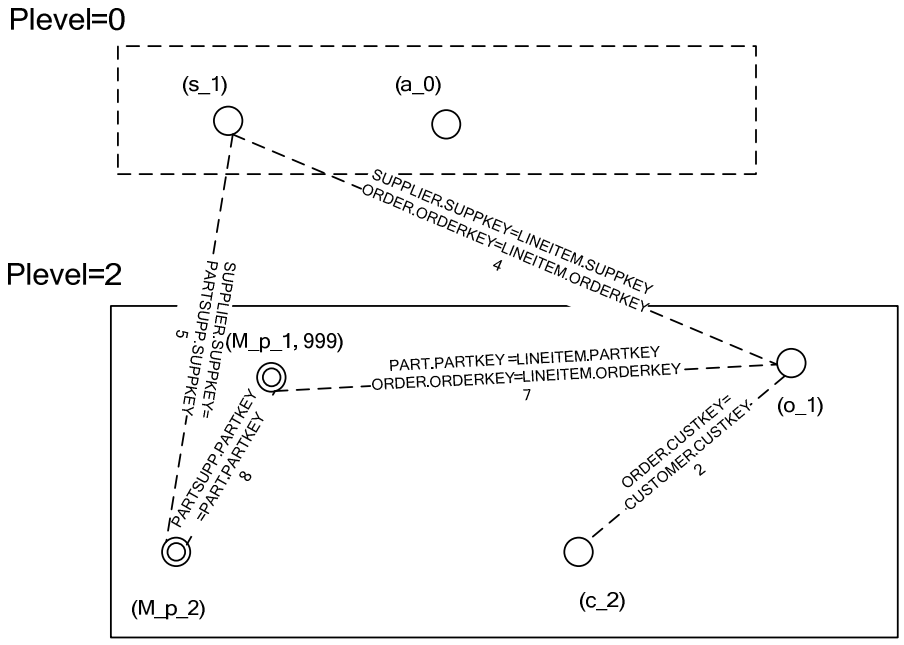


圖 5.2 範例 4.2 產生的 MSTNodes

MSTNodes 的 CVM 欄位如表 5.1 所示。

var	CVM= {Var,ColOrVal,UsedFlag}
s_1	s_1,SUPPLIER,True
a_0	a_0,null,TRUE
M_p_1	M_p_1,PART,FALSE
M_p_2	M_p_2,PARTSUPP,FALSE
o_1	o_1,ORDER,TRUE
c_2	c_2, CUSTOMER, TRUE

表格 5.1 MSTNodes 的 CVM 欄位

CanForest會依最小生成樹決定最後要輸出的資料，產生最小生成樹的演算法如圖5.3的BuildMST（CanForest）所示。在圖5.3中，L01~L03依序處理CanColForest中所有的正規集合樹，L04~L06處理所有的正規值樹，所有非葉節點其CVMs欄位的所有資料都會產生對應的MSTNode，然後MSTNode會加到MSTNodes。其中L02及L05的GetMSTNodes（Node）函式，會為輸入節點Node及其非葉節點的CVMs欄位產生MSTNode，函式如圖5.4所示。L07的MST_Prim函式利用MSTNodes和JE產生最小生成樹，L08的RemoveRedundantMSTNodes函式將最小生成樹中不需要的點和邊去掉，L09將留下的JEConditions放入CanForest的OutQuery欄位，L10則檢查是否所有JEConditions中的表格都在OutQuery->Collections，若不在則放入。

演算法名稱：BuildMST（CanForest） 輸入：CanForest 輸出：CanForest	
L01	For each Node in CanColForest{
L02	MSTNodes+=GetMSTNodes（Node）;
L03	}
L04	For each Node in CanValForest{
L05	MSTNodes+=GetMSTNodes（Node）;

```
L06   }
L07 JEConditions =MST_Prim（MSTNodes,JE）;
L08 JEConditions =RemoveRedundantMSTNodes（MSTNodes）;
L09 CanForest ->OutQuery->Conditions->Add（JEConditions）;
L10 CheckOutCol（CanForest）;
```

圖 5.3 BuildMST 演算法

圖 5.4 的 GetMSTNodes（Node）會為 Node 及其子孫節點產生 MSTNode。L03~L08 取得節點的 CVM 中 usedflag 欄位為 True 的資料，並將這些資料指派給 MSTNodes。L06 和 L11 的 MSTNodes->Add（）函式會加入 CVMs 欄位中 OutValOrCol 不重覆的 MSTNode 到 MSTNodes。L09~L14 中，如果節點的 CVM 中沒有 usedflag 欄位為 True 的資料，則將 CVM 中所有的資料指派給 MSTNodes。最後 L15 回傳 MSTNodes。

演算法名稱：GetMSTNodes（Node）

```
L01 hasMarkCol=false;
L02 MSTNodes=new Collection<MSTNode >;
L03 for（int i=0;i< Node ->getCMsInTree（）->Count;i++）{
L04       if（Node ->getCMsInTree（）[i]->getusedflag（）==true）{
L05             hasMarkCol=true;
L06             MSTNodes->Add（CreateMSTNode（Node ->getCMsInTree（）[i]））;
L07       }
L08 }
L09 if（hasMarkCol==false）{
L10       for（int i=0;i< Node ->getCMsInTree（）->Count;i++）{
L11             MSTNodes->Add（CreateMSTNode（Node ->getCMsInTree（）[i]））;
L12       }
L13       ;
L14 }
L15 return MSTNodes;
```

圖 5.4 GetMSTNodes 演算法

延續範例 4.2 說明如何產生圖 5.2，以及用圖 5.2 如何取得結構的過程。因為

外部查詢句可以影響子查詢句，但子查詢句不會影響到外部查詢句，因此圖 5.3 的 BuildMST 函式中依父階層由低到高處理 CanForests。首先會處理 PLevel 為 2 的 CanForest，L01~L06 會依 CanColForest 及 CanValForest 的 CVMs 欄位產生 MSTNode，生成的 MSTNodes 如圖 5.2 所示。正規集合樹 “o”的 part 節點之 CVMs 欄位中，所有資料的 usedflag 欄位皆為 FALSE，因此 part 節點會產生的 PART 和 PARTSUPP 兩個 MSTNode。

接著處理 PLevel 為 0 的 CanForest，L01~L06 會依 CanColForest 及 CanValForest 產生 MSTNode。PLevel 為 0 的 CanForest 產生的 MSTNodes 如圖 5.2 所示。

圖 5.6 MST_Prim（UnTraveledNodes,JE）參考[WP]產生最小生成樹。因為樹中不需要重覆的表格，所以 L01 的 RemoveRedundantTable 函式移除名稱重覆的表格。L02 記錄最小生成樹應該有的邊個數，邊個數為節點個數減 1。L03~L04 取出 UnTraveledNodes 中第一個 MSTNode，將其放入 TraveledNodes 中並將此 MSTNode 從 UnTraveledNodes 移除，此點為最小生成樹中的第一個點，後續的程式碼會以此點向外擴展最小生成樹。

L05 以 LoopNum 為迴圈的計數器， L08~L24 的雙重迴圈會從已完成的最小生成樹向外擴展。當 MSTNode 為子查詢句的變數宣告時，由等價定義 2.3 得知，外部查詢句轉換後會輸出連結外部查詢句和子查詢句的連結條件句。且子查詢句會輸出連結條件句中和外部查詢句相關的表格欄位。L10 會最後才連結子查詢句，這是為了先建立外部查詢句的最小生成樹，再連結外部查詢句及子查詢句的最小生成樹。在 unTraveledNodes 中只剩下子查詢句的連結節點未處理時，L11 的 JoinSubQuery（node1,node2）函式會連結外部查詢句和子查詢句，其作法是找出 node2 代表的子查詢句中可以連結 node1 的 MSTNode 並取出連結資料，node1 為外部查詢句中的節點，node2 為子查詢句的連結節點。OutQuery.JoinConditions

會記錄子查詢句和外部查詢句的連結條件子句，而子查詢句中的 OutQuery.Values 會額外輸出連結資料代表的表格欄位。以圖 5.2 為例，s_1 會為 node1，a_0 會為 node2，而 s_1 最後將會與 o_1 連結。

利用 UnTraveledNodes 以及 JE 找出與目前最小生成樹具有最低 Cost 的點，MinEdgeCost 會記錄目前 Cost 最少的邊。若目前造訪的兩端點其邊的 Cost 比 MinEdgeCost 小，則 L18~L21 的 MinEdgeCost 記錄會邊的 Cost，MinNode1 記錄 TraveledNodes 使用的點，MinNode2 記錄 UnTraveledNodes 使用的點，MinJEConditions 記錄邊所代表的連結資料。在 L25~L27 中，如果最小生成樹無法擴展，則輸出錯誤訊息，反之則如 L28~L32 所示，MinNode1 和 MinNode2 的 EdgeNum 會各加 1; MinNode2 會放入 TraveledNodes 且從 UnTraveledNodes 中移除。最後輸出最小生成樹中所有邊代表的連結資料。

演算法名稱：MST_Prim（UnTraveledNodes,JE）	
輸入：	UnTraveledNodes 尚未造訪過的點
	JE 連結表格
變數說明：	
	TraveledNodes 已造訪過的 MSTNode
	MinEdgeCost 最小 Cost
	GetMinNode 布林值，記錄 UnTraveledNodes 是否有和 TraveledNodes 連結的點。。
	MinNode1 最小 Cost 邊中已造訪過的點
	MinNode2 最小 Cost 邊中尚未造訪過的點
L01	RemoveRedundantTable（UnTraveledNodes）;
L02	LoopNum= UnTraveledNodes->count-1;
L03	TraveledNodes.Add（UnTraveledNodes[0]）;
L04	unTraveledNodes.remove（UnTraveledNodes[0]）;
L05	For（int i=0;i< LoopNum;i++）{
L06	MinEdgeCost=999;
L07	GetMinNode=false;
L08	For each node1 in TraveledNodes{
L09	For each node2 in UnTraveledNodes {
L10	IsNested（node2 && UnTraveledNodes->count==1））{

```
L11                         MinJEConditions ＝JoinSubQuery（node1 ,node2）;
L12                   }
L13                   Key=（node1.CVM.OutColOrVal+ node2.CVM.OutColOrVal）
L14 or（node2.CVM.OutColOrVal+ node1.CVM.OutColOrVal）
L15                   if（JE.ContainsKey（key））{
L16                         if（JE[key]->edgeCost <MinEdgeCost）{
L17                               GetMinNode=true;
L18                               MinEdgeCost= JE[key]->edgeCost;
L19                               MinNode1=node1;
L20                               MinNode2=node2;
L21                               MinJEConditions=JE[key]->Conditions;
L22                         }
L23                   }//   if（JE.ContainsKey（key））
L24             }//   For each node2 in UnTraveledNodes
L25             If（GetMinNode==false）{
L26                   output Error Message;
L27             }else{
L28                   JEConditions->Add（MinJEConditions）;
L29                   MinNode2->EdgeNum++;
L30                   MinNode1->EdgeNum++;
L31                   MST->Add（MinNode2）;
L32                   UnTraveledNodes->Remove（MinNode2）;
L33             }
L34       }
L35 }
L36 Return JoinConditions;
```

圖 5.5 MST_Prim 演算法

延續圖 5.2 繼續說明產生結構的過程。在處理 PLevel 為 2 的 MSTNods 時，MST_Prim 演算法會將 ORDER 表格代表的點由空心圓形變成實心圓形，這代表 ORDER 表格已造訪過。接著由 ORDER 表格向外延伸最小生成樹，由於 ORDER 表格可以和 CUSTOMER 表格連結，因此 ORDER 和 CUSTOMER 之間的線改為實線，且 CUSTOMER 的空心圓形會變成實心圓形，這代表 ORDER 與 CUSTOMER 的連結表示式被用到，且 CUSTOMER 已造訪。JoinConditions 為

ORDER 與 CUSTOMER 的連結表示式{ O_1.CUSTKEY ,C_1.CUSTKEY }。

用同樣的方法，ORDER 會連結 PARTSUPP 表格，PARTSUPP 表格會連結 PART 表格。至此所有的 MSTNodes 皆已造訪過並成功產生最小生成樹。最後 MST_Prim 函式會回傳 JoinConditions。經 MST_Prim 演算法更新後的 MSTNodes 如圖 5.6 所示，MST_Prim 演算法處理完 PLevel 為 2 以及處理完 PLevel 為 0 後，得到的 JoinCollection 如表格 5.2 所示。

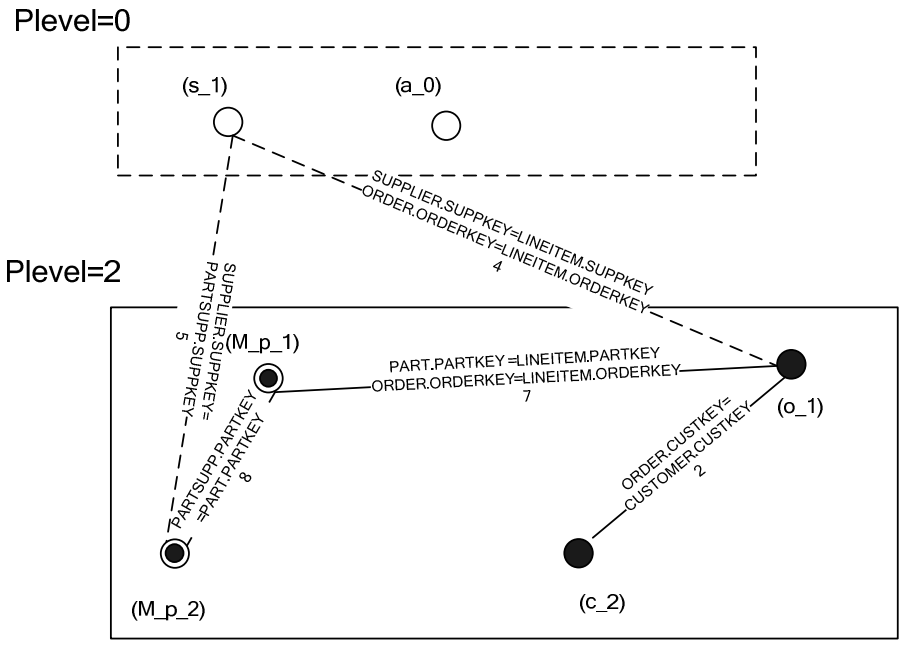


圖 5.6 更新後的 MSTNodes

PLevel	JoinCollection
0	（S_1.PARTRKEY= LINEITEM.PARTKEY）， （LINEITEM.ORDERKEY,a_0.ORDERKEY），
2	（O_1.CUSTKEY =C_2.CUSTKEY）， （M_P_1.PARTRKEY=LINEITEM.PARTKEY）， （O_1.PARTRKEY=LINEITEM.PARTKEY）， （M_P_1.PARTRKEY=M_P_2.PARTKEY）

表格 5.2 MST_Prim 演算法得到的 JoinCollection

而在處理 PLevel 為 0 的 MSTNodes 時， MST_Prim 函式會先造訪 SUPPLIER 表格。在未造訪的 MSTNode 只剩下 a_0 時，SUPPILER 會判斷可否與 a_0 連結，由於 a_0 為子查詢句的變數宣告，a_0 會連結到 PLevel 為 2 的 CanForest，因此 SUPPILER 會到 PLevel2 尋找可連結的 MSTNode，並將 a_0 刪掉。SUPPILER 表格可和 PARTSUPP 表格連結，也可以和 ORDER 表格連結，SUPPLIER 和 ORDER 之間的 Cost 為 4，而 SUPPLIER 和 PARTSUPP 的 Cost 為 5，所以 SUPPLIER 會和 ORDER 表格連結。由於 ORDER 表格位於子查詢句中，子查詢句需要輸出此連結欄位才能和外部查詢句連結，所以 PLevel 為 2 的 OutQuery.Values 會加入（O_1.ORDERKEY）。取得的連結條件中，會用 “a_0”取代 o_1，所以 JoinConditions 會加入對應的資料{（S_1.SUPPKEY,LINEITEM.SUPPKEY），（LINEITEM.ORDERKEY, a_0.ORDERKEY）}。至此所有的 MSTNodes 皆已造訪過並成功產生最小生成樹。更新後的 MSTNodes 的內容如圖 5.7 所示，圖中圓圈為虛線代表 MSTNode 被刪除。

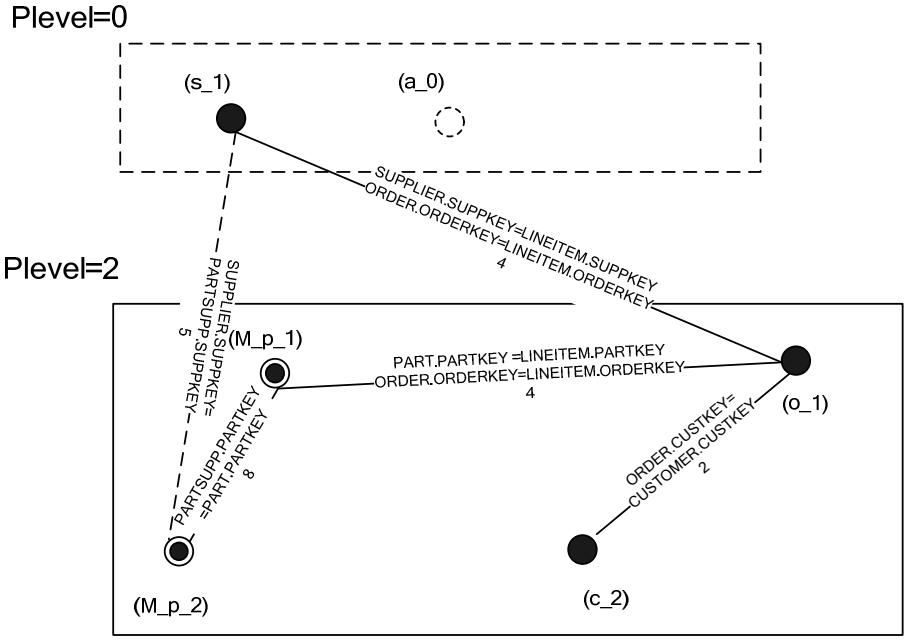


圖 5.7 MST_Prim 演算法產生的 MSTNodes

圖5.8的RemoveRedundantMSTNodes會將最小生成樹中不需要的點和邊去掉。演算法會判斷最小生成樹中每一個葉節點，若葉節點是不輸出的對應集合，則移除此葉節點。

在函式中，L03 判斷最小生成樹中的每一個葉節點代表的表格是否被輸出，若不是則將 JoinConditions 中和此表格相關的資料移除，且 IsAllLeafNodeOutPut 會設成 false。在 IsAllLeafNodeOutPut 為 false 的情況下會一直重覆執行 RemoveRedundantMSTNodes，直到所有葉節點都是被輸出的表格為止。最後回傳 JoinConditions。

演算法名稱： RemoveRedundantMSTNodes（JoinConditions,MST） 輸入： JoinConditions /*最小生成樹中每條邊所代表的 JE 條件*/ MST/*最小生成樹*/ 輸出：JoinConditions 變數說明：IsAllLeafNodeOutPut /*布林值，判斷是否所有葉節點都是被輸出的表格*/	
L01	IsAllLeafNodeOutPut=true;
L02	do{
L03	For each LeafNode in MST {
L04	If(LeafNode->CVM->usedflag==false)
L05	{
L06	RemoveFromJE（JoinConditions, LeafNode）;
L07	IsAllLeafNodeOutPut=false;
L08	}
L09	}
L10	}While(IsAllLeafNodeOutPut==false);
L11	return JoinConditions;

圖 5.8 RemoveRedundantMSTNodes 演算法

延續之前的範例，RemoveRedundantMSTNodes 演算法在處理 PLevel 為 2 的

最小生成樹的葉節點時，由於 PARTSUPP 表格不輸出，所以會將 PARTSUPP 表格移除，JoinConditions 中的（M_P_1.PARTRKEY=M_P_2.PARTKEY）也會被移除。接著再觀察所有的葉節點，此時發現葉節點 PART 表格不輸出，所以移除 PART 表格，JoinConditions 中的（M_P_1.PARTRKEY=LINEITEM.PARTKEY），（O_1.PARTRKEY=LINEITEM.PARTKEY）也會被刪除。

RemoveRedundantMSTNodes 執行完後，更新後的 JoinConditions 會加到 OutQuery-> JoinConditions 。BuildMST 的 L10 會檢查 JoinCollection 的所有表格是否都在 OutQuery->Collections，由於 LINEITEM 表格不在 PLevel 為 0 的 OutQuery->Collections，因此 OutQuery->Collections 會加入 LINEITEM 表格，並給予 LINEITEM 表格變數 R_L。更新後的 OutQuery 如下所示。

PLevel	Values	Collections	ValueConditons	JoinConditions
0	（S_1.NAME），（a_0.NAME）	（a_0, “”），（S_1,SUPPILER），（R_L,LINEITEM）		（S_1.PARTRKEY= R_L ,.PARTKEY），（ R_L .ORDERKEY,a_0.ORDERKEY）
2	（c_2.NAME），（O_1.ORDERKEY）	（O_1,ORDER），（C_2,CUSTOMER）	（O_1.TOTALPRICE>1000）	（O_1.CUSTKEY =C_2.CUSTKEY）

表格 5.3 經 MST_Prim 演算法更新後的 CanForests.OutQuery 欄位

5.3. Constructer

Constructer依階層碼有順序性的收集及合併CVForest的資料，最後輸出轉換後的查詢句。OutPut演算法以PLevel為 “0”的CanForest的OutQuery欄位為輸入參數，SQL變數記錄轉換結果，L02~L06收集Values欄位的資料，L06~L19收集Collections欄位的資料，如果集合為子查詢句，則依集合變數找出對應的CanForest，再用OutPut演算法遞迴處理此子查詢句。L20~L27處理where子句，最後回傳SQL變數。

演算法名稱：OutPut（OutQuery）
輸入：　OutQuery　　/* CanForest 的 OutQuery 欄位*/
輸出：SQL　/*x2s 轉換後的查詢句結果*/

```
L01 SQL="";
L02 SQL +="Select";
L03 for each Value in OutQuery.Values
L04 {
L05       SQL +=Value;
L06 }
L07 SQL +="From";
L08 For each collection in OutQuery.Collection
L09 {
L10       Var=collection.var
L11       If（IsNestQuery（var））{
L12             Nest_PLevel=GetPLevel（var）;
L13             NestQuery= OutPut（CanForests[Nest_PLevel].OutQuery）;
L14             SQL +="（"+ NestQuery +　"）as"+ var;
L15       }
L16       Else{
L17             SQL +=collection;
L18       }
L19 }
L20 SQL +="Where";
L21 for each JoinCondition in OutQuery.JoinConditions;
L22 {
L23       SQL +=Output JoinCondition;
L24 }
L25 for each ValCondition in OutQuery.ValConditions;
L26 {     SQL += ValCondition;
L27 }
L28 Return SQL;
```

圖 5.9 OutPut 演算法

延續之前的例子， 進入OutPut演算法時，輸入參數OutQuery為PLevel為 “0”的CanForest的OutQuery欄位，L02~L06會輸出OutQuery.Values的資料，L06~L19會

回傳OutQuery.Collections的資料，由於變數a連結到子查詢句，因此會遞迴呼叫OutPut函式。L20~L28則會回傳OutQuery.JoinConditions以及OutQuery.ValCondition的資料。轉換後的SQL查詢句如圖5.10所示。OutPut演算法的輸入參數 OutQuery 如表格 5.3所示。

```
Select s_1.NAME,a_0.NAME
From SUPPLIER s_1, LINEITEM R_L（
      Select c_2.NAME, o_1.ORDERKEY
      From ORDER o_1,CUSTOMER c_2
      Where     o_1.TOTALPRICE>10000
                AND o_1.CUSTKEY = c_1.CUSTKEY
      ）as a_0
Where s_1.SUPPKEY = R_L.SUPPKEY
        AND a.ORDERKEY= R_L.ORDERKEY
```

圖 5.10 範例 2.2 轉換後的 SQL 查詢句

第 6 章　SQL 轉換 XQuery

由於論文中主要描述 XQuery 如何轉換成 SQL，本章說明 SQL 轉換成 XQuery 的過程以及所需注意事項，論文中 SQL 轉換成 XQuery 的處理尚不完整，子查詢句僅處理出現在 From 子句的情況。本章以 S2X 簡稱 SQL 轉換成 XQuery，以 X2S 簡稱 XQuery 轉換成 SQL。

6.1. 資料對應表示法

與 X2S 相同，在 S2X 的情況下也需要建立資料對應表示法。S2X 需要建立值、集合的對應表示法以及 DTD 的結構表示法。

6.1.1. 權重

為了處理多元對應，我們給予每一個對應一個權重，權重高的資料會優先輸出。在 S2X 的情況下，權重的計算公式為（表格的欄位對應的葉節點個數/ 此節點之葉節點個數）。以圖 2.1 與圖 2.2 為例，PARTSUPP 表格對應到 supplier 節點和 part 節點， supplier 節點有 4 個葉節點 nkey、skey、name 和 add，其中只有 skey 對應到 PARTSUPP.SUPPKEY，因此權重為 1/4。而 part 節點有 4 個葉節點 pkey、name、type、availqty，其中 pkey 和 availqty 分別對應 PARTSUPP.PARTKEY 和 PARTSUPP.AVAILQTY，因此權重為 2/4。

至於值的權重和 X2S 的處理方法一樣，值的權重和其所在的集合權重相同，這是因為表格與節點的相似度越高，取出此表格的可能越高。以圖 2.1 與圖 2.2 為例，PARTSUPP.PARTKEY 對應到 part@pkey 的權重則與 PARTSUPP 表格對應到 part 節點的權重相同，因此權重為 2/4。

6.1.2. 值對應表示法

【定義 6.1】 S2X **VM**

- vi=（Rname, Aname），VM（vi）=[（XPath, RXPath, Weight）]。其中，Rname為表格名稱，Aname為表格的欄位名稱，XPath為與此欄位對應的DTD葉節點的路徑，RXPath為與此葉節點最接近的祖先層可重覆路徑，Weight則記錄此筆對應資料的權重，若VM（vi）有多筆對應資料時，資料會依Weight由大到小排序。

VM 會針對每一個葉節點與表格欄位之間的關係建立對應資料， VM 記錄 RXPath 的目的是為了取得表格對應的集合。由圖 2.1 與圖 2.2 所建立的部份 VM 內容如表格 6.1 所示。完整的 VM 內容請參考附錄 A。為方便區分所指的資料列，表格中新增欄位「列」，但實際上此欄位不存在 VM 中。

列	Rname	Aname	XPath	RXPath	Weight
1	SUPPLIER	NATIONKEY	/order-ship/suppliers/supplier@nkey	/order-ship/suppliers/supplier	4/4
2	SUPPLIER	SUPPKEY	/order-ship/suppliers/supplier@skey	/order-ship/suppliers/supplier	4/4
3	SUPPLIER	NAME	/order-ship/suppliers/supplier/name	/order-ship/suppliers/supplier	4/4
4	SUPPLIER	ADDRESS	/order-ship/suppliers/supplier/add	/order-ship/suppliers/supplier	4/4
5	PARTSUPP	SUPPKEY	/order-ship/suppliers/supplier@skey	/order-ship/suppliers/supplier	1/4
6	LINEITEM	SUPPKEY	/order-ship/suppliers/supplier@skey	/order-ship/suppliers/supplier	1/4
7	LINEITEM	SHIPMODE	/order-ship/suppliers/supplier/part/order/shipmode	/order-ship/suppliers/supplier/part/order	3/6
8	PART	PARTKEY	/order-ship/suppliers/supplier/part@pkey	/order-ship/suppliers/supplier/part	3/4
9	PART	NAME	/order-ship/suppliers/supplier/part/name	/order-ship/suppliers/supplier/part	3/4
10	PART	TYPE	/order-ship/suppliers/supplier/part/type	/order-ship/suppliers/supplier/part	3/4
11	CUSTOMER	CUSTKEY	/order-ship/customer@ckey	/order-ship/customer	4/5
12	CUSTEL	CUSTKEY	/order-ship/customer@ckey	/order-ship/customer	2/5
13	CUSTEL	TELEPHONE	/order-ship/customer/tel	/ order-ship/customer	2/5
14	INTRODUCE	CUSTKEY1	/order-ship/customer@ckey	/order-ship/customer	1/5

15	INTRODUCE	CUSTKEY2	/order-ship/customer@ckey	/order-ship/customer	1/5

表格 6.1 圖 2.1 與圖 2.2 的 S2X 部分 Value Mapping 內容

6.1.3. 集合對應表示法

定義 6.2 的 Collection Mapping 記錄關聯式資料庫表格與 XML 端內部節點的路徑對應關係。

【定義 6.2】：Collection Mapping（簡稱:CM）

- ci=（ Rname ），CM（ci）= [（xpath，Weight，Condition）]。其中，Rname為關聯式資料庫的表格名稱，XPath為表格對應的XML節點路徑。Weight為記錄表格與節點的相似度，相似度越高Weight的值越大。若CM(ci)回傳多筆資料，資料列會依Weight由大到小排序。Condition記錄節點對應到表格時的條件限制。

CM 會記錄每個表格的對應資訊，若對應到空節點則不建立。例如 PARTSUPP 表格對應到 supplier 節點和 part 節點，因此在表格 6.2 中的 4、5 列記錄此多元對應關係。

列	Rname	XPath	Weight	Condition
1	SUPPLIER	/order-ship/suppliers/supplier	4/4	
2	PART	/order-ship/suppliers/supplier/part	3/4	
3	ORDER	/order-ship/suppliers/supplier/part/order	4/6	
4	PARTSUPP	/order-ship/suppliers/supplier	1/4	
5	PARTSUPP	/order-ship/suppliers/supplier/part	2/4	
6	LINEITEM	/order-ship/suppliers/supplier	1/4	
7	LINEITEM	/order-ship/suppliers/supplier/part	1/4	
8	LINEITEM	/order-ship/suppliers/supplier/part/order	3/6	
9	CUSTOMER	/order-ship/customer	4/5	
10	INTRODUCE	/order-ship/customer	1/5	
11	INTRODUCE	//customer	1/5	
12	CUSTEL	/order-ship/customer	2/5	
13	CUSTEL	/ order-ship/customer/tel	2/5	
14	INTROUUCE	/order-ship/customer/introduce	1/1	

表格 6.2 圖 2.1 與圖 2.2 所建立的 S2X CM

6.1.4. 結構表示法

在S2X中，我們以PE記錄DTD中具有結構意義的內部節點關聯。

【定義6.3】：Path Expression （簡稱:PE）

根據XML綱要給與pi，PE(pi)會回傳綱要中pi代表的資料。pi=(XPath1,XPath2)，PE = [（XID, XPath1, XPath2）]。其中，XID為此結構關係的編號，不同的編號代表不同的意義。編號為X_{Fi}代表於DTD中是屬於扁平（Flat）的連結結構。編號為X_{Ni}代表於DTD中建立兩個可重覆元素的巢狀結構。XPath1 和 XPath2記錄節點路徑，當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。

從圖 2.2 中我們可看出可重覆節點 supplier 是空節點 suppliers 的子節點，因

此我們在表 6.3 以編號 X_{ND1} 記錄此巢狀結構。可重覆節點 part 是 supplier 節點之子節點，此父子關係為一巢狀結構，因此我們以編號 X_{N2} 記錄此巢狀結構。可重覆節點 order 是 supplier 節點之孫節點，此祖孫關係為一巢狀結構，因此我們以編號 X_{N3} 記錄此巢狀結構。至於 customer 節點與 order 節點以屬性 ckey 連結，同時兩者不為巢狀結構，因此以編號 X_{F5} 記錄連結。

XID	XPath1	XPath2
X_{ND1}	/order-ship/suppliers	/order-ship/suppliers/supplier
X_{N2}	/order-ship/suppliers/supplier	/order-ship/suppliers/supplier/part
X_{N3}	/order-ship/suppliers/supplier	/order-ship/suppliers/supplier/order
X_{N4}	/order-ship/suppliers/supplier/part	/order-ship/suppliers/supplier/part/order
X_{F5}	/order-ship/customer@ckey	/order-ship/suppliers/supplier/part/order@ckey
X_{F6}	/order-ship/customer@nkey	/order-ship/suppliers/supplier@nkey
X_{N7}	/order-ship/customer	/order-ship/customer/introduce

表格 6.3 圖 2.2 產生的 Path Expression

6.2. 轉換架構與轉換處理

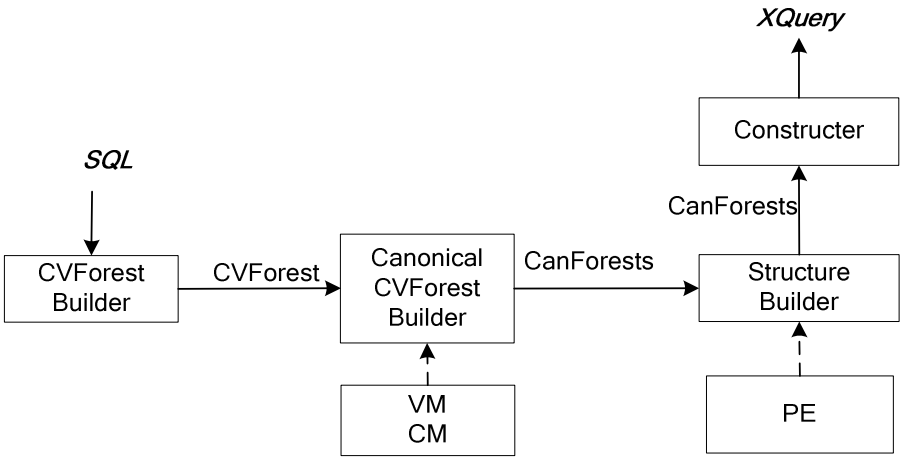


圖 6.1 S2X 的轉換架構

S2X 的轉換架構如圖 6.1 所示。CVForest Builder 根據 SQL 查詢句產生查詢樹。在建立 CVForest 時，程式會先剖析使用者輸入的查詢句，如果查詢句在圖 2.5 的文法範圍，則針對 FROM 子句的每一個表格產生集合樹，並對 SELECT 子句的每一個表格及欄位產生對應的值樹的根節點及子節點。若 SQL 包含子查詢句，則會為子查詢句建立一個集合樹根節點。在 SQL 產生查詢樹時，查詢樹的根節點為 FROM 子句中的表格名稱。若節點路徑為空，則代表連結至子查詢句。

而Canonical CVForest Builder建立CanForest的建立方法與X2S相同，同樣都是根據CVForest的資料建立CanForest。需注意的是，由於關聯式資料表格為扁平結構，扁平結構不具有空節點因此不會有刪除節點的情況發生。同樣的，因為不存在跳層路徑，在建立CanForest時不會在正規樹的根節點和葉節點中間加入節點。建立CanForest時所做的只有加入對應資料。

Structure Builder 會產生最小生成樹時，與 X2S 同樣是利用最小生成樹演算法及對應函數處理集合之間的結構連結，但 S2X 利用的是對應函數 PE 而不是對應函數 JE。

最後 Constructer 會依階層碼有順序性的收集及合併 CVForest 的資料，並輸出轉換後的查詢句。與 X2S 不同的是，X2S 會轉換出關聯式表格，因此轉換出來的表格之間不會有父子祖孫的巢狀關係，而 S2X 轉換出來的是 XML 文件節點路徑，兩個節點可能會是父子祖孫巢狀關係節，所以兩個節點若為父子祖孫巢狀關係，則需要代換變數。例如 $s in /order-ship/suppliers/supplier， $o in /order-ship/suppliers/supplier/part/order，則$o 會改寫成$o in $s/part/order。

6.3. SQL 轉換範例

本節以範例 6.1 說明 SQL 的轉換情形，範例 6.1 在 where 子句中包含子查詢句，此查詢句取出訂單金額大於一萬的顧客名稱，其中 where 子句的子查詢句取

出 ORDER 表格中金額大於一萬的 CUSTKEY，外部查詢句則限定取出 CUSTKEY 位於子查詢句回傳值的顧客名稱。輸入及輸出的查詢句如範例 6.1 所示。

【範例 6.1】

輸入的 SQL	輸出的 XQuery
Select C.NAME From CUSTOMER C , (SELECT O.CUSTKEY FROM ORDER O WHERE O.TOTALPRICE >10000）as Sub1 Where C.CUSTKEY = Sub1.CUSTKEY	For $c in /order-ship/customer Let Sub1::= for $o in /order-ship/suppliers/supplier/part/order Where $o/price>10000 Return <result>$o@ckey</result> Where Sub1@ckey=$c@ckey Return <result>$c/name</result>

BuildCVForest 演算法依外部查詢句建立集合樹 c 及值樹 c，並為子查詢句中資料建立集合樹 o 及值樹 o。建立的 CVForest 如圖 6.2 所示。

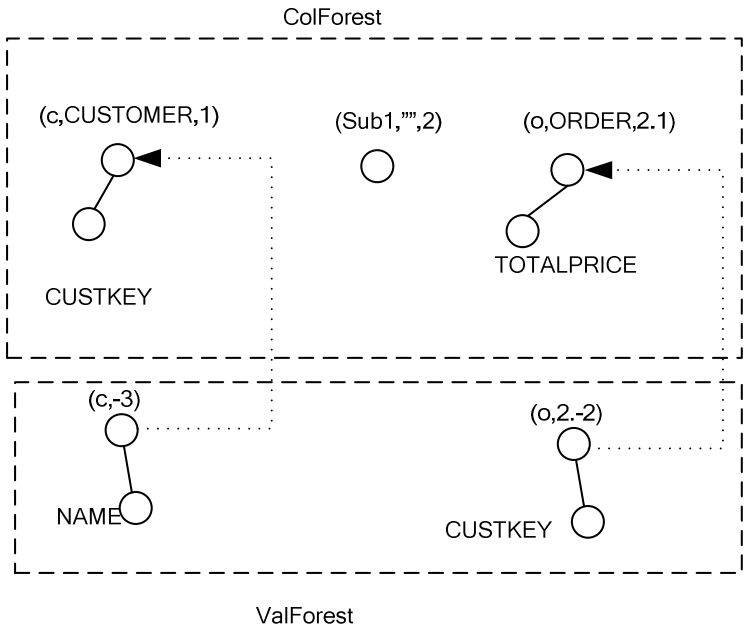


圖 6.2 範例 6.1 建立的 CVForest

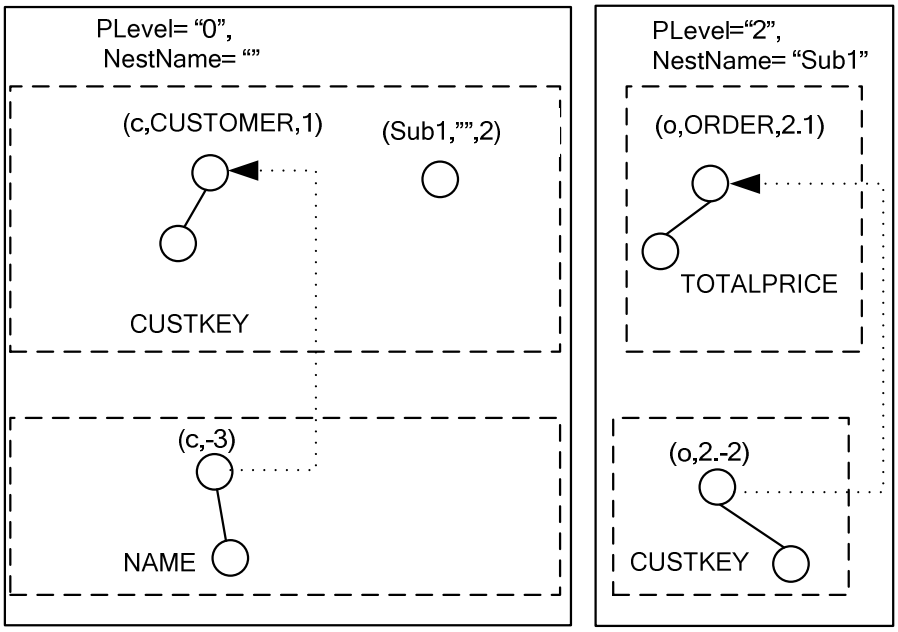


圖 6.3 範例 6.1 建立的 CanForests

TagName	Path	CVMs= {（Var,ColOrVal,UsedFlag）}
CUSTOMER	CUSTOMER	C_1,customer,TRUE
CUSTKEY	CUSTOMER.CUSTKEY	C_1,@ckey,TRUE
Sub1	null	Sub1,null,TRUE
ORDER	ORDER	O_1,order, TRUE
TOTALPRICE	ORDER.TOTALPRICE	O_1,/price, TRUE

表格 6.4 範例 6.1CanForests 中正規集合樹的 CVMs 欄位

TagName	Path	CVMs= {（Var,ColOrVal,UsedFlag）}
CUSTOMER	CUSTOMER	C_1,customer,TRUE
NAME	CUSTOMER.NAME	C_1,/name,TRUE
ORDER	ORDER	O_1,order, TRUE
CUSTKEY	ORDER.CUSTKEY	O_1,@ckey,TRUE

表格 6.5 範例 6.1CanForests 中正規值樹的 CVMs 欄位

BuildCanColForest 演算法會保持樹的結構，並依階層碼分層區分值樹和集合樹，建立的 CanForests 如圖 6.3 所示，CanForests 中正規集合樹的 CVMs 欄位如表格 6.4 所示，CanForests 中正規值樹的 CVMs 欄位如表格 6.5 所示。在表格 6.4 及表格 6.5 中，為了簡化起見，CVMs 中的 ColOrVal 欄位僅記錄 XPath 路徑中最後一個節點，若為葉節點則記錄最後一組軸及節點，此欄位實際上記錄的是完整路徑。

集合樹 “c”會取得對應路徑/order-ship/customer，並將對應資料加入 CVMs 欄位，其他集合樹也用同樣的方法處理。在處理集合樹的 TOTALPRICE 葉節點時，首先會取得對應路徑 /order-ship/suppliers/supplier/part/order/price，而 OutQuery->Collection 會加入父節點 ORDER 的對應路徑 /order-ship/suppliers/supplier/part/order，由於葉節點 TOTALPRICE 是由 where 子句的條件限制式所產生，因此 OutQuery->ValueConditions 會加入 o_1/price>10000。

值樹 “c” 的葉節點 name 取得對應葉節點/order-ship/customer/name，OutQuery->Collections 會加入父節點 CUSTOMER 的對應節點/order-ship/customer，OutQuery->Values 加入 “c_1/name”，其他的葉節點也用同樣的方法處理。BuildCanColForest 執行完成後，更新後的 OutQuery 欄位如下所示。

PLevel	Values	Collections	ValueConditons
0	c_1,/name	Sub1,”” c_1, /order-ship/customer	
2	o_1@ckey	o_1, /order-ship/suppliers/supplier/part/order,	o_1/price>1000

StructureBuilder 模組會建立集合之間的連結關係，MST_Prim 演算法在處理變數 customer 節點時，由於 Sub1 節點代表子查詢句，因此 L10~L12 的 JoinSubQuery 會取出子查詢句中與 customer 節點連結的 order 節點，並取出連結

資料 o_1@ckey=c_1@ckey，此連結條件會連結子查詢句和外部查詢句。

為了連結外部查詢句與子查詢句，子查詢句必需輸出連結欄位，因此 OutQuery.Values 會加入 o_1@ckey=c_1@ckey 的 o_1@ckey。由於 OutQuery->Values 已經有 o_1@ckey，所以不會重覆加入。OutQuery.JoinConditions 則加入 Sub1@ckey=c_1@ckey，這是因為 o_1@ckey 會改寫成 Sub1@ckey。

處理後的 OutQuery 欄位如下所示，轉換後得到的 XQuery 如範例 5.1 所示。

PLevel	Values	Collections	ValueConditons	JoinConditons
0	c_1,/name	Sub,"" c_1, /order-ship/customer		Sub@ckey =c_1@ckey
2	o_1@ckey	o_1, /order-ship/suppliers/supplier/part/order,	o_1/price>1000	

第 7 章　正確性分析與轉換效率評估

7.1. 轉換系統正確性分析

本節說明如何轉換出符合定義 2.4 的等價查詢句，我們將針對綱要的多元對應情況及子查詢句的狀況進行討論。

7.1.1. 多元對應的轉換

以下說明多元對應的處理正確性，正確處理多元對應後可得到定義 2.1 及定義 2.2 的外部查詢句等價定義。

多元對應分成值、集合的多元對應。值的多元對會輸出權重最高的對應值。給與葉節點值 vi， VM（vi）有 m 個對應值，VM（vi）[j]為 VM（vi）中第 j 個對應值，其中 j>0 且 j<m，TopVM（vi）為權重最高的 VM（vi），TargetVM（vi）為輸出的對應值。由於最後會輸出權重最高的對應值，因此 TargetVM（vi）＝TopVM（vi），值的多元對應因此解決。由於 VM（vi）已由權重由大到小排序，因此在 SetVM 演算法中會取出 VM（vi）的第一筆資料，所以 TopVM（vi）為 VM（vi）[0]，TargetVM（vi）＝TopVM（vi）。

針對集合的多元對應，集合有 n 個對應集合，最後最多會輸出 n 個對應集合。若集合節點為 ci，CM（ci）有 m 個對應集合，TargetCM（ci）為輸出的對應集合，CM（ci）[j]為 CM（ci）中第 j 個對應值，其中 j>0 且 j<m。由於 TargetVM（vi）所位於的集合需要輸出，所以 TargetVM（vi）可以決定 TargetCM（ci），假設 CM（ci）[j]為定義 TargetVM（vi）的集合，則 TargetCM（ci）+= CM（ci）[j]。

在 SetVM 演算法中，輸入參數為 TargetVM（vi），演算法的 L06~L15 會在 TargetCM（ci）會加入 CM（ci）[j]。而當 ci 若沒有葉節點 vi 時則無法由 TargetVM（vi）決定 TargetCM（ci），這種情況會由 StructureBuilder 模組決定 TargetCM（ci）。

針對集合節點之間的連結關係，若集合可以組成樹結構則為合理的連結關係，所以 StructureBuilder 模組的 MST_Prim 演算法會產生最小生成樹，最小生成樹的邊即為集合之間合理的連結關係。

當每個 TargetCM（ci）僅有一筆資料時，由於轉換前和轉換後的集合個數相同，因此可得到定義 2.1 的外部查詢句等價定義。而當一個以上的 TargetCM（ci）包含兩筆以上資料時，由於轉換前和轉換後的集合個數不同，所以會得到定義 2.2 的外部查詢句等價定義。

7.1.2. 子查詢句的轉換

而當查詢句包含子查詢句時，子查詢句的轉換問題分成如何處理外部查詢句與子查詢句，以及如何連結轉換後的外部查詢句與子查詢句。

由於外部查詢句可影響子查詢句，但子查詢句不會影響外部查詢句，因此根據階層碼分層處理，先處理子查詢句再處理外部查詢句可正確的轉換子查詢句。對應到轉換系統，BuildCanColForest 和 BuildCanValForest 會產生 CanForests，並將不同層的樹歸類於不同的 CanForest，再根據階層碼由內到外分層處理。

接著探討如何連結轉換後的外部查詢句與子查詢句。當 structure builder 模組處理外部查詢句時，MST_Prim 演算法會輸出連結外部查詢句和子查詢句的連結條件句，且子查詢句會輸出連結條件句中和外部查詢句相關的表格欄位，以上對應到 MST_Prim 演算法中 L10~L12 的 JoinSubQuery 函式。輸出連結外部查詢句和子查詢句的連結條件句會符合定義 2.3-1，而子查詢句會輸出連結條件句中和

外部查詢句相關的表格欄位則是符合定義 2.3-2。

綜合以上，我們的系統會產生符合定義 2.4 的等價查詢句。

7.2. 轉換效率評估

轉換所需的時間主要是由(1)建立 ColValForest、(2)建立 CanColValForests、(3) 建立集合之間的連結 (4) Constructer 模組四大部分所組成。我們針對不同的控制因子設計適當的 Schema 來實驗，並探討其對轉換的效率影響。

在本節中，我們將以不同結構的查詢句評估查詢句轉換系統的執行效率。藉由調整子查詢句深度、變數個數、路徑包含跳層、變數間有上下層關係，來測試以上因素對於轉換效率之影響。針對多元對應，則評估當值與集合為多元對應時，對轉換效率的影響。本章實驗圖表中記錄查詢句轉換一次的時間，時間的單位為ms。

7.2.1. 子查詢句的深度對轉換效率的影響

參照附錄 B 的 Schema 設計與查詢句內容。我們設計 5 個表格 A、B、C、D、E，其對應到 DTD 中的元素分別為 a、b、c、d、e；前後兩表格間皆有關連， A join B、B join C、C join D、D join E。此結構關連對應於 Nested 與 Flat 結構的 DTD 時，分別為元素 a 與 b、b 與 c、c 與 d 之間的結構對應。

實驗中控制子查詢句的深度測試 S2X 以及 X2S 轉換時間。在查詢句中，每個子查詢句各輸出兩個表格欄位，所有的子查詢句都會回傳最深子查詢句的鍵值。

此實驗目的在比較查詢句中深度不同的子查詢句對轉換時間之影響。表格

7.1為S2X的實驗執行時間，表7.2為Nest結構X2S的實驗執行時間，表7.3 則Flat結構的X2S實驗執行時間，曲線結果分別如圖7.1、圖7.2與圖7.3所示。

由實驗結果發現，在四個模組中，建立CVForest花的時間最長，其次是CanForest。當子查詢句的深度越深，則建立CVForest和建立CanForest花的時間會曲線成長，這是因為子查詢句的深度越深，相對所使用的集合個數就越多，因此花費的時間就越多。

比較S2X與Nested結構的X2S發現，S2X所花的時間會比較多，這是由於關聯式表格為扁平結構，表格間是透過鍵值連結，因此每一組連結都會在集合樹中產生對應的葉節點。相對的Nested X2S透過巢狀結構建立連結，因此在建立CVForest和建立CanForest時，S2X花的時間會比X2S多。至於建立結構所花的時間，由於X2S是將巢狀結構轉換成扁平結構，因此X2S花的時間比S2X多。

而比較S2X與Flat結構的X2S發現，兩者所花的時間相似，這是因為兩者皆為扁平結構，表格間透過鍵值連結，而元素間透過葉節點連結，因此每一組連結都會在集合樹中產生對應的葉節點。

子查詢句深度	Execution Time（ms）			
	CVForest Builder	CanForest Builder	Structure Builder	Constructer
1	0.299	0.077	0.015	0.015
2	0.452	0.157	0.015	0.015
3	0.609	0.330	0.031	0.030
4	0.907	0.405	0.046	0.031

表格 7.1　子查詢句的深度對 S2X 轉換效率的影響實驗結果

子查詢句深度	Execution Time（ms）			
	CVForest Builder	CanForest Builder	Structure Builder	Constructer
1	0.187	0.111	0.047	0.015
2	0.249	0.236	0.048	0.047
3	0.377	0.304	0.079	0.079
4	0.564	0.358	0.109	0.109

表格 7.2 子查詢句的深度對 Nested 結構 X2S 轉換效率的影響實驗結果

子查詢句深度	Execution Time（ms）			
	CVForest Builder	CanForest Builder	Structure Builder	Constructer
1	0.325	0.071	0.015	0.015
2	0.432	0.167	0.015	0.015
3	0.599	0.28	0.031	0.03
4	0.927	0.425	0.046	0.031

表格 7.3 子查詢句的深度對 Flat 結構 X2S 轉換效率的影響實驗結果

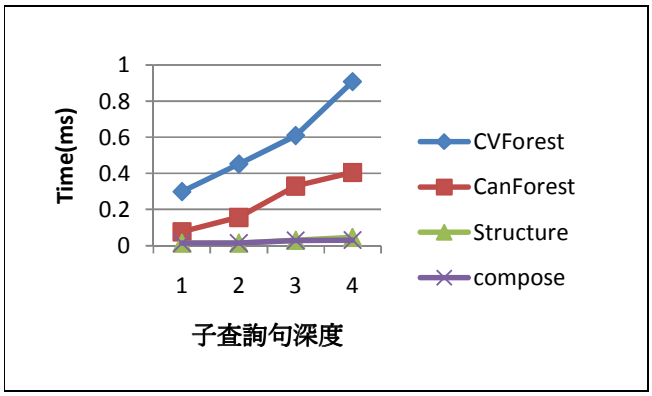


圖 7.1 子查詢句的深度對 S2X 轉換效率的影響

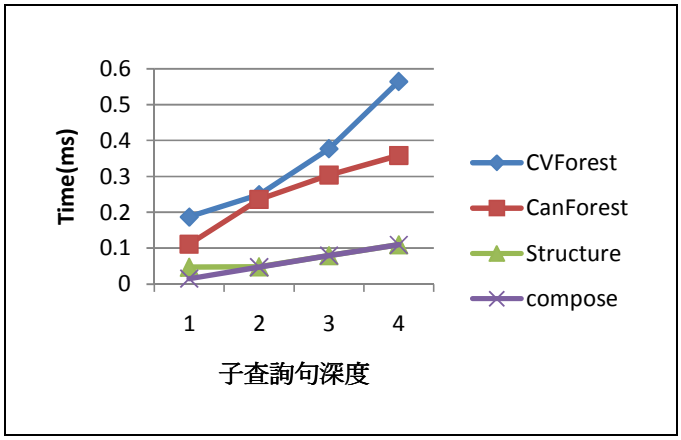


圖 7.2 子查詢句的深度對 Nested 結構 X2S 轉換效率的影響

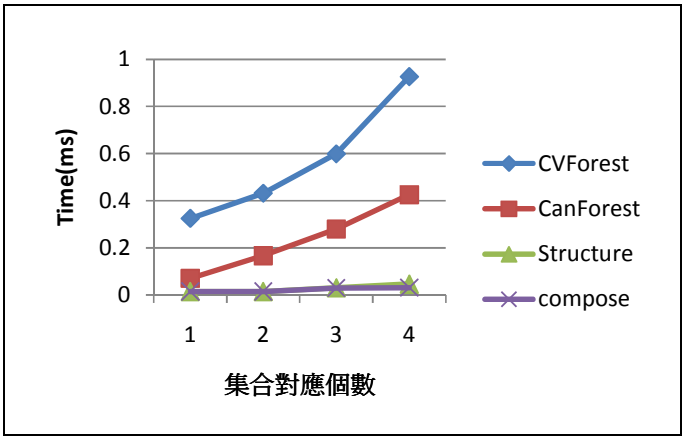


圖 7.3 子查詢句的深度對 Flat 結構 X2S 轉換效率的影響

7.2.2. 直接路徑與跳層路徑的轉換時間

參照附錄B的Schema設計與附錄C的查詢句內容，此節利用集合個數、路徑包含跳層、變數間有上下層關係，來測試以上因素對於轉換效率之影響。在本節中，不包含跳層的路徑稱為「直接路徑」，例如a/b/c，包含跳層的路徑稱為「跳層路徑」，例如//b。而直接路徑中若與其他變數有上下層的巢狀關係，且下層變數的轉換結果受到上層變數轉換結果的影響，則稱為「巢狀直接路徑」，例如$c in a/b/c，$e in $c/d/e，我們稱$c/d/e為巢狀直接路徑。而跳層路徑若與其他變數有上下層的巢狀關係，且下層變數的轉換結果受到上層變數轉換結果的影響，則稱為「巢狀跳層路徑」，例如$c in a/b/c，$e in $c//e，我們稱$c//e為巢狀跳層路徑。

表格7.4為直接路徑的實驗執行時間，表格7.5為跳層路徑的實驗執行時間，表格7.6為巢狀直接路徑的實驗執行時間，最後表格7.7為巢狀跳層路徑的實驗執行時間。而曲線結果分別如圖7.4至與7.5所示。其中表7.4與表7.5所有變數之間無上下層關係，而表格7.6與表格7.7中的變數路徑兩兩成對且有上下層關係。

集合個數	Execution Time（ms）			
	CVForest Builder	CanForest Builder	Structure Builder	Constructer
1	0.065	0.060	0.015	0.016
2	0.094	0.091	0.016	0.020
3	0.157	0.110	0.016	0.030
4	0.188	0.124	0.016	0.040
5	0.208	0.144	0.015	0.045

表格 7.4 直接路徑對轉換時間的影響

集合個數	Execution Time（ms）			
	CVForest Builder	CanForest Builder	Structure Builder	Constructer
1	0.065	0.240	0.015	0.016
2	0.091	0.254	0.016	0.020
3	0.162	0.280	0.032	0.030
4	0.176	0.291	0.032	0.040
5	0.199	0.315	0.032	0.045

表格 7.5 跳層路徑對轉換時間的影響

集合個數	Execution Time（ms）			
	CVForest Builder	CanForest Builder	Structure Builder	Constructer
1	0.065	0.059	0.015	0.016
2	0.094	0.089	0.016	0.020
3	0.157	0.110	0.016	0.030
4	0.188	0.135	0.016	0.040
5	0.208	0.160	0.015	0.045

表格 7.6 巢狀直接路徑對轉換時間的影響

集合個數	Execution Time（ms）			
	CVForest Builder	CanForest Builder	Structure Builder	Constructer
1	0.065	0.250	0.016	0.016
2	0.100	0.269	0.016	0.020
3	0.162	0.291	0.032	0.030
4	0.178	0.302	0.032	0.040
5	0.197	0.326	0.032	0.045

表格 7.7 巢狀跳層路徑對轉換時間的影響

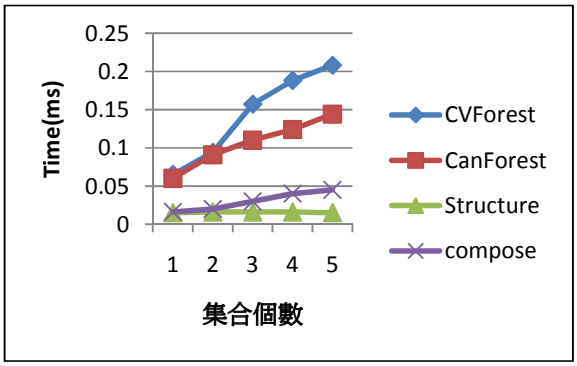


圖 7.4 直接路徑對轉換時間的影響

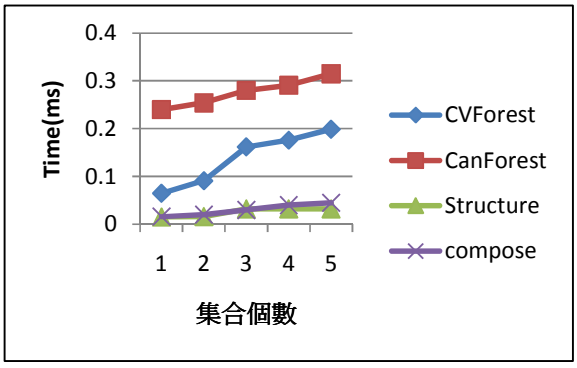


圖 7.5 跳層路徑對轉換時間的影響

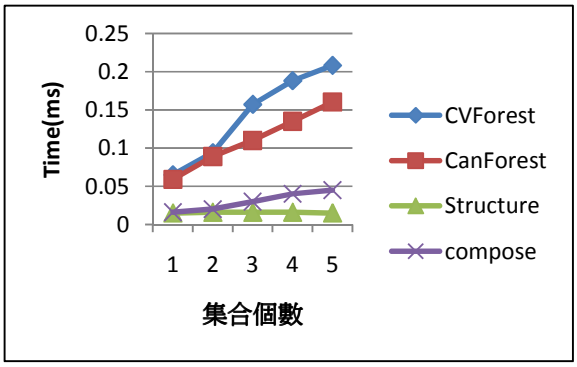


圖 7.6 巢狀直接路徑對轉換時間的影響

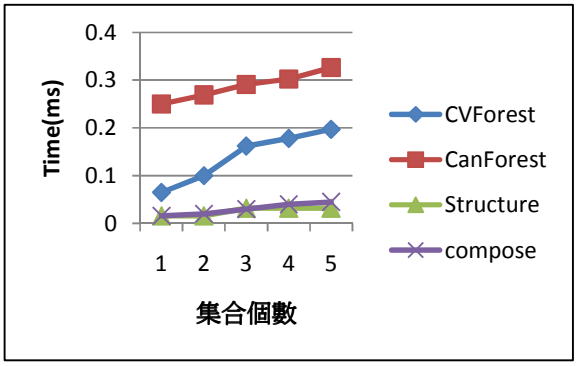


圖 7.7 巢狀跳層路徑對轉換時間的影響

比較圖7.4與圖7.5可以發現，轉換時間會依集合的個數曲線成長。當路徑為直接路徑時，建立CVForest比建立CanForest多，但當路徑為單一跳層路徑時，建立CanForest的時間大於建立CVForest的時間，這是因為CM函數以湊雜表記錄，集合路徑為湊雜表的鍵值。因此當路徑為直接路徑時，我們可以依湊雜表快速的取出對應資料，但當路徑為單一跳層路徑時需逐筆比對湊雜表中的資料。圖7.6、圖7.7則與圖7.4、圖7.5呈現相同的趨勢，轉換時間同樣會依集合的個數線性成長，巢狀跳層路徑建立CanForest的時間大於巢狀直接路徑建立CVForest的時間。

綜合比較四個實驗圖表之後則發現，巢狀直接路徑的轉換時間略大於直接路

徑，巢狀跳層路徑轉換時間略大於跳層路徑。這是因為巢狀路徑需要展開路徑取得相關變數的路徑，因此轉換時間會略大。但兩者的時間幾乎是相同的。

7.2.3. 集合的對應個數對轉換時間的影響

參考附錄 D 中，DTD 中的 A、B、C、D，其對應於關聯式表格中的表格分別為 1 對 1、1 對 2、1 對 3 與 1 對 4 的對應關係；查詢句中固定回傳值的個數為 4 個，控制 DTD 中可重覆元素對關聯式表格一對多對應的個數，比較其對轉換效率的影響，實驗執行時間如表格 7.8 所示，曲線結果如圖 7.8 所示。

從實驗結果發現，轉換時間依輸出的集合個數線性成長。這是因為輸出的集合個數越多，建立集合之間的連結花的時間就越多。但花在集合連結上的時間並不明顯。

集合對應個數	Execution Time（ms）			
	CVForest Builder	CanForest Builder	Structure Builder	Constructer
1	0.108	0.108	0.016	0.015
2	0.108	0.108	0.020	0.015
3	0.108	0.108	0.024	0.015
4	0.108	0.108	0.028	0.015
5	0.108	0.108	0.030	0.015

表格 7.8 集合對應個數與轉換時間的影響

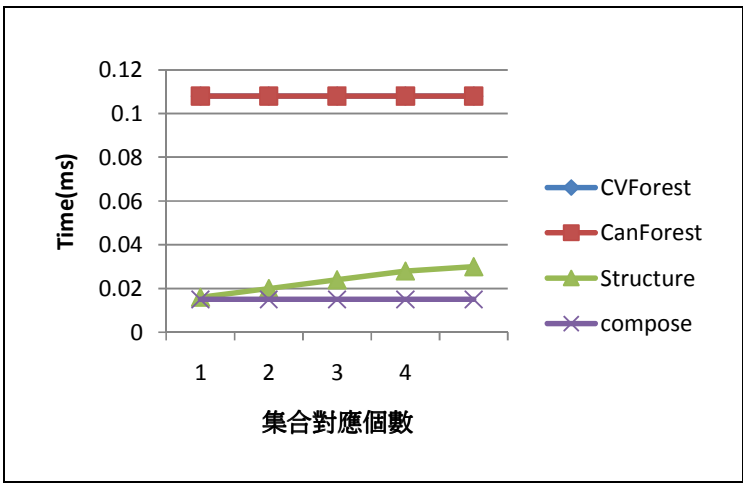


圖 7.8 集合對應個數與轉換時間的影響

7.2.4. 值的對應個數對轉換時間的影響

本節探討值的多元對應對轉換效能的影響。控制值一對多的對應個數，查詢句中僅選擇一個表格欄位。參考附錄 E，附錄 E 使用與附錄 D 相同的關連式表格以及 DTD，DTD 中的 akey、bkey、ckey、dkey，其對應於關聯式表格中的鍵值 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 DTD 的鍵值，測試屬性一對多對轉換效率的影響。DTD 中屬性的權重會與其父節點的權重相同。

表格 7.9 為實驗執行時間結果，圖 7.9 為曲線結果。由圖 7.9 可發現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。這是因為系統在載入依 VM 函定產生的雜湊表時，已經依權重將取得的資料由大到小排序，所以系統在處理值的多元對應時，皆是取用第一筆資料。

值對應個數	Execution Time（ms）			
	CVForest Builder	CanForest Builder	Structure Builder	Constructer
1	0.065	0.070	0.015	0.016
2	0.065	0.070	0.015	0.016
3	0.065	0.070	0.015	0.016
4	0.065	0.070	0.015	0.016
5	0.065	0.070	0.015	0.016

表格 7.9 值對應個數對轉換時間的影響

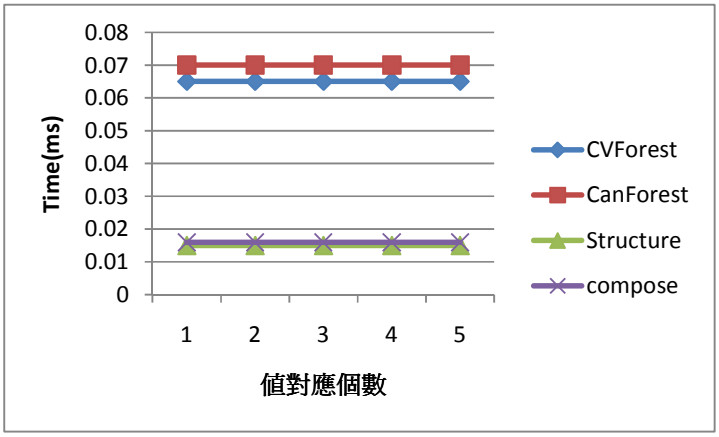


圖 7.9 值對應個數對轉換時間的影響

7.2.5. 轉換效率評估總結

綜合以上小節說明，可發現到轉換時間會依子查詢句的深度、變數個數呈現曲線成長。而跳層路徑時的轉換時間遠大於直接路徑，路徑為直接路徑或巢狀路徑對整體的轉換效率無特別明顯的影響。針對集合的多元對應，輸出的集合個數越多，需要的轉換時間就越多，但轉換時間並無太大差異。針對多元對應，值對應個數不影響轉換效能，這是因為值的多元對應一律取出第一筆資料。

第 8 章　結論與未來研究

本論文提出了一個介於關聯式資料庫及 XML 資料之間的查詢句轉換系統，使用者可方便的依不同類型需求來下達查詢句，使用者可以下達具有巢狀結構的複雜查詢句，轉換後的查詢句可以進一步的進行資料的交換與使用。論文中使用對應函數來表示關聯式綱要與 XML 綱要的值、集合與結構的差異對應資訊，系統利用樹結構處理查詢句，轉換系統會剖析輸入的查詢句，針對查詢句中的集合及值建立樹，再利用對應資料進一步建立正規樹，最後用最小生成樹產生集合的連結。實驗證明系統可以有效率的轉換查詢句，由實驗中發現，路徑中包含跳層對轉換效能的影響最大。

在未來研究方面規劃了數個方向。第一是處理更複雜的遞迴路徑，目前系統在處理遞迴節點時，僅處理查詢句中具有兩個變數，一個變數為可重覆節點，另一個變數為遞迴節點，且兩者在 DTD 為同一個節點的情況，如範例 4.3 的變數 c1 和變數 c2。第二是處理 XQuery Return 子句中的子查詢句，在處理的時候要為子查詢句建立集合樹並給予子查詢句別名，而子查詢句的 NestJoin 也要處理。第三是路徑包含跳層時會逐一比對 Hash Table 的資料，未來希望改善作法加快效率。第四是改善結構的產生方法，目前在建立結構時，除了會放入有值輸出的集合之外，還會放入可能會用到的額外集合，額外集合是用來連結有值輸出的集合。目前的作法是為這些集合產生最小生成樹，但使用 Steiner Tree 可以先為有值輸出的集合建立樹，當樹無法建成時再加入額外的集合以產生樹，因此改用 Steiner Tree 會更符合需求。最後則是深入探討 SQL 轉換 XQuery 查詢句，完善 SQL 轉換 XQuery 系統並為其提出適當的等價定義，等價定義中需考慮子查詢句轉換時會產生的問題。

參考文獻

[ABJ+07] Yuan An, Alex Borgida, Ren´ee J. Miller, John Mylopoulos, "A Semantic Approach to Discovering Schema Mapping Expressions", In Proc. ICDE 2007, Pages： 206 – 215, April 15 – 20, 2007

[ALM09] Shun'ichi Amano, Leonid Libkin, Filip Murlak, "XML Schema Mappings", In Proc. PODS'09, June29–July2, 2009, Providence, RhodeIsland, USA.

[ABM+07] Andrei Arion, V´eronique Benzaken, Ioana Manolescu, Yannis Papakonstantinou, "Structured Materialized Views for XML Queries", In Proc. VLDB, September 23-28, 2007, Vienna, Austria.

[ATV08] Bogdan Alexe, Wang-ChiewTan, Yannis Velegrakis, "STBenchmark: Towards a Benchmark for Mapping Systems", In Proc. VLDB 2008.

[BGW+07] Byron Choi, Gao Cong, Wenfei Fan, Stratis D. Viglas, "Updating Recursive XML Views of Relations" In Proc. ICDE 2007, April 17-20, 2007. Istanbul, Turkey.

[CG09] Rada Chirkova, Michael R. Genesereth, "Equivalence of SQL Queries In Presence of Embedded Dependencies", In Proc. PODS '09, June, 2009.

[CLIO] http://www.cs.toronto.edu/db/clio/

[D09] David DeHaan, "Equivalence of Nested Queries with Mixed Semantics", In Proc. PODS'09, June29–July2, 2009, Providence, RhodeIsland, USA.

[DPX04] Alin Deutsch, Yannis Papakonstantinou, Yu Xu, "Minimization and Group-By Detection for Nested XQueries", In Proc. ICDE 2004, March 30 - April 2, 2004, Boston, USA.

[FCB07] Wenfei Fan, Gao Cong, Philip Bohannon, "Querying XML with Update Syntax", In Proc. SIGMOD 2007, Pages: 293-304, June 12 - 14, Beijing, China.

[FYL+05] Wenfei Fan, Jeffrey Xu Yu, Hongjun Lu, Jianhua Lu, Rajeev Rastogi,

"Query Translation from XPath to SQL in the Presence of Recursive DTDs", In Proc. 31st VLDB Conference,Trondheim, Norway, 2005.

[FYL+09] Wenfei Fan, Jeffrey Xu Yu, Jianzhong Li, Bolin Ding, Lu Qin, "Query translation from XPath to SQL in the presence of recursive DTDs", The VLDB Journal, Vol.18, Page 857–883, August 2009

[K04] Rajasekar Krishnamurthy, "XML-to-SQL Query Translation" PhD Thesis, UNIVERSITY OF WISCONSIN-MADISON, 2004

[KOD09] Patrick Kling, M. Tamer ¨Ozsu, Khuzaima Daudjee, "Optimizing distributed XML queries through localization and pruning", Technical Report CS-2009-13, March 2009

[TPCH] http://www.tpc.org/tpch/

[TS08] Taro L. Saito, Shinichi Morishita, "Relational-Style XML Query", In Proc. SIGMOD, June 9–12, 2008, Vancouver, BC, Canada.

[MPW08] Mauricio A. Hern´andez, Paolo Papotti, Wang-Chiew Tan
"Data Exchange with Data-Metadata Translations" In Proc.VLDB, August 24-30, 2008, Auckland, New Zealand

[W3C] http://www.w3. org/TR/2005/WD-xquery-xpath-parsing-20050404/

[WS] http://en.wikipedia.org/wiki/Minimum_spanning_tree

[WP] http://en.wikipedia.org/wiki/Prim%27s_algorithm

[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。

[劉 06] 劉邦鑑, "支援 SQL 與 XQuery Query 之雙向查詢句轉換系統", 碩士論文，國立台灣海洋大學資訊工程學系, 2006。

附錄

附錄 A 圖 2.1 TPC-H RDB 與圖 2.2 Order-Ship DTD 之 VM 內容

圖 2.1 與圖 2.2 的 X2S Value Mapping 內容如下所示。

XPath	Rname	Aname	RXPath	Weight
/order-ship/suppliers/supplier@nkey	SUPPLIER	NATIONKEY	/order-ship/suppliers/supplier	4/4
/order-ship/suppliers/supplier@skey	SUPPLIER	SUPPKEY	/order-ship/suppliers/supplier	4/4
/order-ship/suppliers/supplier@skey	PARTSUPP	SUPPKEY	/order-ship/suppliers/supplier	1/3
/order-ship/suppliers/supplier@skey	LINEITEM	SUPPKEY	/order-ship/suppliers/supplier	1/5
/order-ship/suppliers/supplier/name	SUPPLIER	NAME	/order-ship/suppliers/supplier	4/4
/order-ship/suppliers/supplier/add	SUPPLIER	ADDRESS	/order-ship/suppliers/supplier	4/4
/order-ship/suppliers/supplier/part@pkey	PART	PARTKEY	/order-ship/suppliers/supplier/part	3/3
/order-ship/suppliers/supplier/part@pkey	PARTSUPP	PARTKEY	/order-ship/suppliers/supplier/part	2/3
/order-ship/suppliers/supplier/part@pkey	LINEITEM	PARTKEY	/order-ship/suppliers/supplier/part	1/5
/order-ship/suppliers/supplier/part/name	PART	NAME	/order-ship/suppliers/supplier/part	3/3
/order-ship/suppliers/supplier/part/type	PART	TYPE	/order-ship/suppliers/supplier/part	3/3
/order-ship/suppliers/supplier/part/availqty	PARTSUPP	AVAILQTY	/order-ship/suppliers/supplier/part	2/3
/order-ship/suppliers/supplier/part/order@ckey	ORDER	CUSTKEY	/order-ship/suppliers/supplier/part/order	4/4
/order-ship/suppliers/supplier/part/order@okey	ORDER	ORDERKEY	/order-ship/suppliers/supplier/part/order	4/4
/order-ship/suppliers/supplier/part/order@okey	LINEITEM	ORDERKEY	/order-ship/suppliers/supplier/part/order	3/5

/order-ship/suppliers/supplier/part/order@linenum	LINEITEM	LINENUMBER	/order-ship/suppliers/supplier/part/order	3/5
/order-ship/suppliers/supplier/part/order/status	ORDER	ORDERSTATUS	/order-ship/suppliers/supplier/part/order	4/4
/order-ship/suppliers/supplier/part/order/price	ORDER	TOTALPRICE	/order-ship/suppliers/supplier/part/order	4/4
/order-ship/suppliers/supplier/part/order/shipmode	LINEITEM	SHIPMODE	/order-ship/suppliers/supplier/part/order	3/5
/order-ship/customer/name	CUSTOMER	NAME	/order-ship/customer	4/5
/ order-ship/customer/tel	CUSTEL	TELEPHONE	/order-ship/customer	2/2
/order-ship/customer@ckey	CUSTOMER	CUSTKEY	/order-ship/customer	4/5
/order-ship/customer@ckey	CUSTEL	CUSTKEY	/order-ship/customer	2/2
/order-ship/customer@ckey	INTRODUCE	CUSTKEY1	/order-ship/customer	2/3
/order-ship/customer@ckey	INTRODUCE	CUSTKEY2	/order-ship/customer	2/3
/order-ship/customer@nkey	CUSTOMER	NATIONKEY	/order-ship/customer	4/5
/ order-ship/customer/comment	CUSTOMER	COMMENT	/order-ship/customer	4/5
/ order-ship/customer/introduce/date	INTRODUCE	DATE	/order-ship/customer/ introduce	1/3

2.1 與圖 2.2 的 S2X Value Mapping 內容如下所示。

Rname	Aname	XPath	RXPath	Weight
SUPPLIER	NATIONKEY	/order-ship/suppliers/supplier@nkey	/order-ship/suppliers/supplier	4/4
SUPPLIER	SUPPKEY	/order-ship/suppliers/supplier@skey	/order-ship/suppliers/supplier	4/4
PARTSUPP	SUPPKEY	/order-ship/suppliers/supplier@skey	/order-ship/suppliers/supplier	1/4
PARTSUPP	PARTKEY	/order-ship/suppliers/supplier/part@pkey	/order-ship/suppliers/supplier/part	2/4
PARTSUPP	AVAILQTY	/order-ship/suppliers/supplier/part/availqty	/order-ship/suppliers/supplier/part	2/4
LINEITEM	SUPPKEY	/order-ship/suppliers/supplier@skey	/order-ship/suppliers/supplier	1/4
SUPPLIER	NAME	/order-ship/suppliers/supplier/name	/order-ship/suppliers/supplier	4/4
SUPPLIER	ADDRESS	/order-ship/suppliers/supplier/add	/order-ship/suppliers/supplier	4/4
PART	PARTKEY	/order-ship/suppliers/supplier/part@pkey	/order-ship/suppliers/supplier/part	3/4
LINEITEM	PARTKEY	/order-ship/suppliers/supplier/part@pkey	/order-ship/suppliers/supplier/part	1/4
PART	NAME	/order-ship/suppliers/supplier/part/name	/order-ship/suppliers/supplier/part	3/4
PART	TYPE	/order-ship/suppliers/supplier/part/type	/order-ship/suppliers/supplier/part	3/4
ORDER	CUSTKEY	/order-ship/suppliers/supplier/part/order	/order-ship/suppliers/supplier/part	4/6

		@ckey	/order	
ORDER	ORDERKEY	/order-ship/suppliers/supplier/part/order @okey	/order-ship/suppliers/supplier/part /order	4/6
LINEITEM	ORDERKEY	/order-ship/suppliers/supplier/part/order @okey	/order-ship/suppliers/supplier/part /order	3/6
LINEITEM	LINENUMBER	/order-ship/suppliers/supplier/part/order @linenum	/order-ship/suppliers/supplier/part /order	3/6
ORDER	ORDERSTATUS	/order-ship/suppliers/supplier/part/order/ status	/order-ship/suppliers/supplier/part /order	4/6
ORDER	TOTALPRICE	/order-ship/suppliers/supplier/part/order/ price	/order-ship/suppliers/supplier/part /order	4/6
LINEITEM	SHIPMODE	/order-ship/suppliers/supplier/part/order/ shipmode	/order-ship/suppliers/supplier/part /order	3/6
CUSTOMER	NAME	/order-ship/customer/name	/order-ship/customer	4/5
CUSTEL	TELEPHONE	/ order-ship/customer/tel	/order-ship/customer	2/5
CUSTOMER	CUSTKEY	/order-ship/customer@ckey	/order-ship/customer	4/5
CUSTEL	CUSTKEY	/order-ship/customer@ckey	/order-ship/customer	2/5
INTRODUCE	CUSTKEY1	/order-ship/customer@ckey	/order-ship/customer	2/5
INTRODUCE	CUSTKEY2	/order-ship/customer@ckey	/order-ship/customer	2/5
CUSTOMER	NATIONKEY	/order-ship/customer@nkey	/order-ship/customer	4/5
CUSTOMER	COMMENT	/ order-ship/customer/comment	/order-ship/customer	4/5
INTRODUCE	DATE	/ order-ship/customer/introduce/date	/order-ship/customer/ introduce	1/1

附錄 B 子查詢句的深度對轉換效率的影響

說明: 我們設計 5 個表格 A、B、C、D、E，其對應到 DTD 中的元素分別為 a、b、c、d、e；前後兩表格間皆有關連， A join B、B join C、C join D、D join E。此結構關連對應於 Nested 結構的 DTD 時，分別為元素 a 與 b、b 與 c、c 與 d 之間的結構對應。實驗中控制子查詢句的深度測試 S2X 以及 X2S 轉換時間。在查詢句中，每個子查詢句各輸出兩個表格欄位，所有的子查詢句都會回傳最深子查詢句的鍵值。

A（AID , A1）
B（ BID , AID , B1）
C（ CID , BID , C1）
D（ DID , DID , D1）
E（ EID , AID , E1）

圖 B-1： 關連式表格架構

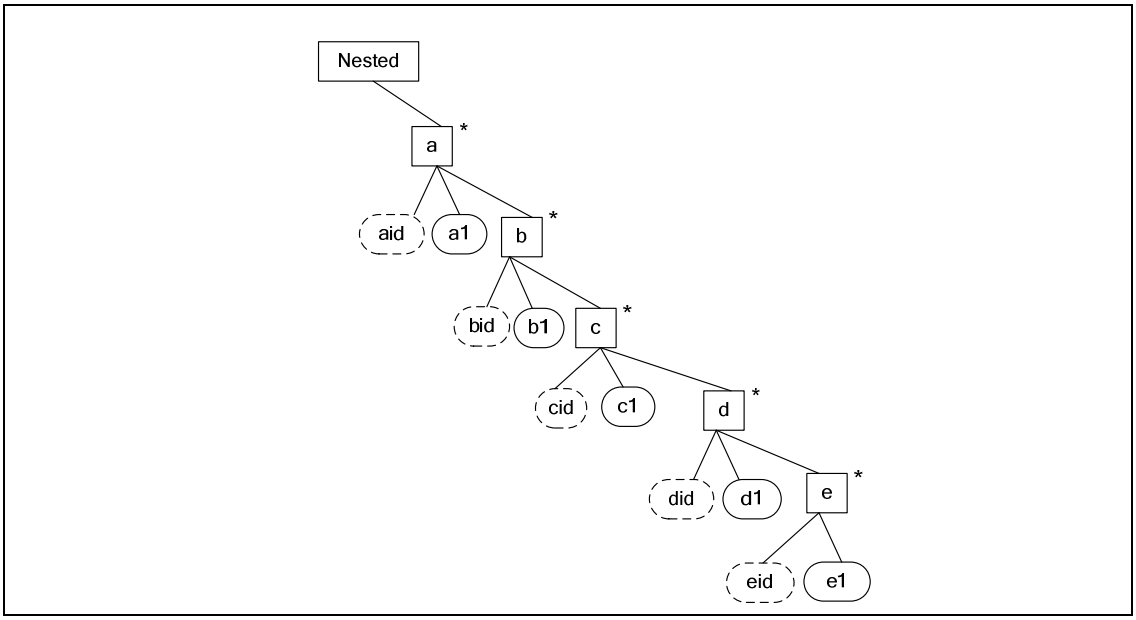


圖 B-2： Nested 結構的 DTD

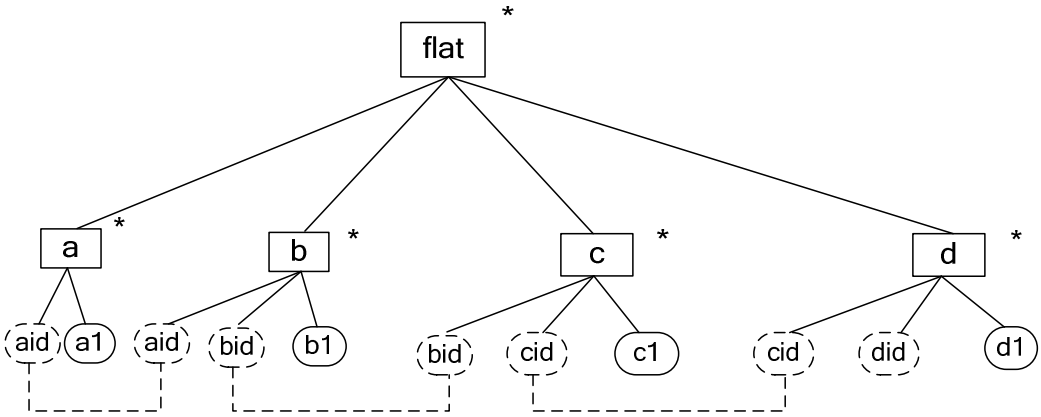


圖 B-3： Flat 結構的 DTD

輸入的 SQL
[子查詢句深度為 1] SELECT A_1.A1, SUBB.BID FROM A A_1, （ SELECT B_1.B1, B_1.AID FROM B B_1）AS SUBB WHERE A_1.AID= SUBB.AID
[子查詢句深度為 2] SELECT A_1.A1, SUBB.CID FROM A A_1, （ SELECT SUBC.CID, B_1.AID FROM B B_1, （ SELECT C_1.C1, C_1.BID FROM C C_1）AS SUBC WHERE B_1.BID=SUBC.BID）AS SUBB WHERE A_1.AID= SUBB.AID
[子查詢句深度為 3] SELECT A_1.A1, SUBB.DID FROM A A_1 ,

```
        （SELECT SUBC.DID, B_1.BID
        FROM    B B_1 ,
                    （SELECT SUBD.DID, C_1.BID
                    FROM    C C_1,
                                （SELECT D_1.D1, D_1.CID
                                FROM D D_1）AS SUBD
                    WHERE C_1.CID=SUBD.CID）AS SUBC
        WHERE B_1.BID=SUBC.BID）AS SUBB
WHERE A_1.AID= SUBB.AID
```

[子查詢句深度為 4]

```
SELECT A_1.A1, SUBB.EID
FROM    A A_1,
        （SELECT SUBB.EID, B_1.AID
        FROM    B B_1,
            （  SELECT SUBC.EID, C_1.BID
                FROM  C C_1,
                    （  SELECT SUBD.EID, D_1.CID
                        FROM D D_1,
                            （SELECT E_1.E1,E_1.DID
                            FROM E E_1）AS SUBE
                        WHERE SUBE.DID=D_1.DID）AS SUBD

                WHERE C_1.CID=SUBD.CID）AS SUBC
        WHERE B_1.BID=SUBC.BID）AS SUBB
WHERE A_1.AID= SUBB.AID
```

輸入的 XQuery（Nested 結構的 DTD）

[子查詢句深度為 1]

```
for   $a in /nested/a
let $leta ::= for $b in $a/b
              return   <result>$b/b1,$b@bid</result>
return   <result>$a/a1,$leta/b1</result>
```

[子查詢句深度為 2]

```
for   $a in /nested/a
let $leta ::= for $b in $a/b
              let $letb ::= for $c in $b/c
```


```
                    return    <result>$c/c1,$c@cid</result>
          return    <result>$letb/c1,$b/b1</result>
return    <result>$leta/b1,$a/a1</result>
```

[子查詢句深度為 3]

```
for    $a in /nested/a
let $leta ::=     for $b in $a/b
                  let $letb ::= for $c in $b/c
                                let $letc ::= for $d in $c/d
                                              return    <result>$d/d1,$d@did</result>
                                return    <result>$letc/d1,$c/c1</result>
                  return    <result>$letb/d1,$b/b1</result>
return    <result>$leta/d1,$a/a1</result>
```

[子查詢句深度為 4]

```
for    $a in /nested/a
let $leta ::=     for $b in $a/b
                  let $letb ::= for $c in $b/c
                                let $letc ::= for $d in $c/d
                                              let $letd::=for $e in $d/e
                                                          return
<result>$e/e1,$d@eid</result>
                                              return    <result>$letd/e1,$d@did</result>
                                return    <result>$letc/e1,$c/c1</result>
                  return    <result>$letb/e1,$b/b1</result>
return    <result>$leta/e1,$a/a1</result>
```

輸入的 XQuery（Flat 結構的 DTD）

[子查詢句深度為 1]

```
for    $a in /flat/a
let $leta ::=     for $b in /flat/b
                  where $a@aid=$b@aid
                  return    <result>$b/b1,$b@bid</result>
return    <result>$a/a1,$leta/b1</result>
```

[子查詢句深度為 2]

```
for    $a in /flat/a
```


```
let $leta ::=    for $b in /flat/b
                 let $letb ::= for $c in /flat/c
                                  where $b@bid=$c@bid
                               return   <result>$c/c1,$c@cid</result>
                 where $a@aid=$b@aid
                 return   <result>$b/b1,$b@bid</result>
return   <result>$a/a1,$leta/b1</result>
```

[子查詢句深度為 3]

```
for   $a in /flat/a
let $leta ::=    for $b in /flat/b
                 let $letb ::= for $c in /flat/c
                                  let $letc ::= for $d in /flat/d
                                                   where $d@cid=$c@cid
                                                   return   <result>$d/d1,$d@did</result>
                                  where $b@bid=$c@bid
                               return   <result>$c/c1,$c@cid</result>
                 where $a@aid=$b@aid
                 return   <result>$b/b1,$b@bid</result>
return   <result>$a/a1,$leta/b1</result>
```

[子查詢句深度為 4]

```
for   $a in /flat/a
let $leta ::=    for $b in /flat/b
                 let $letb ::= for $c in /flat/c
                                  let $letc ::= for $d in /flat/d
                                                   let $letd::=for $e in /flat/e
                                                                  where $e@did=$d@did
                                                               return
<result>$e/e1,$d@did</result>
                                                   where $d@cid=$c@cid
                                                   return   <result>$d/d1,$d@did</result>
                                  where $b@bid=$c@bid
                               return   <result>$c/c1,$c@cid</result>
                 where $a@aid=$b@aid
                 return   <result>$b/b1,$b@bid</result>
return   <result>$a/a1,$leta/b1</result>
```


附錄 C 直接路徑與跳層路徑的查詢句

說明：使用附錄 B 中的巢狀結構 DTD，並利用集合個數、路徑包含跳層、變數間有上下層關係，來測試以上因素對於轉換效率之影響。其中，不包含跳層的路徑稱為直接路徑，包含跳層的路徑稱為跳層路徑。而直接路徑中若與其他變數有上下層的巢狀關係，且下層變數的轉換結果受到上層變數轉換結果的影響，則稱為巢狀直接路徑，而跳層路徑若與其他變數有上下層的巢狀關係，且下層變數的轉換結果受到上層變數轉換結果的影響，則稱為巢狀跳層路徑。

直接路徑	跳層路徑	巢狀直接路徑	巢狀跳層路徑
For $a in /nested/a Return <result>$a/a1</result>	For $a in //a Return <result>$a/a1</result>	For $a in /nested/a Return <result>$a/a1</result>	For $a in //a Return <result>$a/a1</result>
for $a in /nested/a, $b in /nested/a/b return <result>$a/a1</result>	For $a in //a , $b in //b Return <result>$a/a1</result>	for $a in /nested/a, $b in $a/b return <result>$a/a1</result>	For $a in //a , $b in $a//b Return <result>$a/a1</result>
for $a in /nested/a, $b in /nested/a/b, $c in /nested/a/b/c return <result>$a/a1</result>	For $a in //a , $b in //b , $c in //c Return <result>$a/a1</result>	for $a in /nested/a, $b in $a/b, $c in $b/c return <result>$a/a1</result>	For $a in //a ,$b in $a//b ,$c in $b//c Return <result>$a/a1</result>
for $a in /nested/a, $b in /nested/a/b, $c in /nested/a/b/c , $d in /nested/a/b/c/d return	For $a in //a , $b in //b , $c in //c , $d in //d Return <result>$a/a1</res	for $a in /nested/a, $b in $a/b, $c in $b/c , $d in $c/d return <result>$a/a1</res	For $a in //a , $b in $a//b , $c in $b//c , $d in $c//d Return <result>$a/a1</res

<result>$a/a1</result>	ult>	ult>	ult>
for $a in /nested/a, $b in /nested/a/b, $c in /nested/a/b/c , $d in /nested/a/b/c/d, $e in /nested/a/b/c/d/e return <result>$a/a1</result>	For $a in //a , $b in //b , $c in //c , $d in //d ,$e in //e Return <result>$a/a1</result>	for $a in /nested/a, $b in $a/b, $c in $b/c , $d in $c/d, $e in $d/e return <result>$a/a1</result>	For $a in //a , $b in //b , $c in //c , $d in //d ,$e in //e Return <result>$a/a1</result>

附錄 D 集合的對應個數對轉換時間影響的實驗

說明: DTD 中的 A、B、C、D，其對應於關聯式表格中的表格分別為 1 對 1、1 對 2、1 對 3 與 1 對 4 的對應關係；查詢句中固定回傳值的個數為 4 個，控制 DTD 中可重覆元素對關聯式表格一對多對應的個數，比較其對轉換效率的影響。

A1（AKEY,AE1, AE2, AE3, AE4）
B1（ BKEY , BE1 ）
B2（ BKEY,BE2, BE3）
C1（CKEY , CE1）
C2（CKEY , CE2 , CE3）
C3（CKEY , CE4 , CE5 , CE6）
D1（DKEY , DE1）
D2（DKEY , DE2 , DE3）
D3（DKEY , DE4, DE5 , DE6）
D4（DKEY , DE7 , DE8, DE9 , DE10）

圖 D1 關連式表格架構

X2S Collection Mapping 如下所示。

Rname	XPath	Type	Weight	Condition	LoopPaths
A1	/root/flat/a	Repeatable	4/4		
B1	/root/flat/b	Repeatable	2/4		
B2	/root/flat/b	Repeatable	3/4		
C1	/root/flat/c	Repeatable	2/7		
C2	/root/flat/c	Repeatable	3/7		
C3	/root/flat/c	Repeatable	4/7		
D1	/root/flat/d	Repeatable	2/11		
D2	/root/flat/d	Repeatable	3/11		
D3	/root/flat/d	Repeatable	4/11		
D4	/root/flat/d	Repeatable	5/11		

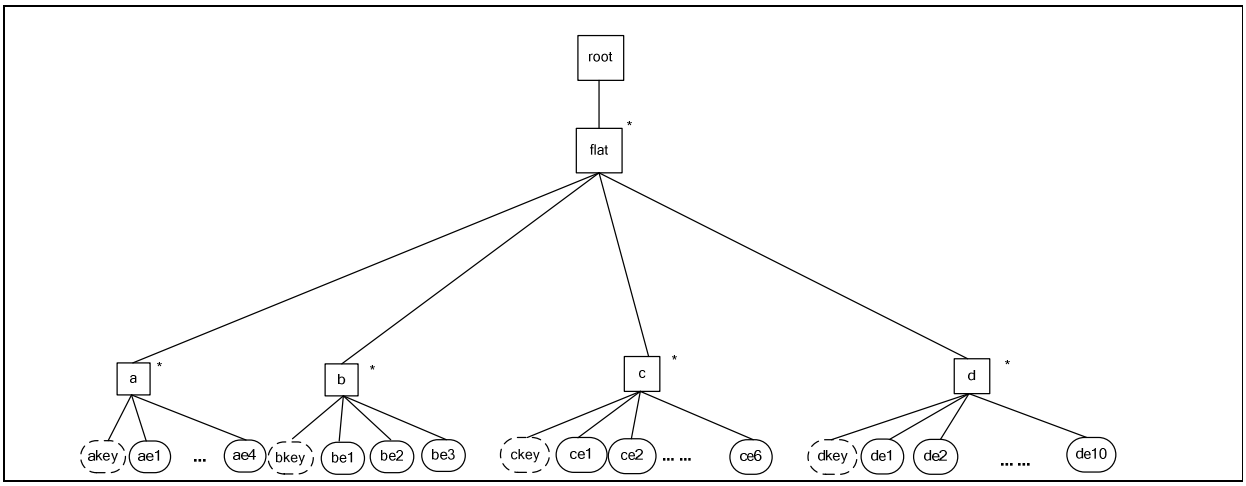


圖 D2 對應的 DTD

實驗使用的查詢句：

集合對應	輸入的 XQuery	轉換出來的 SQL
1 對 1	For $a in /flat/a Return <result>$a@akey,$a/ae1,$a/ae2,$a/ae3</result>	SELECT a_1.AKEY, a_1.AE1, a_1.AE2, A1.AE3 FROM A1 a_1
1 對 2	For $b in /flat/b Return <result>$b@bkey,$b/be1,$b/be2,$b/be3</result>	SELECT b_1.BKEY, b_1.BE1, b_2.BE2, b_2.BE3 FROM B1 b_1,B2 b_2 Where b_1.BKEY= b_2.BKEY
1 對 3	For $c in /flat/c Return <result>$c@ckey,$c/ce1,$c/ce2,$c/ce4</result>	SELECT c_1.CKEY, c_1.CE1, c_2.CE2, c_3.CE4 FROM C1 c_1,C2 c_2,C3 c_3 Where c_1.CKEY= c_2.CKEY AND c_2.CKEY= c_3.CKEY
1 對 4	For $d in /flat/d Return <result>$d/de1,$d/de2,$d/de4,$d/de7</result>	SELECT d_1.DE1, d_2.DE2, d_3.DE4, d_4.DE7 FROM D1 d_1,D2 d_2,D3 d_3,D4 d_4 Where d_1.DKEY= d_2.DKEY AND d_2.DKEY= d_3.DKEY AND d_3.DKEY= d_4.DKEY

附錄 E　X2S 值的對應個數對轉換時間影響的實驗

說明: 與附錄 D 使用相同的關連式表格以及 DTD，DTD 中的 akey、bkey、ckey、dkey，其對應於關聯式表格中的鍵值 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 DTD 的鍵值，測試屬性一對多對轉換效率的影響。DTD 中屬性的權重與其父節點的權重相同。

實驗使用的查詢句：

值的對應個數	輸入的 XQuery	轉換出來的 SQL
1	for $a in /flat/a return<result> $a@akey</result>	SELECT a_1.AKEY FROM A1 a_1
2	for $b in /flat/b return<result> $b@bkey</result>	SELECT b_1.BKEY FROM B2 b_1
3	for $c in /flat/c return<result> $c@ckey</result>	SELECT c_1.CKEY FROM C3 c_1
4	for $d in /flat/d return<result> $d@dkey</result>	SELECT d_1.DKEY FROM D D4 d_1