支援SQL與XQuery之雙向查詢句轉換系統
Bidirectional Query Translation Between SQL And XQuery

劉邦鑑 李佳臻 張雅惠
國立台灣海洋大學資訊工程研究所
{m92570017, m96570008, yahui}@ntou.edu.tw

摘要

由於電子商務及數位內容產業的蓬勃發展，延伸式標記語言 XML 已被提出來且發展為資料交換的標準。但是另一方面，關聯式資料庫已經是相當成熟的技術，也被廣為使用。所以，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。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查詢句，而系統可將該查詢句轉換成合適的表示式，到不同的資料來源取出資料，供使用者之用。我們設計了數個對應表格來記錄關聯式綱要和 XML 綱要之間的資料內容、集合與結構的多元對應關係，並設計一系列的演算法，依據對應表格所提供的資訊來進行查詢句轉換。實驗證明，我們提出的對應表格與轉換系統，可以有效率地轉換出正確的查詢句。

Abstract

Due to the development of electronic commerce, extensible markup language (XML) has recently emerged as the standard of data exchange. However, on the other hand, the technologies of the relational database is relatively mature and have been widely used. Therefore, how to provide the sharing of data between relational format and XML format, has become an important research topic. In this paper, we propose a bi-directional query translation system, which can convert queries into the appropriate expressions to get the required information from different sources. We have designed a set of mapping tables to record the complex mappings among attributes, collections and structural conflicts, and have designed a series of algorithms according to the mapping tables to transform the query statement. Experimental results show that our mapping tables and translation system can produce correct queries effectively.

關鍵字：XML 綱要, 關聯式資料庫, 查詢句轉換, 資料共享

Keywords：XML schema, relational database, query translation, data sharing

一. 緒論

由於電子商務及數位內容產業的蓬勃發展，延伸式標記語言 XML (eXtensible Markup Language) 已被提出來且發展為資料交換的標準。但是另一方面，關聯式資料庫已經是相當成熟的技術，也被廣為使用。所以，隨著 XML 格式文件的日益增多，當資料分別以關聯式格式和 XML 格式儲存時，如何提供兩種格式間的資料整合或共享，就成了重要的研究課題。

本論文提出建立一個查詢語言的轉換系統，來達到資料交流的目的。透過查詢語言的轉換，使用者可方便地依不同類型需求來下達查詢句，而系統可將該查詢句轉換成合適的表示式，以便到不同的資料來源取出相關的資料，以達成使用者的需求。針對 XML 格式，我們考慮 W3C 所提出來的 XQuery 查詢語言，而關聯式資料則採用其標準語言 SQL。我們的系統希望能夠同時考慮兩種語言表示法的差異性，以達到可以雙向轉換（亦即 SQL 轉 XQuery 或 XQuery 轉 SQL）的功能。

另一方面，欲達到資料共享的目的，還必須解決兩類型資料表示方法的不一致性，以表示其對應關係。其中，XML 資料的表示法有較多的彈性，譬如元素間可以形成彼此包含的巢狀結構 (nested structure)，屬於半結構化資料 (semi-structured data)。相較之下，關聯式資料具有較固定的格式，為結構化資料 (structured data)。在本論文中，我們將適當地表示關聯式資料與 XML 資料之間的資料內容 (value)、集合 (collection) 與結構 (structure) 間的對應。同時，我們將特別考量更為複雜的狀況，也就是彼此的表示法有多元的對應關係。譬如，在 XML 文件內集中表示在一個元素下的資料，在關聯式資料庫卻分散由多個表格所表示。這是一個常見的現象，但是目前在文獻上的研究都只限於 1 對 1 的對應，而並沒有針對此點有特別的討論。

綜合而言，本論文的貢獻如下所述：

1. 探討資料表示法的差異：分析關聯式資料與 XML 格式資料表示法的差異，其中又可分為資料內容、集合與結構的表示法差異。

2. 設計對應表格：設計數個對應表格來記錄表示法的差異性，包含資料內容、集合、結構等之間的一元或多元對應資訊。

3. 實作雙向查詢句的轉換系統：提出一系列的演算法進行查詢句轉換，並實際開發此系統。經由實驗證明，我們的轉換系統可正確且有效率地轉換出對應的查詢句。

本篇論文架構如下：在第二節中，首先比較關聯式資料與 XML 格式資料的表示法差異性。第三節則說明我們所設計的對應表格，以及如何表示兩個綱要間的對應。第四節是轉換系統核心的模組介紹，我們將說明相關的演算法及轉換過程，並於第五節探討轉換系統的正確性與效率。最後，在第六節列舉相關文獻，並於第七節提出結論與未來展望。

二.相關定義

在本節中，我們將介紹範例資料與查詢句，並探討其中的差異性。

2.1 關聯式資料與 SQL

範例關聯式資料如圖 1 所示，該範例是依據 TPC-H Benchmark 所簡化而來[1]，共包含九個表格。其中，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；CUSTOMER 表格記錄顧客的名稱與附註資訊；而 CUSTEL 記錄了顧客的電話資料，為從顧客資料中經過正規化獨立出來的表格；ORDER 表格記錄訂單的內容；而 NATION 與 REGION 表格則是記錄了供應商與顧客的地理位置。以上七個表格各自表示不同的實體 (entity) 資料，所以我們將其歸類為「實體表格」。

此外，表格 PARTSUPP 記錄供應商所提供的零件，是利用 SUPPKEY 與 PARTKEY 建立 PART 與 SUPPLIER 表格之間的關聯；LINEITEM 表格除了利用 LINENUMBER 和 SHIPMODE 代表出貨單的編號和運送方式，另外透過 ORDERKEY、SUPPKEY 與 PARTKEY 建立 ORDER、SUPPLIER 及 PART 三個表格間的關聯。所以，我們稱此二表格為「關係表格」。在圖 1 中，我們利用黑色底線表示每個表格的主鍵 (primary key)，而虛線則表示外來鍵 (foreign key)。

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

圖 1 關聯式資料範例

針對關聯式資料庫管理系統所提出的標準查詢語言為 SQL，一個基本的 SQL 資料查詢句，主要可分為 SELECT、FROM、WHERE 三子句，其中 SELECT 子句表示輸出的資料表格欄位名稱，FROM 子句表示提供資料的來源表格，WHERE 子句則允許對資料表格中的欄位做相關限制。範例 1 的查詢句是針對圖 1 關聯式資料所設計的 SQL 查詢句，其目的在找出零件 dvd 的種類和供應商的名稱。其中編號 (1) 的表示式，限制某個特定欄位的值，稱作**選取限制式** (selection condition)，而編號 (2) 的表示式，用來聯結兩個相關的表格，稱做**連結限制式** (join condition)。

【範例 1】

```
SELECT SUPPLIER.NAME, PART.TYPE
FROM  SUPPLIER, PART, PARTSUPP
WHERE  SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY
          AND
          PART.PARTKEY = PARTSUPP.CUSTKEY（２）
          AND   PART.NAME = “dvd”  （１）
```

2.2 XML 資料與 XQuery

XML 資料允許使用者自行定義元素標籤以說明內容資料之意涵，為一個階層式的樹狀結構。XML 資料通常利用 DTD 予以規範文件格式，在本論文中我們將該定義以 DTD Graph 表示，如圖二所示。其中，實線方塊表示**內部節點**元素，實線橢圓表示有值的元素，而虛線橢圓則表示元素的屬性，後兩者我們合稱**葉節點**。節點間的實線表示元素及其子元素的關係，若針對一個父元素可出現零次以上，則於右上角加上星號，並稱其為**可重複元素**。虛線則連結相同意義的葉節點。

我們使用的範例 XML 資料是修改 TPC-H Nested XML Schema [2]而來，其中圖 2 記錄了顧客與供應商間的貨品訂購資訊，而圖 3 則記錄了顧客的資料與地理資訊。值得注意的是，圖 2 的 suppliers 元素僅僅是將所有 supplier 子元素包覆起來，其意涵與 supplier 完全相同，所以我們稱其為**空殼元素** (dummy node)。

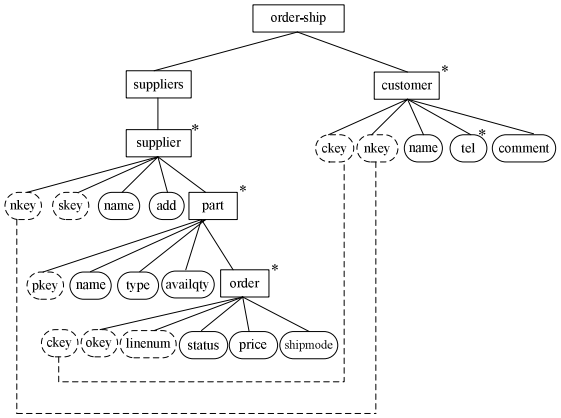


圖 2 訂單 DTD Graph 範例

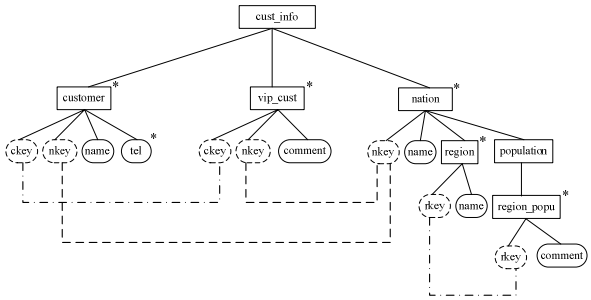


圖 3 顧客 DTD Graph 範例

XML 文件的查詢語言為 XQuery，該查詢句利用路徑表示法 (path expression) 取出 XML 文件中於特定位置的元素，如圖 2 中對應元素 supplier 的路徑表示法是 /order-ship/suppliers/supplier 。XQuery 語言在 W3C 的定義中具有複雜的結構，此論文僅處理核心的表示式，如範例 2 所示。其中，for 子句以遞迴方式取得一個路徑表示法的集合結果，並給予一個變數名稱；where 子句允許對變數所代表的集合內容做條件的限制；return 子句則將建構好的資料回傳給使用者。範例 2 的 XQuery 查詢句對應到圖 2，其語義與範例 1 的 SQL 查詢句相同。

【範例 2】

```
for  $t0 in /order-ship/suppliers/supplier ,
        $t1 in $t0/part
where  $t1/name = “dvd”
return  $t0/name , $t1/type
```

與 SQL 相較之下，XQuery 的 where 子句，如同 SQL 一般，可下達選取限制式和連結限制式。但是，除此之外，XQuery 的 for 子句也可直接表示出兩個變數的結構限制。譬如範例 2 的 $t1 in $t0/part 表示式，即限制$t1 的內容是$t0 變數所代表路徑的子元素 part，我們稱此類表示式為**巢狀限制式** (nested condition)。

2.3 資料表示法差異比較

在本節中我們比較關聯式綱要與 XML 綱要表示法的差異與對應關係。我們將分成資料內容 (value)、集合 (collection) 和結構 (structure) 三方面的表示法進行討論。

首先，就資料內容方面，關聯式資料庫是由欄位所表示，而在 XML 文件中，則是由葉節點所表示。至於兩個綱要中，意義相同的內容，可能由不同名稱的欄位或元素表示。同時，兩者之間，除了常見的一對一關係外，也有可能具有一對多的關係。譬如，圖 1 中 CUSTOMER 表格的 CUSTKEY 欄位，為顧客的識別值，而在圖 3 中，同時由路徑 /cust_info/customer 與 /cust_info/vip/vip_cust 下的屬性節點 ckey 所表示，為 1 個 1 對多的情況。

其次，就集合表示方面，關聯式資料庫中是以表格來儲存一組資料列的集合；但在 XML 文件中，相關的一組資料集合，通常是被建立包含於某個內部節點之下。同時，如同資料內容，此部分的對應差異情形，也有可能產生名稱的差異和多元對應的關係。譬如，顧客的資訊在圖 2 中由一個內部節點元素 customer 所代表，但是在圖 1 中則對應到 CUSTOMER 表格與 CUSTEL 表格。另一方面，同一路徑上的可重覆元素與空殼元素，由於語意相同，所以也會對應到相同的表格。

最後，就結構對應方面，在關聯式資料庫中，儲存資料的結構是以扁平 (flat) 的表格方式存在，表格之間的連結關係是用相同的鍵值來達成。譬如，圖 1 關聯式資料表格 NATION 與 REGION 表格間的關聯是利用 REGIONKEY 來做為連結。而在 XML 方面，資料集合間的關聯性，基本上可分為扁平結構與巢狀結構兩種。其中，除了類似於關聯式資料庫的扁平連結式之外，我們也可以由 DTD 中的一條路徑，來直接表示多個內部節點之間的巢狀結構。譬如，於圖 3 中，我們可以直接以路徑表示法//nation/region 來取得元素 nation 下的 region 元素，不需額外其它的連結資訊。

因為集合對應方式有可能不是一對一的關係，所以結構的對應關係亦可能是一對多。譬如，若一個關聯式表格對應到不同路徑上的內部節點，則會有相同的一組關聯式的連結限制式對應到不同的扁平結構。如圖 1，表格 CUSTOMER 與 NATION 表格利用 NATIONKEY 來連結，但由於 CUSTOMER 表格會對應到圖 3 的 customer 與 vip_cust 兩個元素，因此該關聯式連結會對應到 (nation 元素與 customer 元素) 或是

(nation 元素與 vip_cust 元素)的兩組扁平結構的關係。

反之，若一個可重覆元素對應到兩個以上的關聯式表格，則一組扁平結構會對應到數個關聯式連結。對照圖 1 與圖 2，ORDER 表格會對應到可重覆元素 order，而 CUSTOMER 與 CUSTEL 皆對應到可重覆元素 customer，因此連結限制式 ORDER.CUSTKEY = CUSTOMER.CUSTKEY 與 ORDER.CUSTKEY = CUSTEL.CUSTKEY 皆對應到 (order 元素與 customer 元素) 此組扁平結構關係。

至於巢狀結構方面，圖 1 的關聯式表格 REGION，同時對應到圖 3 的 region 與 region_popu 元素，而此兩元素皆為元素 nation 的子孫元素，因此關聯式表格 NATION 與 REGION 的連結限制式會對應到圖 3 中的//nation/region 或是 //nation //region_popu 的巢狀關係。而另一種比較特殊的對應關係，是介於巢狀結構和關係表格之間。也就是形成巢狀結構的兩個元素，所對應到的兩個實體表格，必須經由另一個關係表格才能建立連結。譬如，圖 2 的巢狀關係 //supplier/part，是對應到圖 1 的兩個實體表格 SUPPLIER、PART，和一個關係表格 PARTSUPP。

本論文提出的轉換系統，將針對以上所討論的差異性，提出有效的解決辦法。

三.對應表格

針對第二節討論的資料表示法差異性，我們設計了數個對應表格 (Mapping Tables)，來記錄關聯式資料與 XML 資料間彼此的資料與結構對應關係，以下將分類加以說明。

3.1 解決基本差異的對應表格

所謂基本差異，是包含資料內容和集合表示法的差異。為了解決此項差異，我們設計了 Collection Mapping Table、Internal Join Table、Value Mapping Table 三個表格來記錄每個關聯式資料表格、欄位與 XML 端的元素、路徑對應關係。此節的範例資料是根據圖 1 的關聯式資料庫與圖 2 的 XML 訂單資料。

首先，Collection Mapping Table (簡稱 CMT) 是用來記錄關聯式資料庫表格與 XML 端內部節點的路徑對應關係，以解決集合表示法的差異性。CMT = {(Rname, XPath, Type)}，其中 Rname 是 關聯式資料庫表格的名稱，XPath 為此表格對應於 XML 端的內部節點路徑，而 Type 則是此內部節點的類型名稱。若 Type 的值為 Repeatable，代表此內部節點為一個可重覆元素，若為 Dummy，則代表此內部節點為一個空殼元素。舉例來說，表格 1 的第二筆資料，表示 SUPPLIER 表格對應到位於路徑/order-ship/suppliers/supplier 的可重複元素。若一個關連表格對到多個元素，則 type 值為 Repeatable 的優先順序會高於 type 值為 Dummy 的元素。

表格 1 Collection Mapping Table

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

同時，當兩邊集合有 1 對多的情形時，有時我們必須利用 Internal Join Table (簡稱 IJT) 來記錄內部連結的關係。IJT = {(Direction, Collection1, Collection2, InternalJoinCondition)} 由四個欄位組成。Direction 為適用此條件的轉換類型對應，其中 S2X 代表 SQL 轉換至 XQuery 的情況，而 X2S 代表 XQuery 轉換為 SQL 的情況。Collection1 和 Collection2 為資料集合的來源，可同為 DTD 中的可重覆元素或是同為關聯式表格名稱。InternalJoinCondition 是此兩資料集合間的連結條件式。

如之前所討論，customer 元素對應到表格 CUSTOMER 和 CUSTEL，我們必須記錄他們之間的聯結，以取得所有的顧客資料。其連結方式如表格 2 的最後一筆資料所示。

表格 2 Internal Join Table

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

針對內容表示法差異，我們定義 Value Mapping Table (簡稱 VMT) 表格記錄關聯式資料庫欄位與 XML 端的葉節點路徑的對應關係。VMT = {(Rname, Aname, XPath, RXPath)}，其中 Rname 為關聯式資料庫表格的名稱，Aname 為此表格內的欄位名稱，XPath 是此欄位對應於 DTD 中的葉節點所代表的路徑，而 RXPath 則是與此葉節點最接近的祖先層之可重覆路徑。在表格 3 中，我們只列出部分的對應內容。

表格 3 部分 Value Mapping Table 內容

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

3.2 解決結構差異的對應表格

為了解決結構差異，我們設計了三個表格，其中 Join Mapping Table 記錄關聯式表格間具有結構意義的連結表示式，Path Mapping Table 記錄於 DTD 中內部節點彼此間具有結構意義的關聯，而 Structure Mapping Table 則是記錄上述兩表格中的結構對應關係。

Join Mapping Table (簡稱 JMT) = {(RID, Condition1, Condition2)}，如表格 4 所示。其中 RID 是此連結限制式的編號，不同的編號代表不同的意義，我們利用 RRi 代表關係表格與關係表格的連結，而其他類型表格間的連結以 Ri 代表。Condition1 和 Condition2 則是兩個表格間，用來連結的鍵值。我們特別區隔出 RRi 的類型，是因為在 1 對多的情況下，RRi 的優先順序較低。

表格 4 Join Mapping Table

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

Path Mapping Table (簡稱 PMT) = {(XID, XPath1, XPath2)} 由三個欄位組成，如表格 5 所示。其中 XID 是 DTD 端此結構關係的編號，不同的編號代表不同的意義：X_{Fi} 代表由兩個葉節點建立的扁平 (flat) 結構， X_{Ni} 代表建立於兩個可重覆元素間的巢狀 (nested) 結構，而 X_{NDi} 特別代表空殼元素 (XPath1) 與可重覆元素 (XPath2) 間的巢狀 (Nested) 結構。同樣，我們區隔 X_N 與 X_{ND} 的目的，是在 1 對多的情況下，X_{ND} 的優先順序較低。

表格 5 Path Mapping Table

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

而兩邊綱要結構表示式的對應，是建立於 Structure Mapping Table (簡稱 SMT) 中，如表格 6 所示。其中，RID 是 JMT 中的連結限制式的編號，而 XID 則是 PMT 中的結構關係編號。一組 (RID, XID) 的資料表示了一個連結限制式與 XML 結構關係的對應。由於結構表示式一般較長，所以我們將其切割出來表示於 SMT 中，且利用編號作為代表。觀察表格 6，R1 對應到 X_{ND1} 和 X_{N1}，也就是關聯表示式：
SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY，同時對應到 DTD 中的兩個巢狀表示式。若是從 SQL 轉到 XQuery 時，我們會選擇 X_{N1} 所代表的結構式。

表格 6 Structure Mapping Table

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

四、轉換演算法

本節將介紹相關的轉換演算法。在轉換過程中，將透過第三節所設計的對應表格，取得轉換時所需要的對應資訊，來進行查詢句轉換的處理。

4.1 轉換系統架構

雙向查詢句轉換系統如圖 4 所示。首先，使用者所輸入的查詢句，將會依據每個子句（如

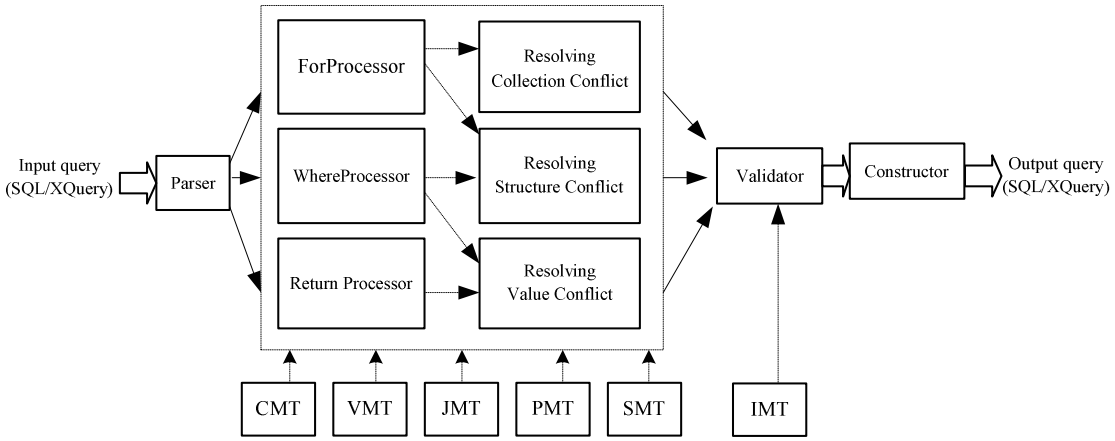


圖 4 雙向查詢句轉換系統架構圖

XQuery 的 for、where、return），分別轉換成內部表示法，然後傳給對應的模組做處理。每個模組將針對該子句所會產生的差異性，至對應表格取得所需的對應資料進行轉換，產生的片段結果會記錄於 ForSet、WhereSet 與 ReturnSet 之中。由於綱要對應在 1 對多的情況，可能在轉換的過程中會產生不必要的資料，所以這三個資料結構會再經過 Validator 濾除多餘的資料並做合理化的動作後，最後透過 Constructor 產生最終輸出的查詢句。

針對之前所討論的差異性，我們的系統會先處理集合的差異性，產生集合的對應之後，接著解決資料內容的差異性，和最複雜的結構差異性，以下各節將分別說明對應的演算法。由於 XQuery 的表示法較為複雜，我們將以範例 2 的 XQuery 作為主要的範例說明。

4.2 解決基本差異的演算法

針對集合表示法的差異性，ForProcessor 會處理 XQuery 的 for 子句，參照 CMT 對應表格的資料，取出對應的關聯式表格，並記錄於 ForSet 中。ForSet = {(XPath, Rname, UsedFlag)}，每一筆資料由三個欄位組成，其中 XPath 是表示於 for 子句中元素的路徑，Rname 是此路徑表示的節點對應到的表格名稱，而 UsedFlag 是用來判斷多元對應時是否會輸出該表格。UsedFlag 一開始皆設為 FALSE，若是在之後的處理發現此表格被其他子句參考到，則會將其 UsedFlag 改為 TRUE。

針對範例 2 中的 XQuery，我們產生的 ForSet 如表格 7 所示。注意到 /order-ship/suppliers/supplier/part 元素對應到 PART 和 PARTSUPP 兩個表格。

表格 7 ForSet 範例

ForSet ={ (/order-ship/suppliers/supplier/part, PART, FALSE), (/order-ship/suppliers/supplier/part, PARTSUPP, FALSE), (/order-ship/suppliers/supplier, SUPPLIER, FALSE) }

針對資料內容的處理，可分做對 XQuery 中 return 子句的處理，和對 where 子句中選取限制式的處理。以 ReturnProcessor 為例，該演算法會針對 XQuery 中的 return 子句裡出現的葉節點，至 VMT 對應表格中取出對應的欄位資料，然後將輸

Algorithm NestedToJoin(JMT, PMT, FromSer, WhereSet)

Input: JMT , PMT , FromSet , WhereSet
Output: FromSet , WhereSet

```
L01  for each tuple P_i that P_i.XID∈ X_Ni or Pi.XID∈ X_NEi from PMT do
L02   if P_i.XPath1 and P_i.XPath2 all found in FromSet's RXpath
L03     XID_i = GetXIDByPMT(Pi.XPath1, Pi.XPath2 , PMT);
L04     if XID_i is not NULL      // XID_i will ∈ X_Ni
L05     Get this XID_i's related RID∈ R_i via SMT and store in x2stmpStructureList;
          // x2stmpStructureList = {(XID_i , RID1, RID2, …, RIDn)}
L06     end if
L07    end if
L08   end for
L09     OutputStructureList = SelectStructure(tmpStructureList)
L10   for each RIDi in OutputStructureList  //重覆的不會處理
L11       J_i = GetTupleByJMT(RID_i, JMT);   //Ji is the tuple with the identified RID
L12       X2SWhere_clause = J_i.Condition1 + " = " + J_i.Condition2;
L13     X2SWhereSet = X2SWhereSet ∪ {(X2SWhere_clause)};
L14   end for
L15   return X2SWhereSet;
```

圖 5 NestedToJoin 演算法

出的結果記錄在 ReturnSet 中。例如： 範例 2 處理後的 ReturnSet 為 {(PART.TYPE), (SUPPLIER.NAME)}。同時，針對被參考到表格，ReturnProcessor 也會更新其在 ForSet 中的 UsedFlag 的值，將其設為 TRUE。以表格 7 為例，第一筆和第三筆資料的 UsedFlag 會被更新為 TRUE。

至於對 where 子句中的選取限制式，也是類似的處理。處理後的結果記錄於 WhereSet 中。例如 {(PART.NAME = "dvd")}。

4.3 解決結構差異的演算法

針對結構表示法的不同差異，由於我們已將關聯式資料的連結資訊記錄於 JMT 表格中，DTD 中的結構表示記錄於 PMT 中，而兩者間的關聯則是以 SMT 記錄，所以轉換過程將以此三個表格所提供的資訊來進行轉換。

4.3.1 結構轉換處理

XQuery 中的結構限制式可能出現在 for 子句或 where 子句中。首先，處理 XQuery 的 for 子句的演算法，如圖 5 的 NestedToJoin 所示。該演算法首先會依序考慮表格 PMT 中每一筆資料列，若該筆資料列的編號格式為 X_{Ni} 及 X_{NDi} ，也就是代表巢狀結構，我們則判斷其 XPath1 與 XPath2 是否同時出現在 ForSet 中，若存在，則代表 for 子句裡表示了具有巢狀結構的關係式。接著，依此 XID 至 SMT 中取得其所有對應的 RID 編號，由於結構間可能有 1 對多的對應，演算法會呼叫 SelectStructure 演算法選擇適當的結構輸出，該演算法將於稍後說明。利用所輸出的 RID 編號，我們至 JMT 找到對應的連結限制式，然後記錄於 WhereSet 中。

以範例 2 的 XQuery 查詢句做說明，因為 X_{N1} 的 XPath1 與 XPath2 同時出現在 ForSet 中，所以可以得知此查詢句具有巢狀結構。依此 XID 至 SMT 中取得對應的 RID 編號 為 R1 和 R5，此資訊 (X_{N1}, R1, R5)暫存於 tmpStructureList 之中，並呼叫 SelectStructure 演算法選擇適當的結構輸出，在此我們先假設此兩個 RID 編號都會被輸出。根據該 RID 至 JMT 找到的連結限制式分別是 SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY 以及 PART.PARTKEY = PARTSUPP.PARTKEY，所以此二連結限制式會被加入到 WhereSet 中。連同之前處理選取限制式的結果，更新後的 WhereSet 內容如表格 8 所示。同時注意到，由於 PARTSUPP 被使用到，所以它在 ForSet 中的 UsedFlag 也會被設為 TRUE。

表格 8 WhereSet 內容範例

WhereSet ={ (PART.NAME = 'dvd'), (SUPPLIER.SUPPKEY= PARTSUPP.SUPPKEY), (PART.PARTKEY= PARTSUPP.PARTKEY)}

至於在 XQuery 中 where 子句中的連結限制式，表示 DTD 中兩個非巢狀關係的元素之間的關係。在進行轉換時，我們會根據該限制式中兩個葉節點的路經，至 PMT 中取得對應的 XID，再依此 XID 至 SMT 中取得對應的 RID 編號。同樣，由於可能會有一對多的情況，所以也必須透過 SelectStructure 演算法選擇適當的結構加以輸出。

4.3.2 選擇對應結構輸出的方式

我們在本節說明演算法 SelectStructure。針對結構多元對應關係的處理，我們是優先輸出結構表示式中包含已經被使用的資料集合。為了便於說明，在此設計一個 2*2 的對應關係，如圖 6 所示。其中，A 元素對應的表格為 a1 與 a2 ，而 B 元素對應的表格為 b1 與 b2，因此元素 A 與元素 B 之結構限制式所對應的 RID 編號會有四組 R_{F1}~R_{F4}。若某一表格在 ForSet 中其 UsedFlag 被設為 TRUE ，則在圖中我們以實心圓圈表示，而線條的編號則代表所對應的結構編號。

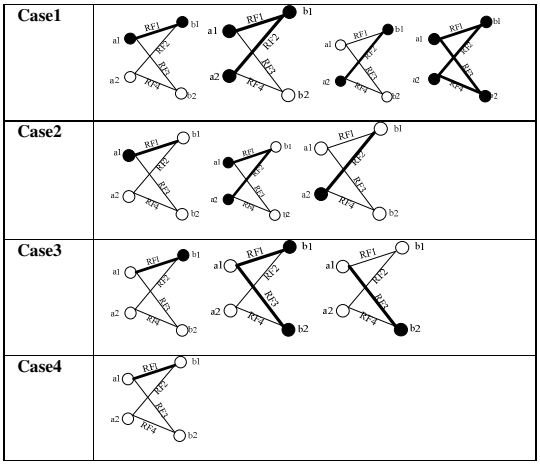


圖 6 選擇輸出結構的示意圖

在圖 6 中我們討論四種情況。首先，若某個 RID 代表的限制式，其兩個條件式連結的兩個表格，在查詢句的其他部分都被使用到，則此連結是必要的，一定會被輸出，情形如 Case1 所示。其次，若僅有一個條件式的表格其 UsedFlag 為 TRUE 時，且該條件對應到多個結構限制式，此時可任選一組包含該條件式的結構式輸出，我們會選擇於對應表格中所找到的第一筆資料。Case2 和 Case3 分別表示 condition1 和 condition2 其表格的 UsedFlag 被設為 TRUE 的情況。最後一種情況，是兩個條件式對應表格的 UsedFlag 皆為 FALSE ，該情形會發生在查詢句列舉多個表格的連結關係，但僅輸出其中一個表格的資料時。同樣，任一組結構式皆可輸出，而我們會選擇第一筆對應的 RID 輸出，如 Case4 所示。所有演算法選擇轉換出的對應結構在圖中皆以粗實線標示。

Algorithm SelectStructure (tmpStructureList)

Input: tmpStructureList
Output: RIDs

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

圖 7 SelectStructure 演算法

完整的演算法如圖 7 所示，我們仍然以 XQuery 轉換成 SQL 為例。不過，在呼叫該演算法之前，必須先針對 JMT 中的每個條件式 (condition1 與 condition2) 進行前處理，也就是若其包含的表格在 ForSet 中的 UsedFlag 已被設為 TRUE 者，則其 ref_flag 也會被設為 TRUE。接著，演算法會依照圖 6 的四個 Case 分別進行選擇。接續其一小節的範例，討論處理{(X_{N1}, R1, R5)}的情形。根據 JMT 的對應資料，R1 代表 SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY，R2 代表 PART.PARTKEY = PARTSUPP.PARTKEY。由於 SUPPLIER 與 PART 的 UsedFlag 為 TRUE，而 PARTSUPP 為 FALSE，所以情況如圖 6 的 Case 2。根據之前的討論，此兩個結構式都是必要的，因此這兩個連結條件式都會輸出。

4.4 查詢句的驗證與輸出

在經過基本差異和結構差異的相關演算法處理後，我們會在驗證階段處理集合的多元對應。我們將檢查 ForSet 中集合的 UsedFlag，若是其值仍為 FALSE，表示該集合並沒有被其他子句使用到，則我們會將該集合刪除。另一方面，若是兩個意義相等的集合皆被使用到，則我們會參考 IJT 將缺少的連結表示式補齊。

經過驗證模組處理後， Constructor 會根據最後 ForSet、WhereSet 與 ReturnSet 的內容組合成完整的查詢句以輸出。範例 2 輸出的 SQL 查詢句結果如下所示：

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

4.5 SQL 至 XQuery 的轉換處理

在前面的小節中，我們主要以 XQuery 轉換至 SQL 做為主要的說明對象，在此小節中我們補充說明從 SQL 轉換至 XQuery 所需注意的事項。

首先，由於 XQuery 的 for 子句中需要宣告變數，所以 ForSet 中也必須記錄被轉換出的內部節點所對應的變數。而處理其他子句時，如 SELECT 子句或 WHERE 子句中的選取限制式，也必須參考 ForSet 找出對應的變數做變數代換。

至於在結構處理方面，若是關聯式連結對應到巢狀結構的話，我們必須根據由 PMT 中取得的 XPath1 與 XPath2 內容，由 ForSet 中取得各自對應的變數名稱，再將路徑較深的路徑以路徑較短的的變數名稱來代換成巢狀結構的表示法，並更新 ForSet 中相對應的內容。若關聯式連結對應到非巢狀結構，則取得的兩個葉節點路徑，必需至 VMT 表格中取得其最接近的可重覆元素路徑，再以該可重覆元素所分配到的變數名稱，進行路徑表示法的代換。

五.實驗結果

在本節中，我們將測試本系統的正確性與效率。針對正確性的實驗，我們將設計不同類型的查詢句，分別進行兩個方向的轉換。譬如，將 SQL 查詢句轉換出的 XQuery，再轉換成對應的

	輸入的查詢句 Q	轉換後的查詢句 Q'	由查詢句 Q'再轉換回的查詢句 Q''
【Q1】	for $t0 in /cust_info/customer return $t0@ckey	SELECT CUSTOMER.CUSTKEY FROM CUSTOMER	for $t0 in /cust_info/customer return $t0@ckey
【Q2】	for $t0 in /cust_info/nation/ population/region_popu return $t0/comment	SELECT REGION.COMMENT FROM REGION	for $t0 in /cust_info/nation/population/region_popu return $t0/comment
【Q3】	SELECT CUSTOMER.NAME, ORDER.ORDERSTATUS FROM CUSTOMER , ORDER WHERE CUSTOMER.CUSTKEY = ORDER.CUSTKEY AND CUSTOMER.CUSTKEY='c01'	for $t0 in /cust_info/customer, $t1 in /order-ship/supplier/ suppliers/part/order where $t0@ckey="c01" and $t0@ckey=$t1@ckey return $t0/name, $t1/status	SELECT CUSTOMER.NAME , ORDER.ORDERSTATUS FROM CUSTOMER , ORDER WHERE CUSTOMER.CUSTKEY = ORDER.CUSTKEY AND CUSTOMER.CUSTKEY = 'c01'
【Q4】	SELECT CUSTOMER.NAME FROM CUSTOMER , NATION WHERE CUSTOMER.NATIONKEY = NATION.NATIONKEY AND NATION.NAME = '台灣'	for $t1 in /cust_info/nation, $t0 in /cust_info/customer where $t1/name = "台灣" And $t0@nkey=$t1@nkey return $t0/name	SELECT CUSTOMER.NAME FROM NATION , CUSTOMER WHERE CUSTOMER.NATIONKEY= NATION.NATIONKEY And NATION.NAME = '台灣'
【Q5】	SELECT SUPPLIER.NAME, PART.NAME FROM SUPPLIER, PARTSUPP, PART WHERE SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY AND PART.PARTKEY= PARTSUPP.PARTKEY	for $t0 in /order-ship/suppliers/supplier, $t1 in $t0/part return $t0/name, $t1/name	SELECT SUPPLIER.NAME, PART.NAME FROM SUPPLIER, PARTSUPP, PART WHERE SUPPLIER.SUPPKEY = PARTSUPP.SUPPKEY AND PART.PARTKEY= PARTSUPP.PARTKEY

圖 8 正確性實驗的範例查詢句

SQL'，若 SQL 與 SQL'語義相同，則稱為正確的轉換。針對效率性的分析，我們將評估轉換輸出的 Structure 個數、DTD 為階層式或與扁平式對轉換效率的影響。實驗測試的電腦配備中央處理器為 P4-2.4 Ghz，記憶體為 512MB DDR，搭配 Windows 2000 Advance Server 中文版作業系統。

5.1 正確性分析

本小節我們將分析查詢句轉換的正確性。我們針對不同的差異性執行多次的查詢句轉換，其中 5 個最具代表性的查詢句如圖 8 所列。這些查詢句包含了 Value 多元對應、集合多元對應、關聯式連結與扁平結構之間的一對一對應及一對多對應等，下面將分別進行討論。

Q1 是一個 Value 多元對應的轉換範例，其中圖 3 的/cust-info/customer@ckey 對應到圖 1 的 CUSTOMER.CUSTKEY 和 CUSTEL.CUSTKEY，具有葉節點與欄位 1 對 2 的對應關係。在本系統中我們會選擇 CUSTOMER.CUSTKEY，轉換後的查詢句與轉換前相同。

Q2 是針對集合多元對應的範例，其中 REGION 表格對應到/cust_info/nation/population 與 /cust_info/nation/population/region_popu，前者為空殼元素而後者為可重複元素，為 1 對 2 的對應關係。在由 Q2' 轉換成 Q2'' 的過程中，由於可重覆元素所對應的路徑會優先於空殼元素，所以轉換出來的 XQuery' 雖然會與原先所輸入的 XQuery 不同，但此雙向轉換結果亦為一個等價的查詢句轉換。

接下來，Q3 表示關聯式連結與扁平結構的 1 對 1 對應。其中，針對圖 1 的 CUSTOMER.CUSTKEY= ORDER.CUSTKEY 對應於圖 2 DTD 中 order@ckey=customer@ckey。雙向轉換皆等價。

Q4 表示關聯式連結與扁平結構 1 對 2 的轉換範例。其中 CUSTOMER.NATIONKEY = NATION.NATIONKEY 對應的結構為 X_{F1} (/cust_info/customer@nkey=/cust_info/nation@nkey) 和 X_{F2} (/cust_info/vip_cust@nkey=/cust_info/nation@nkey)，為一個 1 對 2 的對應。由於輸出的顧客名稱表示於內部節點 customer，因此依據結構轉換法則會選擇編號為 X_{F1} 的結構做轉換，如 Q4'所示。而 Q4'再經由轉換系統轉換時，結構限制式 X_{F1} 僅對應到一組連結限制式，所以 Q4'' 會與原先輸入的 Q4 相同。

最後，Q5 是關係表格與巢狀結構的轉換範例，其中圖 1 的 PARTSUPP 關係表格對應於圖 2 DTD 中 supplier 與 part 組成的巢狀結構。雖然在關聯式資料庫中，PART 和 SUPPILER 表格是由 PARTSUPP.PARTKEY=PART.PARTKEY 和 PARTSUPP.SUPPKEY=SUPPILER.SUPPKEY 兩組連結限制式連結而成，但是 PARTSUPP 和 PART 皆對應到相同的元素，因此其對應連結限制式於 Q5 轉成 Q5'時不會轉換出任何結構輸出。而從 Q5'轉成 Q5''時， supplier 與 part 組成的巢狀結構只對到一組連結限制式，也就是 SUPPLIER.SUPPKEY=PARTSUPP.SUPPKEY。但是，在最後的驗證模組中，會發現 PARTSUPP 和

PART 表格皆在 ForSet 中，所以參考 IJT 再加入 PART.PARTKEY=PARTSUPP.PARTKEY 的表示式。觀察到 Q5'' 會與 Q5 相同。

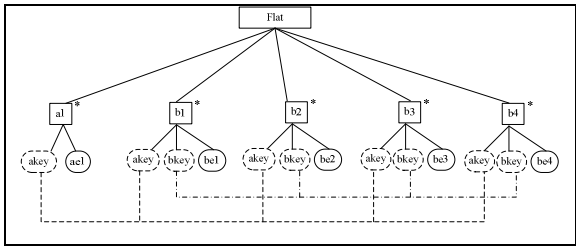
綜合而言，我們的轉換系統，針對不同的差異性時，都可以轉換出正確的查詢句。

5.2 轉換效率評估

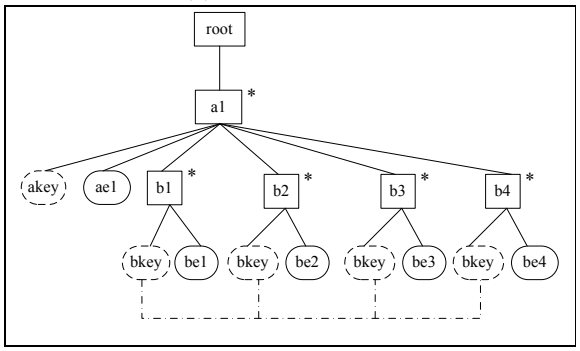
如圖 4 的轉換架構所示，整個轉換過程需經過不同的模組處理，不過其中以結構對應的處理影響轉換效能最大。所以，我們將以結構表示式的輸出個數來探討其對轉換的效率影響。

A (AKEY , AE1)
B (BKEY , AKEY ,BE1, BE2, BE3, BE4)

(a) 關連式綱要



(b) Flat 結構的 DTD



(c) Nested 結構的 DTD

圖 9 實驗一的範例綱要

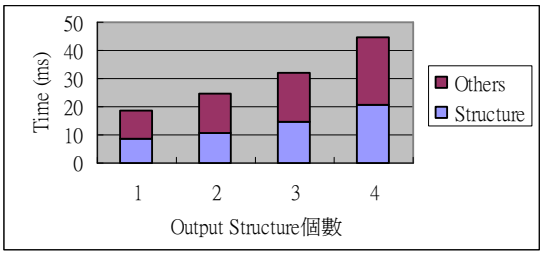
在實驗一中，我們設計一組 1 對 4 的結構對應，來比較當 SQL 轉 XQuery 時，輸出的結構個數對轉換效率的影響。關連式綱要如圖 9(a)所示，而圖 9(b)和(c)分別是對應的扁平結構的 DTD 和巢狀結構的 DTD。其中，關聯式表格 A 對應元素 a1，而關聯式表格 B 對應元素 b1、b2、b3、b4，因此表格 A 與 B 的連結限制式，對應 Flat 與 Nested 的結構表示式，皆為一對 4 的對應。

實驗結果分別如圖 10(a)與 10(b)所示，同時我們特別將轉換結構的時間與其他模組分開表示。我們可以發現到，結構的個數的確很明顯的影響到轉換的時間，不過整體來講，不論是處理結構或其他部分，都是呈現線性的成長。同時，雖然個別模組會因為 DTD 是扁平或是巢狀，而使得轉換時間有所影響，但是整體來講所需的時間是約略相同。

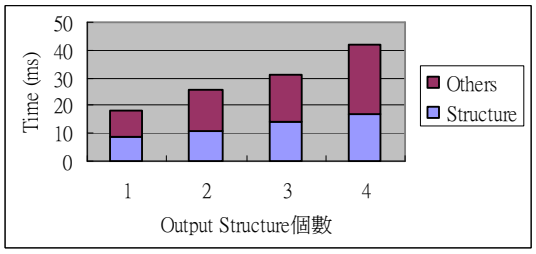
在實驗二中，我們進行 XQuery 到 SQL 的轉換實驗。扁平結構和巢狀結構的 DTD 分別如圖 11(a)與(b)所示，而對應的關聯式資料表格則列於圖 11(c)。我們讓元素 a 對應到關聯式表格的 A1，而元素 b 會對應到關聯式表格 B1、B2、B3、B4，因此，元素 a 與元素 b 之間的結構關聯，對應到關聯式資料表格的連結限制式皆為 1 對 4 的關係。

我們仍舊探討輸出的結構數對轉換效率的影響。實驗結果如圖 12(a)與 12(b)所示。我們可以發現，結構個數的輸出個數，仍然會使得結構處理時間大約是線性遞增。不過，我們發現到 DTD 的結構對轉換時間沒有太大的影響，這是因為關聯式資料皆為扁平結構的關係，且沒有 XQuery 需要路徑代換的問題。

同時由圖 10 及圖 12 我們可發現到 SQL 轉 XQuery 比 XQuery 轉 SQL 還要費時，因為 SQL 轉 XQuery 結構處理時需要另外做路徑代換的處理。

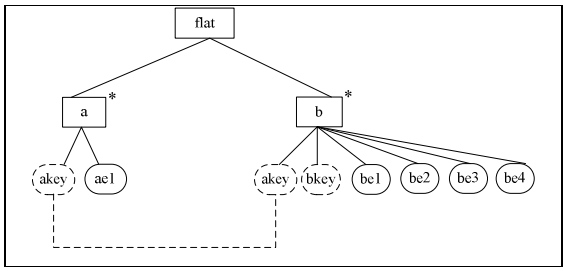


(a)：Flat Structure

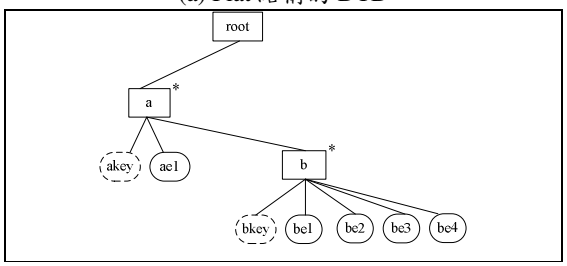


(b)：Nested Structure

圖 10 Structure 輸出個數對轉換時間的影響 (SQL to XQuery)



(a) Flat 結構的 DTD



(b) Nested 結構的 DTD

A1 (AKEY , AE1)
B1 (BKEY , AKEY ,BE1)
B2 (BKEY , AKEY ,BE2)
B3 (BKEY , AKEY ,BE3)
B4 (BKEY , AKEY ,BE4)

(c) 關連式表格架構

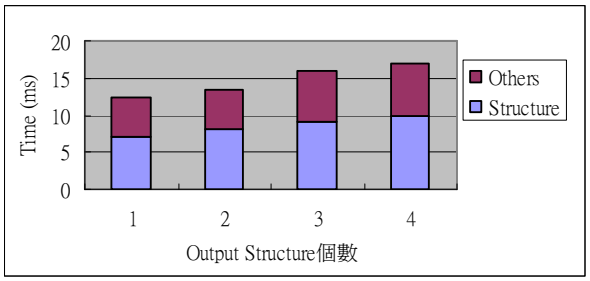
圖 11 實驗二的範例綱要

六.相關研究

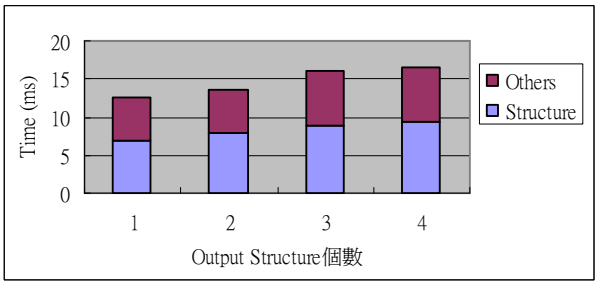
本節討論與查詢句轉換相關的研究成果。首先，很多研究者討論如何將 XML 文件資料轉入關聯式資料庫。有的考慮一般的樹狀結構，所以不需使用 DTD 的資訊 [3]。有的則根據 DTD 建立 DTD Graph，以簡化後的 DTD Graph 結構來定義資料庫的表格 [4]。論文[5] 則討論將關聯式資料以 XML view 的方式 Publish 之後，使用者針對該 XML view 再下達 XQuery 的查詢句。由於 XQuery 和 SQL 的表現能力不同，可能必須重複的下達同一個 SQL 查詢句，所以該篇論文提出一個 middleware 的架構，以達到此功能。雖然這些系統，也必須把使用者下的 XQuery 轉換成 SQL，不過其轉換都針對該作法提出的特定的綱要定義，而我們提出的對應表格則可處理一般的關連式綱要。

有些研究者則針對 XQuery 複雜的語法進行討論。論文[6] 研究如何轉換巢狀結構、並包含複雜路徑的 FLWR 表示式。它使用動態區間編碼對 XML 文件編碼，如此可快速找出元素屬性之間的層次關係，以解決 PC 路徑、AD 路徑和 wildcard 的問題。論文[7]則針對 DTD 結構為 recursive 時，討論對應的 SQL 查詢句轉換，該論文使用 SQL 中的 with 子句來轉換成 recursive 的 SQL 查詢句。而論文[8]則利用 Oracle、IBM DB2、Microsoft SQL Server 皆有支援的 Simple LeastFixpoint Operator 來進行轉換，且經過實驗證明效率比利用 with 語法佳。我們的論文討論最基本的 XQuery 語法，但是這些論文的研究結果都可合併到本系統中。

最後，有些研究者討論 schema 間的對應與整合。論文[9] 提出一個轉換模組化平台－Sangam，使用者可以輸入 XML DTD 或關聯式資料庫綱要，依據 Sangam 的定義轉成 Sangam Graph，再以圖形化方式定義的運算子以整合 Sangam 圖形，之後就可將整合好的 Sangam 圖形套入模組轉換的執行策略中，輸出成 XML DTD 或關聯式資料庫綱要。論文[10]則討論如何計算 DTDs 的相似度，以達到有效率的文件整合。考慮的因素有元素名稱、限制式還有對應路徑的相似度 (semantics similarity) 、 子孫的相似度和內容的相似度 (leaf-context similarity)。目前我們綱要間的對應必須由 DBA 建立，但是我們可以利用這些系統自動產生的對應資料，輔助 DBA 建立對應表格。



(a)：Flat Structure



(b)：Nested Structure

圖 12 Structure 輸出個數對轉換時間的影響 (XQuery to SQL)

七.結論與未來展望

本論文提出了一個介於關聯式資料庫及 XML 資料之間的雙向查詢句轉換系統，透過查詢語言的轉換，使得使用者可方便的依不同類型需求來下達查詢句，再進一步進行資料的交換與使用。我們設計了數個對應表格來記錄關聯式綱要和 XML 綱要之間的資料內容、集合與結構的多元對應關係，並設計對應的演算法依據對應表格所提供的資訊來進行查詢句轉換。經由實驗證明，本轉換系統可以正確地轉換出對應的查詢句，同時也有極佳的效率。

未來，我們考慮對轉換後的查詢句進行最佳化的處理，我們也計畫擴充對應表格與轉換機制，以便處理更複雜的綱要結構對應，及支援更多類型的查詢句轉換。

誌謝

此計畫由國科會贊助，編號為：

NSC95-2422-H-019-001

八.參考文獻

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

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

[3] T. Shimura, M. Yoshikawa and S. Uemura, “Storage and Retrieval of XML Documents Using Object-Relational Databases”, Proceedings of DEXA 1999, pages 206-217, Florence, Italy, 1999.

[4] Jayavel Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David DeWitt and Jeffrey Naughton, “Relational Databases for Querying XML Documents: Limitations and Opportunities”, Proceedings of the 25th VLDB Conference, pages 302-314, Edinburgh, Scotland, 1999.

[5] Mary F. Fernandez, Yana Kadiyska, Dan Suciu, Atsuyuki Morishima, Wang Chiew Tan, “SilkRoute: A Framework For Publishing Relational Data in XML”, ACM Transactions on Database Systems (TODS), 27(4) pages 438-493, 2002.

[6] David DeHaan, David Toman, Mariano P. Consens, and M. Tamer Ozsu, “A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding”, Proceedings of the SIGMOD International Conference on Management of Data, pages 623-634, San Diego, California, 2003.

[7] Rajasekar Krishnamurthy, Venkatesan T. Chakaravarthy, Raghav Kaushik, Jeffrey F. Naughton, “Recursive XML Schemas, Recursive XML Queries, and Relational Storage: XML-to-SQL Query Translation”, Proceedings of the 20th International Conference on Data Engineering, pages 42-53, Boston, Massachusetts, USA, 2004.

[8] Wenfei Fan, Jeffrey Xu Yu, Hongjun Lu, Jianhua Lu and Rajeev Rastogi, “Query Translation from XPath to SQL in the Presence of Recursive DTDs ”, Proceedings of the 31th VLDB Conference, pages 337-348, Trondheim, Norway. 2005.

[9] Kajal T. Claypool, Elke A. Rundensteiner, “Sangam: A Transformation Modeling Framework”, Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA), pages 219-224, Kyoto, Japan, 2003.

[10] Mong Li Lee, Liang Huai Yang, Wynne Hsu, Xia Yang, “Xclust: Clustering XML Schemas for Effective Integration.”, Proceedings of the 7th international conference on Information and knowledge management, pages 292-299, McLean, Virginia, USA , 2002.