在異質性資料環境中查詢效能提升之研究

邱紹豐	許鴻仁	邱已家
大葉大學資訊工程學系	大葉大學資訊工程學系	建國科技大學應用外語系
schiou@mail.dyu.edu.tw	R9606044@mail.dyu.edu.tw	monique@ctu.edu.tw

摘要

關聯式資料庫至今發展已有相當時間，在各方面的技術都較為成熟，但也因不同廠商擁有各自的資料庫而造成格式不一致，導致資料交換上的困難。XML 格式由於具有可自行定義標籤、資料結構化與跨平台的特性，而快速成為異質性資料平台間交換資料的標準。但由於資料來源的多樣化，造成使用者在查詢資料上的困難。本論文提出一個系統化的機制，利用 XML 架構的階層關係，來表示資料表之間的參照關係，藉此將關聯式的資料轉換成 XML 架構，讓使用者能夠利用單一介面來查詢異質性的資料來源。經過多次實驗，依照我們的轉換機制產生的 XML Schema 都能符合原本關聯式資料的架構，欄位也能擁有相對應的型態。

關鍵詞：關聯式資料庫，XML Schema，異質性資料來源。

1. 背景動機

XML 語言(可延伸標記語言，eXtensible Markup Language)自從 W3C 於 1998 年 2 月發布後，已逐漸成為網際網路上資料儲存的標準格式。相較於著重在顯示資料的 HTML，XML 可自行定義標籤，且具備資料結構化與跨平台的特性，因此快速地成為網際網路上異質平台間資料交換的標準。近年來也因 XML 格式在傳遞資料上的便利性，出現了許多 XML 格式文件與關聯式資料庫(RDB，Relational Database)結合方面的研究，但是與結構化的關聯式資料庫相比，半結構化的 XML 要與其結合是較為困難的。

關聯式資料庫由於發展較早，許多方面的技術都較為成熟，但也因此出現不同廠商擁有各自的資料庫系統的狀況。當這些資料庫系統之間需要交換資料時，便產生了格式不相容而無法讀取的情況。雖然已經有公開的存取界標準已經被確立，如 ODBC 等，但是在資料存取的效能上都較原生資料來源差。為了讓關聯式的資料能夠相容於跨平台的系統上，在本論文中我們提出一個系統化的機制，在交換關聯式資料之前先將關聯式資料庫轉換為 XML 格式。利用 XML 的跨平台特性，使資料能夠更方便的在異質性的資料庫間傳遞，使用者端則可以使用標準規格的 XQuery 來查詢經轉換後的 XML 資料。

2. 相關研究

XML 發展至今已有十多年，與關聯式資料庫之間的結合已有不少研究。以下將先介紹現有的 XML DBMS(XML Database Management System)，再介紹關聯式資料庫轉換成 XML 的方法，以及如何將 XML 文件儲存於關聯式資料庫中，最後介紹 XML Schema、XQuery 與異質性資料庫間查詢的研究。

2.1 XML DBMS

XML 在 1998 年發布後，在該年 12 月即由 Mark Birbeck 描述了一個簡單的 XML 資料庫概要[7]。相較於傳統關聯式資料庫以精確的欄位定義資料型態後，再將資料依列(row)的形式來儲存的方式，XML 則是利用 element 與 attribute 來保存資料，再搭配 XML Schema 來控制資料的正確性，只要符合定義便可以新增欄位，彈性較傳統資料庫來的大。XML 資料庫目前主要可分為兩類[1]：

1. XED(XML-Enabled DBMS)：XED 著重

在 XML 與資料庫的轉換上，如 Oracle 的 Oracle 11g[6]。將 XML 文件的資料取出，再利用結構(DTD 或是 XML Schema)來推導出關聯式資料庫中的表格及欄位並存入，因此在轉換混合內容(mixed content)上較為麻煩。轉換時一部分的物理結構會被忽略，如 CDATA 或是註釋等，還原時可能會導致與原檔案有所不同，所以 XED 較適用在架構較規律、內容較簡單的以資料為中心的 XML 文件(data-centric)。在查詢上，由於經過轉換，可直接使用 SQL。

2. NXD(Native XML DBMS)：NXD 是將 XML 文件作為最小儲存單位，直接存進資料庫中，不須經過轉換，如 Tamino XML Database[9]、以及 TEXTML Server[8]。存進資料庫時會保留原本的階層關係，並利用 XML 的樹狀結構特性來查詢、新增與刪除資料。由於 XML 文件的結構不會被破壞，所以可以對應大部分 XML 的新技術，對於格式複雜的文件支援性也較好，適用於內容較不規律的以文件為中心的 XML 文件(document-centric)。

2.2 關聯式資料庫轉換成 XML 的方法

Lee 等學者提出 NeT 演算法,用來將關聯式資料庫轉換成 XML 的演算法[2]。先取得關聯式資料庫中一部份的關係式模組後，利用 Nest Operator 的方式將其轉換成具有階層關係的 XML 文件。Nest Operator 的方式是將一個資料表中的某個或是數個欄位視為一個群組之後，再利用 Flat Translation(FT)的方式來轉換[3]。FT 是一種在結構相近的資料叢集間一對一轉換的方法，此演算法中用來將利用子元素群組後的資料表轉換成平行的 XML 文件。

首先從關聯式資料庫中取得一個資料表(T)(表 2.1)，有 A、B、C 三個欄位。當將表 2.1 中的欄位 A 視為一群組時，可得到如圖 1(a)所示，欄位內容中{1,2,3}在欄

表 2.1 原始資料表

	A	B	C
#1	1	a	10
#2	1	a	20
#3	2	a	10
#4	3	a	10
#5	4	b	10
#6	5	b	20

位 B 中的值都為 a，欄位 C 中的值都為 10 的 $nest_A(T)$ 。同理視 B 與 C 為一群組時，

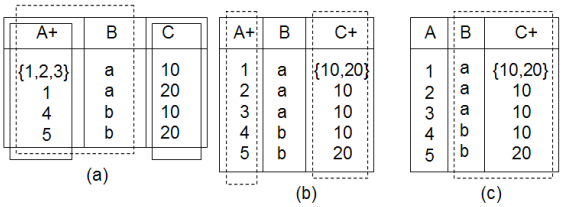

圖 2: 以兩個欄位為群組範例

可得到如圖 1(b)的

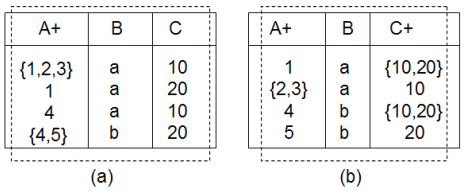

圖 3: 以三個欄位為群組範例

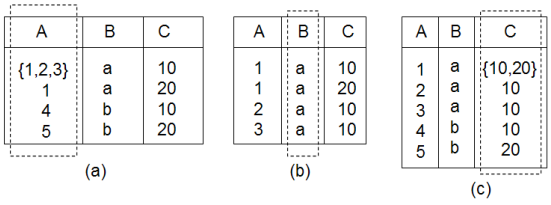

圖 1: 以一個欄位為群組範例

$nest_B(T)$

與圖 1-(c)的

$nest_C(T)$

反覆群組的動作，並逐漸增加欄位的數量再合併轉換，可得到如圖 2(b)的

$nest_A(nest_C(T))$

與圖 2(c)的

$nest_B(nest_C(T))$

最後進行以三個欄位為群組的轉換，如圖 3

所示，可以得到圖 3(a)的

$$nest_B(nest_C(nest_A(T))) = nest_C(nest_B(nest_A(T)))$$

以及圖 3(b)中的

$$nest_B(nest_A(nest_C(T))) = nest_A(nest_B(nest_C(T)))$$

等階層構造。之後再將其轉換成 XML 文件。由於此演算法是取關聯式資料庫中某一個資料表來轉換，因此一次只能轉換一個資料表為其缺點。

2.3 將 XML 文件儲存於關聯式資料庫中的方法

Liu 等學者提出以 XML 文件可轉換成樹狀結構為依據，將其儲存於關聯式資料庫中 [5] 。首先如圖 4 所示，將 XML 文件轉換成樹狀結構，再將各個元素設定其對應的 ID，如表 2.2，再將各條路徑上依照經過的路徑的節點 ID 與資料內容、型態等記錄下來，可以得到如表 2.3 的對應表。例如在圖 4 的 XML 樹中的第三條路徑：//Warehouses/Warehouse/Size/Width=”1015” 在表 2.3 中的路徑是 1、2、4、6，值是 1015，從表 2.2 中可以得到在該條路徑上的父子關係，例如 ID 為 4 的元素 Size 擁有 ID 為 2 的父節點 Warehouse 與 ID 為 6 的子節點 Width，Position 表示了該路徑在其父節點的順序，如方才的例子中 Size 為其父節點的第二個子節點，Value 與 Type 則記錄了各路徑底端的數值與型態，Type 中的 ele 表示 element，att 則表示 attribute。

表 2.2 節點與 ID 對應表

Label	SignatureID
Warehouses	1
Warehouse	2
Name	3
Size	4
Length	5
Width	6
Height	7
Code	8

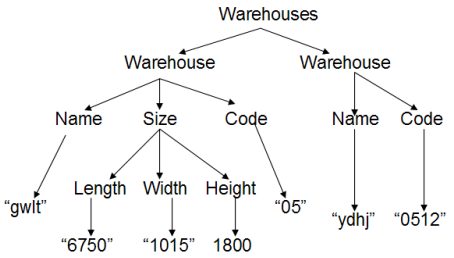

圖 4: XML Tree

表 2.3 節點與路徑對應表

DocID	Path	Position	Values	Type
1	1,2,3	1,1,1	gwlt	ele
1	1,2,4,5	1,1,2,1	6750	att
1	1,2,4,6	1,1,2,2	1015	att
1	1,2,4,7	1,1,2,3	1800	att
1	1,2,8	1,1,3	05	ele
1	1,2,3	1,2,1	ydhj	ele
1	1,2,8	1,2,2	0512	ele

2.4 XML Schema

XML Schema(XML Schema Definition，XSD)由微軟在 2000 年 2 月向 W3C 提出，並於 2001 年 5 月被正式列入建議的規格標準，是用來驗證 XML 文件的語法 [10]。XML Schema 能夠表達出一個 XML 文件必須遵守的規定，與 XML DTD(Document Type Definition)最大的不同在於 DTD 擁有自己的格式，在驗證 DTD 本身的合法性時需要另外處理。而 XML Schema 則本身即為 XML 格式，使用者無需再另外學習，除了擁有 XML 本身的自我描述性以及可延伸性，以及支援命名空間的功能之外，許多現有的 XML 編輯工具及 API 等也可直接套用，主要可分為命名空間(namespace)、架構(structure)以及資料型態(datatypes)三個部分。

1. 命名空間(namespace)：命名空間的目的在於讓一份 XML 文件能與其他 XML 文件擁有相同的定義，相同的定義通常為元素或是資料型態上的定義，大多以 URL 的方式來呈現。一份 XML 文件可以允許同時擁有數個命名空間，通常位於該文件的根標籤(root)中。命名空間並非必要，但在該文件流通時可保持語意的正確性。
2. 架構(structure)：XML Schema 本身為

XML 文件，因此同樣可轉換為樹狀結構，最上層即為根標籤，其下接著子元素，子元素以<element>來定義，element 標籤中會有內容、資料型態等屬性的定義。

3. 資料型態(datatypes)：XML 文件內建了四十種以上常用的資料型態，如 int 或是 string 等，但使用者往往會需要更複雜的資料型態，因此 XML Schema 提供了兩種自訂資料型態的方法：
   A. simpleType(簡單型態)：使用 XML 內建的型態，或是其他經由繼承或是定義 XML 內建的型態而成的資料型態，可透過<restriction>、<list>或是<union>標籤來定義。
   B. complexType(複雜型態)：可以定義包含結構在內的複雜型態，例如當元素中含有子元素時，此時除了該元素的定義外，內部子元素也需要定義，可利用“minOccurs”與“maxOccurs”來代表最小出現次數與最大出現次數。

```
for $book in doc(“text.xml”)//booklist/book
return if ($book/@code = “F4147”)
       then <ntitle>{data($book/title)}</ntitle>
       else $book/title
```

圖 5：XQuery 範例

2.5 XQuery

XQuery 是專門用來對 XML 文件做查詢並從其中提取元素及屬性的語言，就如同關聯式資料庫有 SQL 作為查詢語言。XQury 於 2007 年已成為 W3C 的推薦標準，目前大多數資料庫已支援。 XQuery 由 XPath 路徑運算式、FLWOR 運算式及 XQuery 函數組成。

如在圖 5 中的 XQuery 範例，首先載入 test.xml 文件，選取 booklist 元素之下所有的 book 子元素，並指定給變數$book，回傳時增加條件為該 book 元素中的 code 屬性值為 F4147 時，且將該元素底下的 title 元素傳回並重新命名為 ntitle，否則就直接傳回 title 元素。FLWOR 運算式為 for、let、where、order by 與 return 指令的總稱，功用類似 SQL 中的 SELECT 與 WHERE 等指令。XPath 用來表示搜尋的目標路徑，如上例中表示 booklist 下的 book 元素。XQuery 函數為 XQuery 中內定的常用功能函數，如 data 函數用來傳回參數指定的節點值。

2.6 異質性資料庫間的查詢

Kozlova 等學者提出了 SQXML 的處理需求的系統架構[4]，先由使用者提出查詢(SQL 或 XQuery)，應用程式端接收後，將查詢分割為區域查詢(local query)，區域查詢再對各自的區域資料端(SQL DB 或 XML DB)做查詢，查詢之後再將得到的結果依照語意來映射(mapping)，整合成原始查詢的對應類型文件送回給應用程式端，使用者即可得到由異質性資料來源而來的查詢結果。

由於查詢的資料端不同，可能產生同樣性質的資料有不同的名稱的狀況，例如同樣為書籍清單的資料，書名在 A 資料庫中存為 title，在 B 資料庫可能存為 booktitle，此時就建立一項“title = booktitle”的規則，將映射規則建立完成後，就可依此來整合不同類型的查詢得到的資料。

3.研究方法

為了將關聯式 schema 轉換成 XML Schema 並保持結構的一致性，我們使用資料表之間的主鍵(primary key)與外來鍵(foreign key)的關聯，來定義轉換後的 schema 中父階層與子階層元素之間的關係。關聯式資料結構轉換成 XML 結構對應的原理是以 XML Schema 的 complexType 型態來表示每一個關聯式資料表的結構，而提供外來鍵的關聯式資料表則成為提供主鍵的資料表在轉換之後的子元素。主要的轉換階段可以分為以下幾個步驟：

1. 建立關聯式資料庫的後設資料(metadata)以及資料表的關聯順序。
2. 轉換關聯式架構成 XML Schema

以上各步驟的說明如下列章節所述。

3.1. 建立後設資料

此階段的工作為收集各資料表的資訊，並建立屬於該資料表的後設資料。有一資料庫 D 包含 *n* 個資料表：

$$D: \{R_1, R_2, \ldots, R_n\}, n \geq 1$$

此資料庫的後設資料為：

$$M_D: \{M_1, M_2, \ldots, M_n\}, n \geq 1$$

每一資料表 R_i 的後設資料包含以下資訊：

M_i: (資料表名稱，{欄位名稱，型態，類別，關聯資料表})

每一資料表包含其名稱以及一個以上的欄位的資訊。每個欄位除了包含了其名稱、資料型態外，並記錄欄位的類別及相關聯的資料表。欄位的「類別」可以是主鍵(PK)、外來鍵(FK)、或是一般欄位(Regular)。若類別是主鍵，則「關聯資料表」紀錄了以此欄位為外來鍵的資料表名稱，或者是 null。若類別是外來鍵，則「關聯資料表」紀錄以此欄位為主鍵的資料表名稱。最後，如果類別是一般欄位，「關聯資料表」的值則為 null。

由此後設資料，我們可以建立資料表間的關聯順序。資料表的關聯順序以主鍵及外來鍵的關聯來決定，若資料表 R_1 的主鍵是資料表 R_2 的外來鍵，我們以 $R_1 \rightarrow R_2$ 表是其關聯順序。對來源資料庫我們假設其資料表之間沒有循環關聯，也就是資料表間的關聯順序沒有迴圈的存在。例如以下 R_1、R_2、及 R_3 等三個資料表間就存在循環關聯，因為這三個資料表的關聯順序會造成迴圈，如圖 6 所示。

R_1: (a1, a2, …, c1)
R_2: (b1, b2, …, a1)
R_3: (c1, c2, …, b1)

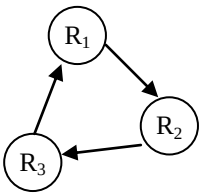

圖 6: 循環關聯示意圖

```
Customer                table name: Customer          id: 0
CNo,int,PK              ------------------------
Cname,char[20],N/A      CNo, int, Primary Key
Caddress,char[40],N/A   Cname, char[20], Regular Column
***                     Caddress, char[40], Regular Column
Order
ONo,int,PK              (PrimaryKey)CNo join Order - CNo
CNo,int,FK              ================================
Odate,char[10],N/A      table name: Order     id: 1
Edata,char[20],N/A      ------------------------
***                     ONo, int, Primary Key
ODetail                 CNo, int, Foreign Key
ONo,int,FK              Odate, char[10], Regular Column
UnitPrice,int,N/A       Edata, char[20], Regular Column
Quantity,int,N/A
                        (ForeignKey)CNo join Customer - CNo
                        (PrimaryKey)ONo join ODetail - ONo
                        ================================
                        table name: ODetail  id: 2
                        ------------------------
                        ONo, int, Foreign Key
                        UnitPrice, int, Regular Column
                        Quantity, int, Regular Column

                        (ForeignKey)ONo join Order - ONo
                        ================================
```

圖 7: 關聯式架構與對應的後設資料

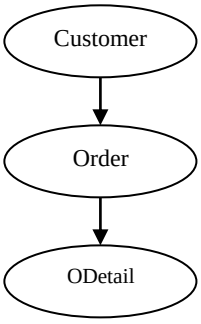

圖 8: 架構 DAG 示意圖

這個假設主要是避免在轉換成 XML Schema 時無法決定何者為其根元素(root element)，而在實作上若關聯式資料表間存在循環關聯時，也可以加入一資料表以打破此循環的關係。

讀取關聯式 schema 的內容後，在本階段首先建立每一個資料表的後設資料，分析各資料表的主鍵與外來鍵後，將資料庫中所有資料表的關聯以一有向非循環圖(DAG, Directed Acyclic Graph)來表示其間的關係。例如，圖 7 所示的關聯式架構與其對應的後設資料。藉由分析主鍵及外來鍵，資料表之間的關聯則可以用圖 8 的有向非循環圖來表示。

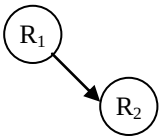

圖 9: 一對一資料表參照示意圖

```
<xs:element name ="R1" type = R1/>
                ...
<xs:complexType name="R1">
  <xs:choice minOccurs="1" maxOccurs="unbounded">
    <xs:element name="tid" type="xs:int" />
    <xs:element name="tdata" type="xs:string" />
      <xs:element name="R2_complex">
        <xs:complexType>
        <xs:choice>
        <xs:element minOccurs="1" maxOccurs="unbounded"
          name="R2" type="R2" />
        </xs:choice>
        </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>
```

圖 10: 一對一資料表參照範例

3.2. 轉換方法

關聯式 schema 經由建立後設資料並產生 DAG 後，接著利用 DAG 來進行 XML Schema 的轉換。

由於每個資料表中都含有多筆欄位資料，所以設定成 complexType，complexType 子元素使用 choice，設定 minOccurs 為 1，maxOccurs 則為 unbounded，用來表示任意數量的子元素。最後不直接將資料表轉換成元素，而是設定以各資料表為名稱的型態(type)，藉由設定元素的型態來連結資料表元素與資料，可以有效的減少參照關係中部分 schema 片段容易重複出現的狀況。以下就各種轉換時產生的情況敘述：

1. 一對一資料表參照：

如圖 9 所示，資料表 R_1 與資料表 R_2 是屬於一對多的關係，例如一個部門下會有很多位員工，我們將有參照關係的資料表(R_2)轉換成被參照的資料表(R_1)中的元

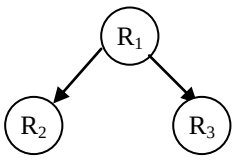

圖 11: 多對一資料表參照示意圖

```
<xs:element name ="R1" type = R1/>
                ...
<xs:complexType name="R1">
  <xs:choice minOccurs="1" maxOccurs="unbounded">
    <xs:element name="tid" type="xs:int" />
    <xs:element name="tdata" type="xs:string" />
      <xs:element name="R2_complex">
        <xs:complexType>
        <xs:choice>
        <xs:element minOccurs="1" maxOccurs="unbounded"
          name="R2" type="R2" />
        </xs:choice>
        </xs:complexType>
    </xs:element>
      <xs:element name="R3_complex">
        <xs:complexType>
        <xs:choice>
        <xs:element minOccurs="1" maxOccurs="unbounded"
          name="R3" type="R3" />
        </xs:choice>
        </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>
```

圖 12: 多對一資料表參照範例

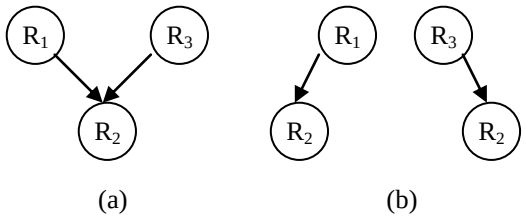

圖 13: 一對多資料表參照及轉換示意圖

```
<xs:element name ="R1" type = R1/>
<xs:element name ="R3" type = R3/>
                ...
<xs:complexType name="R1">
  <xs:choice minOccurs="1" maxOccurs="unbounded">
    <xs:element name="tid" type="xs:int" />
    <xs:element name="tdata" type="xs:string" />
      <xs:element name="R2_complex">
        <xs:complexType>
        <xs:choice>
        <xs:element minOccurs="1" maxOccurs="unbounded"
          name="R2" type="R2" />
        </xs:choice>
        </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>
<xs:complexType name="R3">
  <xs:choice minOccurs="1" maxOccurs="unbounded">
    <xs:element name="sid" type="xs:int" />
    <xs:element name="sdata" type="xs:string" />
      <xs:element name="R2_complex">
        <xs:complexType>
        <xs:choice>
        <xs:element minOccurs="1" maxOccurs="unbounded"
          name="R2" type="R2" />
        </xs:choice>
        </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>
```

圖 14: 一對多資料表參照範例

素，並將資料型態設定為 complexType，最後再設定型態加以連結。如圖 10 的範例中，資料表 R_2 有參照資料表 R_1 的關係。

2. 多對一資料表參照：

如圖 11 所示，資料表 R_2 及 R_3 同時參照到資料表 R_1，表示 R_2 與 R_3 屬於同一階層，轉換後都為 R_1 的子元素，型態皆為 complexType，如圖 12 的範例所示。

3. 一對多資料表參照：

當一個資料表參照到多個資料表時，如圖 13(a)所示，R_1 與 R_3 同時擁有 R_2 子元素，代表 R_2 同時參照 R_1 和 R_3。由於 XML 格式為一個樹狀結構，因此我們將此種關聯視為兩條不同的路徑，也就是將其視為兩個單一資料表參照的情況來處理，如圖 13(b)所示。圖 14 的範例可看到 R_1 與 R_3 中都有元素 R_2。

圖 15 為用來產生 schema 的 BFS 演算法(Breadth-First-Search, 寬度優先搜尋)，首先第 2 行產生一虛擬根節點 root，該節點指向 DAG 中的所有沒有 incoming edge 的節點，接著產生對應 root 的 XML 元素並

Generate XML document
Input: DAG G:(V, E)
Output: XML document
1. Initialize queue Q
2. Create a virtual root v_{root} for G
3. Create XML element E_{root} corresponding to v_{root}
4. Q.Enqueue(v_{root})
5. while (Q is not empty)
6. $v_{head} \leftarrow$ head of Q
7. $E_{head} \leftarrow v_{head}$'s corresponding XML element
8. foreach $v_i \in \{v_{child} | v_{head} \rightarrow v_{child}, v_{child}$ is the direct child of $v_{head}\}$
9. Insert E_i as sub-element into E_{head}, where E_i is the corresponding XML element of v_i
10. Q.Enqueue(v_i)
11. end foreach
12. Q.Dequeue()
13. end while

圖 15：產生 schema 演算法

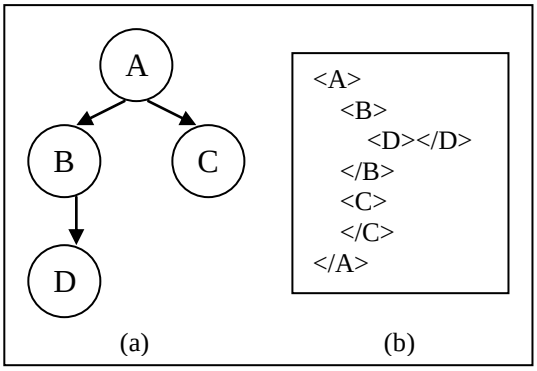

圖 16: 演算法轉換結果範例

將 root 放入佇列 Q 中，此元素將作為 XML schema 最外層的元素。第 5 行開始為 BFS 演算法的遍歷過程。v_{head} 為 Q 中的第一個元素，E_{head} 為對應 v_{head} 的 XML schema 元素，而 v_{child} 則是所有與 v_{head} 有直接的父子關係的節點的集合，由於每個節點皆代表一個資料表，第 8 行開始的 foreach 迴圈對所有屬於 v_{child} 的 v_i，將其對應的 XML 元素 E_i 加入 schema 中作為 E_{head} 的子元素，然後將 v_i 放入佇列 Q 中。當 v_{head} 的所有子節點放入 Q 後，將其從 Q 中移除，代表完成該節點的遍歷過程，依序遍歷所有節點後，即產生 XML Schema。如圖 17 的範例中，圖 16(a)的 DAG 結構將產生如圖 16(b)結構的 XML Schema。

4. 實驗結果

本次研究是以微軟 WindowsXP 作業系統為平台開發，使用微軟的 Visual Studio 2008 與 C#做為開發系統及程式語言。

使用者經由使用圖 17 的介面，選擇要輸入的關聯式架構資料檔案。關聯式架構的範例如圖 18 所示，該架構中有 7 個資料

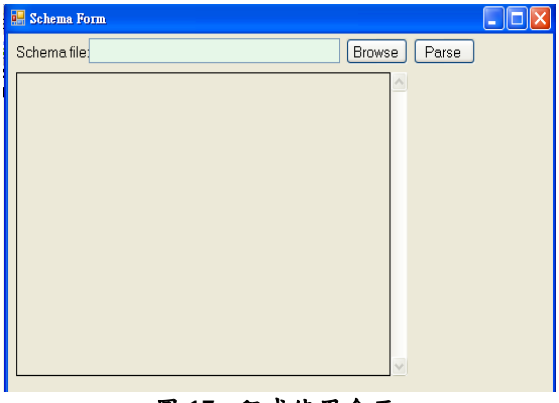

圖 17: 程式使用介面

Order
OrderNo,int,PK
EmpNo,int,FK
CustomerNo,inf,FK
OrderDate,char[20],N/A
PaymentMethod,char[20],N/A
DeliveryMethod,char[20],N/A
***
Order_detail
OrderNo,int,FK
ProductNo,int,FK
UnitPrice,int,N/A
Quantity,int,N/A
***
Employee
EmpNo,int,PK
Name,char[20],N/A
Title,char[40],N/A
Gender,char[10],N/A
Address,char[40],N/A
ExtNo,char[10],N/A,
***
Customer
CustomerNo,int,PK
CompanyName,char[40],N/A
Contact,char[40],N/A
ContactTitle,char[20],N/A
ContactGender,char[10],N/A
ZipCode,char[10],N/A
Address,char[40],N/A
PhoneNo,char[20],N/A
***
Product
ProductNo,int,PK
CategoryNo,int,FK
ProviderNo,int,FK
Name,char[40],N/A
ListPrice,int,N/A
Stock,int,N/A
SafetyStock,int,N/A
***
Provider
ProviderNo,int,PK
Name,char[40],N/A
Contact,char[40],N/A
ContactTitle,char[20],N/A
ContactGender,char[10],N/A
ZipCode,char[10],N/A
Address,char[40],N/A
PhoneNo,char[20],N/A
***
Product_Category
CategoryNo,int,PK
CategoryName,char[20],N/A

圖 18: 原始的關聯式架構

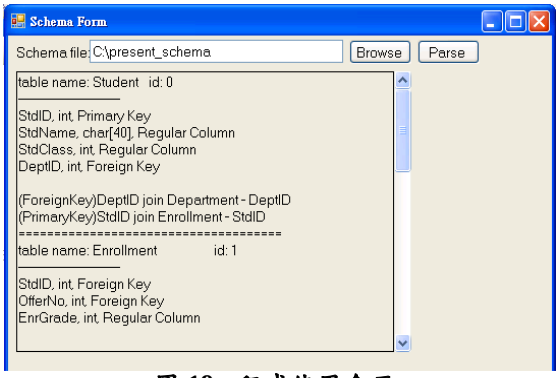

圖 19: 程式使用介面

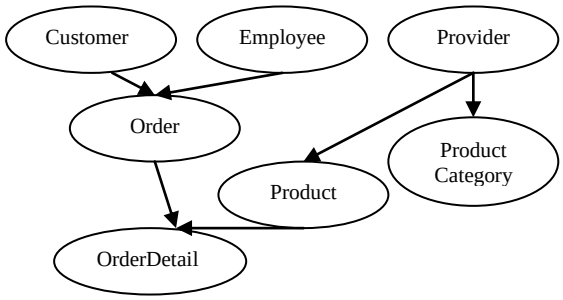

圖 20: DAG 示意圖

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="present">
  <xs:complexType>
   <xs:choice minOccurs="1" maxOccurs="unbounded">
    <xs:element name="Employee" type="Employee" />
    <xs:element name="Customer" type="Customer" />
    <xs:element name="Provider" type="Provider" />
    <xs:element name="Product_Category" type="Product_Category" />
   </xs:choice>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="Order">
  <xs:choice minOccurs="1" maxOccurs="unbounded">
   <xs:element name="OrderNo" type="xs:int" />
   <xs:element name="OrderDate" type="xs:string" />
   <xs:element name="PaymentMethod" type="xs:string" />
   <xs:element name="DeliveryMethod" type="xs:string" />
   <xs:element name="Order_detail_complex">
    <xs:complexType>
     <xs:choice>
      <xs:element minOccurs="1" maxOccurs="unbounded"
          name="Order_detail" type="Order_detail" />
     </xs:choice>
    </xs:complexType>
   </xs:element>
  </xs:choice>
 </xs:complexType>
 <xs:complexType name="Order_detail">
  <xs:choice minOccurs="1" maxOccurs="unbounded">
   <xs:element name="UnitPrice" type="xs:int" />
   <xs:element name="Quantity" type="xs:int" />
  </xs:choice>
 </xs:complexType>
```

圖 21: 轉換完成的 XML schema 片段

表，每個欄位資料後面紀錄了該欄位的資料型態與參照關係。在讀入關聯式架構資料後，將會依各欄位中顯示的參照關係在各個資料表間建立連結來產生如圖 19 所示的後設資料。每個資料表後會紀錄該資料表是否與其他資料表有參照關係，並標明該資料表在關係中是屬於 PK 方或是 FK 方。此後設資料可表示出一個如圖 20 的 DAG 架構。圖 21 為最後轉換完成的 XML Schema 片段，可看到由最外層由根元素構成，之後依照型態的宣告，形成階層關係。

5.結論與未來展望

本論文的研究制定了一套機制，將關聯式的資料架構轉換成相符結構的 XML schema，並且維持了原有關聯式的架構以及各欄位的形態。這個轉換的機制有助於網路上異質的資料庫之間，以標準的 XML 格式交換資料，並以單一的格式來整合異質的資料，能夠有效地降低查詢的複雜度。我們以應用層面較為廣泛的關聯式資

料為例，實作了一個有效率的轉換系統，在多次實驗中均能正確將關聯式的架構轉換成 XML 架構。

然而除了關聯式資料外，尚有許多不同的資料結構需要整合。網路上隨處可見的各種文件已逐漸成為主要的資料來源，例如常見的 MS Word 文件(doc)、e-mail、部落格文章、討論區文章、pdfs、或是各種 odf(OpenDocumentFormat)文件等，各擁有不同程度結構嚴謹性，都適合以 XML 文件來描述，因此未來的發展朝向實現一個單一介面整合各種不同資料來源，以利使用者能夠查詢異質性的資料來源。

參考文獻

[1] Bart Steegmans, Ronald Bourret, Owen Cline, Olivier Guyennet, Shrinivas Kulkarni, Atephen Priestly, Valeriy Sylenko, Ueli Wahli, *XML for DB2 Information Integration*, http://www.redbooks.ibm.com/, 2004, pp 11-21.

[2] D. Lee, M. Mani, and W. W. Chu, *Schema conversion methods between XML and relational models*, Knowledge Transformation for the Semantic Web(2003), 2003, pp.1-17.

[3] Dongwon Lee, and Frank Chiu, and Murali Mani, Wesley W. Chu, *Nesting-based Relational-to-XML Schema Translation*, International Workshop on the Web and Databases, Santa Barbara, California (2001).

[4] Iryna Kozlova, and Norbert Ritter，and Olga Reimer, *Towards Integrated Query Processing for Object-Relational and XML Data Sources*, 10th International Database Engineering and Applications Symposium, 2006, pp.295-300.

[5] Liu Sanian, and Liu Caifeng, and Guan Liming, *A Storage Method for XML Document based on Relational Database*, 2008 International Symposium on Computer Science and Computational Technology, Shanghai, China, 2008, pp.50-53.

[6] Oracle Database 11*g* Release 2, http://www.oracle.com/us/products/dat–abase/index.html.

[7] Ronald Bourret, XML and Databases, http://www.rpbourret.com/index.htm.

[8] TEXTML server, http://www.ixiasoft.com/default.asp?xml=/xmldocs/webpa–ges/textml-server.xml.

[9] Tamino The XML Database, http://www.softwareag.com/corporate/products/wm/tamino/default.asp.

[10] W3C, http://www.w3.org/.