在 XML 資料倉儲中實體化資料方體選取之研究

邱紹豐　　李義偉
大葉大學資訊工程學系
r9206026@mail.dyu.edu.tw

摘要

為了能夠充分運用資料庫中的資料，且加快資料的查詢速度及協助經營的決策方向，因此採用資料倉儲的方式來解決。隨著網際網路的興起，許多的資料都是藉由網路傳遞，而在不同的平台或不同的系統之間交換資料方體（Data Cube）時需使用中介軟體來轉換。為了解決資料方體交換的問題，本文提出使用 XML（eXtensible Markup Language）來建立資料倉儲，並加入 Lattice 架構，在儲存空間有限的情況下，加快資料方體（Data Cube）的查詢速度。

關鍵詞：Lattice，資料方體，XML。

1. 前言

當資料庫中的資料尚未經過處理（processed）時，查詢這樣的資料庫通常會花費不少時間。為了加快資料查詢的速度，且充份的利用這些資料，都會建立資料倉儲。建立好資料倉儲後，除了加快資料的查詢外，同時可分析資料倉儲中的資料，得到重要的資訊。

如果每公司間彼此要交換資料倉儲中的資料方體，而公司又各自使用不同的資料庫，就會有異質性資料庫間相容的問題。而 XML 使用純文字編寫，能夠直接存取，因此擁有跨平台、跨系統的特性。故本文使用 XML 建立資料倉儲，提出 XML 資料倉儲的架構，用以解決資料方體交換時所遇到的問題。

在儲存空間允許的情況下，資料倉儲中所有的資料方體都可實體化（Materialize），此時的資料查詢速度最快。但在儲存空間有限的情形下，並非每個資料方體都能實體化。因此，如何在有限的儲存空間下，挑選出資料方體實體化，以達到較佳的查詢速度是十分重要的。故本文在 XML 資料倉儲中加入 Lattice 架構，表示資料方體與資料方體間的關係，並提出在 Lattice 架構中查詢資料方體的演算法，在儲存空間有限的情況下，善用儲存空間以加速資料方體的查詢速度。

本論文第二章介紹相關的研究資料，第三章說明建立 XML 資料倉儲系統的流程以及建立 XML 資料倉儲的詳細過程與演算法，第四章結論與未來的發展方向。

2. 相關研究

XML 為 W3C 組織所提出的標記語言，被廣泛的應用在網際網路上，因為是純文字檔的特性，故能夠跨平台無需經過中介軟體的轉換就可直接讀取，故成為網際網路上普遍的資料交換格式[1]。

將整合過的資料視為一個多維度的資料方體，如圖 1 左邊所示為存放銷售資料之資料方體，其包含了時間、產品及地區，資料方體中的每格，被視為單元（Cell），表示在某個時間點某種產品於某個地區的銷售資料；在此例子中每個單元存放銷售數量。藉由資料方體，從不同的角度可得到不同的資訊；從產品的角度來看可得知產品的銷售情形，從時間的角度來看可得知某段時間的銷售情況，從地區的角度來看可得知某些地區的銷售情況；並且能對資料方體使用向上提升（roll-up）與深入探測（drill-down）以得到不同的資訊，如圖 1 右邊表示由縣市向上提升到區域的結果。

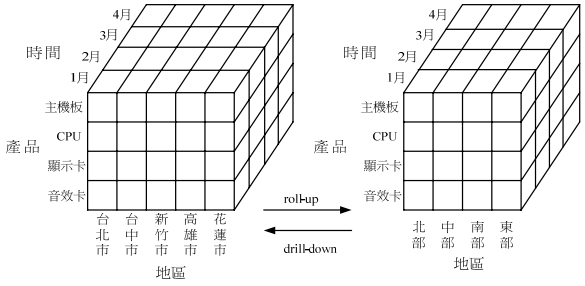

圖 1 資料方體

以往常使用關聯式資料庫等發展成熟的資庫料庫系統建立資料倉儲，而 Wolfgang Hümmer 等學者則提出 XCube[2]。XCube 是利用 XML 所建立的資料倉儲架構，一共包含了三份文件，分別為 XCubeSchema 、 XCubeDimension 、 XCubeFact 。 XCubeSchema 包含了整個 XML 資料倉儲的架構，包含了每個資料方體的維度（Dimension）和事實（Fact），XCubeDimension 則包含了在每個維度中值（value）的階層關係（hierarchies），而 XCubeFact 包含了每個資料方體中事實（Fact）的值。

XCubeSchema 其 Schema 如圖 2 所示，“*”表示此節點（element）於 XML 文件中出現次數最少為 0 次，最多不限；在<cubeSchema>存放資料方體的相關資訊，而<classSchema>存放維度的階層關

係，利用<classLevel>就能實現資料倉儲中向上提升的功能，例如月向上提升到季。

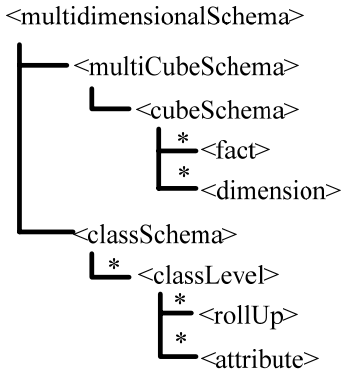

圖 2 XCubeSchema

XCubeDimension 包含了每個維度的值之階層關係。其 Schema 如圖 3 所示，利用此 Schema 就可描述在維度中值的階層關系，例如由一月份向上提升到第一季。

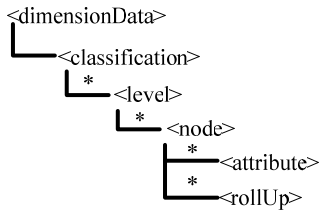

圖 3 XCubeDimension

XCubeFact 包含每個資料方體維度和事實資料表的值，其 Schama 如圖 4 所示。

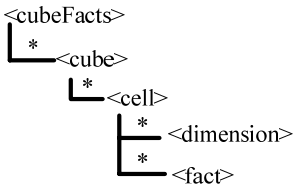

圖 4 XCubeFact

Laura Irina Rusu 等學者在[3]提出建立 XML 資料倉儲時並沒有自訂出 XML 資料倉儲的結構，而是利用大量的 XQuery[4]的語法把 XML 原始文件中需要資料抽取出來，存成多份 XML 文件；一個維度存成一份文件，有多個維度則有多份維度文件；另一份事實文件（fact document）存放事實資料，於事實文件與維度文件間使用鍵值(key)連結，流程如圖 5 所示。此做法相當簡單，但使用 XQuery 於大量的資料時會花費相當多的時間。

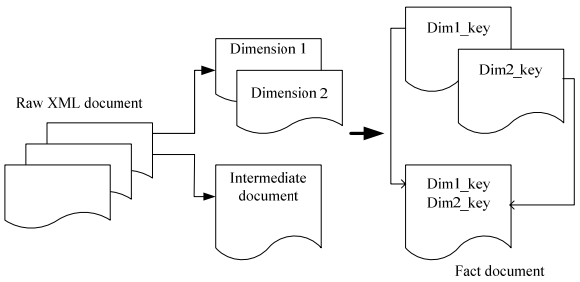

圖 5 XML 資料倉儲建立示意圖

Stelios Paparizos 等學者在[5]則提到因為 XML 文件允許某些元素（Element）不存在，架構十分靈活，不像關聯式資料庫的格式固定。故提出分析 XQuery 的語法，將之轉成 Pattern Tree，使用 Pattern Tree 去比對 XML 文件的架構，與 Pattern Tree 相同架構的資料即是查詢結果，且利用此方法於 XML 文件的群組（Group by）上。

Lattice 為一圖形架構，用以表示資料方體間的父子關係[6]。如圖 6 為一 Lattice 架構，由顧客(C)、零件(P)、供應商(S)所構成的三維的 Lattice 架構，每個節點表示一個資料方體；資料方體旁，括號內的數字表示此資料方體之大小（size），和其查詢頻率，0 表最小，1 表示最大；none 表示原始資料。

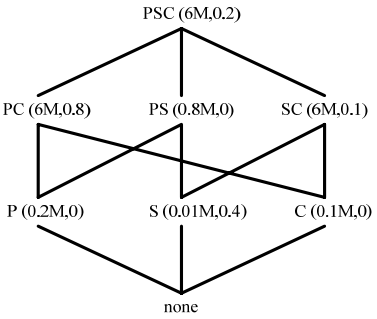

圖 6 Lattice 架構

在此結構中，若使用者要查詢包含零件和供應商兩個維度的資料方體時，查詢 PS 這個資料方體會比查詢 PSC 還要快，利用這種方式來加快查速度[6]。在空間允許的情況下，Lattice 中的所有資料方體都可以實體化；其查詢速度最快，幾乎沒有必要查詢到原始的資料庫，但在儲存空間有限的情況下，部份的資料方體能實體化，此時的查詢速度會降低，且因查詢不到資料轉而查詢原始資料庫的機率大增。

而在儲存空間有限的情況下，挑選哪些資料方體來實體化成為研究的重點。這個挑選資料方體的

問題已被証實為 NP 問題（NP-Complete），故很難得到一個最佳解[6]。Venky Harinarayan 等學者提出一演算法計算每個資料方體之效益值，挑選效益值最高之資料方體實體化，在此演算法中 Lattice 架構中最頂端資料方體一定要實體化[6]。林志麟等學者在[7]提出反向的貪婪演算法，其特性正好與正向貪婪演算法相反，且 Lattice 架構中頂端資料方體可不實體化。陳耀輝等學者提出二階段演算法，第一階段以節省儲存空間為目標，對實體化的資料方體做刪選；第二階段則以提升資料方體的查詢效能為目標，增加或刪除資料方體[8]。

3. 系統流程

在建立 XML 資料倉儲時，使用本文所提出的公式預估每個資料方體的大小。依照資料方體的大小和查詢頻率計算每個資料方體的增益值（Benefit）；其原理是以查詢新加入之資料方體時比查舊資料方體少掉的資料量為計算方式。少掉的資料量愈多其增益值愈高，查詢頻率愈高的資料方體增益值也愈高。將實體化的資料方體置入已修改過的 XCube 架構。本文先大概說明建立的流程，在下面的章節中再仔細說明其詳細的作法與演算法；其流程如圖 7。

步驟 1：一開始對原始 XML 文件執行群組之前，必須預估每個資料方體的大小；避免因為詳細計算資料方體大小而花費過多的時間。

步驟 2：在儲存空間允許的情況下，所有資料方體都能實體化，但在儲存空間不夠的情況下須執行 XCGA 演算法，挑選出實體化資料方體集合。再利用預估出來之資料方體的大小執行貪婪演算法，在不大於空間限制的情況下挑選出欲實體化之資料方體集合。

步驟 3：依照指定實體化之資料方體對原始 XML 文件執行群組，把群組後的結果存入已支援 Lattice 架構之 XCube，建立 XML 資料倉儲。

當 XML 資料倉儲建立完成後，為了能充份的利用 Lattice 架構，本文利用深先搜尋的方式提出 **XCSA** 演算法以加快資料方體的查詢速度。

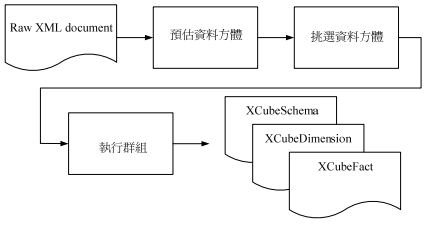

圖 7 系統流程示意圖

3.1 資料方體大小預估

由於之後的 XCGA 演算法須利用到每個資料方體的大小來進行運算，故必須先得到每個資料方體的大小。若對原始的資料執行群組則所求得的資料方體大小最為精確，但如此的做法會花費不少時間。在關聯式資料庫中預估資料方體可用統計學的公式或抽樣等方式預估[9]。在 XML 方面本文提出一公式來預估每個資料方體的大小。一開始先掃描原始 XML 文件，得到每個維度的相異值（distinct value）之數量，再套用公式。其公式如式 1 所示，資料庫容量表示原始 XML 資料庫的大小。因為執行群組後，某些群組（group）並不存在於資料方體中，故須額外除以係數，而係數的求法會使用實驗來求得一個合理的值。

$$\text{資料方體大小} = \frac{\text{原始XML文件大小}}{\text{維度的迪卡爾乘積} \times \text{係數}}$$

式 1 資料方體大小預估公式

例如有一原始 XML 資料庫大小為 100M，維度 p 相異值的數量為 5，維度 s 相異值的數量為 10，係數為 0.1。欲預估包含 p、s 二維的資料方體大小，則計算 p 與 s 的迪卡爾乘積 10×5=50，再將 50×0.1=5，最後結果為 100M/5=20M。

3.2 挑選資料方體實體化之演算法

此演算法是參考 Venky Harinarayan 等學者在[7]提出的貪婪演算法，本文額外加入每個資料方體的查詢率，以提高資料方體的查詢效能。演算法會計算出每個資料方體的增益值，挑選出增益值最大的資料方體來實體化；本文命名為 XCGA 演算法（XML Cube Greedy Algorithm）。其演算法流程如下：

步驟 1：先判斷實體化資料方體集合（MCs）的總大小是否小於空間限制（S），在小於空間限制的情況下挑選資料方體。

步驟 2：將未實體化之資料方體實體化後總大小小於空間限制之資料方體加入候選集合（CandidateSet）中。若候選集合為空時，表示所有資料方體已實體化或其大小大於空間限制，即跳出迴圈，結束演算法。

步驟 3：計算候選集合中每個資料方體的增益值，將增益值最大之資料方體置入實體化集合中。

增益值的計算方式是以實體化某個資料方體後，能夠減少的資料查詢量和查詢率的比例。其演

算法流程如下：

```
▷ 全域變數，實體化資料方體之集合
MCs = null

01    XCGA
02        ▷ 在 MCs 之總大小小於空間限制下挑
03        選資料方體
04        While C(MCs) < S
05            ▷ 將未實體化之方體且其方體實
06            體化後總大小小於 S 加入集合
07            CandidateSet = {x | x ∉ MCs And
08            C(MCs)+C(x)<S}
09            if CandidateSet = null
10                exit while loop
11            Max = 0
12            ▷ 計算每個方體之增益值，將增益
13            值最大之資料方體加入 MCs
14            for each node x ∈ CandidateSet
15                if Benefit(x,MCs) > Max
16                    v = x
17                    Max = Benefit(x)
18            MCs = MCs ∪ { v }
19
20    Benefit(v)
21        Benefit = 0
22        ▷ 將能用 v 回答之子節點包含 v 本身加
23        入集合
24        DependSet = {w | w ∈ v}
25        for each cube w ∈ DependSet
26            ▷ Min 為原始資料之大小
27            Min = C(RawData)
28            for each cube u ∈ MCs
29                if (C(u)<Min And w ∈ u)
30                    z = u
31                    Min = C(z)
32            if C(v) < Min
33                Bw = ( Min – C(v) ) *
34                w.frequency
35            else
36                Bw = 0
37            Benefit = Benefit + Bw
```

圖 8 XCGA 演算法

步驟 1：將欲計算增益值之資料方體和其子資料方體置入相依集合（DependSet）中；表示查詢子資料方體都用此資料方體查詢。

步驟 2：計算相依集合中每個資料方體之增益值；當實體化資料方體集合中沒有任何的資料方體被實體化時，會先計算查詢此資料方體會比查詢原始 XML 文件少掉多少的查詢量再乘上此資料方體的查詢頻率，而查詢其子立方體時，也用相同的計算方式。

步驟 3：當實體化集合中已有資料方體時，查詢新資料方體的資料量會比查詢舊資料方體的資料量少掉的資料量，再乘上資料方體的查詢頻率就是增益值，查詢其子資料方體也用相同的計算方式。

步驟 4：累計增益值且回傳結果。

依照上述演算法流程，演算法如圖 8 所示。例如有一 Lattice 架構如圖 6 所示，其原始 XML 文件大小為 50M，若選擇 PSC 實體化，則 PSC 的增益值為(50-6)*0.2+(50-6)*0.8+(50-6)*0+(50-6)*0.1+(50-6)*0+(50-6)*0.4+(50-6)*0=83.6。

3.3 修改 XCube 架構

確定哪些資料方體要實體化後，需要一個 XML 資料倉儲儲存這些資料方體，且 XML 資料倉儲須能夠支援 Lattice 架構，故本文修改原本 XCube 的架構，使其能夠支援 Lattice 架構。修改後的 XCubeSchema 如圖 9 所示，在<classSchema>中加入新節點<lattice>，其用意表示在 Lattice 中資料方體與資料方體間的父子關係，在<latticeLevel>表示 Lattice 架構中最頂端的資料方體，而<rollUp>的屬性會指向與其有關係之資料方體，<latticeAttribute>則表示此資料方體一些額外的資訊，例如資料方體的大小等。

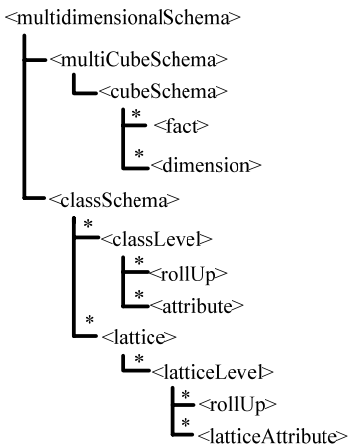

圖 9 修改後之 XCubeSchema

例如有一已實體化之 Lattice 架構如圖 10 所示，

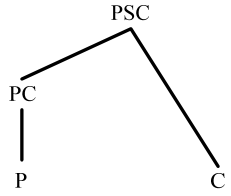

圖 10 已實體化之 Lattice

則在 XCubeSchema 之文件內容如下所示：

```
<multidimensionalSchema>
        .
        .
    <classSchema>
        .
        .
        <lattice>
            <latticeLevel id=“PSC”>
                <rollup toLevelCube=“PC”/>
                <rollup toLevelCube=“C” />
            </latticeLevel>
            <latticeLevel id=“PC” >
                <rollup toLevelCube=“P”/>
            <latticeLevel>
        </lattice>
    </classSchema>
</multidimensionalSchema>
```

3.4 搜尋資料方體之演算法

建立好支援 Lattice 架構的 XML 資料倉儲後，為了充分利用 Lattice 的架構以加快查詢資料的速度，本文提出 XCSA（XML Cube Search Algorithm）演算法來達成此目地。因為在 Lattice 架構中愈下層的資料方體其大小愈小，或與上層的資料方體大小相等，故本文使用深先搜尋的方式搜尋整個 Lattice 架構。演算法流程如下：

```
▷ 全域變數
MinCube ← null
Cubesize ← ∞
01    SelectCube
02        ▷ 判斷頂點之方體是否符合搜尋條件，
03        若符合則搜尋其子方體
04        for each TopCube
05            if u ∈ TopCube
06                if TopCube size < Cubesize
07                    MinCube ← top
08                    cube
09                Cubesize ← top cube
10                size
11                SearchCube(top cube)
12
13    SearchCube(n)
14        ▷ 判斷 n 的子方體是否符合查詢條件
15        for each v ∈ child(n)
16            if v is not mark And u ∈ v
17                If v size < Cubesize
18                    MinCube ← v
19                    Cubesize ← v size
20                SearchCube(v)
21        ▷ 標記 n
22        Mark(n)
```

圖 11 XCSA 演算法

步驟 1：先判斷 Lattice 架構中最頂端的資料方體是否符合使用者的查詢條件，若符合再繼續往下搜尋，落不符合則直接跳出演算法，因為子節點的維度一定會被父節點所包含住。

步驟 2：往下搜尋時，若符合查詢條件則再往下一層搜尋，若不符合則往其第二個子節點搜尋，直到搜尋完整個 Lattice 架構。

步驟 3：搜尋的同時把已經搜尋過且符合查詢條件的節點標記(Mark)起來，避免重覆搜尋。

依照上述演算法流程，演算法如圖 11 所示。例如圖 12 所示為一已實體化之 Lattice 架構，若使用者欲查詢包含維度 S 之資料方體，演算法先判斷 PSC 是否符合使用者查詢條件，若是則往下搜尋，判斷 PC 是否符合使用者查詢，若不是則搜尋 PS，若 PS 符合使用者查詢，則繼續往搜尋，搜尋到 S 則程式結束。

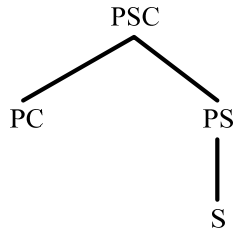

圖 12 已實體化之 Lattice 架構

4. 結論與未來工作

在本文的研究中，提出修改原本的 XCube 使其能夠支援 Lattice 架構，並提出 XCSA 演算法查詢 Lattice 架構中的資料方體；同時提出 XCGA 演算法，在有限的儲存空間下挑選實體化之資料方體，在儲存空間與查詢時間上達到較佳的平衡點。至今為止本文只提出一種理論，而未來會實作系統以驗證理論。

另一方面在 Lattice 架構中查詢資料方體的演算法上，若欲查詢的資料方體不包含在 Lattice 架構中，會轉而查詢原始的 XML 文件。未來將思考是否有其他方法能避免查詢原始的 XML 文件以加快查詢速度，例如將 Lattice 架構中某幾個資料方體執行交集（Join）以回答使用者的查詢。

參考文獻

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

[2] Andreas Bauer, Gunnar Harde and Wolfgang Hü

mmer, “XCube – XML For Data Warehouses” The 6th ACM international workshop on Data warehousing and OLAP., pp.33-40, 2003.

[3] David Taniar, J. Wenny Rahayu and Laura Irina Rusu, “A Methodology for Building XML Data Warehouses” IDEAL 2004., LNCS 3117, pp.3293-299, 2004.

[4] XQuery, http://www.w3.org/XML/Query/

[5] Andrew Nierman, Divesh Srivastava, H.V. Jagadish, Laks Lakshmanan, shurug Al-Khalifa, Stelios Paparizos and Yuqing Wu, “Grouping in XML” EDBT 2002 Workshops., LNCS 2409, pp.128-147, 2002.

[6] Anand Rajaraman, Jeffrey D. Ullman and Venky Harinarayan, “Implementing Data Cubes Efficiently” SIGMOD Record, 25:2, pp.205-227, 1996.

[7] 林志麟與邱承凡, “資料倉儲實體化視域選取之研究 – 以資料方體之建置為例” 2000.

[8] 陳耀輝與劉宇昌,”在資料倉儲中針對查詢選擇實體化視域之研究”, 1997.

[9] Amit Shukla, Jeffrey F. Naughton, Karthikeyan Ramasamy, Kristin Tufte, Prasad Deshpande and Yihong Zhao, “Cubing Algorithms, Storage Estimation, and Storage and Processing Alternatives for OLAP” IEEE Data Eng.Bull.20(1). pp.3-11.