

第十章

Transact-SQL 在 XML 方面的支援





本章內容

- 10.1 Transact-SQL 的 For XML 子句
- 10.2 Openxml 函數之應用
- 10.3 Transact-SQL 所支援的 XQuery
- 10.4 XML 與網路服務 (Web Service)



SQL Server 2008 的 XML 資料型態

- SQL Server 2008 已支援 XML 資料型態
 - 使用者可宣告 Transact-SQL 變數為 XML 資料型態
 - 將某個關聯表屬性宣告成 XML 資料型態
 - 讓 XML 資料當作參數傳入預儲程序以及使用者自訂函數，並經計算後同樣回傳 XML 型態的資料。



XML 資料型態 vs. 一般資料型態

- 要特別注意 XML 資料型態有下列幾種不允許的情況：
 - XML 資料型態的屬性不可以宣告成為「主鍵」(Primary Key)、「外來鍵」(Foreign Key)，或宣告成為 Unique 的限制條件。
 - 不可用於 Where 子句，或進行相互比較、也不可以用在 SQL 的 Group By 或 Order By 子句中，除非利用 Cast() 或 Convert() 函數先行轉換成為字串資料。



XML 資料型態 vs. 一般資料型態

- 具有 XML 資料型態的屬性不可以配合 Collate 關鍵字宣告進行大小寫區別、或全形/半形的區別。XML 有其自我的編碼方式。
- XML 資料型態的屬性雖然可以建立「索引」(Index)，但是並不是用來加速 SQL 本身的比較運算，而是用於加速 XML 專屬的函數或方法 (Method) 之運算。



XML 查詢的輸出

- 不同資料型態轉成 XML 輸出，會依據其資料格式來產生。
 - 實數 (real, float) 輸出會有小數點。
 - 若屬性資料型態為 `char(n)`，但是內含字串長度小於 *n*，則輸出成為 XML 資料時，系統會補上空白符號
 - 可以透過 SQL 指令中的 `rtrim()` 函數消去字串右方的空白。
 - 以下的 XML 範例中，為了簡潔起見，我們將假設：所有字串資料在 SQL 指令的 `select` 子句中，都已經加上 `rtrim()` 函數將多餘的空白刪除了。



Transact-SQL 的 for xml 子句

- 放置在 select 指令的最後，語法如下：

```
for xml {raw [('ElementName')] | auto | explicit | path [('ElementName')] }  
[, xmldata] [, elements] [, binary base64] [, type] [, root [('ElementName')]]
```

- **raw**：將查詢結果逐列轉為 XML 元素，並以 <row /> 做為元素標記。
- **auto**：將查詢結果以巢狀 XML 元素傳回。
- **explicit**：撰寫特定的查詢以傳回明確定義的 XML 文件。
- **path**：融合了上述三者的優點，提供更強大的用法。

(輸出控制選項 xmldata, element, binary base64, type, root 下頁說明)



for xml 子句的輸出控制選項

- 輸出控制選項用來指示結果的輸出樣式
 - xmldata：傳回描述文件結構的結構描述 (Schema)，但不加入根元素到文件中。
 - elements：指定資料行應傳回為子元素。
 - binary base64：以 base64 編碼格式，傳回二進位資料。
 - type：要求以 xml 資料型態傳回 for xml 子句的查詢結果，以便進一步處理 for xml 查詢的結果。例如，將結果指派至 xml 類型變數或撰寫 巢狀 for xml 查詢。
 - root ['*RootName*']：用來指定使用 `<RootName>` 與 `</RootName>` 將查詢結果包起來，**當作根節點**。若不指定 ('*RootName*')，則預設以 `<root>` 與 `</root>` 為根節點。



Raw 模式

- 是三種模式中最陽春的，也是使用限制較少的，幾乎可以搭配所有 SQL 子句來使用。
- 它會將查詢結果轉換成一筆一筆的 XML 資料列，並以 “row” 做為資料列的元素名稱，任何不允許虛值的欄位名稱都會直接拿來當作 “row” 標籤的屬性，而且其欄位值會對應到相對的屬性值。
- 見下頁範例...

Raw 模式的範例

- `select id, SUM(quantity) as total_quantity`
`from Orders`
`group by id`
`order by id`

執行結果

<i>id</i>	<i>total_quantity</i>
1	60
2	100
3	40
4	50
5	50
6	10

- `select id, SUM(quantity) as total_quantity`
`from Orders`
`group by id`
`order by id`
`for xml raw`

```
<row id="1" total_quantity="60"/>  
<row id="2" total_quantity="100"/>  
<row id="3" total_quantity="40"/>  
<row id="4" total_quantity="50"/>  
<row id="5" total_quantity="50"/>  
<row id="6" total_quantity="10"/>
```



Raw 模式的範例 (有根元素)

- `select id, SUM(quantity) as total_quantity`
`from Orders`
`group by id`
`order by id`
`for xml raw, root('Order')`

```
<order>  
  <row id="1" total_quantity="60"/>  
  <row id="2" total_quantity="100"/>  
  <row id="3" total_quantity="40"/>  
  <row id="4" total_quantity="50"/>  
  <row id="5" total_quantity="50"/>  
  <row id="6" total_quantity="10"/>  
</order>
```



Raw 模式

- Raw 模式後面可以跟隨著 `xmldata`、`binary base64`、`type` 三種輸出控制選項
 - `xmldata` 這個選項是用來要求系統在輸出 XML 資料時，將查詢結果所對應到的 XML Schema 一併輸出；
 - `binary base64` 這個選項則是用在查詢結果有二進位的資料時。
 - `type` 選項將於討論 “`for xml auto` 子句” 的章節中說明
 - 見下頁範例...



Raw 模式的範例 (xmldata)

- select *id*, SUM(*quantity*) as *total_quantity*
from Orders
group by *id*
order by *id*
for xml raw, xmldata

執行結果



```
<Schema name="Schema1" xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="row" content="empty" model="closed">
    <AttributeType name="id" dt:type="i4"/>
    <AttributeType name="total_quantity" dt:type="i4"/>
    <attribute type="id"/><attribute type="total_quantity"/>
  </ElementType>
</Schema>
```

```
<row xmlns="x-schema:#Schema1" id="1" total_quantity="60"/>
<row xmlns="x-schema:#Schema1" id="2" total_quantity="100"/>
<row xmlns="x-schema:#Schema1" id="3" total_quantity="40"/>
<row xmlns="x-schema:#Schema1" id="4" total_quantity="50"/>
<row xmlns="x-schema:#Schema1" id="5" total_quantity="50"/>
<row xmlns="x-schema:#Schema1" id="6" total_quantity="10"/>
```



binary base64 的 raw 模式

- 照片在關聯表 Employees 中是屬於 image 資料型態，所以要指定 binary base64 選項，讓系統將二進位資料以 Base64 編碼格式傳回，否則會出現錯誤訊息。
- ```
select photo from Employees
where id = 1 for xml raw, binary base64
```
- 輸出結果為  

```
<row photo="Binary data in base64 format" />
```



# Auto 模式

---

- 是最方便的模式，可由系統自行判別輸出的型式，因此如果查詢結果有二進位的資料時，那麼系統就會自動以 Base64 的編碼方式輸出，不需再指定 binary base64 選項。
- 在此選項下
  - 關聯表名稱會變成資料列的元素名稱，
  - 查詢結果的欄位名稱會變成該元素的屬性，
  - 欄位值則變成相對應的屬性值。





# Auto 模式的簡易範例

---

- *select id, bookname, price, author*  
from Books  
for xml auto

```
<Books id="1" bookname="三國演義" price="120" author="羅貫中"/>
<Books id="2" bookname="水滸傳" price="170" author="施耐庵"/>
<Books id="3" bookname="紅樓夢" price="170" author="曹雪芹"/>
<Books id="4" bookname="西遊記" price="140" author="吳承恩"/>
<Books id="5" bookname="水經注" price="120" author="酈道元"/>
<Books id="6" bookname="道德經" price="190" author="老子"/>
```



# Auto 模式用到計算式時的改寫方式

- Auto 模式中，輸出的欄位中若有用到計算式時，一定要使用 *as alias* 另取別名才行：
- `select id, bookname, '打八折價格 =', price*0.8, author`  
`from Books`  
`for xml auto`                      必須改寫如下，方能執行
- `select id, bookname, '打八折價格 =' as discount,`  
`price * 0.8 as new_price, author`  
`from Books`  
`for xml auto`



# Auto 模式用到兩個關聯表的情況

---

- 系統會先看第一個出現在 select 子句中的欄位是來自哪一個關聯表，然後以該關聯表做為外層元素，
- 再將另一個關聯表做為其子元素，把資料以巢狀方式先列出外層元素，再列出其子元素。
- 如果同一份外層元素中重複與不同的子元素出現在數筆查詢結果時，則會使用一份外層元素來產生內容。
- 見下頁例子...
- 將另一個關聯表的欄位反過來的範例請見例 10.4

# 看一個例子就知道 (例 10.3)

- `select Books.id, bookname, price, author, quantity  
from Books, Orders  
where Books.id = Orders.id  
for xml auto`

```
<Books id="1" bookname="三國演義" price="120" author="羅貫中">
 <Orders quantity="30"/>
</Books>
<Books id="2" bookname="水滸傳" price="170" author="施耐庵">
 <Orders quantity="20"/>
</Books>
<Books id="3" bookname="紅樓夢" price="170" author="曹雪芹">
 <Orders quantity="40"/>
</Books>
<Books id="4" bookname="西遊記" price="140" author="吳承恩">
 <Orders quantity="20"/>
</Books>
<Books id="5" bookname="水經注" price="120" author="酈道元">
 <Orders quantity="10"/>
</Books>
<Books id="6" bookname="道德經" price="190" author="老子">
 <Orders quantity="10"/>
</Books>
```

...

```
<Books id="1" bookname="三國演義" price="120" author="羅貫中">
 <Orders quantity="30"/>
</Books>
<Books id="2" bookname="水滸傳" price="170" author="施耐庵">
 <Orders quantity="40"/>
 <Orders quantity="20"/>
</Books>
<Books id="4" bookname="西遊記" price="140" author="吳承恩">
 <Orders quantity="30"/>
</Books>
<Books id="5" bookname="水經注" price="120" author="酈道元">
 <Orders quantity="40"/>
</Books>
```

<u>id</u>	bookname	price	author	quantity
1	三國演義	120	羅貫中	30
2	水滸傳	170	施耐庵	20
3	紅樓夢	170	曹雪芹	40
4	西遊記	140	吳承恩	20
5	水經注	120	酈道元	10
6	道德經	190	老子	10
1	三國演義	120	羅貫中	30
2	水滸傳	170	施耐庵	40
2	水滸傳	170	施耐庵	20
2	水滸傳	170	施耐庵	20
4	西遊記	140	吳承恩	30
5	水經注	120	酈道元	40

當同一個 Books 元素重複與不同的 Orders 元素  
出現時，會將這些 Orders 元素放在同一個 Books 元素內



# Auto 模式配合 element 選項

---

- 用來要求系統在輸出 XML 資料時，
  - 不要將查詢結果中的欄位名稱直接拿來當做元素的屬性，
  - 而是將它們當作獨立的元素 (element)，
  - 而且要放在以關聯表來命名的元素內。
- *select id, bookname, price, author*  
*from Books*  
*for xml auto, elements* (結果如下頁所示)



# 將欄位名稱當作獨立元素

---

```
<Books>
 <id>1</id>
 <bookname>三國演義</bookname>
 <price>120</price>
 <author>羅貫中</author>
```

```
</Books>
```

```
<Books>
 <id>2</id>
 <bookname>水滸傳</bookname>
 <price>170</price>
 <author>施耐庵</author>
```

```
</Books>
```

```
<Books>
 <id>3</id>
 <bookname>紅樓夢</bookname>
 <price>170</price>
 <author>曹雪芹</author>
```

```
</Books>
```

```
<Books>
 <id>4</id>
 <bookname>西遊記</bookname>
 <price>140</price>
 <author>吳承恩</author>
```

```
</Books>
```

```
<Books>
 <id>5</id>
 <bookname>水經注</bookname>
 <price>120</price>
 <author>酈道元</author>
```

```
</Books>
```

```
<Books>
 <id>5</id>
 <bookname>道德經</bookname>
 <price>190</price>
 <author>老子</author>
```

```
</Books>
```



# 使用別名 (Alias) 命名機制

- 將關聯表與欄位名稱各取其中文別名就可以產生中文標籤
- `select id as 書籍編號, bookname as 書名, price as 價格, author as 作者`  
`from Books 書籍列表`  
`for xml auto`

```
<書籍列表 書籍編號="1" 書名="三國演義" 價格="120" 作者="羅貫中"/>
<書籍列表 書籍編號="2" 書名="水滸傳" 價格="170" 作者="施耐庵"/>
<書籍列表 書籍編號="3" 書名="紅樓夢" 價格="170" 作者="曹雪芹"/>
<書籍列表 書籍編號="4" 書名="西遊記" 價格="140" 作者="吳承恩"/>
<書籍列表 書籍編號="5" 書名="水經注" 價格="120" 作者="酈道元"/>
<書籍列表 書籍編號="6" 書名="道德經" 價格="190" 作者="老子"/>
```





# Auto 模式配合 type 選項

- Auto 模式配合 type 輸出控制選項，表示查詢結果會被轉換成 xml 資料型態。
- 以下舉兩個範例說明如何利用 type 控制選項來撰寫 巢狀的 for xml 查詢

```
select name 書局名稱, (select id as 書籍編號, rtrim(bookname) as 書名,
 price as 價格, rtrim(author) as 作者, quantity 數量
 from Books 書籍列表
 where orders.id = 書籍列表.id for xml auto, type) as xmlstring
from orders, bookstores
where orders.no = bookstores.no
```

- 此例會產生關聯表中內含 xml 資料型態的結果型式

# 產生內含 xml 資料的關聯表

```
select name 書局名稱, (select id as 書籍編號, rtrim(bookname) as 書名,
 price as 價格, rtrim(author) as 作者, quantity 數量
from Books 書籍列表
where orders.id = 書籍列表.id for xml auto, type) as xmlstring
from orders, bookstores
where orders.no = bookstores.no
```

書局名稱	xmlstring
天秤書局	<書籍列表 書籍編號="2" 書名="水滸傳" 價格="170" 作者="施耐庵" 數量="20" />
天秤書局	<書籍列表 書籍編號="4" 書名="西遊記" 價格="140" 作者="吳承恩" 數量="30" />
天秤書局	<書籍列表 書籍編號="5" 書名="水經注" 價格="120" 作者="酈道元" 數量="40" />
水瓶書局	<書籍列表 書籍編號="2" 書名="水滸傳" 價格="170" 作者="施耐庵" 數量="20" />
巨蟹書局	<書籍列表 書籍編號="1" 書名="三國演義" 價格="120" 作者="羅貫中" 數量="30" />
巨蟹書局	<書籍列表 書籍編號="2" 書名="水滸傳" 價格="170" 作者="施耐庵" 數量="20" />
巨蟹書局	<書籍列表 書籍編號="3" 書名="紅樓夢" 價格="170" 作者="曹雪芹" 數量="40" />
巨蟹書局	<書籍列表 書籍編號="4" 書名="西遊記" 價格="140" 作者="吳承恩" 數量="20" />
巨蟹書局	<書籍列表 書籍編號="5" 書名="水經注" 價格="120" 作者="酈道元" 數量="10" />
巨蟹書局	<書籍列表 書籍編號="6" 書名="道德經" 價格="190" 作者="老子" 數量="10" />
射手書局	<書籍列表 書籍編號="1" 書名="三國演義" 價格="120" 作者="羅貫中" 數量="30" />



# 以子查詢將 xml 資料轉成 xml 的外層查詢子元素

- 以下範例以子查詢將 xml 資料轉成 xml 的外層查詢子元素

```
select name 書局名稱, (select id as 書籍編號, rtrim(bookname) as 書名,
 price as 價格, rtrim(author) as 作者, quantity 數量
 from Books 書籍列表
 where orders.id = 書籍列表.id for xml auto, type) as xmlstring
from orders, bookstores
where orders.no = bookstores.no
for xml auto
```

- 此例與前例一樣，只是在外層查詢最後加上 **for xml auto**



# 產生內含 xml 資料的 xml 輸出

```
<orders 書局名稱="天秤書局">
 <xmlstring>
 <書籍列表 書籍編號="2" 書名="水滸傳" 價格="170" 作者="施耐庵" 數量="20" />
 </xmlstring>
</orders>
<orders 書局名稱="天秤書局">
 <xmlstring>
 <書籍列表 書籍編號="4" 書名="西遊記" 價格="140" 作者="吳承恩" 數量="30" />
 </xmlstring>
</orders>
<orders 書局名稱="天秤書局">
 <xmlstring>
 <書籍列表 書籍編號="5" 書名="水經注" 價格="120" 作者="鄺道元" 數量="40" />
 </xmlstring>
</orders>
<orders 書局名稱="水瓶書局">
 <xmlstring>
 <書籍列表 書籍編號="2" 書名="水滸傳" 價格="170" 作者="施耐庵" 數量="20" />
 </xmlstring>
</orders>
<orders 書局名稱="巨蟹書局">
 <xmlstring>
 <書籍列表 書籍編號="1" 書名="三國演義" 價格="120" 作者="羅貫中" 數量="30" />
 </xmlstring>
</orders>
<orders 書局名稱="巨蟹書局">
 <xmlstring>
 <書籍列表 書籍編號="2" 書名="水滸傳" 價格="170" 作者="施耐庵" 數量="20" />
 </xmlstring>
</orders>
```

```
<orders 書局名稱="巨蟹書局">
 <xmlstring>
 <書籍列表 書籍編號="3" 書名="紅樓夢" 價格="170" 作者="曹雪芹" 數量="40" />
 </xmlstring>
</orders>
<orders 書局名稱="巨蟹書局">
 <xmlstring>
 <書籍列表 書籍編號="4" 書名="西遊記" 價格="140" 作者="吳承恩" 數量="20" />
 </xmlstring>
</orders>
<orders 書局名稱="巨蟹書局">
 <xmlstring>
 <書籍列表 書籍編號="5" 書名="水經注" 價格="120" 作者="鄺道元" 數量="10" />
 </xmlstring>
</orders>
<orders 書局名稱="巨蟹書局">
 <xmlstring>
 <書籍列表 書籍編號="6" 書名="道德經" 價格="190" 作者="老子" 數量="10" />
 </xmlstring>
</orders>
<orders 書局名稱="射手書局">
 <xmlstring>
 <書籍列表 書籍編號="1" 書名="三國演義" 價格="120" 作者="羅貫中" 數量="30" />
 </xmlstring>
</orders>
<orders 書局名稱="射手書局">
 <xmlstring>
 <書籍列表 書籍編號="2" 書名="水滸傳" 價格="170" 作者="施耐庵" 數量="40" />
 </xmlstring>
</orders>
```



# Explicit 模式

---

- Raw 與 Auto 兩種模式用起來方便，但轉換的結果比較制式，無法讓我們隨心所欲地安排輸出結果
- 如果想把關聯表的一些屬性轉成元素，另一些轉成元素的屬性，就必須靠 Explicit 模式來達成。
- 透過 explicit 模式，還可以設定輸出 XML 文件的階層關係 (Tree Hierarchy)
- 但必須犧牲轉換效能，
- 其轉換的複雜度也會提高
- 其轉換的依據是透過「通用表格」(Universal Table) 來達成



# 通用表格的使用方式

---

- 從 XML 文件，可以還原「通用表格」的內容，
- 先根據所要的 XML 文件求得「通用表格」，
- 再依「通用表格」的內容找出產生它的 SQL 指令，然後在這個 SQL 指令後面加上 **for xml explicit** 去執行，即可得到所要的 XML 文件



# 「通用表格」各個欄位的內容

- 第一欄：欄位名稱固定為 **Tag**
  - 是存放元素的標記編號 (Tag Number)。
  - 查詢結果的最外層元素 (非根元素)，其 Tag 值 (i.e. 標記編號) 為 1，
  - 同一階層的元素具有相同的 Tag 值
- 第二欄：欄位名稱固定為 **Parent**，
  - 用來存放「父元素」的標記編號。
  - 如果一個元素是空元素 (i.e. 沒有 Text 部分)，那麼其上一層元素就稱作它的「父元素」。
  - 元素的 Parent 值等於其 Tag 值減 1。
- 其他欄位：欄位名稱的格式為 `[element-name ! tag-num ! attribute-name ! directive]`
  - **directive** 用來指定 XML 資料的編碼方式 (後續討論)





# 先看一個簡單的欄位例子

---

- 假設某個元素名稱為  $B$ ，上一層元素名稱為  $A$  且其 Tag 值 = 1，那麼儲存此元素資料的欄位名稱就是： $[A!1!B!element]$ ，
  - 其中 *directive* 的指引值 **element**，是用來告訴系統：在 XML 文件中，將  $A$  元素的屬性  $B$  以**元素**的型態來表示。



# 再看一個 XML 的例子

---

- 假設想要輸出的 XML 文件如下所示：

```
<Books bookname="三國演義" author="羅貫中" price="120"/>
<Books bookname="水滸傳" author="施耐庵" price="170"/>
<Books bookname="紅樓夢" author="曹雪芹" price="170"/>
<Books bookname="西遊記" author="吳承恩" price="140"/>
<Books bookname="水經注" author="酈道元" price="120"/>
 <Books bookname="道德經" author="老子" price="190"/>
```

- Books 是最外層的元素，所以其 Tag 值 = 1。
- Books 有 bookname、author、price 三個屬性，因此通用表格有：  
Tag、Parent、[Books!1! bookname]、[Books!1! author]、  
[Books!1!price] 五個欄位。（如下頁所示）



# 前述 XML 範例的通用表格

Tag	Parent	[Books!1! bookname]	[Books!1! author]	[Books!1!price]
1	0	三國演義	羅貫中	120
1	0	水滸傳	施耐庵	170
1	0	紅樓夢	曹雪芹	170
1	0	西遊記	吳承恩	140
1	0	水經注	酈道元	120
1	0	道德經	老子	190

- 產生此通用表格的 SQL 指令是：  
`select        1 as Tag, 0 as Parent,  
              bookname as [Books!1!bookname],  
              author as [Books!1!author], price as [Books!1!price]  
from Books`
- 再加上 `for xml explicit`，就可以產生前述的 XML 文件



# 更複雜的例子 (例 10.6)

## ■ 假設要由 BOB 資料庫產生下列的 XML 文件

```
<書局 書局編號="1" 書局名稱="巨蟹書局" 城市="臺北市">
```

```
 <書籍 書名="三國演義"/>
```

```
 <書籍 書名="水經注"/>
```

```
 <書籍 書名="水滸傳"/>
```

```
 <書籍 書名="西遊記"/>
```

```
 <書籍 書名="紅樓夢"/>
```

```
 <書籍 書名="道德經"/>
```

```
</書局>
```

```
<書局 書局編號="2" 書局名稱="射手書局" 城市="高雄市">
```

```
 <書籍 書名="三國演義"/>
```

```
 <書籍 書名="水滸傳"/>
```

```
</書局>
```

```
<書局 書局編號="3" 書局名稱="水瓶書局" 城市="新竹市">
```

```
 <書籍 書名="水滸傳"/>
```

```
</書局>
```

```
<書局 書局編號="4" 書局名稱="天秤書局" 城市="臺中市">
```

```
 <書籍 書名="水經注"/>
```

```
 <書籍 書名="水滸傳"/>
```

```
 <書籍 書名="西遊記"/>
```

```
</書局>
```

```
<書局 書局編號="5" 書局名稱="獅子書局" 城市="臺南市" />
```

- 書局是最外層的元素，所以 Tag 值 = 1。
- 書籍元素是空元素，不能往上合併成屬性，所以其 Tag 值 = 2。
- 書局有：書局編號、書局名稱、城市，三個屬性
- 書籍有：書名一個屬性，
- 因此通用表格有：Tag、Parent、[書局!1!書局編號]、[書局!1!書局名稱]、[書局!1!城市]、[書籍!2!書名] 六個欄位。(如下頁所示)

# 複雜的通用表格對照

<書局 書局編號="1" 書局名稱="巨蟹書局" 城市="臺北市">

<書籍 書名="三國演義"/>

<書籍 書名="水經注"/>

<書籍 書名="水滸傳"/>

<書籍 書名="西遊記"/>

<書籍 書名="紅樓夢"/>

<書籍 書名="道德經"/>

</書局>

<書局 書局編號="2" 書局名稱="射手書局" 城市="高雄市">

<書籍 書名="三國演義"/>

<書籍 書名="水滸傳"/>

</書局>

<書局 書局編號="3" 書局名稱="水瓶書局" 城市="新竹市">

<書籍 書名="水滸傳"/>

</書局>

<書局 書局編號="4" 書局名稱="天秤書局" 城市="臺中市">

<書籍 書名="水經注"/>

<書籍 書名="水滸傳"/>

<書籍 書名="西遊記"/>

</書局>

<書局 書局編號="5" 書局名稱="獅子書局" 城市="臺南市">

Tag	Parent	[書局!1!書局編號]	[書局!1!書局名稱]	[書局!1!城市]	[書籍!2!書名]
1	0	1	巨蟹書局	臺北市	NULL
2	1	1	巨蟹書局	臺北市	三國演義
2	1	1	巨蟹書局	臺北市	水經注
2	1	1	巨蟹書局	臺北市	水滸傳
2	1	1	巨蟹書局	臺北市	西遊記
2	1	1	巨蟹書局	臺北市	紅樓夢
2	1	1	巨蟹書局	臺北市	道德經
1	0	2	射手書局	高雄市	NULL
2	1	2	射手書局	高雄市	三國演義
2	1	2	射手書局	高雄市	水滸傳
1	0	3	水瓶書局	新竹市	NULL
2	1	3	水瓶書局	新竹市	水滸傳
1	0	4	天秤書局	臺中市	NULL
2	1	4	天秤書局	臺中市	水經注
2	1	4	天秤書局	臺中市	水滸傳
2	1	4	天秤書局	臺中市	西遊記
1	0	5	獅子書局	臺南市	NULL



# 如此複雜的 XML, SQL 如何寫?

- 因為有兩層元素：書局和書籍，所以要個別撰寫它們的 SQL 指令。
  - 屬於第一層的書局元素，在此通用表格內共有 5 筆資料，產生這些資料的 SQL 指令是：  
`select 1 as Tag, 0 as Parent, no as [書局!1!書局編號],  
name as [書局!1!書局名稱], city as [書局!1!城市], null as [書籍!2!書名]  
from Bookstores`
  - 屬於第二層的書籍元素，在此通用表格內共有 12 筆資料，產生這些資料的 SQL 指令是：  
`select 2 as Tag, 1 as Parent, bookstores.no, name, city, bookname  
from Books, Bookstores, Orders  
where Bookstores.no = Orders.no and Books.id = Orders.id`
- 使用 union all 將以上兩個 SQL 指令合併，再依 [書局!1!書局編號]、[書籍!2!書名] 排序，就能得到前述的通用表格。

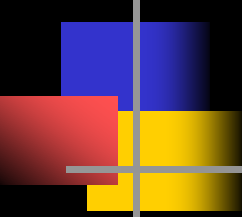


## 完整的 SQL (例 10.6)

---

- `select 1 as Tag, 0 as Parent, no as [書局!1!書局編號],  
name as [書局!1!書局名稱], city as [書局!1!城市], NULL as [書籍!2!  
書名]  
from Bookstores  
union all  
select 2 as Tag, 1 as Parent, Bookstores.no , name, city, bookname  
from Bookstores, Books, Orders  
where Bookstores.no = Orders.no and Books.id = Orders.id  
order by [書局!1!書局編號], [書籍!2!書名]  
for xml explicit`
- 範例 10.7 有類似用法, 請自行參見課本說明





# Explicit 模式的輸出指引

---

- Explicit 模式的輸出指引 *directive* 有 8 種：
  - **ID**、**IDREF**、**IDREFS**: 用來告訴系統：資料的編碼方式。若在 `explicit` 後面加上 “, `xmldata`” 則系統會將資料的編碼方式放在 XML Schema 中，讓收到這份 XML 文件的系統可以用來驗證文件是否合法 (Valid)
  - **hide**、**element**、**xml**、**xmltext** 與 **cdata**: 告訴系統如何在 XML 文件中表示字串資料。



# 輸出指引 ID 的用法

---

- ID：指示系統將元素的屬性設定為 ID 型態。
- 具有 ID 型態的屬性值必須要在該 XML 文件中具有唯一性才合法。
- 這類屬性通常是對應到關聯表的主鍵或候選鍵。
- 下頁的範例會將 XML 文件中的 *price* 屬性設定為 ID 型態 (是一個錯誤示範, 驗證時會有 Error!)

# 使用 ID 指引的錯誤範例

- select 1 as Tag, 0 as Parent, *bookname* as [Books!1!*bookname*],  
*price* as [Books!1!*price*!id]  
from Books  
for xml explicit, xmldata

```
<Schema name="Schema1" xmlns="urn:schemas-microsoft-com:xml-data"
 xmlns:dt="urn:schemas-microsoft-com:datatypes">
 <ElementType name="Books" content="mixed" model="open">
 <AttributeType name="bookname" dt:type="string"/>
 <AttributeType name="price" dt:type="id"/>
 <attribute type="bookname"/>
 <attribute type="price"/>
 </ElementType>
</Schema>
<Books xmlns="x-schema:#Schema1" bookname="三國演義" price="120"/>
<Books xmlns="x-schema:#Schema1" bookname="水滸傳" price="170"/>
<Books xmlns="x-schema:#Schema1" bookname="紅樓夢" price="170"/>
<Books xmlns="x-schema:#Schema1" bookname="西遊記" price="140"/>
<Books xmlns="x-schema:#Schema1" bookname="水經注" price="120"/>
<Books xmlns="x-schema:#Schema1" bookname="道德經" price="190"/>
```

- *Price* 的 dt:type 的屬性值由原來的 i4 (整數型態) 變成為 id (見黃色部分)。
- 由於 *price* 的屬性值不具有唯一性，所以在XML 剖析器驗證文件時，會產生錯誤訊息。



# 輸出指引 IDREF 的用法

---

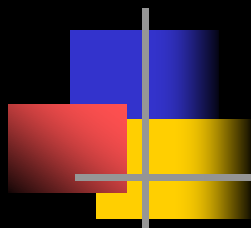
- 將元素的屬性設定為 IDREF 型態，以參考某個具有 ID 型態的屬性，形成文件內部的參考鏈結。
- 具有 IDREF 型態的屬性值必須與所參考的 ID 型態屬性值能夠找到互相匹配者才算合法。
- 這類屬性通常是對應到關聯表的**外來鍵**。因此 IDREF 會與 ID 搭配出現。
- 下例會將 XML 文件中“訂單”元素的屬性“書局編號”設定為 IDREF 型態，“書局”元素的屬性“書局編號”設定為 ID 型態以供參考



# 使用 IDREF 指引的範例

---

- select 1 as Tag, 0 as Parent, *no* as [書局!1!書局編號!**id**],  
*name* as [書局!1!書局名稱], NULL as [訂單!2!書局編號!**idref**],  
NULL as [訂單!2!數量]  
from Bookstores  
**union all**  
select 2 as Tag, 1 as Parent, **Bookstores.no**, *name*, **orders.no**, *quantity*  
from Books, Orders, Bookstores  
where Books.*id* = Orders.*id* and Orders.*no* = Bookstores.*no*  
order by [書局!1!書局編號!**id**], [訂單!2!書局編號!**idref**]  
**for xml explicit, xmldata**
- 結果如下頁所示



• 元素“訂單”中屬性“書局編號”的所有值 {1, 2, 3, 4} 都存在於所參考的元素“書局”中屬性“書局編號”的所有值 {1, 2, 3, 4, 5} 內。

• 所以此文件是一個合法的 XML 文件。

• 還有 IDREFS 與 ID 搭配的用法也很類似，請參考 11-20, 21 頁範例

```
<Schema name="Schema1" xmlns="urn:schemas-microsoft-com:xml-data"
 xmlns:dt="urn:schemas-microsoft-com:datatypes">
 <ElementType name="書局" content="mixed" model="open">
 <AttributeType name="書局編號" dt:type="id"/>
 <AttributeType name="書局名稱" dt:type="string"/>
 <attribute type="書局編號"/>
 <attribute type="書局名稱"/>
 </ElementType>
 <ElementType name="訂單" content="mixed" model="open">
 <AttributeType name="書局編號" dt:type="idref"/>
 <AttributeType name="數量" dt:type="i4"/>
 <attribute type="書局編號"/>
 <attribute type="數量"/>
 </ElementType>
</Schema>
<書局 xmlns="x-schema:#Schema1" 書局編號="1" 書局名稱="巨蟹書局">
 <訂單 書局編號="1" 數量="30"/>
 <訂單 書局編號="1" 數量="20"/>
 <訂單 書局編號="1" 數量="40"/>
 <訂單 書局編號="1" 數量="20"/>
 <訂單 書局編號="1" 數量="10"/>
 <訂單 書局編號="1" 數量="10"/>
</書局>
<書局 xmlns="x-schema:#Schema1" 書局編號="2" 書局名稱="射手書局">
 <訂單 書局編號="2" 數量="30"/>
 <訂單 書局編號="2" 數量="40"/>
</書局>
<書局 xmlns="x-schema:#Schema1" 書局編號="3" 書局名稱="水瓶書局">
 <訂單 書局編號="3" 數量="20"/>
</書局>
<書局 xmlns="x-schema:#Schema1" 書局編號="4" 書局名稱="天秤書局">
 <訂單 書局編號="4" 數量="20"/>
 <訂單 書局編號="4" 數量="30"/>
 <訂單 書局編號="4" 數量="40"/>
</書局>
<書局 xmlns="x-schema:#Schema1" 書局編號="5" 書局名稱="獅子書局"/>
```



# hide 輸出指引

---

- 將元素的屬性資料隱藏起來。
- 使用時機：利用某個屬性來對資料排序，但是卻不想顯示該屬性值的情況下。
- 舉例來說，假設我們希望在 XML 文件中，產生書局與其所訂購書籍的階層關係，並要求“書局”元素以“書局編號”屬性來排序，而在“書局”元素內的“書籍”元素則是以不顯示內容的“書籍編號”屬性來排序。那麼可以使用以下的 SQL 指令來達成



# 使用 hide 的範例

---

- ```
select 1 as Tag, 0 as Parent, no as [書局!1!書局編號],  
       name as [書局!1!書局名稱], NULL as [書籍!2!書籍編號!hide],  
       NULL as [書籍!2!書名]  
from Bookstores  
union all  
select 2 as Tag, 1 as Parent, Bookstores.no, name, Books.id, bookname  
from Books, Bookstores, Orders  
where Bookstores.no = Orders.no and Books.id = Orders.id  
order by [書局!1!書局編號], [書籍!2!書籍編號!hide]  
for xml explicit
```
- 結果如下頁所示



輸出結果 (依書局/書籍編號排序)

```
<書局 書局編號="1" 書局名稱="巨蟹書局">
  <書籍 書名="三國演義"/>
  <書籍 書名="水滸傳"/>
  <書籍 書名="紅樓夢"/>
  <書籍 書名="西遊記"/>
  <書籍 書名="水經注"/>
  <書籍 書名="道德經"/>
</書局>
<書局 書局編號="2" 書局名稱="射手書局">
  <書籍 書名="三國演義"/>
  <書籍 書名="水滸傳"/>
</書局>
<書局 書局編號="3" 書局名稱="水瓶書局">
  <書籍 書名="水滸傳"/>
</書局>
<書局 書局編號="4" 書局名稱="天秤書局">
  <書籍 書名="水滸傳"/>
  <書籍 書名="西遊記"/>
  <書籍 書名="水經注"/>
</書局>
<書局 書局編號="5" 書局名稱="獅子書局"/>
```



輸出指引| element

- Element 指示系統在 XML 文件中將元素的屬性以子元素的型態表示

```
select 1 as Tag, 0 as Parent,  
no as [書局!1!書局編號],  
name as [書局!1!書局名稱!element]  
from Bookstores  
for xml explicit
```

```
<書局 書局編號="1">  
  <書局名稱>巨蟹書局</書局名稱>  
</書局>  
<書局 書局編號="2">  
  <書局名稱>射手書局</書局名稱>  
</書局>  
<書局 書局編號="3">  
  <書局名稱>水瓶書局</書局名稱>  
</書局>  
<書局 書局編號="4">  
  <書局名稱>天秤書局</書局名稱>  
</書局>  
<書局 書局編號="5">  
  <書局名稱>獅子書局</書局名稱>  
</書局>
```

查詢結果中如果有 XML 的特殊符號：<、>、&、'、"，那麼在所產生的 XML 文件內，會以其替代符號來表示 (特殊符號與替代符號的對照請參考本書 2-25 頁或投影片第 2-56 頁)

輸出指引 xml

- 功能與 element 一樣，只不過對於資料中 XML 的特殊符號不予處理，以其原貌呈現。
- 除了 hide 之外，此指引不能和其他的指引一起使用。

```
<書局 書局編號="1">  
  <書局名稱>巨蟹書局</書局名稱>  
</書局>  
<書局 書局編號="2">  
  <書局名稱>射手書局</書局名稱>  
</書局>  
<書局 書局編號="3">  
  <書局名稱>&水瓶書局</書局名稱>  
</書局>  
<書局 書局編號="4">  
  <書局名稱>天秤書局</書局名稱>  
</書局>  
<書局 書局編號="5">  
  <書局名稱>獅子書局</書局名稱>  
</書局>
```

不會
輸出
成為



```
<書局 書局編號="1">  
  <書局名稱>巨蟹書局</書局名稱>  
</書局>  
<書局 書局編號="2">  
  <書局名稱>射手書局</書局名稱>  
</書局>  
<書局 書局編號="3">  
  <書局名稱>&水瓶書局</書局名稱>  
</書局>  
<書局 書局編號="4">  
  <書局名稱>天秤書局</書局名稱>  
</書局>  
<書局 書局編號="5">  
  <書局名稱>獅子書局</書局名稱>  
</書局>
```



輸出指引 xmltext

- 當關聯表的某個屬性，其所有的值均為 XML 文件的元素格式時，我們可以使用 xmltext 來指示系統將這些元素融入所產生的 XML 文件中。
- 使用 xmltext 的方式有以下兩種：
 - 指定屬性名稱：系統會拿這個所指定的名稱來取代原來存在屬性值中的 XML 元素名稱。
 - 沒有指定屬性名稱：系統會將原來存在屬性值中的 XML 元素，其屬性變成所指定元素的屬性，其資料變成所指定元素的資料。

使用 xmldata 時指定屬性名稱

- 假設 Mbooks 關聯表的 *publisher* 屬性存放著XML 元素 “出版商” 的資料，其內容如下所示 (注意：屬性 *publisher* 的 XML 內容有多種不同的型式)

MBooks

| <i>id</i> | <i>bookname</i> | <i>author</i> | <i>price</i> | <i>publisher</i> |
|-----------|-----------------|---------------|--------------|---------------------------------------|
| 1 | 三國演義 | 羅貫中 | 120 | <出版商 名稱="古文出版社"/> |
| 2 | 水滸傳 | 施耐庵 | 170 | <出版商>中庸出版社</出版商> |
| 3 | 紅樓夢 | 曹雪芹 | 170 | <出版商 名稱="春秋出版社">
台北總公司</出版商> |
| 4 | 西遊記 | 吳承恩 | 140 | <出版商>聊齋出版社</出版商> |
| 5 | 水經注 | 酈道元 | 120 | <出版商>易經出版社
<分公司>高雄</分公司>
</出版商> |
| 6 | 道德經 | 老子 | 190 | <出版商 成立日期="1988">
大唐出版社</出版商> |



拿指定名稱來取代原 XML 元素名稱

```
select      1 as Tag, 0 as Parent,
bookname as [書局!1!書籍名稱],
publisher as [書局!1!發行人!xmltext]
from MBooks
for xml explicit
```

拿“發行人”來取代 *publisher*
屬性值中的元素名稱—“出版商”，
將屬性值的元素內容融入 XML 文件中

```
<書局 書籍名稱="三國演義">
  <發行人 名稱="古文出版社"/>
</書局>
<書局 書籍名稱="水滸傳">
  <發行人>中庸出版社</發行人>
</書局>
<書局 書籍名稱="紅樓夢">
  <發行人 名稱="春秋出版社">台北總公司</發行人>
</書局>
<書局 書籍名稱="西遊記">
  <發行人>聊齋出版社</發行人>
</書局>
<書局 書籍名稱="水經注">
  <發行人>易經出版社
    <分公司>高雄</分公司>
  </發行人>
</書局>
<書局 書籍名稱="道德經">
  <發行人 成立日期="1988">大唐出版社</發行人>
</書局>
```

使用 xmldata 時沒有指定屬性名稱

- 系統會將原來存在屬性值中的 XML 元素，其屬性變成所指定元素的屬性，其資料變成所指定元素的資料。
- 以前述的 MBooks 為例 (如下所列)

MBooks

| <i>id</i> | <i>bookname</i> | <i>author</i> | <i>price</i> | <i>publisher</i> |
|-----------|-----------------|---------------|--------------|---------------------------------------|
| 1 | 三國演義 | 羅貫中 | 120 | <出版商 名稱="古文出版社"/> |
| 2 | 水滸傳 | 施耐庵 | 170 | <出版商>中庸出版社</出版商> |
| 3 | 紅樓夢 | 曹雪芹 | 170 | <出版商 名稱="春秋出版社">
台北總公司</出版商> |
| 4 | 西遊記 | 吳承恩 | 140 | <出版商>聊齋出版社</出版商> |
| 5 | 水經注 | 酈道元 | 120 | <出版商>易經出版社
<分公司>高雄</分公司>
</出版商> |
| 6 | 道德經 | 老子 | 190 | <出版商 成立日期="1988">
大唐出版社</出版商> |

拿指定名稱來取代原 XML 元素名稱

MBooks

```
select      1 as Tag, 0 as Parent,
bookname as [書局!書籍名稱],
publisher as [書局!! !xmltext]
from MBooks
for xml explicit
```

| <i>id</i> | <i>bookname</i> | <i>author</i> | <i>price</i> | <i>publisher</i> |
|-----------|-----------------|---------------|--------------|---------------------------------------|
| 1 | 三國演義 | 羅貫中 | 120 | <出版商 名稱="古文出版社"/> |
| 2 | 水滸傳 | 施耐庵 | 170 | <出版商>中庸出版社</出版商> |
| 3 | 紅樓夢 | 曹雪芹 | 170 | <出版商 名稱="春秋出版社">
台北總公司</出版商> |
| 4 | 西遊記 | 吳承恩 | 140 | <出版商>聊齋出版社</出版商> |
| 5 | 水經注 | 酈道元 | 120 | <出版商>易經出版社
<分公司>高雄</分公司>
</出版商> |
| 6 | 道德經 | 老子 | 190 | <出版商 成立日期="1988">
大唐出版社</出版商> |

產生的XML文件如右

```
<書局 書籍名稱="三國演義" 名稱="古文出版社" />
<書局 書籍名稱="水滸傳">中庸出版社</書局>
<書局 書籍名稱="紅樓夢" 名稱="春秋出版社">台北總公司</書局>
<書局 書籍名稱="西遊記">聊齋出版社</書局>
<書局 書籍名稱="水經注">易經出版社
    <分公司>高雄</分公司></書局>
<書局 書籍名稱="道德經" 成立日期="1988">大唐出版社</書局>
```




輸出指引| cdata

- 只適用在 ntext、nvarchar、text、varchar 文字型態的欄位上，用來指示系統將屬性值以 `<![CDATA[put data here]]>` 包裝，然後放在所指定的元素內，其中 *put data here* 就是放屬性值的地方。
- 包裝的用意是希望 XML 的剖析器不要去解釋裡面的資料，讓這些資料以原貌呈現。
- **注意**：使用 cdata 的時候，不能註明元素的屬性名稱，否則會有錯誤訊息。



使用 cdata 輸出指引的範例

- 我們想將下述表格的 *publisher* 屬性文字資料以原貌寫入 XML 文件中

NBooks

| <i>id</i> | <i>bookname</i> | <i>author</i> | <i>price</i> | <i>publisher</i> |
|-----------|-----------------|---------------|--------------|-------------------|
| 1 | 三國演義 | 羅貫中 | 120 | <出版商 名稱="古文出版社"/> |
| 2 | 水滸傳 | 施耐庵 | 170 | <出版商>中庸出版社</出版商> |
| 3 | 紅樓夢 | 曹雪芹 | 170 | <春秋出版社> |
| 4 | 西遊記 | 吳承恩 | 140 | <出版商 名稱="聊齋出版社"> |
| 5 | 水經注 | 酈道元 | 120 | !易經出版社 |
| 6 | 道德經 | 老子 | 190 | "大唐出版社" |

使用 cdata 輸出指引的範例(續)

- Select 1 as Tag, 0 as Parent,
bookname as [Books!1!bookname], *author* as [Books!1!author],
price as [Books!1!price], *publisher* as [Books!1!!cdata]
from Nbooks
for xml explicit

NBooks

| <u>id</u> | bookname | author | price | publisher |
|-----------|----------|--------|-------|-------------------|
| 1 | 三國演義 | 羅貫中 | 120 | <出版商 名稱="古文出版社"/> |
| 2 | 水滸傳 | 施耐庵 | 170 | <出版商>中庸出版社</出版商> |
| 3 | 紅樓夢 | 曹雪芹 | 170 | <春秋出版社> |
| 4 | 西遊記 | 吳承恩 | 140 | <出版商 名稱="聊齋出版社"> |
| 5 | 水經注 | 酈道元 | 120 | !易經出版社 |
| 6 | 道德經 | 老子 | 190 | "大唐出版社" |

```
<Books bookname="三國演義" author="羅貫中" price="120">
  <![CDATA[<出版商 名稱="古文出版社"/>]]>
</Books>
<Books bookname="水滸傳" author="施耐庵" price="170">
  <![CDATA[<出版商>中庸出版社</出版商>]]>
</Books>
<Books bookname="紅樓夢" author="曹雪芹" price="170">
  <![CDATA[<春秋出版社>]]>
</Books>
<Books bookname="西遊記" author="吳承恩" price="140">
  <![CDATA[<出版商 名稱="聊齋出版社">]]>
</Books>
<Books bookname="水經注" author="酈道元" price="120">
  <![CDATA[!易經出版社]]>
</Books>
<Books bookname="道德經" author="老子" price="190">
  <![CDATA["大唐出版社"]]>
</Books>
```



Path 模式

- 雖然 explicit 模式功能強大，但使用上卻很複雜，所以 SQL Server 2008 便提供了 path 模式供使用者選擇。
- 此模式可以讓你在多個階層下，產生混合的元素與屬性，提供類似 raw 與 auto 模式一樣簡易的寫法，但是功能卻不輸explicit模式。
- 作法也是利用替查詢結果取別名 (alias) 的機制來進行XML文件的輸出控制
- 當系統看到alias中有斜線 (/) 的時候，就會產生一個新的層級，並將別名中以斜線隔開的名稱中，把相同的名稱當作新層級的共同父節點



Path 模式的範例

```
select id,  
       rtrim(bookname) as [name/bookname],  
       rtrim(author) as [name/author_name],  
       price  
from Books  
where id > 3  
for xml path
```

```
select id,  
       rtrim(bookname) as [name/bookname],  
       rtrim(author) as [name/@author_name],  
       price  
from Books  
where id > 3  
for xml path -- 換成 @author_name 試試看
```

```
<row>  
  <id>1</id>  
  <name>  
    <bookname>三國演義</bookname>  
    <author_name>羅貫中</author_name>  
  </name>  
  <price>120</price>  
</row>  
<row>  
  <id>2</id>  
  <name>  
    <bookname>水滸傳</bookname>  
    <author_name>施耐庵</author_name>  
  </name>  
  <price>170</price>  
</row>  
<row>  
  <id>3</id>  
  <name>  
    <bookname>紅樓夢</bookname>  
    <author_name>曹雪芹</author_name>  
  </name>  
  <price>170</price>  
</row>
```



利用 OPENXML 函數做 增/刪/改

- OPENXML 函數讓 XML 文件的資料可以以關聯表的形式呈現。

```
<書籍資料>
  <書籍 書籍編號="1">
    <書名>三國演義</書名>
    <價格>120</價格>
  </書籍>
  <書籍 書籍編號="2">
    <書名>水滸傳</書名>
    <價格>170</價格>
  </書籍>
</書籍資料>
```

可以看成



書籍編號	書名	價格
1	三國演義	120
2	水滸傳	170



使用 OPENXML 函數的觀念

- XML 文件要先化成樹狀結構才能使用 OPENXML 函數。
- OPENXML 函數的做法是：在樹狀結構上，依據所設定的路徑找出相關的節點做為「節點集」，然後使用「節點集」的節點取得所要的資料，將它們表示成關聯表的形式。
- 在說明 OPENXML 函數的使用方法之前，要先了解產生 XML 文件樹狀結構的預儲程序 `sp_xml_preparedocument` 與訂定「節點集」的 XPath 位置路徑



sp_xml_preparedocument 介紹

- 這個預儲程序會使用 MSXML 剖析器來剖析XML 文件，以產生文件的樹狀結構
- 若格式正確，系統會配置一塊記憶體來存放其樹狀結構，然後將控制代碼傳回，以利後續存取使用（如下面的@idoc）。
- 語法：
`sp_xml_preparedocument @idoc output, xmltext [, namespaces]`
 - *xmltext* 是內容為 XML 文件的字串變數；
 - *Namespaces* 是用來註明 XML 文件元素的「命名空間」(Namespace)。不同性質的元素只要定義在不同的「命名空間」，就可以有相同的元素名稱。
 - 「命名空間」可以用來識別元素，提供剖析器正確的資料所在位置。
- 使用 `sp_xml_removedocument` 可以歸還 @idoc 的記憶空間

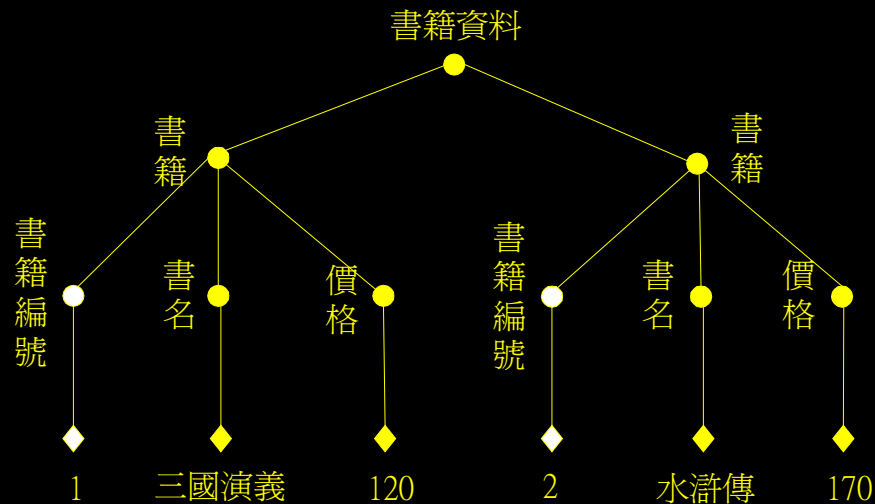
範例介紹

```
declare @xmltext nvarchar(max)
declare @idoc integer
set @xmltext = '
```

```
<書籍資料>
<書籍 書籍編號="1">
  <書名>三國演義</書名>
  <價格>120</價格>
</書籍>
<書籍 書籍編號="2">
  <書名>水滸傳</書名>
  <價格>170</價格>
</書籍>
</書籍資料>'
```

```
exec sp_xml_preparedocument @idoc output, @xmltext
```

產生的樹狀結構如下



- : 元素節點
- : 屬性節點
- ◆ : 元素本文節點
- ◇ : 屬性值節點

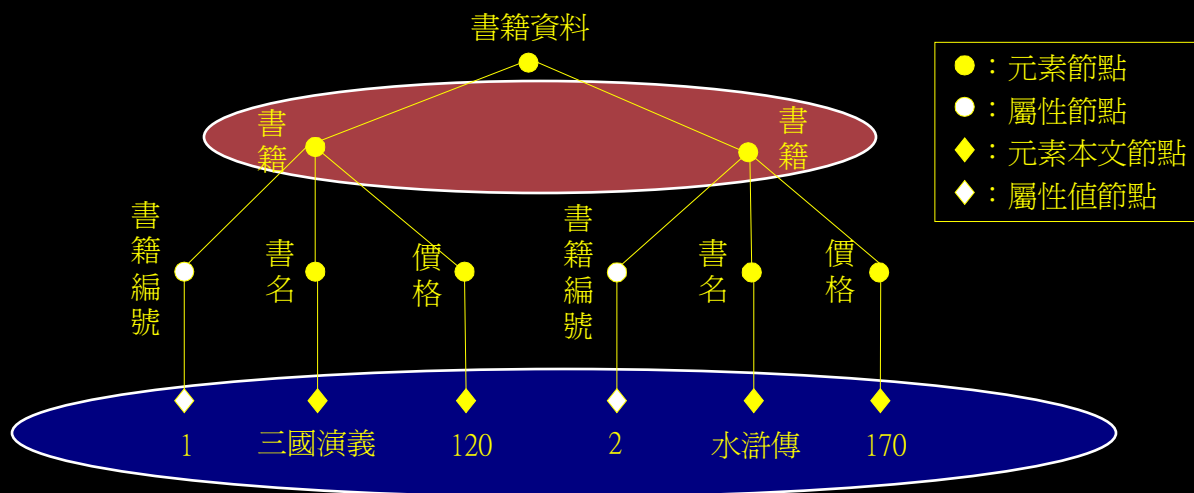


Xpath: 路徑運算式語言

- 根據運算式的運算結果，在樹狀結構上取得「節點集」，以取得所要的資料。
- 位置路徑有兩種表示法：
 - **絕對位置路徑**：是從根節點到達目的節點的路徑，以斜線 (/) 來區隔沿路所經過的節點名稱，如：'/書籍資料/書籍/書名'。注意：屬性節點的名稱前面須加上@符號。
 - **相對位置路徑**：是從目前節點集的節點開始，到達目的節點的路徑，其中單點 (.) 代表目前所在位置的節點，而雙點 (..) 則代表目前所在位置的父節點，並以斜線 (/) 來區隔沿路所經過的節點名稱。
 - E.g., '/書名' 與 '../價格'。

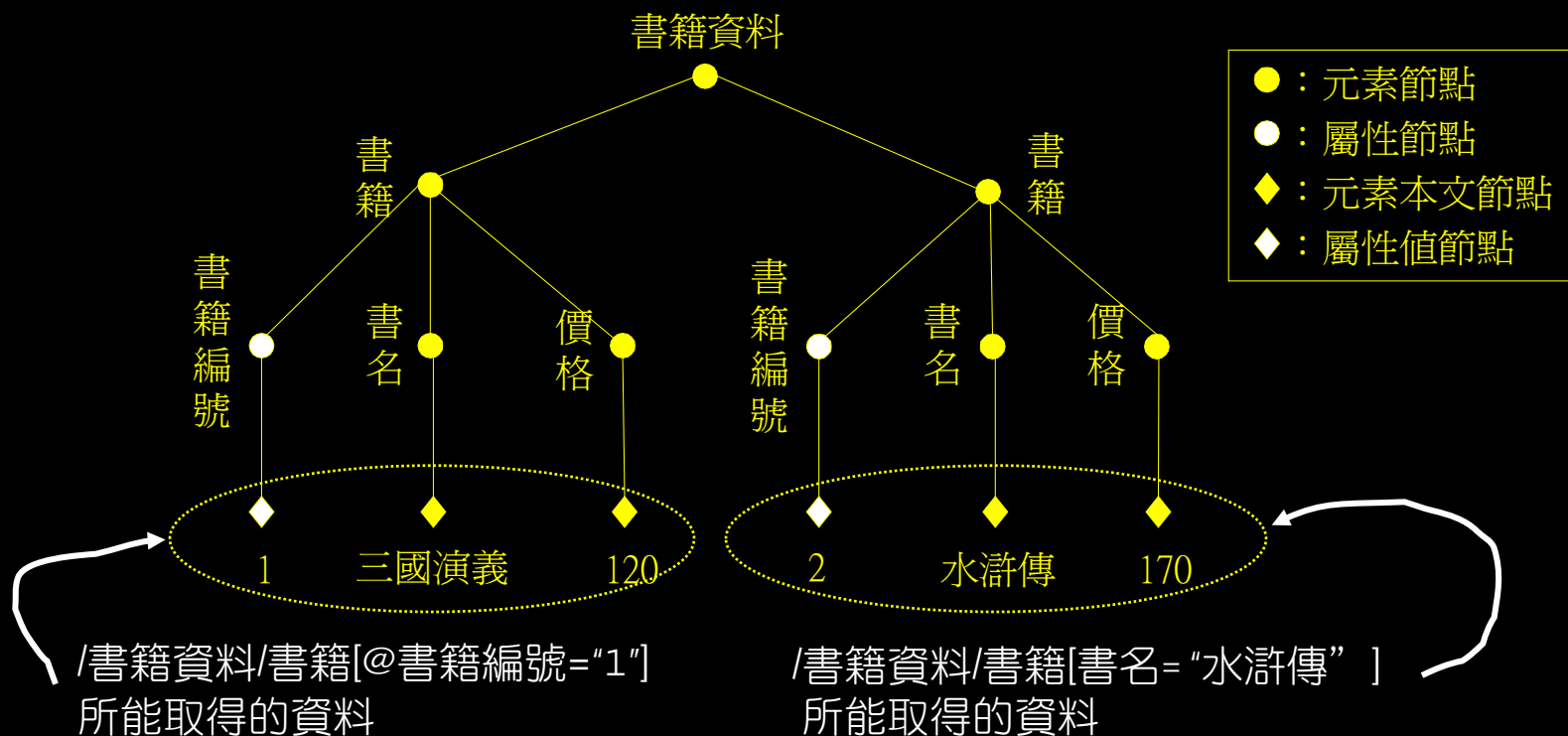
「節點集」所能取得的資料

- 節點集: 能夠取得所要資料的節點所構成的集合
 - 元素節點: 系統可以取得它的本文、屬性值，以及子元素的本文三種資料；
 - 屬性節點: 系統可以取得它的屬性值資料。
- 如果「節點集」的節點為“書籍”這些元素節點（下圖紅色部分），那麼系統就可以取得“書籍編號”的屬性值，以及子元素“書名”和“價格”的本文資料（下圖藍色部分）。



路徑篩選條件

- 我們還可以設定條件以篩選出所要的節點集，其語法為：位置路徑[條件式]





OPENXML 的語法

- OPENXML 函數讓我們可以在 XML 文件的樹狀結構中查詢資料，並將查詢的結果以關聯表的格式來表示。
- 使用時，必須註明：
 - 樹狀結構的控制代碼：由 `sp_xml_preparedocument` 傳回
 - 搜尋資料所需的位置路徑：XPath 型式
 - 關聯表的欄位名稱、
 - 資料型態等資訊。

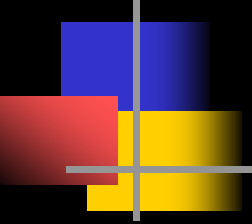


OPENXML 的語法 (續)

■ 語法如下：

`OPENXML(idoc int [in], row-pattern nvarchar [in], [flags byte [in]])`
`[with (schema-declaration | table-name)]`

- *idoc*：是樹狀結構的控制代碼
- *row-pattern*：是一個 Xpath 的位置路徑
- *schema-declaration*：用來宣告關聯表的欄位名稱與資料型態等資訊 (說明如下頁所示)
- *table-name*：關聯表的名稱，該關聯表具有輸出結果所要使用的綱要結構。或使用 *schema-declaration* 來描述綱要。
- *flags*：用來告訴系統：*schema-declaration* 或 *table-name* 的欄位名稱是對應到節點集內的屬性節點、元素節點、還是兩者都有 (flag 的值請見下頁)。



flags 值	說 明
0	此為預設值，表示欄位名稱是對應到節點集內屬性節點的名稱。
1	明確地表示欄位名稱是對應到節點集內屬性節點的名稱。
2	明確地表示欄位名稱是對應到節點集內元素節點的名稱。
3	表示欄位名稱有的是對應到節點集內屬性節點的名稱，有的則是對應到節點集內元素節點的名稱。
8	表示欄位名稱是對應到節點集內屬性節點的名稱。對於找不到對應的欄位名稱，則以@mp:xmltext 這個元屬性將節點集內剩餘的資料輸出做為該欄的資料。
9	明確地表示欄位名稱是對應到節點集內屬性節點的名稱。對於找不到對應的欄位名稱，則以@mp:xmltext 這個元屬性將節點集內剩餘的資料輸出做為該欄的資料。
10	明確地表示欄位名稱是對應到節點集內元素節點的名稱。對於找不到對應的欄位名稱，則以@mp:xmltext 這個元屬性將節點集內剩餘的資料輸出做為該欄的資料。
11	表示欄位名稱有的是對應到節點集內屬性節點的名稱，有的則是對應到節點集內元素節點的名稱。對於找不到對應的欄位名稱，則以@mp:xmltext 這個元屬性將節點集內剩餘的資料輸出做為該欄的資料。



schema-declaration 格式說明

- *schema-declaration* 格式為：

*col-name col-type [col-pattern | metaproperty-name]
[, col-name col-type [col-pattern | metaproperty-name]...]*

- *col-name* 是導出關聯表的欄位名稱，它們在樹狀結構內不是元素節點就是屬性節點。
- *col-type* 是欄位 *col-name* 的資料型態（SQL 所定義的資料型態）。
- *col-pattern* 是一個 Xpath 型式的相對位置路徑，用來描述從 *row_pattern* 這個位置路徑的目的節點 (i.e. 節點集的節點) 開始，到達取得欄位 *col-name* 資料的節點，所經過的路徑。
- *Metaproperty-name* 如下頁說明



XML 文件的元屬性 (Meta-Properties)

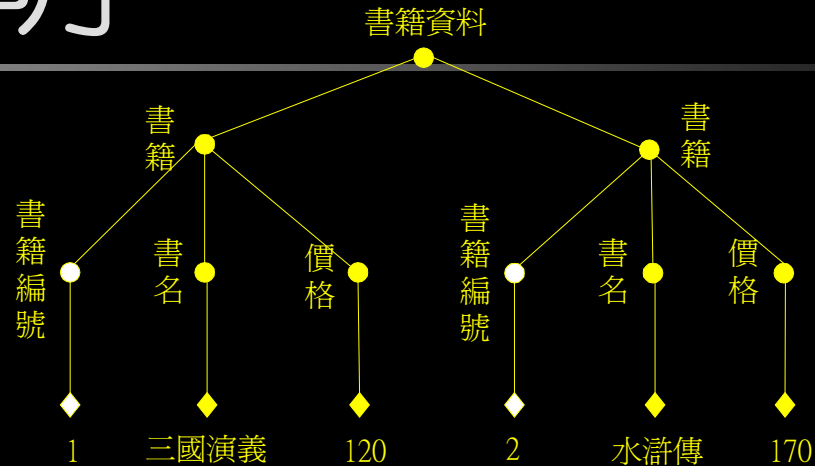
- XML 文件的「元屬性」(Meta-Properties) 描述了 XML 項目的特性，讓我們可以在樹狀結構上，獲得更多的節點資訊。
- OPENXML 所提供的元屬性可以 (見課本 11-32, 33) :
 - 提供所在節點的識別編號 id (以 @mp:id 可取得)
 - 提供所在節點的父節點其識別編號 id (@mp:parentid)。
 - 提供所在節點在原始 XML 文件中的名稱 (@mp:localname)。
 - 提供所在節點的父節點，在原始 XML 文件中的名稱。
 - 提供所在節點的命名空間 URI, 與其前導字串。
 - 提供所在節點的父節點，其命名空間 URI, 與前導字串。
 - 提供目前所在節點的前一個兄弟節點之識別編號 id。
 - 處理 OPENXML 的資料的「溢位」(Overflow)。

範例說明

```

declare @xmltext nvarchar(max)
declare @idoc integer
set @xmltext = '
<書籍資料>
  <書籍 書籍編號="1">
    <書名>三國演義</書名>
    <價格>120</價格>
  </書籍>
  <書籍 書籍編號="2">
    <書名>水滸傳</書名>
    <價格>170</價格>
  </書籍>
</書籍資料>'

```



- : 元素節點
- : 屬性節點
- ◆ : 元素本文節點
- ◆ : 屬性值節點

書籍編號	書名	價格
1	三國演義	120
2	水滸傳	170

```

exec sp_xml_preparedocument @idoc output, @xmltext
select * from OPENXML(@idoc, '/書籍資料/書籍', 3)
  with (書籍編號int, 書名char(20), 價格 int)
exec sp_xml_removedocument @idoc

```



使用 OPENXML 擷取 XML 文件的資料 (範例)

```
declare @xmltext nvarchar(max)
declare @idoc integer
set @xmltext = '
    <Root>
      <row id="1" total_quantity="60"/>
      <row id="2" total_quantity="100"/>
      <row id="3" total_quantity="40"/>
      <row id="4" total_quantity="50"/>
      <row id="5" total_quantity="50"/>
      <row id="6" total_quantity="10"/>
    </Root>'
exec sp_xml_preparedocument @idoc output, @xmltext
select * from OPENXML(@idoc, '/Root/row')
  with (id int, total_quantity int)
exec sp_xml_removedocument @idoc
```

<i>id</i>	<i>total_quantity</i>
1	60
2	100
3	40
4	50
5	50
6	10



另一個查詢範例

```
declare @xmltext nvarchar(max)
declare @idoc integer
set @xmltext = '
```

```
<Root>
```

```
<Books id="1" bookname="三國演義" price="120" author="羅貫中"/>
```

```
<Books id="2" bookname="水滸傳" price="170" author="施耐庵"/>
```

```
<Books id="3" bookname="紅樓夢" price="170" author="曹雪芹"/>
```

```
<Books id="4" bookname="西遊記" price="140" author="吳承恩"/>
```

```
<Books id="5" bookname="水經注" price="120" author="酈道元"/>
```

```
<Books id="6" bookname="道德經" price="190" author="老子"/>
```

```
</Root>'
```

```
exec sp_xml_preparedocument @idoc output, @xmltext
```

```
select * from OPENXML(@idoc, '/Root/Books')
```

```
with (id int, bookname char(50), price int)
```

```
exec sp_xml_removedocument @idoc
```

<i>id</i>	<i>bookname</i>	<i>price</i>
1	三國演義	120
2	水滸傳	170
3	紅樓夢	170
4	西遊記	140
5	水經注	120
6	道德經	190



將上例換成 element-centric

```
declare @xmltext nvarchar(max)
declare @idoc integer
set @xmltext = '
  <Root> <Books>
    <id>1</id><bookname>三國演義</bookname>
    <price>120</price><author>羅貫中</author>
  </Books>
  <Books>
    <id>2</id><bookname>水滸傳</bookname>
    <price>170</price><author>施耐庵</author>
  </Books>
  <Books>
    <id>3</id><bookname>紅樓夢</bookname>
    <price>170</price><author>曹雪芹</author>
  </Books>
  <Books>
    <id>4</id><bookname>西遊記</bookname>
    <price>140</price><author>吳承恩</author>
  </Books>
```

```
<Books>
  <id>5</id><bookname>水經注</bookname>
  <price>120</price><author>酈道元</author>
</Books>
<Books>
  <id>6</id><bookname>道德經</bookname>
  <price>190</price><author>老子</author>
</Books>
</Root>'
```

```
exec sp_xml_preparedocument @idoc output, @xmltext
select * from OPENXML(@idoc, '/Root/Books', 2)
  with (id int, bookname char(50), price int)
exec sp_xml_removedocument @idoc
```

使用參數 *col-pattern*

<書籍資料>

<書籍 書籍編號="1">

<書名>三國演義</書名>

<價格 單位="台幣">120</價格>

</書籍>

<書籍 書籍編號="2">

<書名>水滸傳</書名>

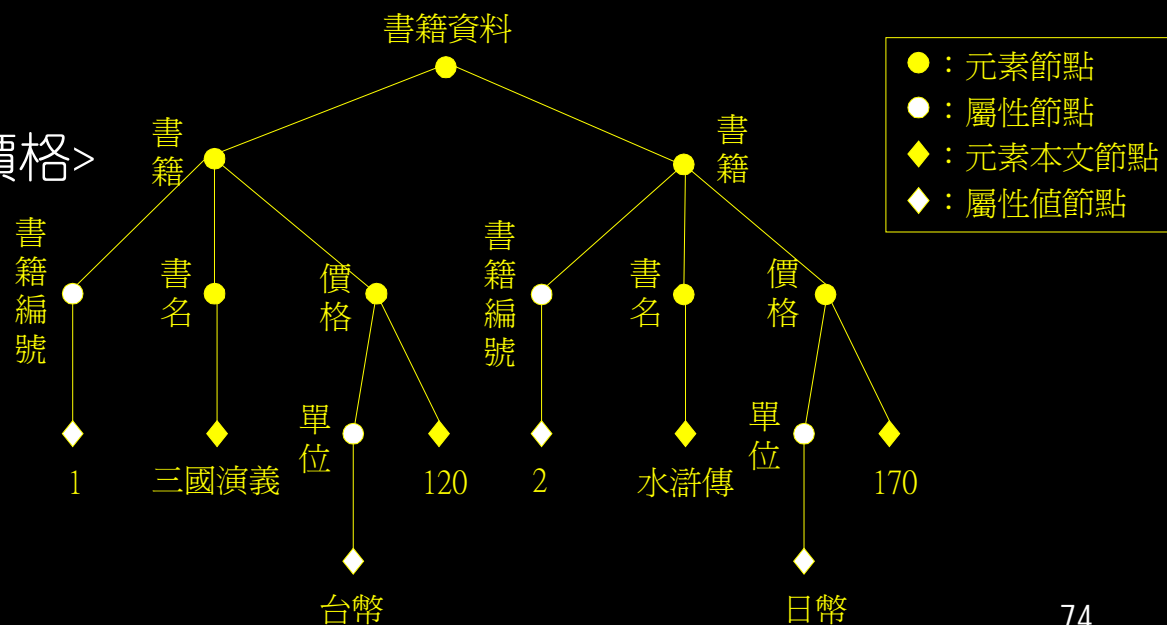
<價格 單位="日幣">170</價格>

</書籍>

</書籍資料>

書籍編號	書名	價格	單位
1	三國演義	120	台幣
2	水滸傳	170	日幣

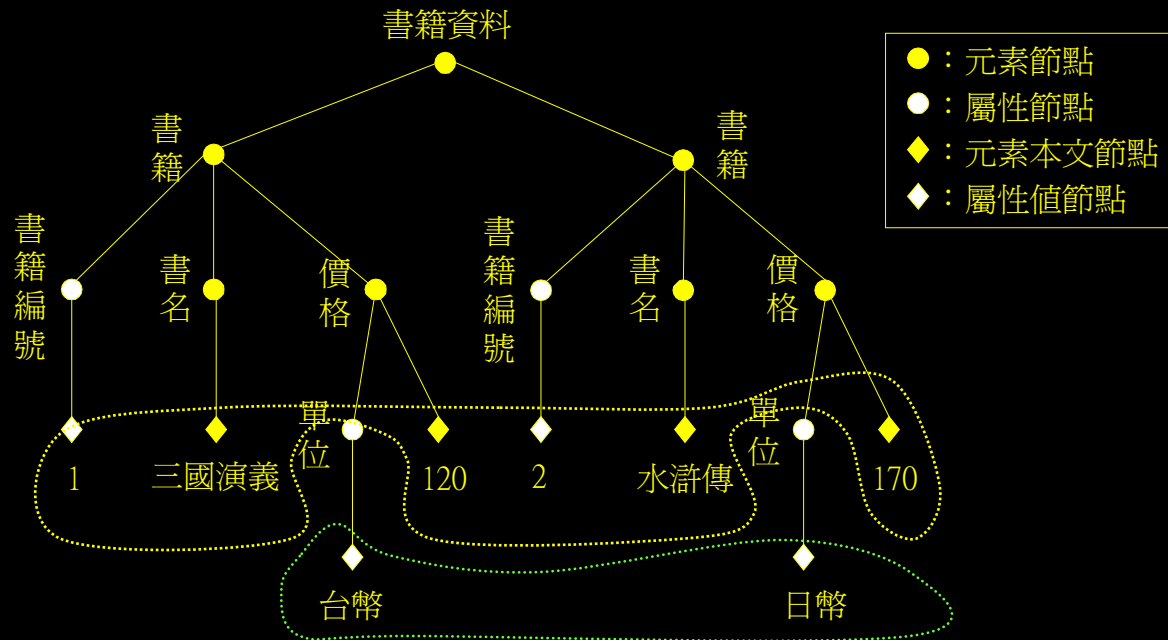
其指令如下頁所示



使用參數 *col-pattern* 的範例

```
declare @xmltext nvarchar(max)
declare @idoc integer
set @xmltext = '
<書籍資料>
<書籍 書籍編號="1">
  <書名>三國演義</書名>
  <價格 單位="台幣">120</價格>
</書籍>
<書籍 書籍編號="2">
  <書名>水滸傳</書名>
  <價格 單位="日幣">170</價格>
</書籍>
</書籍資料>'
```

```
exec sp_xml_preparedocument @idoc output, @xmltext
select * from OPENXML(@idoc, '/書籍資料/書籍', 3)
  with (書籍編號 int, 書名 char(20), 價格 int, 單位 char(8) '!.價格/@單位')
exec sp_xml_removedocument @idoc
```



使用參數 *table-name*

```
if exists (select 1 from sysobjects where name='書局與書籍' and type='U')
    drop table 書局與書籍
create table 書局與書籍
    (書局編號 int, 書局名稱 char(20), 城市 nchar(20), 書籍 char(50))
declare @xmltext nvarchar(max)
declare @idoc integer
set @xmltext = '
<Root>
  <書局書局編號="1" 書局名稱="巨蟹書局" 城市="臺北市" 等級="20">
    <書籍>三國演義</書籍>
  </書局>
  <書局 書局編號="2" 書局名稱="射手書局" 城市="高雄市" 等級="10">
    <書籍>水滸傳</書籍>
  </書局>
</Root>'
exec sp_xml_preparedocument @idoc output, @xmltext
select * from OPENXML(@idoc, '/Root/書局', 3)
  with 書局與書籍
exec sp_xml_removedocument @idoc
```

書局編號	書局名稱	城市	書籍
1	巨蟹書局	臺北市	三國演義
2	射手書局	高雄市	水滸傳

使用參數

metaproperty-name : @mp:id

```
declare @xmltext nvarchar(max)
declare @idoc integer
set @xmltext = '
<Root>
  <Books id="1" bookname="三國演義" price="120" author="羅貫中"/>
  <Books id="2" bookname="水滸傳" price="170" author="施耐庵"/>
  <Books id="3" bookname="紅樓夢" price="170" author="曹雪芹"/>
  <Books id="4" bookname="西遊記" price="140" author="吳承恩"/>
  <Books id="5" bookname="水經注" price="120" author="酈道元"/>
  <Books id="6" bookname="道德經" price="190" author="老子"/>
</Root>'
exec sp_xml_preparedocument @idoc output, @xmltext
select * from OPENXML(@idoc, '/Root/Books')
with (id int, bookname char(50), price int, node_id int '@mp:id')
exec sp_xml_removedocument @idoc
```

<i>id</i>	<i>bookname</i>	<i>price</i>	<i>node_id</i>
1	三國演義	120	2
2	水滸傳	170	7
3	紅樓夢	170	12
4	西遊記	140	17
5	水經注	120	22
6	道德經	190	27

使用參數 *metaproperty-name* : *@mp:xmltext*

```
declare @xmltext nvarchar(max)
declare @idoc integer
set @xmltext = '
```

```
<Root>
  <Books id="1" bookname="三國演義" price="120" author="羅貫中"/>
  <Books id="2" bookname="水滸傳" price="170" author="施耐庵"/>
  <Books id="3" bookname="紅樓夢" price="170" author="曹雪芹"/>
  <Books id="4" bookname="西遊記" price="140" author="吳承恩"/>
  <Books id="5" bookname="水經注" price="120" author="酈道元"/>
  <Books id="6" bookname="道德經" price="190" author="老子"/>
</Root>'
```

```
exec sp_xml_preparedocument @idoc output, @xmltext
select * from OPENXML(@idoc, '/Root/Books', 9)
  with (bookname char(50), price int, other varchar(100) '@mp:xmltext')
exec sp_xml_removedocument @idoc
```

<i>bookname</i>	<i>price</i>	<i>other</i>
三國演義	120	<Books id="1" author="羅貫中"/>
水滸傳	170	<Books id="2" author="施耐庵"/>
紅樓夢	170	<Books id="3" author="曹雪芹"/>
西遊記	140	<Books id="4" author="吳承恩"/>
水經注	120	<Books id="5" author="酈道元"/>
道德經	190	<Books id="6" author="老子"/>



用 OPENXML 對資料庫做異動

- 前面所討論的查詢，都可以搭配
 - select into 、
 - insert into 、
 - delete from ，以及
 - update from 等指令，把 XML 文件的資料拿來對資料庫做新增、刪除、修改的動作。



將 XML 資料新增到關聯表內

```
if not exists (select 1 from sysobjects where name='Books' and type='U')
create table Books
(id int, bookname char(50), author char(30), price int, publisher char(20))
declare @xmltext nvarchar(max)
declare @idoc integer
set @xmltext = '
  <Root>
    <書籍 書籍編號="7">
      <書名>吳建雄傳</書名><作者>江才健</作者><價格>350</價格>
      <出版社>天下出版社</出版社>
    </書籍>
    <書籍 書籍編號="8">
      <書名>楊振寧傳</書名><作者>江才健</作者><價格>400</價格>
      <出版社>天下出版社</出版社>
    </書籍>
  </Root>'
exec sp_xml_preparedocument @idoc output, @xmltext
insert into Books(id, bookname, author, price, publisher)
select * from OPENXML(@idoc, '/Root/書籍', 3)
with (書籍編號 int, 書名 char(50), 作者 char(30), 價格int, 出版社 char(20))
exec sp_xml_removedocument @idoc
```



依 XML 資料刪除關聯表資料

```
declare @xmltext nvarchar(max)
declare @idoc integer
set @xmltext = '
  <Root>
    <訂單 書局編號="1" 書局名稱="巨蟹書局" 城市="臺北市" 等級="20">
      <書籍 書籍編號="2">水滸傳</書籍>
    </訂單>
    <訂單 書局編號="2" 書局名稱="射手書局" 城市="高雄市" 等級="10">
      <書籍 書籍編號="1">三國演義</書籍>
    </訂單>
  </Root>'
exec sp_xml_preparedocument @idoc output, @xmltext
delete from Orders where exists
  (select * from OPENXML(@idoc, '/Root/訂單')
   with (書局編號 int, 書籍編號 int './書籍/@書籍編號') D
   where Orders.no=D.書局編號 and Orders.id=D.書籍編號)
exec sp_xml_removedocument @idoc
```



依 XML 資料修改關聯表資料

```
declare @xmltext nvarchar(max)
```

```
declare @idoc integer
```

```
set @xmltext = '
```

```
<Root>
```

```
<書局 書局編號="1" 書局名稱="高科書坊" 城市="臺北市" 等級="20"/>
```

```
<書局 書局編號="2" 書局名稱="陽明書城" 城市="高雄市" 等級="10"/>
```

```
</Root>'
```

```
exec sp_xml_preparedocument @idoc output, @xmltext
```

```
update Bookstores set name = B.書局名稱
```

```
from OPENXML(@idoc, '/Root/書局') with (書局編號 int, 書局名稱 char(20)) B
```

```
where Bookstores.no=B.書局編號
```

```
exec sp_xml_removedocument @idoc
```



Transact-SQL 支援的 XQuery

- 上節的做法都是說明如何將 XML 資料與關聯式資料庫進行相互的轉換，並進行傳統的 SQL 查詢。
- Transact-SQL 也支援直接在 XML 文件中進行查詢的方法：XQuery。
- 有了 XQuery，即使不透過上述轉換，也可以查詢 XML 文件了。



Transact-SQL 支援的 XQuery

- XQuery 是 W3C 為了讓 XML 文件中的任何元素、屬性值得以被篩選、排序，以及輸出所制定的 XML 查詢標準，
- 相關資料在 <http://www.w3c.org/XML/Query>
- XQuery 指令的語法簡稱為 FLWOR (唸做Flower)，主要是由 For、Let、Where、Order By、Return 五大子句所構成 (但目前 SQL Server 2008未支援 Let 子句)



Transact-SQL 支援的 XQuery

- FLOWR 子句
 - For \$var in <Sequence_or_context>
 - Let \$var := <Sequence_or_context> -- SQL Server 2008 未支援
 - Where <Expression>
 - Order By <Expression>
 - Return <Literal_and_nodes>
- 上述語法要放在 7.10.5 節所提到的 exist(XQuery) 方法中
- 在 <Sequence_or_context> 、 <Expression> 或 <Literal_and_nodes> 中可配合 XPath 指定要篩選的目標



直接用 XPath 過濾部份元素的指令

```
declare @xmltext nvarchar(max)
```

```
declare @xml xml
```

```
set @xml = '
```

```
<Root>
```

```
<Books id="1" bookname="三國演義" price="120" author="羅貫中"/>
```

```
<Books id="2" bookname="水滸傳" price="170" author="施耐庵"/>
```

```
<Books id="3" bookname="紅樓夢" price="170" author="曹雪芹"/>
```

```
<Books id="4" bookname="西遊記" price="140" author="吳承恩"/>
```

```
<Books id="5" bookname="水經注" price="120" author="酈道元"/>
```

```
<Books id="6" bookname="道德經" price="190" author="老子"/>
```

```
</Root>'
```

```
select @xml.query('/Root/Books[@price < "150"]')
```

結果為

```
<Books id="1" bookname="三國演義" price="120" author="羅貫中" />
```

```
<Books id="4" bookname="西遊記" price="140" author="吳承恩" />
```

```
<Books id="5" bookname="水經注" price="120" author="酈道元" />
```



完整的 XQuery 指令

```
declare @xmltext nvarchar(max)
```

```
declare @xml xml
```

```
set @xml = '
```

```
<Root>
```

```
<Books id="1" bookname="三國演義" price="120" author="羅貫中"/>
```

```
<Books id="2" bookname="水滸傳" price="170" author="施耐庵"/>
```

```
<Books id="3" bookname="紅樓夢" price="170" author="曹雪芹"/>
```

```
<Books id="4" bookname="西遊記" price="140" author="吳承恩"/>
```

```
<Books id="5" bookname="水經注" price="120" author="酈道元"/>
```

```
<Books id="6" bookname="道德經" price="190" author="老子"/>
```

```
</Root>'
```

```
select @xml.query('for $i in /Root/Books
```

```
  where data($i/@id) > "3"
```

```
  order by $i/@id
```

```
  return <GoodBooks>{data($i/@bookname)}</GoodBooks>')
```

```
<GoodBooks>西遊記</GoodBooks>
```

```
<GoodBooks>水經注</GoodBooks>
```

```
<GoodBooks>道德經</GoodBooks>
```



結果



將上述結果加上 <NewRoot> 當根節點

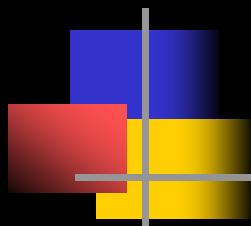
修改 select 指令為：

```
select @xml.query('<NewRoot> { for $i in /Root/Books  
  where data($i/@id) > "3" order by $i/@id  
  return <GoodBooks>{data($i/@bookname)}</GoodBooks>} </NewRoot>')
```

結果為



```
<NewRoot>  
  <GoodBooks>西遊記</GoodBooks>  
  <GoodBooks>水經注</GoodBooks>  
  <GoodBooks>道德經</GoodBooks>  
</NewRoot>
```



本章結束
The End.