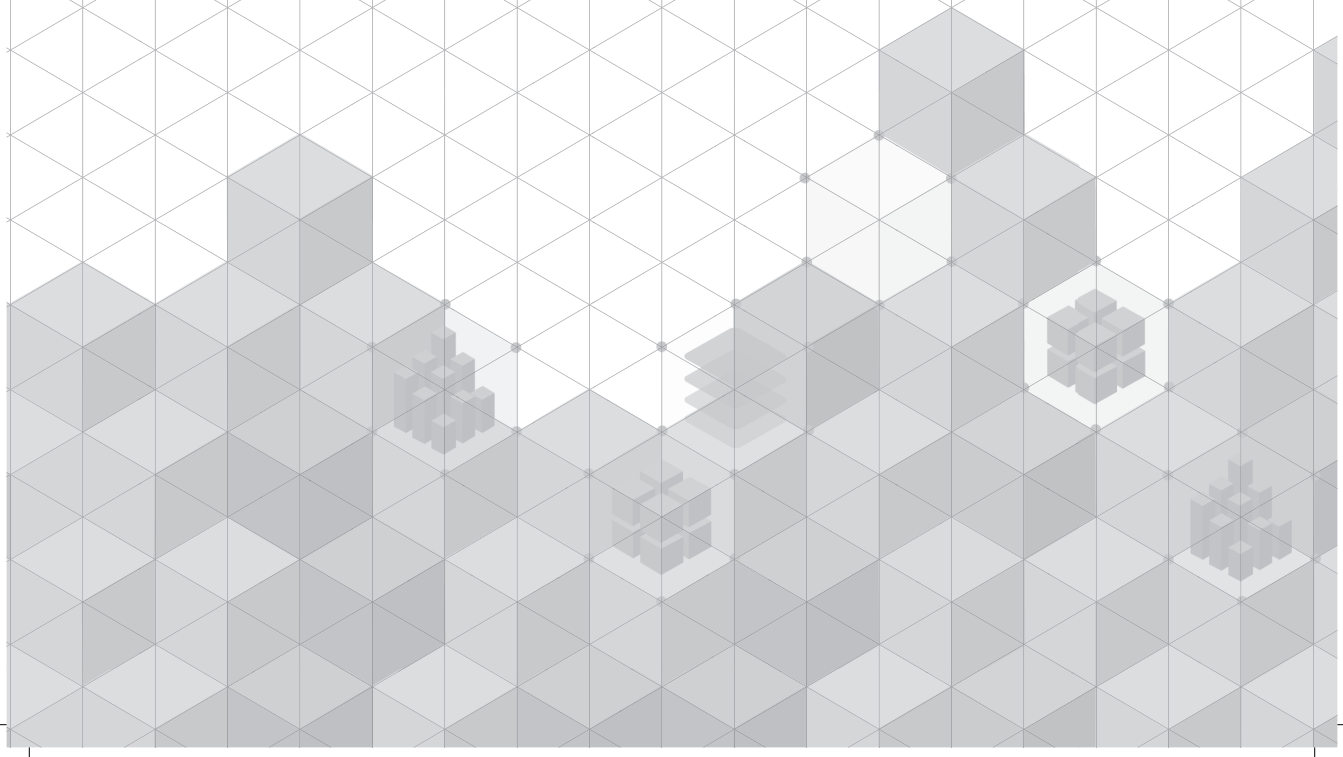
SQL Server 2012
T-SQL資料庫設計

第10章BISM表格式模型管理與效能

表格式模型的日常維運
BISM表格式模型記憶體使用最佳化
BISM表格式模型效能監控與管理



我們已經介紹過如何設計、部署、處理表格式模型。接下來，在本章中，我們要討論的是上線後的日常維運、管理與效能調校等議題。

10-01 表格式模型的日常維運

10-1-1 表格式模型備份

在系統維護議題之中，備份與還原是不可或缺的一環。備份表格式模型的方法與備份多維度模型的方式完全相同，只需開啟SSMS，點選欲備份之資料庫按右鍵，選取「備份」後，如圖10-1。

圖10-1：設定備份

此時，會彈出「備份資料庫」視窗，即可定義備份選項。

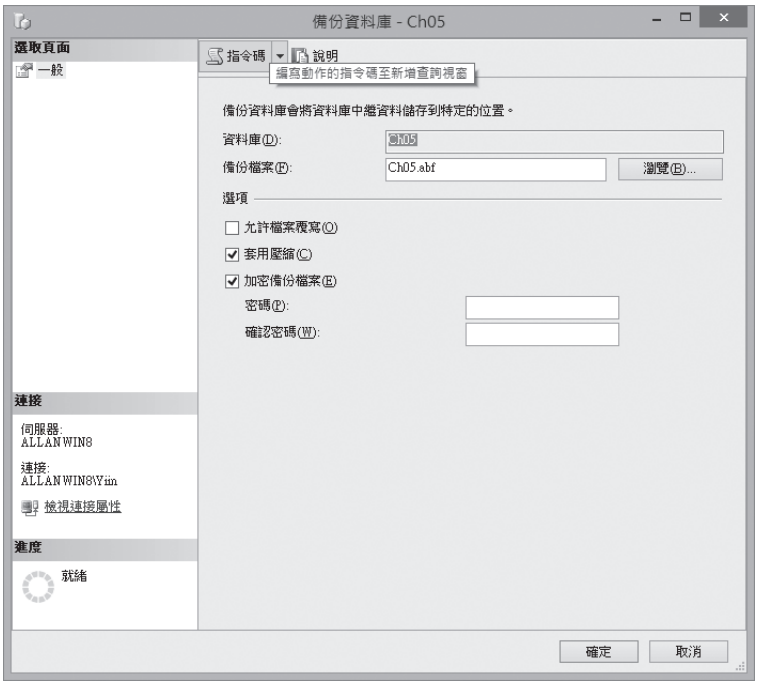


圖10-2：備份選項

備份設定選項包括：

- **備份檔案**：指定備份檔案存放之路徑和檔名。點選「瀏覽」即可檢視儲存的路徑。根據預設，只能將備份檔儲存在安裝路徑下的「MSAS11.MSSQLSERVER\OLAP\ Backup」以及「MSAS11.MSSQLSERVER\OLAP\ Log」資料夾，如果希望能夠開放其他的資料夾，可至伺服器屬性（利用SSMS點選伺服器名稱按右鍵選取「屬性」，並勾選「進階」），將AllowedBrowsingFolders屬性加入欲顯示的路徑即可（路徑間是利用「|」符號做分隔符號）。

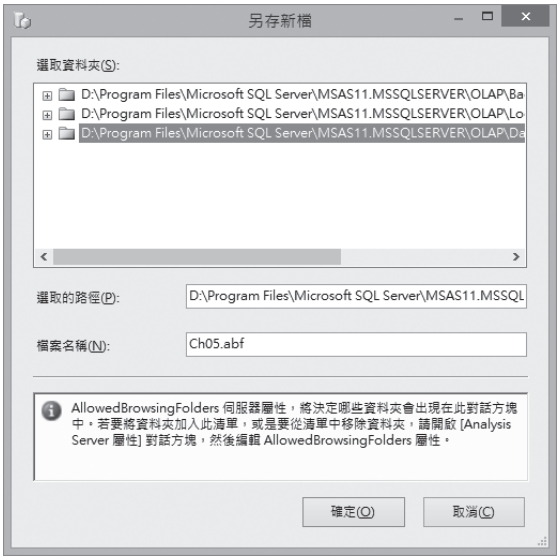


圖10-3：備份路徑

■ 允許檔案覆寫：可覆寫目錄下之同名備份檔案。

■ 套用壓縮：可針對備份資料進行壓縮以節省儲存空間。

■ 加密備份檔案：可輸入密碼以控管還原權限。

請注意，若您的表格式模型是xVelocity模式，則可以同時備份中繼資料以及彙總資料；若您的表格式模型是DirectQuery模式，則只能備份中繼資料。

若希望能透過排程的方式定期備份表格式模型，您可以在設定完所有備份選項後，如圖10-2，點選上方工具列的「指令碼」選項，即可將設定內容匯出為XMLA指令碼。有了這個指令碼，我們就可以利用SSIS中的「Analysis Services執行DDL工作」來執行指令，方法請參閱本書《第9-2-2節：整合SSIS進行模型處理》。

```
<Backup xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">
  <Object>
    <DatabaseID>Ch05</DatabaseID>
  </Object>
  <File>Ch05.abf</File>
</Backup>
```

10-1-2 表格式模型還原

至於還原資料庫同樣是利用SSMS，你只需點選欲還原之伺服器，按右鍵後選取「還原」，即可進入還原設定視窗，如圖10-4。

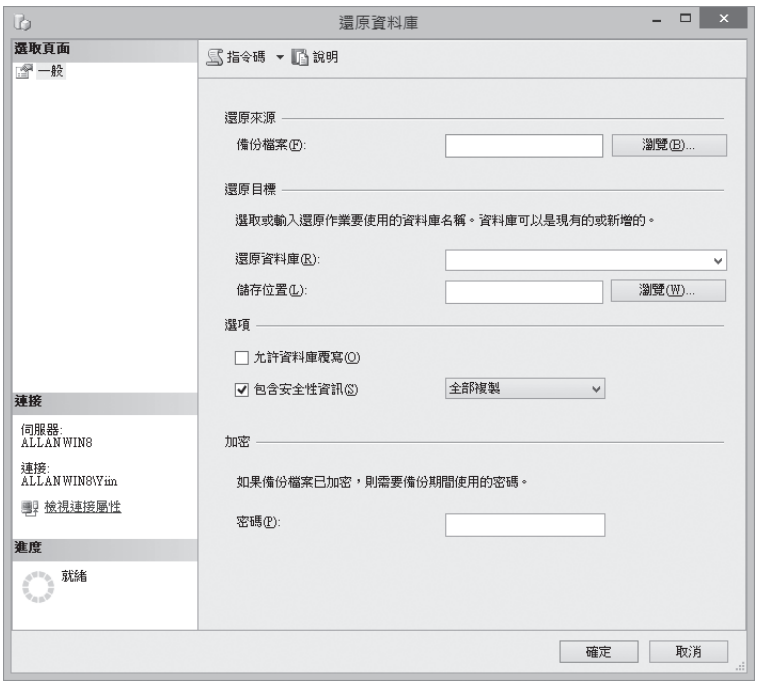


圖10-4：還原設定視窗

在還原設定視窗的「一般」頁籤中設定值如下：

- **還原來源**：指定還原檔案之路徑。
- **還原資料庫**：可利用下拉式選單指定欲覆寫之資料庫，或自行輸入還原資料庫名稱。
- **允許資料庫覆寫**：可以覆寫既有之資料庫。
- **包含安全性資訊**：指是否將備份資料檔之安全性設定覆寫既有資料庫，選項包括「全部複製」與「略過成員資格」。
- **加密**：如果資料庫備份有加密，則需輸入密碼始可還原。

在此需要注意的是，在SQL Server 2012中，不同模式的Analysis Services備份檔雖然都是.abf檔，但彼此間是無法通用的。若您將表格式模型的備份檔還原至多維度伺服器，就會發生如圖10-5的錯誤訊息。

圖10-5：還原至不正確的Analysis Services執行個體

10-1-3 表格式模型的事件記錄

由於表格式模型在這一版本並未提供事件記錄的功能，雖然表格式模型與多維度模型相同，但在點選資料庫圖示按右鍵選取「屬性」後，會出現如圖10-6的屬性式窗。裡面的確存在可設定QueryLog的功能，並預留這些屬性，但目前表格式模型並未提供查詢記錄之功能，連帶地也未提供包括飛行記錄器（Flight Recorder）等機制，僅在多維度模型中可以使用。

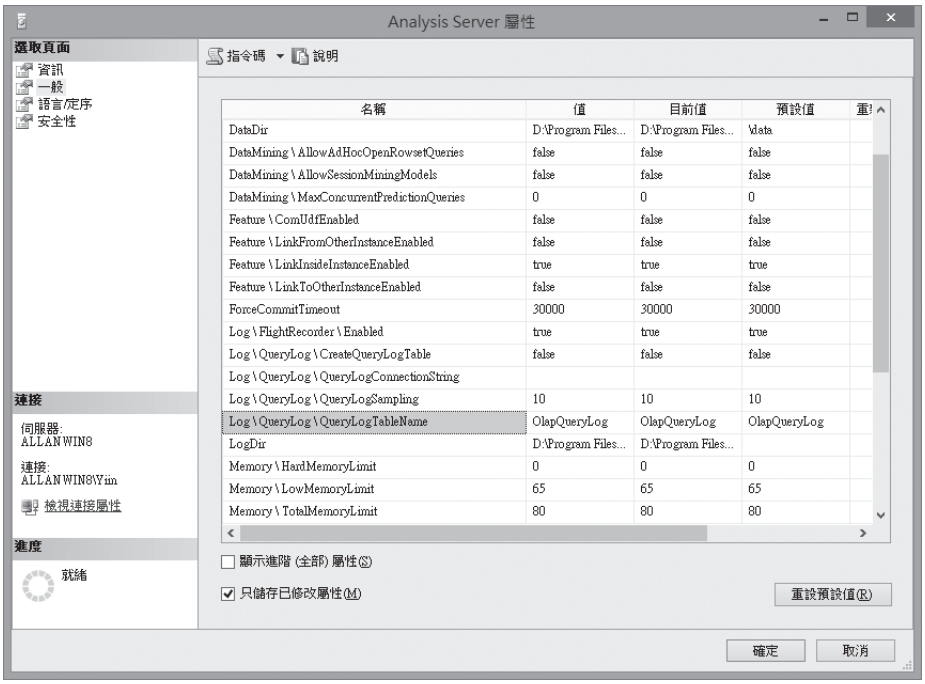


圖10-6：查詢事件檔屬性

10-02 BISM表格式模型記憶體使用最佳化

既然表格式模型是基於記憶體運算技術，所以表格式模型的效能會與記憶體耗用密切相關。在本節中，我們將先介紹表格式模型如何耗用記憶體，並提出在設計表格式模型時，該注意的設計準則。

10-2-1 表格式模型的記憶體管理原則

在評估xVelocity的記憶體耗用量時，應考慮到以下三個層面：

- 資料儲存記憶體耗用量
- 資料查詢記憶體耗用量
- 資料處理記憶體耗用量

大多數人在評估導入表格式模型時，僅考量到**資料儲存記憶體耗用量**。資料儲存的確是導入表格式模型時最重要的考量重點，畢竟xVelocity的儲存體都是存放於記憶體之中。但相較於一般資料庫的硬碟耗用評估，其硬碟耗用空間與資料的筆數是呈正比的，但xVelocity的記憶體耗用，卻非呈現線性的關係。這是因為xVelocity是以資料行為基礎的儲存結構，它會先將資料行內的值編列索引，再透過索引值代換原始資料值，以達到資料壓縮的效果。因此，當資料筆數倍增時，記憶體的耗用並不會是倍數的增長。

會影響xVelocity儲存體記憶體耗用的因素包括了：

- 資料行的數量（在xVelocity儲存體中，資料行倍增的影響會比資料列倍增來的大）。
- 資料行內值的多元性（這會影響資料行基礎索引的大小）。
- 資料型別（這會影響當透過索引值代換原始值的壓縮效率，我們將於《第10-2-3節：資料型別選擇》中再深入討論）。
- 資料列的筆數。

目前這些因素並沒有可套用的公式來協助我們算出預估的記憶體耗用量，所以筆者還是比較建議各位採用實證法來評估記憶體耗用狀態，也就是準備資料規模相近的測試資料來進行評估。我們在後續的章節中，將介紹如何利用動態管理檢視（DMV）來取得各資料表的記憶體耗用，以供評估使用。但在準備測試資料時，

必須要注意各資料行的值域分布應盡量與實際狀態吻合，以免低估了資料行索引的大小。

此外，需要注意的是，由於開啟SSDT的表格式模型時，系統會同時建立一個暫存的資料庫，這部分也會造成記憶體的耗用。因此，比較建議的方式是在正式上線環境透過佈署精靈來執行上線佈署（操作方法詳見《第9-1-3節：使用部署精靈部署模型》），來避免額外的記憶體耗用。

資料處理記憶體耗用量是經常被忽略的另一個重要的記憶體耗用考量重點。我們在此列舉幾種經常發生的情境，來說明其對應的記憶體耗用：

- **使用「完整處理」來處理整個資料庫**：毫無疑問的，此模式是最耗用記憶體的方法。完整處理是以「交易」的模式來執行，因此，當交易認可之前，原有記憶體中的xVelocity並不會消滅，但另一方面，系統又從資料源讀入資料產生新的xVelocity儲存體。故使用完整處理的模式起碼會需要資料儲存記憶體耗用量的兩倍（這還未計算執行作業過程中，所需的記憶體），所以，這也是筆者最不建議的執行模式。
- **先使用「處理清除」清除資料表內的資料，再利用「完整處理」來處理整個資料庫**：此做法當然不會耗用兩倍的記憶體，但相對地，在處理模型的過程中，使用者將無法存取資料模型（如果是夜間執行批次的話，影響不大；若是日間要定期更新的話，就不建議採用此種模式）。
- **維度資料表使用「完整處理」，較龐大的事實資料表僅「完整處理」較近的資料分割**：這種模式會是比較合理的處理方式，因為維度資料表通常資料筆數不多。若為較大的事實資料表，透過合理的資料分割便能縮限完整處理的範圍，耗用的記憶體也只有資料分割的兩倍。
- **各資料表依序執行「處理資料」，然後在執行資料庫層級的「處理重新計算」**：使用這種模式的重點在於「依序」執行，所以記憶體的耗用會最低（僅需總體儲存體記憶體耗用+最大資料表記憶體耗用即可），但相對來說，這種模式處理時間也會被拖長。

各位在考慮記憶體耗用時，務必將所採取的處理模式納入考量。此外，對多數的企業來說，在夜間執行處理作業的好處是，此時並不會有人執行查詢。因此「處理記憶體耗用」與「查詢記憶體耗用」兩者僅需取其大者即可。但若上線環境會有上班時間資料更新的需求（例如，每小時更新一次），兩者就必須疊加考量。

資料查詢記憶體耗用量其實是最難估算的一塊。如果是一般應用程式，由於裡面存取資料庫的操作是固定的，因此，可透過Profiler錄製或分析執行計畫以調整查詢語法的方式來提升效能。但商業智慧平台卻很難以預料使用者可能採行的查詢模式，再者，大多數使用者並不會自己下語法，而是透過視覺化工具（例如，Power View）來產出執行語法，所以語法的複雜度與調整其實不具太大意義。因而這部分記憶體的需求仍需以實證法的方式來評估，如可透過測試環境中的壓力測試，來評估各項查詢的記憶體耗用。

10-2-2 資料模型結構

在上一節中，曾討論到記憶體耗用的主要因素，但我們能否在一開始設計資料模型的階段，就降低模型的記憶體耗用呢？在此，我們先列出幾項比較容易做到的基本準則：

- **若為沒有分析意義的資料行，就從資料模型中刪除**：自資料模型中刪除，自然就不會耗用記憶體。
- **除了鍵值資料行外，盡量不要放入不會重複的資料行**：xVelocity會針對每個資料行先編列索引值，因此，當該資料行內沒有重複值時，索引的記憶體耗用會最大。鍵值資料行肩負著資料庫關聯的重責大任，無法省掉。但其他非重複性資訊資料行，例如，敘述性文字…等，就不建議放入模型之中。
- **透過資料行分割，來降低資料行的多元性**：有時資料行是由幾個具意義的區段所構成的，例如，銀行業的貸放序號通常是由「產品代號」+「分行代號」+「序號」所構成，因此，產生的貸放序號排列組合數就會暴增，而增加了記憶體耗用量。較好的方法是拆分成三個資料行，讓各個資料行的多元性降低，這樣三個資料行的資料行索引總和，會比原有的一個資料行還來得小，便能大幅節省記憶體耗用。當分析需顯示完整的貸放序號時，則可以自訂一個新的導出資料行（將這三個資料行串接）。基本上，若串接的邏輯不複雜，並不會導致過多的查詢記憶體耗用，卻能有效減少總體的資料儲存記憶體耗用。其他如時間戳記資料行，也建議拆分為日期資料行與時間資料行，以有效降低儲存體大小。
- **減少意義重複的資訊**：例如，在事實資料表中，可能同時存放著「銷售數量」、「單價」以及「銷售金額」三個資料行，但其實「銷售金額」=「銷售

數量」*「單價」，所以只需保存兩個資料行，就能夠呈現完整的資訊。建議可移除實體的「銷售金額」資料行，但在此資料表中加入「銷售金額」的導出資料行。由於導出資料行耗用的是查詢時的記憶體，但因為公式並不複雜，不會造成查詢時的負擔，反而能有效地降低總體資料儲存體的大小。

- **以圖檔Url取代二進位圖檔資料行**：圖檔Url是以字串的方式儲存自然記憶體耗用，會較二進位圖檔資料行少許多，但相對來說，圖檔的產製會在查詢階段執行，所以耗用的查詢記憶體也會較高。

10-2-3 資料型別選擇

在SQL Server資料庫中，資料型別的選擇與資料庫的空間使用息息相關，光是數值型別，就有一系列可供選擇。不過，在表格式模型中，資料型別大致可分為兩大類型：

- **數值型態**：包括數值（最多64位）、十進位數字（最多64位與17位小數）、貨幣（最多64位整數值與4位小數）、日期、二進位等資料型別。
- **字串型態**：字串型別。

數值型態的資料平均儲存成本為20~30 bytes（索引值+壓縮後資料值），每百萬筆記錄（一個數值資料行）的記憶體耗用量約為20~30 MB。實際值仍會受到該數值資料行內值的分布模式而有所差異。

至於字串型態的資料型別，儲存時會先壓縮再編列索引值，然後進行索引值替換。一般字串的基礎儲存成本為16 bytes，若字串長度更長，常見的儲存成本估算方式為16+0.5*字串長度。也就是說，若一個資料行平均字串長度為50，則計算的平均儲存成本為41 bytes，則每百萬筆記錄的記憶體耗用量約為41 MB左右。

基本上，除非過長的字串，否則字串的儲存成本（16 bytes）會略低於數值（20~30 bytes）。然而這意味著，我們應該將數值資料行以字串來儲存嗎？在此整理出兩個評估重點：

- 如果數值資料行涉及數學運算，那麼還是建議各位以數值型態資料型別來儲存，雖然數值型態的記憶體耗用較高，但執行運算的速度相對較快，特別是以DAX來看。將數值以字串型態儲存的做法，雖省了資料儲存，但卻多了查詢耗用，總和來看，其實並未討到什麼便宜。

■ 如果數值型態資料行不涉及數學運算，例如，為代碼、序號…等資料行。筆者建議將這些資料行的資料型別改為字串，可省下不少資料儲存記憶體耗用。

另一個與資料型別相關的重點在於資料型別的精確度。其中，影響最大的是日期型態的資料（不會有公司為了節省記憶體，而讓數字產生誤差），因為日期資料行實際是以浮點數的方式來儲存，如果資料裡面包含時分秒，甚至於毫秒，都會造成檔案暴增。若此資料行僅需分析到日，建議可在資料源頭將時間部分截掉，便能省下很多的記憶體耗用。若希望執行時段的分析，則建議將時間與日期分開成兩個資料行，並將時間資料行保留至時段分析所需的精確度即可。

10-03 BISM表格式模型效能監控與管理

以下我們要來介紹表格式模型記憶體相關的管理設定，與記憶體效能監控工具。

10-3-1 記憶體相關設定

伺服器屬性中包含了一系列與記憶體相關的設定值。若要檢視伺服器屬性，請點選伺服器圖示按右鍵，選取「屬性」即可開啟「Analysis Services屬性」視窗。勾選畫面下方的「顯示進階（全部）屬性」即可顯示所有的伺服器屬性。

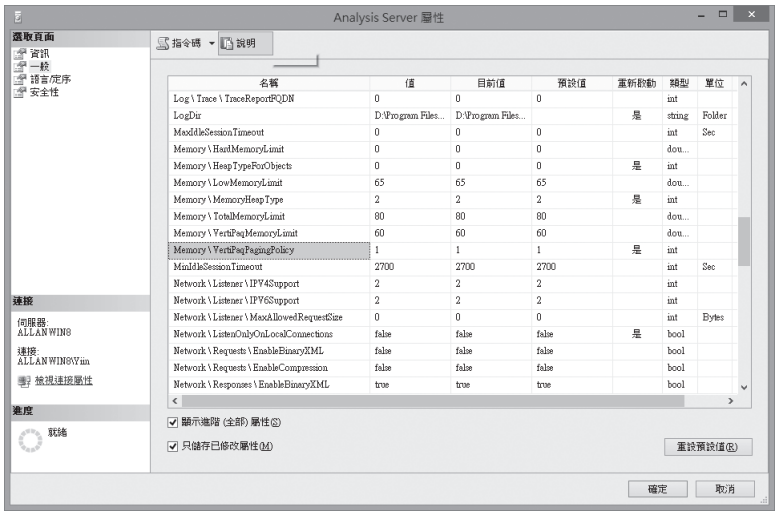


圖10-7：記憶體相關屬性

首先，最重要的設定值是「Memory / VertiPaqPagingPolicy」，這個屬性共有兩個可用的設定值：

- 0：表示鎖定，鎖定xVelocity（VertiPaq）資料能存放於實體記憶體中。
- 1：表示允許系統分頁。僅鎖定記憶體中的雜湊索引，其餘xVelocity（VertiPaq）資料則可以存放於虛擬記憶體。

當系統記憶體資源不足時，作業系統會在硬碟上切割出一塊區域，用以模擬實體記憶體，將實體記憶體中放置過久，或是較無急切性的資料挪放置此區域。以邏輯上來說，等同於加大實體記憶體的容量，使程式在執行時，較不受實體記憶體的容量所限制。當「Memory / VertiPaqPagingPolicy」設定為0時，表示不使用虛擬記憶體；預設值為1，表示除了索引一定得放在實體記憶體之外，其餘資料揭允許放置在虛擬記憶體中。但要注意的是，使用虛擬記憶體雖然看來可讓系統使用超過實體上限的記憶體資源，但畢竟仍是透過模擬的方式來達成，反而會造成效能低落。筆者個人比較建議將此屬性值設定為0。

此外，以下3個屬性可用來定義系統記憶體壓力區：

- Memory / HardMemoryLimit ：預設值為0
- Memory / LowMemoryLimit：預設值為65
- Memory / TotalMemoryLimit ：預設值為80

- HardMemoryLimit是系統能夠使用的記憶體上限。當Analysis Services耗用的記憶體超過此門檻值，系統將會開始執行中斷連線階段，以釋放出記憶體。而被中斷的連線階段，則會收到標示因記憶體壓力而被終止連線階段之錯誤訊息。如果HardMemoryLimit設定為0，則預設值為TotalMemoryLimit與實體可用記憶體的中間值。
- LowMemoryLimit是用來定義系統記憶體壓力區的門檻值下限。當系統記憶體耗用高於LowMemoryLimit，則系統會開始清除快取資料。
- TotalMemoryLimit是定義系統記憶體壓力區的門檻值上限。超過此值，表示系統呈現高度記憶體負載狀態。若是記憶體耗用超過此門檻值，系統將會自動清除所有的快取。

請注意，LowMemoryLimit的設定值應該要低於TotalMemoryLimit設定值，且TotalMemoryLimit的設定值也應該要低於HardMemoryLimit設定值。

上述三個門檻值是用來定義整個Analysis Services的記憶體耗用門檻值。至於VertiPaqMemoryLimit則是定義xVelocity（VertiPaq）儲存體的可用實體記憶體上限。**這四個門檻值若是數值低於100，則表示百分比；若是數值高於100，則表示「位元數」，也就是16,000,000,000代表16GB。**

若是VertiPaqPagingPolicy設定為1，此時，放置在虛擬記憶體的部分是不會納入以上限額的計算，相對來說，也就比較不會被記憶體清理機制給清除。但這會給系統帶來更嚴重的系統資源負擔。

10-3-2 使用動態管理檢視管理記憶體耗用

設定了記憶體管理上限之後，該如何掌握到底是哪個資料庫、哪個資料表造成了較重的記憶體負擔？在Analysis Services中，提供了動態管理檢視（DMV, Dynamic Management View）的機制，可提供更深入的管理資訊。

查詢動態管理檢視的方法很簡單，只需執行以下語法：

```
DAX範例：查詢動態管理檢視
select *
from $system.Discover_object_memory_usage
```

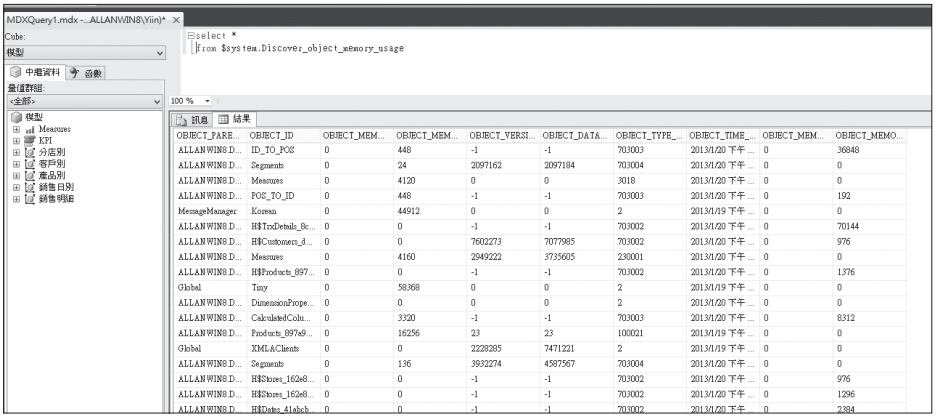


圖10-8：查詢動態管理檢視

Discover_object_memory_usage檢視的資料表規格如下：

資料行名稱	資料型別	資料行描述
OBJECT_PARENT_PATH	DBTYPE_WSTR	此物件的父層路徑
OBJECT_ID	DBTYPE_WSTR	物件編號
OBJECT_MEMORY_SHRINKABLE	DBTYPE_I8	此物件所耗用的可壓縮記憶體，數值應該皆為0
OBJECT_MEMORY_ NONSHRINKABLE	DBTYPE_I8	此物件所耗用的記憶體
OBJECT_VERSION	DBTYPE_I4	此物件之版本號，當此物件變動此版本號，即會更新累加
OBJECT_DATA_VERSION	DBTYPE_I4	此物件中資料的歷程編號。當物件處理時，則版本號會累加
OBJECT_TYPE_ID	DBTYPE_I4	
OBJECT_TIME_CREATED	DBTYPE_DBTIMESTAMP	物件建立之UTC伺服器時間
OBJECT_MEMORY_CHILD_ SHRINKABLE	DBTYPE_I8	此物件之子物件所耗用的可壓縮記憶體，數值應該皆為0
OBJECT_MEMORY_ CHILD_ NONSHRINKABLE	DBTYPE_I8	此物件之子物件所耗用的記憶體

表10-1：Discover_object_memory_usage資料表規格

其中，OBJECT_PARENT_PATH定義了記憶體耗用的物件父子式關係階層，並可使用DAX語法中的PATH函數來拆解其階層關係。為了簡化動態管理檢視，筆者在本書範例中，準備了名為「BISMServerMemoryReport.xlsx」的活頁簿，便是利用PowerPivot存取動態管理檢視，並於其中加入DAX公式，以整理各個物件階層與記憶體耗用數據，再透過樞紐分析表以儀表板的方式呈現。各位之後可自行調整PowerPivot中的Analysis Services執行各體連線字串，作為日後管理使用。

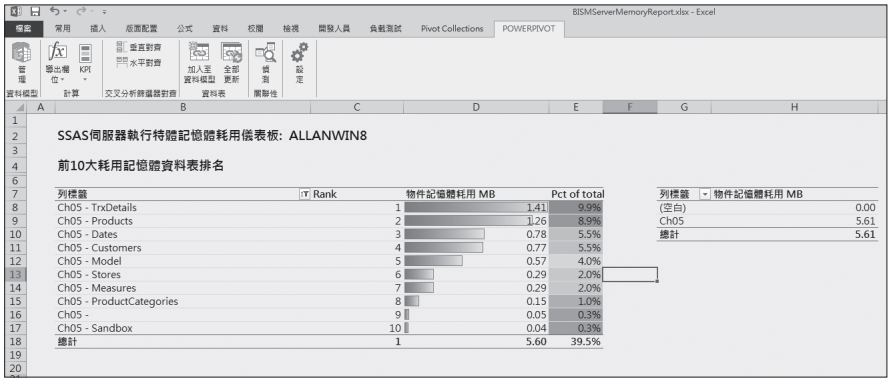


圖10-9：BISMServerMemoryReport

10-3-3 使用效能計數器管理記憶體耗用

另一種用來追蹤伺服器使用狀態的方式就是利用Windows效能監視器（Windows Performance Monitor, 效能監視器）。正如SQL Server Profiler的作用，效能監視器可用來即時監控本機或遠端伺服器的使用狀態，並捕捉事件至檔案中，以便日後進行分析。

與使用Profiler不同的是，無法使用效能監視器識別效能問題的原因。舉例來說，效能監視器可能顯示高CPU使用率，但卻無法顯示傳送至伺服器的DAX查詢。使用效能監視器最常見的狀況在於，當需要實作高層級的效能或負載測試，但卻必須再花額外的心力才能找出效能瓶頸的原因。

在安裝SSAS過程中，同時也安裝了一系列效能計數器，以處理必要的伺服器統計值，包括：快取、連線管理、記憶體使用率、DAX查詢覆蓋率、處理等等。以下列舉一些可供效能測試使用的重要的效能計數器。

屬性	描述
MSAS11:Memory / VertiPaq Memory-mapped KB:	xVelocity儲存體於總體記憶體之使用量
MSAS11:Memory / VertiPaq None-Paged KB:	xVelocity儲存體於實體記憶體之使用量
MSAS11:Memory / VertiPaq Paged KB:	xVelocity儲存體於虛擬記憶體之使用量

表10-2：重要效能計數器（續）

屬性	描述
MSAS11:Memory Limit Hard kb	對應到伺服器屬性HardMemoryLimit，此數值應不會隨時間變動，可用來做為參考值
MSAS11:Memory Limit High kb	對應到伺服器屬性TotalMemoryLimit，此數值應不會隨時間變動，可用來做為參考值。
MSAS11:Memory Limit Low kb	對應到伺服器屬性LowMemoryLimit，此數值應不會隨時間變動，可用來做為參考值。
MSAS11:Memory Limit VertiPaq kb	對應到伺服器屬性VertiPaqMemoryLimit，此數值應不會隨時間變動，可用來做為參考值。
MSAS11:Memory\Memory Usage KB	顯示伺服器處理的記憶體使用量
MSAS11: Connection\Current	
Connections	連接至伺服器的連接數量

表10-2：重要效能計數器

以下為設定效能監視器以顯示SSAS效能計數器的步驟：

Step01：使用命令提示字元中輸入「perfmon」，以開啟效能監視器。

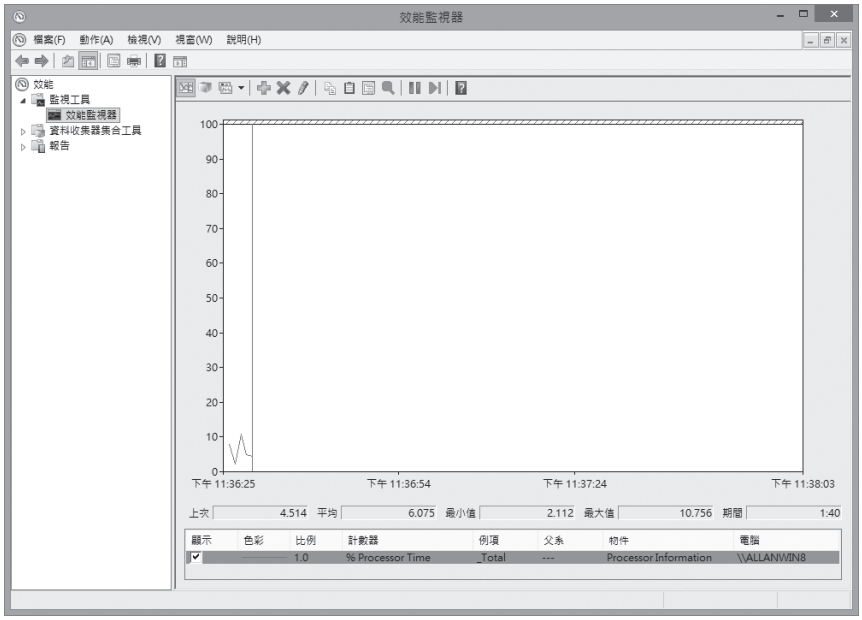


圖10-10：SSAS效能監視器

Step02：點選「+」工具列按鈕，啟動效能計數器。

Step03：在「從下列電腦選取計數器」下拉式選單中輸入伺服器名稱。

Step04：展開效能物件下拉式選單，將捲軸拉至MSAS11效能物件類別。

Step05：選取SSAS效能物件，像是MSAS11: Memory。

Step06：在「從清單選取計數器」清單方塊中，選取想要監控的計數器，點選「解說」按鈕，可看到該選取效能計數器的簡要說明。

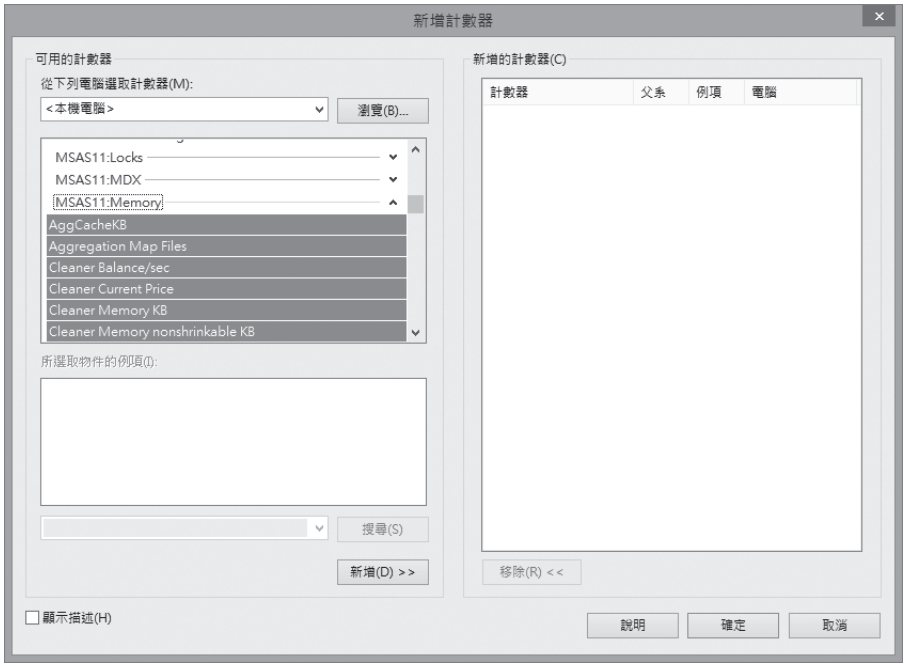


圖10-11：SSAS效能計數器

Step07：點選「加入」以加入該校能計數器至效能監視器。點選「關閉」以關閉「加入計數器」對話方塊回到效能監視器，即可看到效能計數器隨時間變動的狀況。

在格子中選取效能計數器，就能觀察到效能計數器藉著監控時間跨時的「上次」、「平均」、「最小」以及「最大」值。

不過，在執行SSAS伺服器效能測試時，應先清除所有的查詢結果快取，以避免影響到測試結果。

此時可使用ClearCache XMLA命令。舉例來說，若要清除「Ch05」資料庫的查詢結果快取，可使用以下XMLA查詢陳述句以清除快取，以確保獲得最接近真實的效能數據。

```
<Batch xmlns="http://schemas.microsoft.com/
analysisservices/2003/engine">
<ClearCache>
<Object>
<DatabaseID> Ch05</DatabaseID>
</Object>
</ClearCache>
</Batch>
```