如何寫出高效能 TSQL – 深入淺出 SQL Server Relational Engine (含 SQL 2014 in-memory Engine)

**本文將分成四大單元，分別帶您了解：**

[**簡介**](https://blogs.technet.microsoft.com/technet_taiwan/2015/01/16/tsql-sql-server-relational-engine-sql-2014-in-memory-engine/#a)

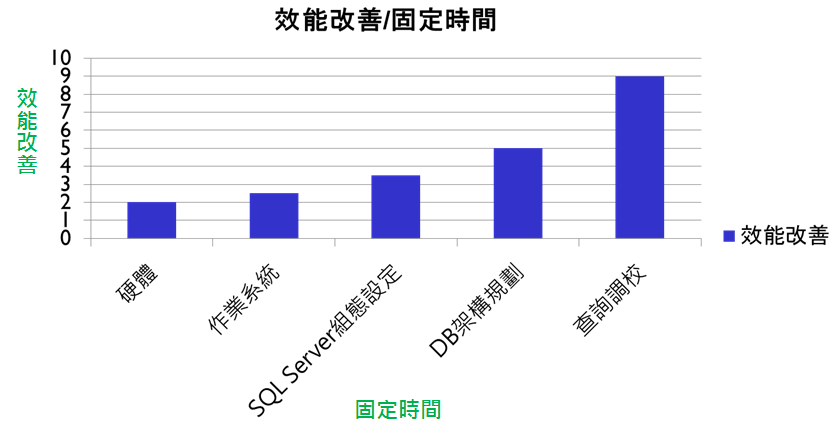
[**SQL Server 關聯式引擎**](https://blogs.technet.microsoft.com/technet_taiwan/2015/01/16/tsql-sql-server-relational-engine-sql-2014-in-memory-engine/#b)

[**SQL 2014 in-memory OLTP 引擎**](https://blogs.technet.microsoft.com/technet_taiwan/2015/01/16/tsql-sql-server-relational-engine-sql-2014-in-memory-engine/#d)

[**進階學習**](https://blogs.technet.microsoft.com/technet_taiwan/2015/01/16/tsql-sql-server-relational-engine-sql-2014-in-memory-engine/#c)

**簡介**

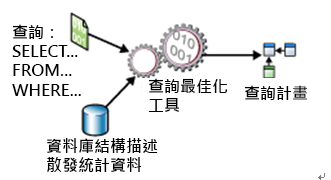
良好的 TSQL 和正確索引是大幅提高查詢效能最快的捷徑，同時 TSQL 也是使用 SQL Server 的核心，任何應用程式想要和 SQL Server 溝通，都無法避免撰寫 TSQL，所以各種效能調校方法中，我們認為查詢調校是最省成本、最快能感受到效果的方法 (如下圖)，這一系列文章將為大家介紹如何寫出高效能 TSQL，以及開發人員必須了解 SQL Server 相關基本知識和觀念，這樣才能活用相關技術，並發揮 SQL Server 無限潛能。

[](https://msdnshared.blob.core.windows.net/media/TNBlogsFS/prod.evol.blogs.technet.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/75/38/7206.1.png)

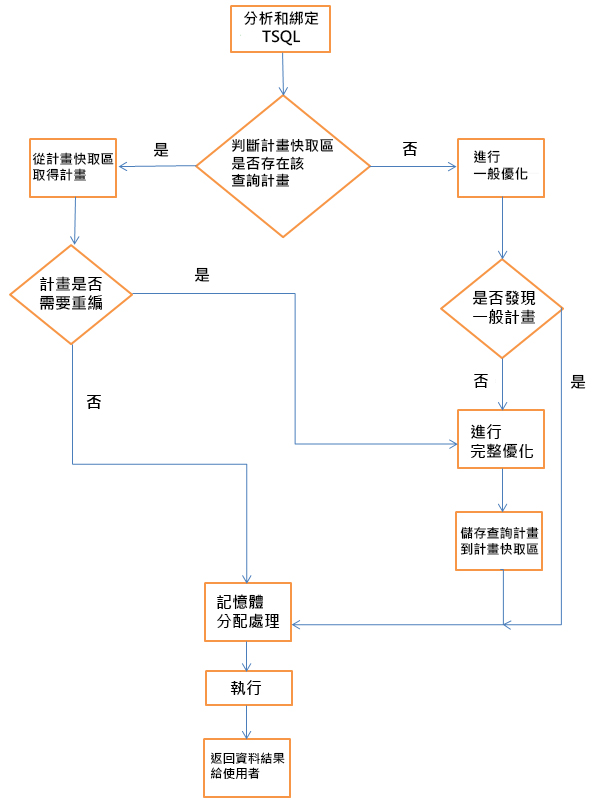
**SQL Server 關聯式引擎**

SQL Server 資料庫引擎主要有兩部分，這我們裡不討論儲存引擎 (storage engine) 架構，而將會著重在關聯式引擎。

由於 SQL Server 處理查詢過程步驟相當繁瑣，當一句 TSQL 送給 SQL Server 時，我們須了解關聯式引擎是如何工作的，因為每一句 TSQL 都必須透過關聯式引擎分析處理，最後才會透過儲存引擎執行並返回使用者所需的資料結果集，理解關聯式引擎不僅可以幫我們預先避開效能陷阱，同時也有助於我們減少查詢調校和除錯時間，下面我將說明幾個處理關鍵步驟。

[](https://msdnshared.blob.core.windows.net/media/TNBlogsFS/prod.evol.blogs.technet.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/75/38/A.jpg)

圖一：SQL Server 處理查詢簡單示意

[](https://msdnshared.blob.core.windows.net/media/TNBlogsFS/prod.evol.blogs.technet.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/75/38/B.jpg)

圖二：SQL Server 處理查詢關鍵步驟流程圖

**分析和綁定**

一開始查詢最佳化工具會先確認語法正確性，如有錯誤將會立即返回錯誤訊息給使用者，如果沒有錯誤就會建立分析樹並進行物件綁定 (如資料表欄位是否存在、資料型別是否正確、函式是否異常..等)，主要是因為 TSQL 並非程序性語言，它無法告知資料庫應該用什麼樣正確步驟來擷取資料，所以這階段會幫你處理基本語法優化、資料型別轉換，簡單來說就是會改寫 TSQL，如把 between 轉換為 >= and <=，型別比較不一致時，自動增加轉換函數來確保資料正確性 (這就可能會大大影響查詢效能)，group by 位置..等，最後將輸出邏輯 (操作) 查詢樹，後面將會依照所輸出邏輯查詢樹步驟來一一執行。

**查詢優化**

查詢優化幾乎可說是最複雜且耗時的階段 (所以大家要有一點想像力)，一開始會先確認計畫快取區是否有適當的執行計畫，如果沒有找到適當執行計畫，就會進行一般優化 (如果可以的話)，然後再次從計畫快取區尋找是否有適當的一般計畫，如果存在就會分配相關記憶體並執行，這是因為不必為只有一種可能執行方法的 TSQL 進行完整優化，如此一來可以省下編譯和建立執行計畫時間，因為我們知道查詢效能一部分取決於建立執行計畫效率 (執行查詢時間是產生執行計畫時間 + 本身資料查詢時間)，所以查詢最佳化工具並不會尋找最佳執行計畫，而是盡可能尋找最低成本 (以 CBO 為基礎的 CPU、I/O 成本) 及傳回結果速度最快的執行計畫，如果該 TSQL 有達到一般計畫條件的話，那麼查詢最佳化工具將可不必進行完整優化而浪費不必要時間，下面我簡單列出建立一般計畫的情況

1. 只有查詢單一資料表，且沒有任何 order by or group by。

2. 只有查詢單一資料表，且該資料表沒有任何索引。

3. 只有查詢單一資料表，且符合 SARG (**search argrment**) 格式及條件比對使用唯一值 (unique key)。

4. 查詢只使用系統預設函示 (如 select getdate()..)。

5. 使用 insert..values 只新增單一資料表資料。

現在我們知道，如果查詢語法符合一般計畫的情況，那麼 SQL Server 在建立計畫時將會省下很多時間，但如果不符合的話，SQL Server 將針對該查詢擷取所有可用統計值資料(欄位和索引)，並先從計畫快取區尋找是否有符合該查詢的執行計畫，如果有找適當執行計畫的話，就在判斷是否需要經過完整優化 (如查詢使用了recompile 提示就必須進行完整優化)，如果沒有找到適當執行計畫的話，那麼就必須進行完整優化，這個時候查詢最佳化工具，會以所輸出的邏輯查詢樹為基準，自我設計各種可能執行的方法，雖然這裡會產生多種排列組合，但之前有提過，SQL Server 並不會尋找最佳執行計畫，因為尋找最佳執行計畫所花費時間成本可能還會高出本身查詢所需時間成本，所以只會尋找最低成本執行計畫。

為了加速這處理過程，SQL Server 將會採取平行處理來建立執行計畫，前提是 Server 有多核心且 cost threshold for parallelism and max degree of parallelism 兩個參數設定正確。

一般會選擇非平行執行計畫，但如果非平行計畫成本超過平行計畫成本，那麼 SQL Server 將會把負載傳送給每個可用 CPU，這時將會建立平行執行計畫，一般來說，在 OLTP 環境中需特別注意採用平行執行計畫的查詢，因為大多數都是不良 TSQL、不正確索引或索引遺漏..等造成 (發生高 CPU 情況)。

當完整優化後將產生低成本適當的執行計畫，這時就會將該計畫存放到計畫快取區中，並繼續下一執行階段。

**計畫快取和查詢執行**

取得相關執行計畫後，這時就會將該計畫送給儲存引擎，由儲存引擎來執行該查詢並且返回使用者所需的資料結果集，這裡須注意 SQL Server 可能會改變當初估計的執行計畫，如果有達到以下條件

1. 所需資料表和欄位統計值過時。

2. 執行期間所觸及資料或統計值異動過大。

3. 非平行執行計畫執行時間超過 cost threshold for parallelism 設定值。

如果執行期間資料、統計值異動過大，將會導致發生重新編譯，你可以想像在最後一步驟才知道需要重新編譯的話，這無疑對效能是一大傷害。

再來就是計畫快取區只保留最常執行的查詢計畫，太久沒執行的查詢計畫可能會自動清除，底下有幾個情況會自動清除計畫快取區，並再次發生完整優化處理

1. 當緩衝池 (buffer pool) 針對另一物件需要更多記憶體

2. 當查詢計畫不被任何連線使用

3. 當查詢計畫成本因子為 0

這裡簡單說明一下計畫成本因子，每個執行計畫都有該成本因子，每一次執行該執行計劃就會自動累加 1，SQL Server 並不會自動遞減該成本因子，但如果計畫快取區大小達到 buffer pool 大小 50% 時，當下一次計畫快取區被存取時，這是就會將快取區所有執行計畫的成本因子都減 1，這時如有成本因子為 0 的就會被清除掉。

為了要善用計畫快取區，所以我們要盡量避免 SQL Server 發生記憶體不足問題，如果你遇到記憶體壓力問題，可以開啟 optimize for ad hoc workloads (SQL 2008 以後才有) 來減輕記憶體壓力

-- 開啟 optimize for ad hoc workloads

sp\_CONFIGURE 'show advanced options',1

reconfigure

go

sp\_CONFIGURE 'optimize for ad hoc workloads',1

reconfigure

go

select \* from master.sys.configurations

where name='optimize for ad hoc workloads'

go

同時我們也要避免在正式環境中手動清除計畫快取區 (使用 DBCC FREEPROCCACHE)，因為這將導致所有查詢都需要重新編譯且降低查詢效能。

再來就是我們要盡可能提高執行計畫重用率並小心誤用錯誤執行計畫所造成效能問題，我簡單整理以下 5 點可幫助你執行查詢時，提高執行計畫重用機率

1. 避免 ad hoc 查詢類型

2. 動態組 SQL 字串請使用 sp\_executesql 取代 exec

3. 查詢異動值請明確使用參數取代

4. 盡量使用 SP 但須注意參數探測問題

5. 盡量使用兩字節表示 (dbo.usp\_getdate)，避免隱式解析

**SQL 2014 查詢最佳化工具改善 -- 新基數估計演算法**

取得最低成本執行計畫雖然對大多數情況來說是好的，但並不表示該計畫一定是最佳的，為了取得更好執行計畫，SQL 2014 採用新基數估計演算法來進行改善，簡單來說就是查詢效能會比以前的版本更好，而以前的 SQL Server 版本都將會遇到舊基數估計演算 (Plan Regressions) 而造成的效能問題，該問題詳細可參考 [[SQL SERVER]統計值能吃嗎?](http://www.dotblogs.com.tw/ricochen/archive/2014/02/09/142874.aspx)一文。

**SQL 2014 in-memory OLTP 引擎**

SQL 2014 可以讓我們建立 in-memory 資料表來優化 OLTP 效能，這意味者過去資料表使用硬碟存放並有高資源競爭問題都將獲得改善，最大改變就是 in-memory 資料表將不會有任何 latch 和 lock，那你可能會擔心交易過程資料一致問題 (ACID)，這部分在 SQL 2014 則是使用 Multiversion Concurrency Control (MVCC) 技術來隔離交易不被干擾，同時也避免 lock 問題，也可讓任何使用者隨時存取 in-memory 資料表資源，因為記憶體中的操作對使用者來說都是一閃即逝的過程 (短暫交易最有利)，使用者幾乎不會有任何等待的感覺出現。

SQL 2014 還提供了原生編譯的儲存程序，透過原生編譯的儲存程序來存取 in-memory 資料表可將效能提高是以前版本約 50% 效益 ([微軟官方資料](https://blogs.technet.microsoft.com/b/franmer/archive/2014/04/25/sql-2014-technologie-in-memory.aspx))，主要是因為產生的機器碼(machine code)，只針對該查詢建立應該要執行處理器指令 (無須進一步編譯或解釋)，可以更有效率執行查詢，並且更快速得擷取資料，下面我們會使用 in-memory OLTP 引擎來簡單測試不同持久性設定並執行新增資料效能差異。

建立 in-memory DB 和 Table (schema\_and\_data & schema\_only)

CREATE DATABASE memoryDB

    ON PRIMARY (

    NAME = [E:\SQLDataFile\memoryDB\_data]

    ,FILENAME = 'E:\SQLDataFile\memoryDB\_data.mdf'

    )

    ,FILEGROUP [memoryDB\_FG] CONTAINS MEMORY\_OPTIMIZED\_DATA (

    NAME = [memoryDB\_dir]

    ,FILENAME = 'E:\memoryDB\_dir'

    )

    LOG ON (

    NAME = [memoryDB\_log]

    ,Filename = 'E:\SQLDataFile\memoryDB\_log.ldf'

    )

GO

CREATE TABLE MemorySchemaAndData

(

       id int NOT NULL,

       c1 nchar(1) NOT NULL,

       CONSTRAINT PK\_MemorySchemaAndData PRIMARY KEY NONCLUSTERED HASH (id) WITH (BUCKET\_COUNT = 150000)

) WITH (MEMORY\_OPTIMIZED = ON, DURABILITY = SCHEMA\_AND\_DATA)

CREATE TABLE MemorySchemaOnly

(

       id int NOT NULL,

       c1 nchar(1) NOT NULL,

       CONSTRAINT PK\_MemorySchemaOnly PRIMARY KEY NONCLUSTERED HASH (id) WITH (BUCKET\_COUNT = 150000)

) WITH (MEMORY\_OPTIMIZED = ON, DURABILITY = SCHEMA\_ONLY)

進行測試之前我簡單說一下建立 in-memory 資料表注意事項

Schema\_and\_data：OLAP 系統中，該記憶體資料表類型最常被使用，透過持久性選項設定，可以確保 SQL Server 發生崩潰後相關資料表結構和資料依然會保留。

Schema\_only：記憶體中只存放資料表結構(metadata)，SQL Server 發生崩潰後相關資料會消失。

前面我們有提到，in-memory OLTP 引擎會建立 machine code，所以每一個 in-memory 資料表都有一顆相對應 dll (如下圖)，當該資料表被載入時，可以避免編譯或解釋，因此可以更快取得查詢資料，同時因為都會被載入到記憶體中，所以限制每個資料表記憶體大小也是相當重要的事，微軟有提供一個[估算記憶體大小公式](http://msdn.microsoft.com/zh-tw/library/dn282389.aspx)，當你要將資料表移轉到in-memory資料表時，還請不要忘記該重要環節。

**測試新增資料到 in-memory 資料表**

Schema\_and\_data

DECLARE @step INT

SET @step = 1

WHILE @step <= 120000

  BEGIN

      INSERT INTO dbo.MemorySchemaAndData

      VALUES      (@step, '1'),

                  (@step+1, '2'),

                  (@step+2, '3'),

                    (@step+3, '4'),

                    (@step+4, '5')

      SET @step = @step + 5

  END

12 萬筆約花 8 秒。

Schema\_only

DECLARE @step INT

SET @step = 1

WHILE @step <= 120000

  BEGIN

      INSERT INTO dbo.MemorySchemaOnly

      VALUES      (@step, '1'),

                  (@step+1, '2'),

                  (@step+2, '3'),

                    (@step+3, '4'),

                    (@step+4, '5')

      SET @step = @step + 5

  END

12 萬筆幾乎 0 秒。

確認 in-memory 資料表資料

當 in-memory 資料表使用 schema\_only 選項時，新增資料處理幾乎沒花費任何時間，但我前面有提到過，schema\_only 在 SQL Server 發生崩潰後，將不會保留任何資料，所以當你使用該選項時必須要清楚了解該特性，以免發生資料無法復原的慘劇。

重新啟動 SQL Server DB (模擬 SQL Server 崩潰)，並確認 in-memory 資料表資料狀況

Use Master

go

ALTER DATABASE memoryDB SET OFFLINE WITH ROLLBACK IMMEDIATE

GO

ALTER DATABASE memoryDB SET ONLINE

GO

使用 Schema\_only 的 in-memory 資料表資料都無保留。

**TSQL on Azure**

如有你有使用 Azure SQL Database 的話，那麼在撰寫 TSQL 時需要留意以下幾個重點。

1. 善用資料表值參數

SQL 2014 有加強資料表值參數 (SQL Azure 也一樣)，就是資料表值參數可以使用索引來提高查詢效能，以前大家可能比較常用 temp table 來處理中繼資料，但在 Azure 上需要當心使用過多的 tempdb 資源，畢竟 tempdb 只有一個 (大家共用)，所以當使用過多 tempdb 資源時，Azure 可能會自動切斷連線，所以建議使用資料表值參數取代temp 資料表來處理中繼資料。

2. 減少資料網路來回次數

由於網路品質我們無法掌握，所以一定要減少資料在 Azure 和 Client 之間的往來次數，同時也要建立一套安全 retry 機制，基本上都建議採取批次處理 (如一次撈所需的資料結果集)，少用 row by row 方式 (這也增加 Azure 費用成本) 來查詢資料或進行資料異動 (如c ursor)，且所有 TSQL 請使用 try.. catch 包起來，如遇到問題 (如網路斷線..等)才可以有相對應處理方法。

3. 善用快取

可以在資料表中新增 [rowversion](http://msdn.microsoft.com/zh-tw/library/ms182776.aspx) 欄位，這樣我們就可以輕易使用該欄位(注意該欄位不建議成為索引鍵值)來判斷上次讀取資料列後，資料列中任何值是否有改變，如果沒有改變的話就讀取本地快取資料，否則就讀取Azure上資料，這不僅可以大幅提高效能，同時也可以省下不少 Azure 費用成本。