T-SQL 語言基礎與資料類型

4

悅知文化
Delight Press

Transact-SQL(T-SQL) 是與 SQL Server 溝通的核心。凡存取 SQL Server 執行個體的所有應用程式，不論其使用者介面為何（例如，我們一般透過 .NET、VB、ASP 寫的應用程式、SQL Server 自己的管理介面 SQL Server Management Studio、sqlcmd…等），都是藉由傳遞 T-SQL 陳述式給伺服器，以要求 SQL Server 完成某項任務。

本章中，我們將介紹組成 T-SQL 語言的基本元素，在之後的章節中，再針對查詢、新增、修改、刪除等動作，各別介紹不同型態的 T-SQL 語法。

4-1 T-SQL 語言所包含的語法元素

T-SQL 元素包含了註解、運算子、資料類型…等，這些是使用 T-SQL 的基礎。接下來，分別說明各元素所提供的內容。

4-1-1｜註解

「註解 (comment)」是 T-SQL 陳述式中不會執行的字串，可用於說明，或是暫時取消部分程式碼執行。使用註解說明，可使後續的維護工作較為容易。註解經常用來記錄程式技巧、功能、參數、相關物件、原作者、修改者、審閱者姓名，以及發佈、修改日期、版本等，並用來描述複雜的計算或撰寫程式的邏輯。

SQL Server 支援兩種類型的註解字元，詳細說明如下：

■-- 雙連字號

「雙連字號」註解字元可與程式碼放在同一行執行，或是自己占據一行。從「雙連字號」開始到該行結尾之間，全部都是註解的一部分。對於多行註解而言，「雙連字號」必須出現在每一行註解的前面，如範例程式碼 4-1 所示：

```
-- 選擇 Northwind 資料庫。
USE Northwind
GO

-- 查詢 dbo.Customers 資料表所有的欄位。
```

```
SELECT Country,CustomerID,CompanyName,ContactName FROM dbo.Customers
ORDER BY Country DESC  -- 指定資料集排序方式。
```

範例程式碼 4-1：透過「雙連字號」撰寫單行註解

■/* ... */ 斜線-星號字元配對

「斜線-星號字元配對」註解字元可與程式碼放在同一行來執行，也可以占據全行，甚至參雜在可執行的程式碼中。從開始註解配對 (/*) 到結束註解配對 (*/) 之間，全部都會視為註解的一部分。對於多行註解而言，自 /* 起始直到 */ 結束間的多行文字全為註解。它與 C 系列語言註解標示的規則相同。

範例程式碼 4-2 利用註解來說明程式碼區段應該執行的動作。

```
/* 本段使用多行註解說明
   多行註解不得跨越批次 */
SELECT ProductID,ProductName,UnitPrice,UnitsInStock
FROM dbo.Products
```

範例程式碼 4-2：透過 /**/ 符號撰寫多行註解

多行註解必須包含在批次之中，不得跨越批次。例如，在「SSMS 程式碼編輯器」與 sqlcmd 公用程式中，GO 命令代表批次的結尾。若直接反白標示以下語法的上半段：

```
USE Northwind
GO
SELECT * FROM dbo.Customers
/* 查詢Employee資料表
GO
```

```
這段註解文字，會把上一行的 GO 一起當作註解解釋  */
SELECT * FROM dbo.Products
GO
```

範例程式碼 4-3：註解符號 /**/ 不可以跨批次

讓 SSMS 僅傳遞前半的 /* 註解符號，則會有如下的錯誤訊息：

```
訊息113，層級15，狀態1，行3
遺漏結尾的註解記號'*/'。
```

若範例程式碼 4-3 上下兩段一起執行，讓 GO 出現在 /* 與 */ 分隔符號之間，則會將 GO 當作註解的一部分，而不會導致語法錯誤。

4-1-2｜認識「識別碼」

資料庫物件名稱又稱為「識別碼 (identifiers)」。SQL Server 中每個物件都具有「識別碼」，例如，伺服器、資料庫與資料庫物件 (如資料表、資料行、索引、條件約束、規則、觸發程序、檢視、預存程序、函數…等) 都有「識別碼」。大多數物件需要「識別碼」，但對部分物件如條件約束，則是選擇性的需求，若在定義條件約束時未指定「識別碼」，系統會自動為你產生。

定義物件時會建立物件「識別碼」[①]，之後就可以使用「識別碼」來參考物件。例如，以下陳述式會建立具有「識別碼」 tbA 的資料表，以及具有「識別碼」 KeyCol 與 Description 的兩個資料行：

```
CREATE TABLE tbA(
KeyCol INT PRIMARY KEY,
Description nvarchar(80)
)
```

定義上述資料表時，還有一個未命名的條件約束 PRIMARY KEY。若以 sp_help tbA 檢視與 tbA 相關的定義時，可以看到 SQL Server 自動隨機賦予了 PK__tbA__D7CB9CCC0A9D95DB 一類的名稱，當然，你看到的名稱應該與我的不同。

「識別碼」可以分為兩類，以下兩小節會依序說明。

4-1-2-1 一般識別碼

符合規則的「識別碼」稱為「一般識別碼」，在 T-SQL 陳述式中可以省略符號分隔。「一般識別碼」的格式規則視資料庫相容性層級而定。這個層級可使用 sp_dbcmptlevel 系統預存程序或 SQL Server Management Studio 來設定。如果相容性層級是 90 (SQL Server 2005 版本，2008 依然可行)，則適用以下規則：

① 「識別碼」的定序由其定義階層決定。SQL Server 執行個體階層物件的「識別碼」 (如登入與資料庫名稱)，會被指派執行個體的預設定序。資料庫內物件的「識別碼」 (如資料表、檢視與資料行名稱) 會被指派資料庫的預設定序。例如，可區分大小寫的定序，在資料庫中建立名稱相同且大小寫不同的兩個資料表，但不可在不區分大小寫定序的資料庫中建立這兩個資料表名稱。

■ 第一個字元必須符合以下任一項：

- Unicode Standard 3.2 中所定義的字母。Unicode 的字母定義包括從 a 到 z 以及從 A 到 Z 的拉丁字元，還有其他語系中的字母字元。
- 第一個字元可以是 @ 符號、# 符號或底線 (_)。

■ 「識別碼」開頭的某些符號在 SQL Server 中具有特殊意義。例如：

- 以 @ 符號開頭的一般「識別碼」代表本機變數或參數，不能做為任何其他類型之物件的名稱。
- 部分 T-SQL 功能的名稱會以兩個 @ 符號為開頭。為了避免與這些功能混淆，不應該使用以 @@ 為開頭的名稱。
- 開頭為 # 符號的「識別碼」代表暫存資料表或預存程序②。
- 開頭為兩個 ## 符號的「識別碼」代表全域暫存物件。雖然 # 符號或兩個 ## 符號字元可做為其他類型之物件的名稱開頭，但是不建議此用法。

■ 可包含的後續字元如下：

- Unicode Standard 3.2 中所定義的字母。
- 其他 Basic Latin 或國家 (地區) 字母中的十進位數字。
- @、$、# 等符號或底線 (_)。

■ 「識別碼」不可以是大、小寫的 T-SQL 保留字。

■ 不允許內嵌的空格或特殊字元。

4-1-2-2 分隔識別碼

SQL Server 的變數名稱和函數、預存程序參數，必須符合一般「識別碼」規則。在 T-SQL 陳述式中使用「識別碼」時，如果「識別碼」與上述規則不符，則必須使用雙引號 ("") 或方括號 ([ ]) 加以分隔。以下為「分隔識別碼」的格式規則：

---

② 你可以在連接裡執行 CREATE TABLE #t… 或 CREATE PROC #p… 等語法建立物件，它們實際都建立在 tempdb 系統資料庫內，而非連接所在的資料庫。這些物件僅在該連接內可以使用，同一時間其它連接看不到這些物件；一旦連接結束後，系統會刪掉這些暫存物件。

- 「分隔識別碼」可包含的字元數與一般「識別碼」相同。不包括分隔符號字元在內，可有 1 到 128 個字元。本機暫存資料表的「識別碼」最多為 116 個字元。
- 「識別碼」本身可以包含目前字碼頁內任何字元的組合，但分隔符號除外。例如，「分隔識別碼」中可以包含空白，以及對一般「識別碼」有效的任何字元。

範例如下所示：

```
SELECT *
FROM [My Table]       --識別碼包含空格且使用保留字
WHERE [order] = 10    --識別碼是保留字
```

符合規則的「識別碼」，不一定要使用分隔符號。不符合一般「識別碼」格式則要分隔。另外，當連接的 QUOTED_IDENTIFIER 選項設為 ON 時，雙引號必須括起「識別碼」，而非字串。程式碼如下所示：

```
SET QUOTED_IDENTIFIER ON
SELECT * FROM "My Table" WHERE "order" = 10
```

依預設，Microsoft OLE DB Provider for SQL Server、SQL Server ODBC 及 .NET SqlClient 等驅動程式會在連接時，將 QUOTED_IDENTIFIER 設為 ON[③]。這時，在 T-SQL 陳述式中使用雙引號 (") 與單引號 (') 會遵循 SQL-92 規則。例如：**雙引號只能用來分隔識別碼，不能用來分隔字元字串[④]。單引號只能用來括住字元字串，不能用來分隔識別碼**。而不論 QUOTED_IDENTIFIER 的設定是什麼，都可以使用方括號當分隔識別碼。如果字串內嵌單引號，則應以兩個單引號代表一個單引號，程式碼如下所示：

```
SELECT * FROM "My Producet Table"
WHERE "Product Name" = 'A''Box'
```

③ 你可以透過 SQL Profiler 公用程式錄製 SQL Server 所觸發的 Audit Login 事件，當程式碼中的 Connection 物件執行 Open 方法時，在伺服器端會收到 set quoted_identifier on… 等語法。

④ 不過，為維持與現有應用程式的相容性，SQL Server 沒有完全強制執行這項規則。如果字元字串未超過識別碼的長度，就可以將字串括在雙引號中。但我們不建議採用此方式。

當 QUOTED_IDENTIFIER 設為 OFF 時，SQL Server 解釋單引號及雙引號的規則為：雙引號不能用來分隔識別碼。不過，雙引號可以用來作為字元字串分隔符號。換句話說，可以使用單引號或雙引號來括住字元字串。如果使用雙引號，就不必再使用兩個單引號來表示內嵌的單引號。例如：

```
SET QUOTED_IDENTIFIER OFF
SELECT * FROM [My Table]
WHERE [Last Name] = "O'Brien"
```

保留關鍵字不應作為物件名稱使用。但從舊版 SQL Server 升級的資料庫，「識別碼」可能會包含在先前版本中不是保留字，但卻是 SQL Server 2008 的保留字。在物件名稱變更之前，可以先使用「分隔識別碼」來參考物件。

SQL Server 可以在分隔識別碼中使用目前字碼頁的任何字元。不過，於物件名稱任意使用特殊字元，會使 SQL 陳述式和指令碼難以讀懂和維護。例如，當建立名為 MyProduct] 的資料表時，其中的右方括號是名稱的一部分。若要這麼做，則必須多加上兩個方括號，使右方括號逸出，程式碼如下所示：

```
CREATE TABLE [MyProduct]]]
(
ProductID int IDENTITY (1,1) NOT NULL,
ProductName varchar(30),
Price int
)
```

範例程式碼 **4-4**：在分隔識別碼中使用特殊字元

另外，如果物件在命名尾端包含空白，存取時可以不包含名稱尾端的空白，當然，儘量避免這些混淆。

SQL Server 保留一些關鍵字，用來定義、操作和存取資料庫。例如，在 sqlcmd 或 SQL Server 程式碼編輯器工作階段中，使用「BACKUP」關鍵字即告訴 SQL Server 要備份資料庫或記錄檔。

自訂的物件名稱最好不要使用保留關鍵字。如果取相同名稱，就必須以「分隔識別碼」來參考物件。雖然這種方法允許物件名稱為保留字，但建議不要使用。如果物件名稱必須像保留關鍵字可以移除子音或母音。例如，將程序命名為 bckup，來為使用者自訂資料庫執行 BACKUP 陳述式。有關於保留關鍵字的詳細列表，請參考《Microsoft SQL Server 2008 線上叢書》的說明。

4-1-3｜使用運算子

運算子是一個符號，負責一或多個運算式所執行的動作。運算子包含算術、比較、串連、指派…等等。例如，透過「比較運算子」測試資料，檢查客戶的國家 / 地區資料行是否 NULL，而「算術運算子」則可用於數學運算。列舉運算子可執行的作業如下：

- 變更資料。
- 搜尋符合指定條件的資料列或資料行。
- 在開始或認可交易，或是執行特定的程式碼之前，測試特定條件。

當簡單的運算式合併為複雜的運算式時，結果的資料類型是由運算子資料類型優先順序的規則來決定。如果結果是字元或 Unicode 值，那麼結果的定序則是由運算子定序優先順序來決定。而決定結果的有效位數、小數位數及長度的規則，則是根據運算式的有效位數、小數位數，以及長度來決定。

算術運算子

「算術運算子」會針對數值資料類型中，一或兩種資料類型的兩個運算式執行數學運算。T-SQL提供的「算術運算子」包含如下：加 (+)、減 (-)、乘 (*)、除 (/)、取得除法的餘數 (%)。加號 (+) 和減號 (-) 運算子也可以對 datetime 和 smalldatetime 執行算術運算。例如：

```
declare @date datetime='20080115'
select @date+1
```

會回傳：

```
2008-01-16 00:00:00.000
```

但 SQL Server 2008 所新增的 Date、Time、DateTime2 和 DateTimeOffset 等資料類型，則不可如此與整數類型混合運算，若上述範例改成如下寫法：

```
declare @date date='20080115'
select @date+1
```

將會傳回錯誤訊息：

```
訊息 206，層級 16，狀態 2，行 2
運算元類型衝突: date 與 int 不相容
```

比較運算子

「比較運算子」用來測試兩個運算式是否相同。除了 text、ntext 或 image 資料類型的運算式，所有運算式都可以使用「比較運算子」。T-SQL提供的「比較運算子」包含：

等於(=)、大於(>)、小於(<)、大於或等於(>=) 、小於或等於(<=) 、不等於(<>)、不等於(!=)、不小於(!<)、不大於(!>)

邏輯運算子

「邏輯運算子」會測試是否符合某些狀況，如同「比較運算子」，邏輯運算子會傳回含 TRUE、FALSE 或 UNKNOWN 值的 Boolean 資料類型。表 4-1 是 T-SQL 較常見的邏輯運算子，大多用於 WHERE 子句中。

運算子	意義
AND	如果兩個布林運算式都是 TRUE 時，便是 TRUE。
OR	如果任一個布林運算式是 TRUE，便是 TRUE。
NOT	反轉任何其他布林運算子的值。
BETWEEN	如果運算元在範圍內，便是 TRUE。
EXISTS	如果子查詢包含任何資料列，便是 TRUE。
IN	如果運算元等於運算式清單中的某個運算式，便是 TRUE。
LIKE	如果運算元符合某個模式，便是 TRUE。

表 4-1：邏輯運算子

■指派運算子

等號 (=) 是 T-SQL 唯一的「指派運算子」。下列範例中，透過「指派運算子」將常數 1 指定給變數 @MyCounter。

```
DECLARE @MyCounter INT
SET @MyCounter = 1
```

你也可以利用「指派運算子」來建立資料行標題，和定義資料行值與運算式之間的關聯性。下列範例會顯示資料行標題 FirstColumnHeading 和 SecondColumnHeading。FirstColumnHeading 標題的資料行內，所有資料列都會顯示 xyz 字串。之後，Products 資料表中的每個產品識別碼都會列在標題為 SecondColumnHeading 的資料行中：

```
SELECT FirstColumnHeading = 'xyz',
       SecondColumnHeading = ProductID
FROM dbo.Products
```

範例程式碼 4-5：使用指派運算子建立資料行標題以及定義資料行的內容

■字串串連運算子

加號 (+) 除了是算數運算子，也是串連字串的「字串串連運算子」。其他字串操作則是利用 SUBSTRING 之類；系統提供的字串函數來處理。

在串連 varchar、char 或 text 等資料類型的資料時，會將空字串解譯成空字串。例如，'abc' + '' + 'def' 會儲存成 'abcdef'[5]。

■位元運算子

「位元運算子」會在整數資料類型之類別下 (bigint[6]、int、smallint、tinyint)；任何資料類型的兩個運算式之間，執行位元操作。表 4-2 列出 T-SQL 的「位元運算子」：

⑤ 以往 SQL Server 6.5 時，會將空字串當作單一空白字元來處理，將 'abc' + '' + 'def' 儲存成 'abc def'。

⑥ SQL Server 2005 的線上說明似乎漏掉了 bigint 資料類型，就筆者測試 bigint 也可以執行位元運算。

運算子	意義
& (位元 AND)	位元 AND (兩個運算元)。
| (位元 OR)	位元 OR (兩個運算元)。
^⑦ (位元互斥 OR)	位元互斥 OR (兩個運算元)。

表 4-2：位元運算子

下列範例分別對 7 和 5 兩個數字各位元作 AND、OR、XOR 的運算：

```
select 7 & 5 [111 & 101],7 | 5 [111 | 101],7 ^ 5 [111 ^ 101]
```

其執行結果如下：

```
111 & 101   111 | 101   111 ^ 101
----------- ----------- -----------
5           7           2
```

■一元運算子

「一元運算子」只用在數值資料類型的單一運算式上。表 4-3 為 T-SQL 的「一元運算子」：

運算子	意義
+ (正)	傳回正的的數值資料。
- (負)	傳回負的的數值資料。
~ (位元 NOT)	只能用在整數資料類型類別與 bit 資料類型中，任一資料類型的運算式上，用於傳回數字的補數。

表 4-3：一元運算子

下列範例會將變數設為正值。

```
DECLARE @MyNumber decimal(10,2);
SET @MyNumber = +123.45;
SELECT @MyNumber;
```

⑦ 要特別注意的是 ^ 常被人誤認為次方運算子，但 T-SQL 需要用 Power 函數，例如，2 的 3 次方要寫作 Power(2,3)，而非 2^3。2^3 是二進位的 10 和 11 做 XOR，變成 01 了。

雖然加號可以出現在任何數值運算式之前，但對運算式傳回的值並不做任何處理。明確地說，如果運算式是負的，經過加號運算也不會傳回正值。如果負運算式要傳回正值，應使用 ABS 函數。以範例程式碼 4-6 為例，使用 ABS 函數的運算式才會傳回正值。

```
USE tempdb;
GO
DECLARE @Num1 int;
SET @Num1 = -5;
SELECT +@Num1, ABS(@Num1);
GO
```

範例程式碼 4-6：比較一元運算子與 ABS 函數

4-2 資料類型

資料類型用來定義資料種類 (例如，字元、整數或日期..等等)，告知電腦如何解釋連串的 0 與 1。

善用資料庫的第一步就是善用資料類型，對各類型的特徵、值域、精確度、所占空間的位元組數、數值類型小數點的進位或捨去、資料類型的隱含與明顯轉換、可用的函數…等等，都了然於胸才能正確有效地使用資料。

為了精確有效率地使用資源，同時提供彈性與廣泛應用，SQL Server 每個版本都會新增一些資料類型，以強化其功能。在接下來的小節中，將為你整理 SQL Server 2008 版本，可用的各種資料類型。

4-2-1｜SQL Server 2008 預設提供的資料類型

SQL Server 提供一組「資料類型」，以定義資料儲存的種類，或變數型態，例如：整數、浮點數、貨幣、日期和時間、字元、二進位、位元…等。瞭解它們才能適當地定義資料表的資料行以及變數。以下是具有資料類型的應用：

- 資料表與檢視中的資料行
- 預存程序中的輸出 / 入參數
- 含有傳回碼的預存程序，其資料類型一定是整數 (integer)
- 傳回一或多個特定資料類型的 T-SQL 函數與參數
- 變數
- 運算式的結果

「資料類型」提供下列四種屬性用於定義物件：

- 資料種類
- 儲存值的長度或大小
- 數字的精確度 (只限數值資料類型)
- 數字的小數位數 (數值與時間資料類型)

T-SQL 提供的系統資料類型如表 4-4 所示：

日期時間	字元字串	數值	二進位	大型資料	其他
date** datetime datetime2** datetimeoffset** smalldatetime Time**	char nchar nvarchar varchar	bigint decimal float int money numeric real smallint smallmoney tinyint	binary varbinary	CLR* image^ ntext^ nvarchar (max)* text^ varbinary (max)* varchar (max)* Xml*	bit CLR UDT* cursor geography** geometry** hierarchyid** rowversion (timestamp) sql_variant table uniqueidentifier Xml* 使用者自訂資料表類型**

表 4-4：T-SQL 提供的系統資料類型

表 4-4 中，各符號所代表的意義如下：

■ *：代表該資料類型在 SQL Server 2005 版本後才提供。

■ **：代表該資料類型在 SQL Server 2008 版本後才提供。

■ ^：建議不再使用的資料類型。

表 4-4 中，有類型同時列在兩欄，因為它們同時兼具兩種屬性。

你也可以利用 T-SQL 或 .NET Framework，定義自己的「資料類型」。T-SQL 提供的「別名」是以系統的「資料類型」為基礎，給特定格式一個別名，例如，稱 CHAR(10) 為“身分證字號”。SQL Server 2005 版後新增「使用者自訂類型」，讓你利用 .NET Framework 支援的程式語言建立類別，透過其內的方法和運算子定義它們的性質，以延伸 SQL Server 既有的資料類型。

以下將個別介紹不同資料類型的特徵，我們就從 SQL Server 2008 增加最多的日期和時間資料類型開始談起。

4-2-2｜日期和時間

SQL Server 2008 支援的日期和時間資料類型如表 4-5 所示：

名稱	範圍	長度	精確度	資料格式	備註
Date*	西元1年1月1號到9999年12月31號	3	1 天	YYYY-MM-DD	
Datetime	西元1753年1月1號到9999年12月31號	8	千分之三秒	YYYY-MM-DD hh:mm:ss [.nnn]	
Smalldatetime	1900年1月1號到2079年6月6號	4	分	YYYY-MM-DD hh:mm:ss	
Datetime2*	西元1年1月1號到9999年12月31號	6-8	100奈秒 (10 的 -7 次方秒)	YYYY-MM-DD hh:mm:ss [.nnnnnnn]	資料長度隨小數點精確位數而不同

表 4-5：SQL Server 2008 預設支援的日期時間資料類型

名稱	範圍	長度	精確度	資料格式	備註
Datetimeoffset*	西元1年1月1號到9999年12月31號	8-10	100奈秒	YYYY-MM-DD hh:mm:ss[.nnnnnnn][+|-]hh:mm	資料長度隨小數點精確位數而不同
Time*	0時0分0秒到23時59分59.9999999秒	3-5	100奈秒	hh:mm:ss[.nnnnnnn]	資料長度隨小數點精確位數而不同

表 4-5：SQL Server 2008 預設支援的日期時間資料類型（續）

* 代表SQL 2008 後才新增的資料類型

表 4-5 中，所有類型都支援關係運算子 (<、<=、>、>= 和 <>)、比較運算子 (=、<、<=、>、>=、<>、!< 和 !>) 和邏輯運算子 (IS NULL、IS NOT NULL、IN、BETWEEN、EXISTS、NOT EXISTS 和 LIKE)。SQL Server 同時針對日期和時間資料類型，提供一組專屬的函數，例如，以 DATEADD 和 DATEDIFF 加減日期，以 GETDATE、GETUTCDATE 來取回系統時間…等。而 SQL Server 也為了各種資料類型準備不同的系統函數，你可以直接在 SQL Server Management Studio 的「物件總管」視窗內，展開任一個資料庫後，繼續展開其內的「可程式性」→「函數」→「系統函數」節點，就可以看到各函數粗略的使用方式，並透過線上叢書查詢詳細的說明，其畫面如圖 4-1 所示：

圖 4-1：SQL Server Management Studio 所提供系統函數的中繼資料

在正式討論日期資料類型前，先看一下與曆法 / 日期相關的歷史沿革：

不知你是否會好奇，為何 SQL Server 的 DateTime 日期類型其起始是西元 1753 年，而非西元 1 年？

相傳是為了不處理西方 Julian 和 Gregorian 兩種曆法的差異。其故事起源如下 (內容擷取自 http://www.geocities.com/calshing/westerncalendar.htm)：

西方古代的曆法非常混亂，直到公元前 46 年，羅馬的凱撒大帝 (Julius Caesar) 根據天文學家 Sosigenes 建議，修訂古羅馬曆改成新的 Julian 曆法。將一年分為十二個月，規定單數月為 31 日，雙數月為 30 日，通常二月是 29 日 (平年)，每四年設置一閏年，閏年的二月加多一日成為 30 日。該曆法的結果是，與地球繞日的回歸周期，每 128 年累積的偏差達一日。

公元前 8 年，羅馬議會將八月改成奧古斯都皇帝 (Augustus Caesar) 之名，稱為 August。同時將八月改為大月而成 31 日，使它和紀念凱撒 (Julius Caesar) 的七月 (July) 日數相同，以顯示他和凱撒的功業同等偉大 ☺。

直到 16 世紀，天主教教皇 Gregory 十三世根據天文學家 Aloysius Lilius 及 Christopher Clavius 建議，從當時曆法中減去十天，使春分出現在 3 月 21 日，因為他們觀察到春分點發生在 3 月 11 日。執行以下改革並稱為 Gregorian 曆，其 3 千 3 百多年才會累積一日的偏差。

- 將 1582 年 10 月 5 日至 1582 年 10 月 14 日這 10 日取消。
- 每個可被 4 整除的年份是一個閏年。
- 但 00 結尾的年份一定要被 400 整除，才能算是閏年。否則不是閏年。

Gregorian 曆也就是現今世界通用的曆法，羅馬教廷宣佈 1852 年 10 月 4 日後面緊跟著就是 15 日。其他天主教國家也很快跟著這麼做了，例如，義大利、波蘭、葡萄牙和西班牙等；但新教國家不願意追隨。尤其是希臘等東正教國家直到 20 世紀初才修改。英國及其殖民地 (包括現在的美國) 在 1752 年執行。所以，1752 年 9 月 2 日後面跟著 1752 年 9 月 14 日，我們則在辛亥革命後才修改。

換句話說，西方的國家與國家間曆法，在換與未換成 Gregorian 曆之前，其日期需要加減 10 多天，因為舊的 Julian 曆每 128 年即多偏差一日。而 SQL Server 的前身 Sybase 工程師不想實作各國家間的日期轉換，因此，選擇英國改制 Gregorian 曆之後的那一年，也就是 1753 年 ☺

在輸入日期類型時，首先需要注意的是表示方法，因為我們大多利用 T-SQL 語法跟資料庫引擎溝通，而在撰寫 T-SQL 時，是以字串形式表示日期，並讓資料庫引擎自行判讀字串中哪一段代表年、月、日，將會造成一些混淆，例如：

```
'01/02/03'
```

這到底是哪年哪月哪日？

根據連接上不同的 Language 或 Dateformat 設定，會影響 SQL Server 對字串轉日期的解讀，其範例程式碼如下：

```
declare @d date,@d2 date
--設定Session 的Langauge 會自動影響Dateformat
set language '繁體中文'
set @d='01/02/03'
set @d2='20010203'
select @d,@d2,language,date_format from sys.dm_exec_sessions where session_
id=@@spid

set language 'English'
set @d='01/02/03'
set @d2='20010203'
select @d,@d2,language,date_format from sys.dm_exec_sessions where session_
id=@@spid

--設定Session 的Dateformat 不會連帶改變Langauge
set dateformat 'dmy'
set @d='01/02/03'
set @d2='20010203'
select @d,@d2,language,date_format from sys.dm_exec_sessions where session_
id=@@spid
```

範例程式碼 4-7：解釋字串所代表的時間內容時，須參考連接的相關設定

其執行結果如圖 4-2 所示：

	(沒有資料行名稱)	(沒有資料行名稱)	language	date_format
1	2001-02-03	2001-02-03	繁體中文	ymd

	(沒有資料行名稱)	(沒有資料行名稱)	language	date_format
1	2003-01-02	2001-02-03	us_english	mdy

	(沒有資料行名稱)	(沒有資料行名稱)	language	date_format
1	2003-02-01	2001-02-03	us_english	dmy

圖 4-2：根據 Session 不同的 Language 或 Dateformat 設定，會影響字串轉日期的解讀

從以上測試可以發現，設定 language 會影響 dateformat，dateformat 則影響對有格式的日期字串之解讀。需要強調的是：最佳的表現方式是不寫日期格式，而是僅以年月日的數字呈現 (在此列出了最長的日期時間表示)：

```
YYYYMMDD HH:MI:SS[.mmmmmmm][+|-]HH:MI
```

另外，Datetime 資料類型最小單位是 0.003 秒，當資料不是正好等於該值時，Datetime 是採用最接近的值，而三位小數可以出現的值之模式為 [0-9][0-9][037]，也就是最後一位數僅能是 0、3 或 7。而何謂最接近值呢？其範例如下：

```
declare @i int=1,@s nvarchar(30)=''
declare @t table(s nvarchar(30),d datetime)
while @i<10
begin
   set @s=' 20081218 23:59:59.99' + CONVERT(char(1),@i)
   insert @t values(@s,@s)
   set @i+=1
end
select * from @t
```

範例程式碼 4-8：測試 Datetime 資料類型的誤差

執行結果如圖 4-3 所示：

	s	d
1	20081218 23:59:59.991	2008-12-18 23:59:59.990
2	20081218 23:59:59.992	2008-12-18 23:59:59.993
3	20081218 23:59:59.993	2008-12-18 23:59:59.993
4	20081218 23:59:59.994	2008-12-18 23:59:59.993
5	20081218 23:59:59.995	2008-12-18 23:59:59.997
6	20081218 23:59:59.996	2008-12-18 23:59:59.997
7	20081218 23:59:59.997	2008-12-18 23:59:59.997
8	20081218 23:59:59.998	2008-12-18 23:59:59.997
9	20081218 23:59:59.999	2008-12-19 00:00:00.000

圖 4-3：DateTime 資料類型會有的 0.003 秒誤差

而這個誤差最可能造成的問題是：若要設小於某一天，例如，要 20081218 這一天的紀錄，若寫成如下的查詢語法：

```
where date between  '20081218 00:00:00.000'  and  '20081218 23:59:59.999'
```

則因為上述的原因，會包含 20081219 00:00:00.000 的資料。換句話說，比較時，用小於 (<) 會比小於等於 (<=) 正確。

SQL Server 2008 為了加強日期與時間類型的資料處理，新增下列四種資料類型：

- Date：僅儲存日期的資料類型。
- Time：僅儲存時間的資料類型。
- datetime2：與datetime 相似，但儲存的日期與時間範圍較大，且精確度更高。
- datetimeoffset：允許儲存相對於格林威治標準時間的時區資料。

相較於既有的 DateTime、SmallDatetime 資料類型，現在可以將含有日期與時間元素的資料內容分別存放於不同的資料類型欄位中，當使用過濾條件「between...and...」取日期間隔時，就變得方便許多。參照表 4-5 ，儲存日期元素時，改用新的資料類型「DATE」佔用的空間，最大可由「DateTime」資料類型的 8 byte 減少到 3 byte。資料表的筆數成長到一定數量時，除了可節省為數可觀的硬碟空間外，執行時也能降低記憶體需求，建立索引、比對資料都較有效率。

新的資料類型擴大儲存日期的範圍，並將精確度由原先的微秒增加到 100 奈秒 (10 的負九次方秒)。而 T-SQL 除了舊有的時間函數仍可繼續延用於上述新的資料類型，SQL Server 2008 也提供新的時間函數，以取得更精確的時間。而帶有時間的新資料類型，如：time、datetime2、datetimeoffset 等型態的資料長度可變動，能在宣告時，指定秒數以下小數點的精確位數，讓設計師可在精確度與節省空間兩者間求取平衡。

4-2-2-1 Date 與 Time 資料類型

若要儲存員工的生日、到職日期、訂單日期、出貨日期...等，這類的資料只須存放日期部分即可，但 SQL Server 2008 之前的版本中，沒有可明確儲存日期的資料類型。要存放日期的資料，就只能選用 DATETIME 或 SMALLDATETIME 資料類型。當輸入日期資料或是透過 GetDate() 函數，取得當下的日期存入該欄位時，時間會一併存入，例如，12:00:00.000 AM。這時就需要透過格式化來呈現日期部分，此外，在查詢時也要格外小心時間所帶來的潛在錯誤。

下列範例使用兩種不同的查詢語法，會回傳不同的結果值。這是因為指定變數 @date 為 '20040123' 時，沒有指定時間內容，因此，過濾條件在比較變數 @date 時，會預設成 '2004-01-23 00:00:00.000' 進行比對。

```
Use AdventureWorks
GO
-- 在 HumanResources.JobCandidate 資料表內有兩筆記錄是 2004/01/23
-- 但有時間，所以會比對不到
declare @date datetime set @date='20040123'
select * from HumanResources.JobCandidate where ModifiedDate=@date

--若有 2004/01/24 的資料也一併被加進來了
select * from  HumanResources.JobCandidate
where ModifiedDate between @date and dateadd(d,1,@date)
```

你可以利用 SQL Server 2008 提供的 DATE 資料類型，指定僅儲存日期資料於欄位中，以避免過濾資料時，因疏忽時間而導致錯誤。如下列範例所示，可以感受新資料類型帶來的便利。

```
DECLARE @Day as DATE
SET @Day = getdate()
PRINT @Day
```

回傳值如下：

```
2008-04-23
```

另外，時間被分為 24 小時，每小時 60 分，分又被分為 60 秒。在有些語系中，像是波蘭語和阿拉伯語，秒也以 60 進位制被再細分，但現今大都是以十進位法來細分小數點以下的時間。例如： 1 秒= 1000 毫秒 (millisecond)、 1 毫秒= 1000 微秒 (microsecond)、1 微秒= 1000 奈秒 (nanoseconds)

以往，SQL Server 處理時間的最小單位為毫秒 (Millisecond，千分之一秒)⑧，這是軟、硬碟或光碟機在儲存資料時常用的計算單位。

⑧ 雖然也可以擴增時間，例如，ks (千秒)、Ms (百萬秒) 和 Gs (十億秒)，但實際上很少這樣使用，大家都還是習慣用 60 進位的分、時，以及 24 進位的日做為秒的擴充。

而在 SQL Server 2008 版本中，將時間的精確度延伸至奈秒，例如，Time、Datetime2、Datetimeoffset 資料類型。下列程式碼在宣告變數時，直接賦與該資料類型的精確值。若沒有宣告數值，該資料類型會以最大的精確度當作預設值。

```
--設定@T5 變數的精確值到小數以下5位
DECLARE @T5 Time(5)=GETDATE()  -- GETDATE 延續以往的格式，精確度不夠
PRINT @T5

--設定@DT5 變數的精確值到小數以下7位
DECLARE @DT7 datetime2(7)= SYSDATETIME ()
PRINT @DT7
```

上述範例中若延用舊的 GetDate() 函數，將無法取得所要的精確度，因此，筆者以 SQL Server 2008 新增的 SysDateTime 函數來做個比對，它可以提供足夠的精確度。

下列範例資料表是以錄影帶出租店為例，在資料表 Product 中記錄該店的商品、進貨日期、各錄影帶的時間長度與相關資料。首先，示範以往沒有新的時間型態時，使用者較常選擇的設計方式。資料表的進貨日期 StockDate 欄位使用 Datetime 資料類型，錄影帶撥放時間 playLength 欄位使用 float 資料類型，如範例程式碼 4-9 所示：

```
USE tempDB
IF  EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].
[Product]') AND type in (N'U'))
DROP TABLE [dbo].[Product]

-- STEP 2: 建立訂單資料表
CREATE TABLE dbo.Product(
   ProductID int primary key,
   FilmName nvarchar(30),
   direct nvarchar(10),
   Rent money NULL,
   Discount real NULL,
   StockDate datetime Null,
   PlayLength float NULL
   )
GO

insert into Product values(1,'熊貓家族(電影版)','宮崎駿',70,0.2,'2004/06/30','6000')
insert into Product values(3,'天空之城','宮崎駿',70,0.2,'2004/06/30 15:30','6000')
insert into Product values(4,'兒時的點點滴滴','宮崎駿',60,0.2,'2004/06/30 16:15','4380')
insert into Product values(5,'貓的報恩','宮崎駿',60,0.2,'2004/06/30 16:20','4500')
insert into Product values(2,'霍爾的移動城堡','宮崎駿',80,0.2,'2006/01/10','7140')

select * from Product
```

範例程式碼 4-9：使用舊有的資料類型建立資料表

在儲存錄影帶播放時間的 PlayLength 欄位中，以秒為單位紀錄，範例程式碼 4-9 執行後的結果如圖 4-4 所示：

	ProductID	FilmName	direct	rent	Discount	StockDate	PlayLength
1	1	熊貓家族(電影版)	宮崎駿	70.00	0.2	2004-06-30 00:00:00.000	6000
2	2	霍爾的移動城堡	宮崎駿	80.00	0.2	2006-01-10 00:00:00.000	7140
3	3	天空之城	宮崎駿	70.00	0.2	2004-06-30 15:30:00.000	6000
4	4	兒時的點點滴滴	宮崎駿	60.00	0.2	2004-06-30 16:15:00.000	4380
5	5	貓的報恩	宮崎駿	60.00	0.2	2004-06-30 16:20:00.000	4500

圖 4-4：將時間的長度使用 float 資料類型紀錄

接下來，改用新的資料類型，記錄先前的資料內容，如範例程式碼 4-10 所示：

```
Use tempDB
IF  EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].
[ProductNew]') AND type in (N'U'))
DROP TABLE [dbo].[ProductNew]
-- STEP 2: 建立訂單資料表
CREATE TABLE dbo.ProductNew(
    ProductID int primary key,
   FilmName nvarchar(30),
   direct nvarchar(10),
    rent money NULL,
   Discount real NULL,
    StockDate date Null,
   PlayLength time NULL
   )
GO

insert into ProductNew values
(1,N'熊貓家族(電影版)',N'宮崎駿',70,0.2,'2004/06/30','01:40'),
(3,N'天空之城',N'宮崎駿',70,0.2,'2004/06/30 15:30','01:40'),
(4,N'兒時的點點滴滴',N'宮崎駿',60,0.2,'2004/06/30 16:15','01:13'),
(5,N'貓的報恩',N'宮崎駿',60,0.2,'2004/06/30 16:20','01:15'),
(2,N'霍爾的移動城堡',N'宮崎駿',80,0.2,'2006/01/10','01:59')

SELECT * FROM  ProductNew
```

範例程式碼 4-10：利用新的資料類型建立資料表

透過新的 Time 資料類型，在儲存錄影帶播放時間的 PlayLength 欄位中，可以更直觀地儲存時間長度。

	ProductID	FilmName	direct	rent	Discount	StockDate	PlayLength
1	1	熊貓家族(電影版)	宮崎駿	70.00	0.2	2004-06-30 00:00:00.000	01:40:00
2	2	霍爾的移動城堡	宮崎駿	80.00	0.2	2006-01-10 00:00:00.000	01:59:00
3	3	天空之城	宮崎駿	70.00	0.2	2004-06-30 00:00:00.000	01:40:00
4	4	兒時的點點滴滴	宮崎駿	60.00	0.2	2004-06-30 00:00:00.000	01:13:00
5	5	貓的報恩	宮崎駿	60.00	0.2	2004-06-30 00:00:00.000	01:15:00

圖 4-5：將時間的長度使用 time 資料類型紀錄

透過下列語法，分別對新舊兩個資料表查詢錄影帶撥放長度大於 1.5 小時的資料。由於 Product 資料表的 PlayLength 欄位是以秒數儲存，所以，過濾時需要以 90 分 *60 秒取得總秒數後才能比較。而 ProductNew 資料表就方便多了，由於 PlayLength 欄位的資料類型為 Time，因此，可以使用更直觀的方式過濾：

```
--列出錄影帶播放時間大於 1 .5 小時的資料

SELECT ProductID,
       FilmName,
       PlayLength
  FROM Product
 WHERE PlayLength>= 90 * 60


SELECT ProductID,
       FilmName,
       PlayLength
  FROM ProductNew
 WHERE PlayLength >= '01:30:00'
```

範例程式碼 4-11：比較兩種類型的過濾方式

4-2-2-2 DateTimeOffset 資料類型

以往儲存日期與時間時，沒有提供屬於哪個時區的表示法。現在處理來自於不同時區的資料時，可利用 SQL Server 2008 提供的 DateTimeOffset 資料類型儲存當地時間與該時區的資料。

例如，日本、台灣、美國三地相對於格林威志標準時間差如下：

- 日本格林威治標準時間 (G.M.T.)+9 小時。
- 台灣格林威治標準時間 (G.M.T.)+8 小時。
- 美國太平洋沿岸，如西雅圖、洛杉磯等城市，時區為格林威治標準時間 (G.M.T.)-8 小時。

因此，假設 2008 年 5 月 1 日，各地各有一筆資料於當地晚間 8 點存入資料庫。這時，各資料內容應分別記錄為 2008/05/01 20:45 +9:00、2008/05/01 20:45 +8:00、2008/05/01 20:45 -8:00

利用 DATEDIFF 函數，可以計算台灣與美國兩地區的時差，程式碼如下所示：

```
--使用DateDiff函數比較 兩個時區的時間差
DECLARE @dt DATETIMEOFFSET(0)
SET @dt = '2008-05-01 20:45:00 +8:00'

DECLARE @dt1 DATETIMEOFFSET(0)
SET @dt1 = '2008-05-01 20:45:00 -8:00'
SELECT DATEDIFF(hh,@dt,@Dt1)
```

其結果是 16 個小時。

透過以下的方式可以取得包含時區資訊的完整日期時間：

```
DECLARE @DATE2 DATETIMEOFFSET SET @DATE2= sysdatetimeoffset()
SELECT @DATE2
```

其結果如下：

```
4/30/2008 4:23:19 PM +08:00
```

4-2-2-3 新的日期時間函數

與時間相關的函數中，除了較常用的 GetDate、Year、Month、Day 等系統函數外，還有 CURRENT_TIMESTAMP、DATEADD、DATEDIFF、DATENAME、DATEPART、GETUTCDATE 等。其用途與使用方式如表 4-6 所示：

函數名稱	說明
CURRENT_TIMESTAMP	傳回目前的日期和時間。相當於 GetDate 函數。
DATEADD	根據新增間隔到特定的日期，傳回新的 datetime 值。 `DATEADD (datepart ,number,date )`
DATEDIFF	傳回兩個指定日期之間所跨過的日期和時間界限數目。 `DATEDIFF ( datepart , startdate , enddate )`

表 4-6：SQL Server 2005 與日期相關的函數

函數名稱	說明
DATEPART	傳回一個整數，代表指定的日期部分。 `DATEPART ( datepart , date )`
DATENAME	功能與 DATAPART 相似，但月份與星期會以各國語文呈現。 `DATEPART ( datepart , date )`
GETUTCDATE	傳回系統目前的格林威治標準時間和日期。

表 4-6：SQL Server 2005 與日期相關的函數（續）

利用這些函數，可支援大部分的時間運算。另外，DATEPART 、DATENAME 系統函數的 DatePart部分新增了四個選項：microsecond、nanosecond、Tzoffset、ISO_WEEK。前三個選項應該都可望文生義，第四個 Iso_Week 代表 ISO 8601 定義，規範兩年間交界的星期，依星期四所在，決定該星期屬於前 / 後哪一年。以 2004 年第一個星期為例，2003/12/31 是星期三，2004/1/1 是星期四，所以，該星期屬於 2004 年。這會決定一年內有 52 或 53 個星期，而這個標準通常執行於歐洲。

以下以簡單的範例說明 DateName 函數搭配 DatePart 的各種設定，將回傳不同的結果：

```
SELECT DATENAME(datepart,'2007-10-30 12:15:32.1234567 +05:10')
```

上述語法中，DatePart 分別設定不同的內容，其回傳值如表 4-7 所示：

datepart	傳回值
year, yyyy, yy	2007
quarter, qq, q	4
month, mm, m	十月
dayofyear, dy, y	303
day, dd, d	30
week, wk, ww	44

表 4-7：SQL Server 2008 後 DateName 系統函數在 DatePart 部分可以設定的內容

datepart	傳回值
iso_week, isowk, isoww	44
weekday, dw	星期二
hour, hh	12
minute, n	15
second, ss, s	32
millisecond, ms	123
microsecond, mcs	123456
nanosecond, ns	123456700
TZoffset, tz	310

表 4-7：SQL Server 2008 後 DateName 系統函數在 DatePart 部分可以設定的內容（續）

除上述函數外，SQL Server 2008裡，新增了 5 個與日期時間相關的系統函數： SYSDATETIME、SYSDATETIMEOFFSET、SYSUTCDATETIME、SWITCHOFFSET 和TODATETIMEOFFSET。分別說明如表 4-8：

函數名稱	說明
SYSDATETIME	與 GETDATE 相似，傳回目前的時間，但 SYSDATETIME 的精確度為 100 奈秒，日期格式為 YYYY-MM-DD hh:mm:ss[.nnnnnnn] 與 DATETIME2 資料類型相同。
SYSDATETIMEOFFSET	與 SYSDATETIME 功能相同，但包括時區。時間格式為 YYYY-MM-DD hh:mm:ss[.nnnnnnn] [+|-]hh:mm 與DATETIMEOFFSET 資料類型相同。
SYSUTCDATETIME	以 100 奈秒的精確度傳回 UCT(Universal Coordinated Time) 時間，也就是格林威治當地的日期與時間。由執行時間與 SQL Server 伺服器的時區所計算出來的。 日期格式為 YYYY-MM-DD hh:mm:ss[.nnnnnnn] 與 DATETIME2 資料類型相同。

表 4-8：SQL Server 2008 提供的新函數

函數名稱	說明
SWITCHOFFSET	取得某個時區時間加減時差後，換算到另一個時區的時間值。 `SWITCHOFFSET ( DATETIMEOFFSET, time_zone )`
TODATETIMEOFFSET	將本地日期時間值搭配指定的時區時差。

表 4-8：SQL Server 2008 提供的新函數（續）

簡單的測試範例如下：

```
print SYSDATETIME()
print SYSDATETIMEOFFSET()
print SYSUTCDATETIME()
```

執行結果如下：

```
2008-04-28 11:25:31.5242848          目前伺服器的執行時間
2008-04-28 11:25:31.5242848 +08:00   目前伺服器的執行時間(含當地時區)
2008-04-28 03:25:31.5242848          相對於格林威治當地的標準時間 (不含當地時區)
```

SWITCHOFFSET 函數轉換時間到對應的時區，並回傳 DATETIMEOFFSET 類型的資料。範例程式碼如下所示：

```
PRINT SYSDATETIMEOFFSET()
PRINT SWITCHOFFSET (SYSDATETIMEOFFSET(), '-08:00') -取回西雅圖當地的時間
```

回傳值：

```
2008-04-30 16:41:00.6766816 +08:00
2008-04-30 00:41:00.6766816 -08:00
```

上述程式碼中，第一個回傳值取得目前的日期與當下的時區時間。以美國西雅圖為例，其標準時區為 -08。因此，以筆者所執行的伺服器為準，作業系統的時區選擇為台灣，所以，回傳2008-04-30 16:41:00.6766816 +08:00。而第二個回傳值為西雅圖當地時間：2008-04-30 00:41:00.6766816 -08:00[⑨]。

⑨ 其實，西雅圖當地時間還需要考慮美國政府所頒佈的日光節約時間，不一定僅是考慮時差而已 ☹

TODATETIMEOFFSET 函數，則可將本地日期時間值搭配指定的時區時差，程式碼如下所示：

```
print SYSDATETIME()
print TODATETIMEOFFSET(SYSDATETIME(),'-08:00')
```

回傳值如下：

```
2008-05-05 10:08:40.0017680
2008-05-05 10:08:40.0017680 -08:00
```

對於上述 SQL Server 2008 新增的日期資料類型，相信你已能充分掌握了。在此要提醒你，萬一需要將「DATE」與「TIME」資料類型之欄位內容相加時，若沒有經過轉換處理，將會出現下列錯誤訊息：

```
Operand data type date is invalid for add operator.
```

運算上述兩個資料類型的內容時，正確作法需要分別將其轉換為「SMALLDATETIME」資料類型才能運算。程式碼如下所示：

```
Declare @dt as DATE
Set @dt = getdate()
Declare @dtt as TIME
Set @dtt = getdate()
PRINT @dt
PRINT @dtt
PRINT cast(@dt as smalldatetime)  + cast(@dtt as smalldatetime)
```

回傳結果如下：

```
2008-05-05
11:02:02.8170000
May  5 2008 11:02AM
```

討論完日期時間類型後，接著來看字元字串資料類型。

4-2-3｜字元字串

SQL Server 2008 預設提供的字元字串如表 4-9 所示：

名稱	最大長度	說明
Char	8000 個字元	固定長度非 Unicode 字元
Varchar	8000 個字元	可變動長度非 Unicode 字元
Varchar(Max)*	2G 個字元	可變動長度非 Unicode 字元
Text	2G 個字元	可變動長度非 Unicode 字元，建議不要再使用，改以 Varchar(Max) 取代
Nchar	4000 個字元	固定長度 Unicode 字元
Nvarchar	4000 個字元	可變動長度 Unicode 字元
Nvarchar(max)*	1G 個字元	可變動長度 Unicode 字元
Ntext	1G 個字元	可變動長度 Unicode 字元，建議不要再使用，改以 Nvarchar(Max) 取代

表 4-9：SQL Server 2008 預設支援的字元字串資料類型

* 代表 SQL 2005 後才新增的資料類型

「字元」類型可用於儲存各種文數字與符號，非 Unicode 的編碼長度與方式隨各國語言不同而有所不同。Unicode 則統一編碼全世界的字元，支援的字元 (character) 範圍較廣。運用數字與字元的對照表，可以將全球的所有語言符號納入，可以說它是全世界最廣泛使用的編碼配置。當所有電腦都以一致的方式使用 Unicode 格式時，資料可以自由地在不同的資訊系統轉換，不須擔心接收端的系統會將資料解釋成錯誤的字元。在支援多國語言的系統時，若沒有特殊需求的限制，建議你應選擇使用 Unicode 類型，將字元轉換的問題減到最少。關於 Unicode 的解釋可以參照如下的網址：

http://zh.wikipedia.org/w/index.php?title=Unicode&variant=zh-tw

若沒有多國語言需求，例如，代碼 / 編號一類的應用，只有英文字母與數字的組合，則用非 Unicode 字元較省空間。

使用 char[(n)] / varchar[(n)] / nchar[(n)] / nvarchar[(n)] 等字元類型時，儲存空間的大小可由使用者自行決定。如範例程式碼 4-12 所示，將變數的資料類型定義為 varchar(10) / nvarchar(10)，則代表儲存空間可存放 10 個字元。當儲存的資料長度超出允許的字元數，就會截斷資料。

```
DECLARE @Word1 varchar(10)
DECLARE @Word2 nvarchar(10)
SET @Word1='This is a  long character string'
SET @Word2='This is a  long character string'
SELECT @Word1
SELECT @Word2
```

範例程式碼 4-12：測試字元的長度

執行結果如圖 4-6 所示，超過 10 個字元的資料將截斷為 "This is a"。

使用 char、varchar、nchar、nvarchar 資料類型時 ，最好指定足夠的字元長度，避免字元被截斷。變數宣告中使用 char 或 varchar 資料類型時，若沒有指定字元長度，其預設值為 1，變數指派超過 1 個字元時，字元會被截斷。

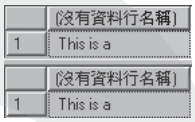


圖 4-6：超過指定長度的字元會被 SQL Server 截斷

如範例程式碼 4-13 所示，將變數 @myVariable 與 @myNextVariable 的資料類型分別設定為 varchar 與 char ，並指派三個字元給予變數。接著，使用 DATALENGTH 函數檢視變數的位元組 (byte) 長度時，其結果為 1。

```
DECLARE @myVariable AS varchar
DECLARE @myNextVariable AS char
SET @myVariable = 'abc'
SET @myNextVariable = 'abc'

--使用SELECT 陳述是查詢變數結果將只回傳一個字元 'a'
SELECT  @myVariable, @myNextVariable
--使用DATALENGTH 撿視變數的字元長度
SELECT DATALENGTH(@myVariable), DATALENGTH(@myNextVariable);
```

範例程式碼 4-13：使用 DATALENGTH 函數檢視變數的位元長度

使用型態轉換函數 (CAST、CONVERT) 時，char 與 varchar 的字元長度預設值為 30，如範例程式碼 4-14 所示，雖然宣告變數 @myVariable 的字元長度為 40 個字元。但在查詢陳述式，分別搭配 CAST、CONVERT 等系統函數時，預設長度是 30：

```
DECLARE @myVariable AS varchar(40)
SET @myVariable = 'This string is longer than thirty characters'

--
SELECT CAST(@myVariable AS varchar)
SELECT DATALENGTH(CAST(@myVariable AS varchar)) AS 'VarcharDefaultLength'

SELECT CONVERT(char, @myVariable)
SELECT DATALENGTH(CONVERT(char, @myVariable))
AS 'VarcharDefaultLength'
```

範例程式碼 4-14：varchar 與 char 搭配型別轉換函數時，n 的預設值為 30

由於 char 與 varchar 的預設位元組數為 30，造成變數內的字元截斷。試著使用 DATALENGTH 測試預設的位元長度，執行結果圖 4-7 所示：

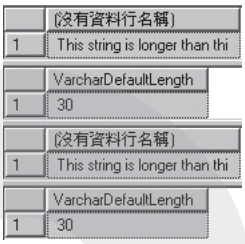


圖 4-7：搭配資料類型轉換函數時，varchar 與 char 的字元長度預設值為 30

試將範例程式碼 4-14 更改為 Unicode 類型的 nchar 與 nvarchar 資料類型，測試執行的結果如圖 4-8 所示。字元長度的預設值為30，使用 DATALENGTH 函數測試變數的位元組長度則為 60。

將資料新增到定義為 nchar 或 nvarchar 資料類型的資料行時，在字元前面加上大寫的 N 關鍵字，可指定以 Unicode 編碼新增到資料行中，這在不同國別輸入資料時，要特別小心。

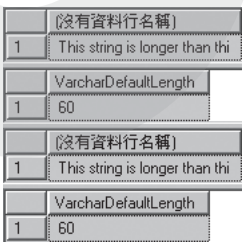


圖 4-8：搭配資料類型轉換函數時，nvarchar 與 nchar 的字元長度預設值為 60

```
INSERT Book (Name) values(N'SQL Server 資料庫')
```

若沒有使用 N 這個關鍵字，SQL Server 就會使用預設的 code page 儲存這些資料。伺服器上的 code page 若無法辨識這些資料時，就會存入不正確值。不過，只有當使用的延伸字元沒有辦法被常用的 code page 辨識時才會發生。在此建議你應養成指定 Unicode 字元的習慣，測試語法如範例程式碼 4-15：

```
DECLARE @n nchar(3),@n2 nchar(3),@c char(6),@c2 char(6)
DECLARE @sn nchar(3),@sn2 nchar(3),@sc char(6),@sc2 char(6)
SET @c='簡體字'
SET @c2=N'簡體字'
SET @n='簡體字'
SET @n2=N'簡體字'
SET @sc='简体字'
SET @sc2=N'简体字'
SET @sn='简体字'
SET @sn2=N'简体字'
SELECT CONVERT(BINARY(6),@c) ,CONVERT(BINARY(6),@c2),CONVERT(BINARY(6),@n),
CONVERT(BINARY(6),@n2)
,CONVERT(BINARY(6),@sc),CONVERT(BINARY(6),@sc2),CONVERT(BINARY(6),@sn),CONV
ERT(BINARY(6),@sn2)

SELECT @c ,@c2,@n,@n2,@sc,@sc2,@sn,@sn2
```

範例程式碼 4-15：比較未輸入 N 的問題

在上述範例中，由於透過鍵盤 / 螢幕輸入文字時，作業系統是採用 Unicode，因此可以選到簡體中文字。當存入系統時，若宣告為 char 或 varchar，預設採用筆者系統所預設的字碼 (big5)，而該字碼頁並沒有「简」這個字，因此，都自動轉成問號。

而變數若宣告為 nchar，但在設定值未指定 N，則會以為是 big5 要轉成 Unicode。則在輸入時，由 Unicode 先轉成 Big5，再從 Big5 換成 Unicode，而失去了可對應的字。

	(沒有資料行名稱)	(沒有資料行名稱)	(沒有資料行名稱)	(沒有資料行名稱)	(沒有資料行名稱)	(沒有資料行名稱)	(沒有資料行名稱)	(沒有資料行名稱)
1	0xC2B2C5E9A672	0xC2B2C5E9A672	0x217CD49A575B	0x217CD49A575B	0x3FCA5EA67220	0x3FCA5EA67220	0x3F00534F575B	0x807B534F575B

	(沒有資料行名稱)	(沒有資料行名稱)	(沒有資料行名稱)	(沒有資料行名稱)	(沒有資料行名稱)	(沒有資料行名稱)	(沒有資料行名稱)	(沒有資料行名稱)
1	簡體字	簡體字	簡體字	簡體字	?体字	?体字	?体字	简体字

圖 4-9：比較字元型別的宣告與賦與值

若要儲存資料的長度超出允許的字元數，就會截斷資料；但若輸入的資料長度小於定義的字元數呢？這個判讀有些複雜。它會與定義資料行時，連接的 ANSI_PADDING 設定為何，以及該資料行是否可以 NULL 有交互影響的關係。

而 ANSI_PADDING 設定只會在建立資料表時，影響新資料行的定義。之後，SQL Server 會根據建立資料行當時的設定來儲存值。更新資料時，在連接對 ANSI_PADDING 設定的變更並不會影響現有資料行。

若資料行指定 NOT NULL，char 資料類型就會是固定長度的資料類型。插入短於資料行長度的值，則值的右邊會填上空白，以補滿該資料行的大小。例如，將資料行定義為 char(10)，並儲存 "music" 資料，SQL Server 會將此資料儲存為 "music"，也就是 music 後五個空白。

建立 CHAR 類型且允許 NULL的資料行時，如果 ANSI_PADDING 是 ON，其作用就相當於資料行類型為 CHAR 且 NOT NULL：會在值的右邊填上空白，以補滿該資料行的大小。

建立 CHAR 且允許 NULL的資料行時，如果 ANSI_PADDING 是 OFF，則截斷尾端的空白，等同 ANSI_PADDING 設為 OFF 的 VARCHAR 資料行。

VARCHAR 是可變長度的資料類型。輸入值如果短於資料行的大小，也不會在右邊填上空白。建立資料行時，如果 ANSI_PADDING 選項的設定為 OFF，會截斷儲存在資料行中尾端的空白。如果 ANSI_PADDING 設為 ON，就不截斷後隨的空白。其定義如表 4-10：

ANSI_PADDING設定	char(n) NOT NULL 或 binary(n) NOT NULL	char(n) NULL 或 binary(n) NULL	varchar(n) 或 varbinary(n)
ON	將原始值填補到資料行的長度，char 資料行尾端補空白，binary 資料行尾端補零。	遵照 char(n) 或 binary(n) NOT NULL 的相同規則。	不截斷 varchar 資料行尾端的空白。不截斷 varbinary 資料行尾端的零 (二進位值)。不填滿資料行的長度。
OFF	將原始值填補到資料行的長度，char 資料行尾端補空白，binary 資料行尾端補零。	遵照 varchar 或 varbinary 的相同規則。	截斷 varchar 資料行的尾端空白。截斷 varbinary 資料行之二進位值的尾端零。

表 4-10：CHAR、VARCHAR、BINARY、VARBINARY 類型與 NULL、NOT NULL 和 ANSI_PADDING 設定間的關係

SET ANSI_PADDING 設定不會影響 nchar、nvarchar、ntext、text、image 和大數值。它們一律會顯示 SET ANSI_PADDING ON 行為。這表示 nchar 皆會補齊資料長度，nvarchar 不會修剪尾端空格。測試與示範的程式碼如下：

```
SET ANSI_PADDING ON;
GO

CREATE TABLE t1 (
   charNull CHAR(16) NULL,
   charNotNull CHAR(16) NOT NULL,
   varcharNull VARCHAR(16) NULL,
   varcharNotNull VARCHAR(16) NOT NULL,
   varbinarycol VARBINARY(8)
);
GO
INSERT INTO t1 VALUES ('補空白', '補空白', '補空白', '補空白', 0x00ee);
INSERT INTO t1 VALUES ('截去尾端空白    ', '截去尾端空白    ', '截去尾端空白    ',
'截去尾端空白    ', 0x00ee00);

--透過與 > < 符號的整合，呈現空白字元
SELECT '>' + charNull + '<' AS ANSI_PaddintOn_CharNull,
 '>' + charNotNull + '<' AS ANSI_PaddintOn_CharNotNull,
 '>' + varcharNull + '<' AS ANSI_PaddintOn_varharNull,
 '>' + varcharNotNull + '<' AS ANSI_PaddintOn_varcharNotNull,
   varbinarycol AS ANSI_PaddintOn_varbinary
FROM t1;
GO
DROP TABLE t1
GO

SET ANSI_PADDING OFF;
GO

CREATE TABLE t1 (
   charNull CHAR(16) NULL,
   charNotNull CHAR(16) NOT NULL,
   varcharNull VARCHAR(16) NULL,
   varcharNotNull VARCHAR(16) NOT NULL,
   varbinarycol VARBINARY(8)
);
GO
INSERT INTO t1 VALUES ('補空白', '補空白', '補空白', '補空白', 0x00ee);
INSERT INTO t1 VALUES ('截去尾端空白    ', '截去尾端空白    ', '截去尾端空白    ',
'截去尾端空白    ', 0x00ee00);

SELECT '>' + charNull + '<' AS ANSI_PaddintOff_CharNull,
 '>' + charNotNull + '<' AS ANSI_PaddintOff_CharNotNull,
```

```
 '>' + varcharNull + '<' AS ANSI_PaddintOff_varharNull,
 '>' + varcharNotNull + '<' AS ANSI_PaddintOff_varcharNotNull,
   varbinarycol AS ANSI_PaddintOff_varbinary
FROM t1;
GO
DROP TABLE t1
```

範例程式碼 4-16：比較Ansi_Padding、欄位允許 NULL 與否、Char/Varchar 對補空白和截去尾端空白的差異

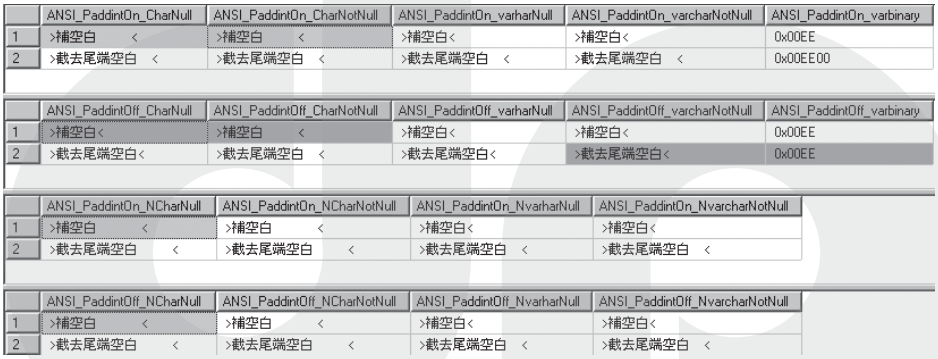
以下是換成 Unicode 的對照組：

```
SET ANSI_PADDING ON;
GO

CREATE TABLE t1 (
   ncharNull NCHAR(16) NULL,
   ncharNotNull NCHAR(16) NOT NULL,
   nvarcharNull NVARCHAR(16) NULL,
   nvarcharNotNull NVARCHAR(16) NOT NULL
);
GO
INSERT INTO t1 VALUES (N'補空白', N'補空白', N'補空白', N'補空白');
INSERT INTO t1 VALUES (N'截去尾端空白     ', N'截去尾端空白     ', N'截去尾端空白
', N'截去尾端空白    ');

SELECT '>' + ncharNull + '<' AS ANSI_PaddintOn_NCharNull,
 '>' + ncharNotNull + '<' AS ANSI_PaddintOn_NCharNotNull,
 '>' + nvarcharNull + '<' AS ANSI_PaddintOn_NvarharNull,
 '>' + nvarcharNotNull + '<' AS ANSI_PaddintOn_NvarcharNotNull
FROM t1;
GO
DROP TABLE t1
GO

SET ANSI_PADDING OFF;
GO

CREATE TABLE t1 (
   ncharNull NCHAR(16) NULL,
   ncharNotNull NCHAR(16) NOT NULL,
   nvarcharNull NVARCHAR(16) NULL,
   nvarcharNotNull NVARCHAR(16) NOT NULL
);
GO
INSERT INTO t1 VALUES (N'補空白', N'補空白', N'補空白', N'補空白');
INSERT INTO t1 VALUES (N'截去尾端空白     ', N'截去尾端空白     ', N'截去尾端空白
```

```
', N'截去尾端空白     ');

SELECT '>' + ncharNull + '<' AS ANSI_PaddintOff_NCharNull,
 '>' + ncharNotNull + '<' AS ANSI_PaddintOff_NCharNotNull,
 '>' + nvarcharNull + '<' AS ANSI_PaddintOff_NvarharNull,
 '>' + nvarcharNotNull + '<' AS ANSI_PaddintOff_NvarcharNotNull
FROM t1;
GO
DROP TABLE t1
```

範例程式碼 4-17：Ansi_Padding 的設定不影響 NChar 補空白， NVarchar不截去尾端空白之運作

上述兩個範例程式碼的執行結果如圖 4-10 所示：



	ANSI_PaddintOn_CharNull	ANSI_PaddintOn_CharNotNull	ANSI_PaddintOn_varharNull	ANSI_PaddintOn_varcharNotNull	ANSI_PaddintOn_varbinary
1	>補空白 <	>補空白 <	>補空白<	>補空白<	0x00EE
2	>截去尾端空白 <	>截去尾端空白 <	>截去尾端空白 <	>截去尾端空白 <	0x00EE00

	ANSI_PaddintOff_CharNull	ANSI_PaddintOff_CharNotNull	ANSI_PaddintOff_varharNull	ANSI_PaddintOff_varcharNotNull	ANSI_PaddintOff_varbinary
1	>補空白<	>補空白 <	>補空白<	>補空白<	0x00EE
2	>截去尾端空白<	>截去尾端空白 <	>截去尾端空白<	>截去尾端空白<	0x00EE

	ANSI_PaddintOn_NCharNull	ANSI_PaddintOn_NCharNotNull	ANSI_PaddintOn_NvarharNull	ANSI_PaddintOn_NvarcharNotNull
1	>補空白 <	>補空白 <	>補空白<	>補空白<
2	>截去尾端空白 <	>截去尾端空白 <	>截去尾端空白 <	>截去尾端空白 <

	ANSI_PaddintOff_NCharNull	ANSI_PaddintOff_NCharNotNull	ANSI_PaddintOff_NvarharNull	ANSI_PaddintOff_NvarcharNotNull
1	>補空白 <	>補空白 <	>補空白<	>補空白<
2	>截去尾端空白 <	>截去尾端空白 <	>截去尾端空白 <	>截去尾端空白 <

圖 4-10：比較 Ansi_Padding、欄位允許 NULL 與否、Char/Varchar/NChar/NVarchar 對補空白和截去尾端空白的差異

參照前文的說明，比對範例程式碼 4-16、4-17 與圖 4-10 後，相信你可以釐清這些差異。需要提醒的是：未來的 SQL Server 版本中，ANSI_PADDING 一定是 ON，若明確將此選項設定為 OFF將會產生錯誤。請避免在新的專案中使用該功能，並規劃修改目前使用這項功能的應用程式。

文字字串類型就討論到此，接著來看數值資料類型。

4-2-4｜數值資料

SQL Server 2008 支援的數值資料類型如表 4-11：

<table>
<tr><th>類型</th><th>資料類型</th><th>長度
(位元組)</th><th>說明</th></tr>
<tr><td rowspan="4">整數</td><td>tinyint</td><td>1</td><td>0 到 255。</td></tr>
<tr><td>smallint</td><td>2</td><td>-2^15 次方 (-32,768) 到 2^15 次方– 1 (32,767)。</td></tr>
<tr><td>int</td><td>4</td><td>-2^31 次方 (-2,147,483,648) 到
2^31 次方 –1 (2,147,483,647)。</td></tr>
<tr><td>bigint</td><td>8</td><td>-2^63 (-9,223,372,036,854,775,808) 到
2^63-1 (9,223,372,036,854,775,807)。</td></tr>
<tr><td>十進位數值</td><td>decimal / numeric [p[,(s)]]</td><td>4-17</td><td>用來儲存含小數的數字，而且資料以指定的方式儲存，範圍從 -10^38-1 到 10^38-1。

參數中的 p 代表精確度 (precision)，也就是物件可包含的位數，最大 38。s 指定小數位數，也就是可放在小數點右邊的位數。p 和 s 必須遵守規則：0 <= s <= p <= 38。

準確度決定儲存空間的大小，定義如下所示：

<table>
<tr><th>精確度 (p 值)</th><th>儲存空間</th></tr>
<tr><td>1~9</td><td>5位元</td></tr>
<tr><td>10~19</td><td>9位元</td></tr>
<tr><td>20~28</td><td>13位元</td></tr>
<tr><td>28~38</td><td>17位元</td></tr>
</table>
精確度的設定與儲存空間的關係

範例如下：

<pre>declare @i decimal(38)
set @i=12345678901234567890123456789012345678
select datalength(@i)</pre>

在 T-SQL 中，numeric 的功能等同於 decimal 資料類型。</td></tr>
</table>

表 4-11：SQL Server 2008 所支援的數值資料類型

<table>
<tr><th>類型</th><th>資料類型</th><th>長度
(位元組)</th><th>說明</th></tr>
<tr><td>近似值</td><td>float[(n)]</td><td>4-8</td><td>浮點數字，範圍從 -1.79E+308~1.79E+308。遵循 IEEE 754⑩有關近似數值資料類型的規格。

其中 n 是用來儲存科學記號標記法 float 尾數的位元數目，因此，它規定了有效位數和儲存體大小。如果指定 n，它必須是在 1 和 53 之間的值。n 的預設值是 53。

<table>
<tr><th>n 值</th><th>有效位數</th><th>儲存空間</th></tr>
<tr><td>1~24</td><td>1-9</td><td>4位元</td></tr>
<tr><td>25~53</td><td>10~19</td><td>8位元</td></tr>
</table>
精確度 n 值的設定與儲存空間的關係</td></tr>
<tr><td></td><td>real</td><td>4</td><td>浮點數字，範圍從 -3.40E+38~3.40+38。遵循 IEEE 754 有關近似數值資料類型的規格。</td></tr>
<tr><td>貨幣資料</td><td>money</td><td>8</td><td>範圍由 -922,337,203,685,477.5808 到 922,337,203,685,477.5807。19 為有效位數、4 位小數。</td></tr>
<tr><td></td><td>smallmoney</td><td>4</td><td>範圍由 - 214,748.3648 到 214,748.3647。10 為有效位數、4 位小數。</td></tr>
</table>

表 4-11：SQL Server 2008 所支援的數值資料類型（續）

雖然使用數值的資料類型非常簡單，只要針對資料行或是變數評估資料範圍，選擇足夠容納該資料的類型就可以了。不過，部分數值類型所配置的資料儲存空間是固定的，當定義了不適當的資料類型，會浪費不少空間。例如，某個資料行定義為 int 類型，但實際上儲存的數值都小於 50，理當宣告 tinyint 類型就夠了。若資料表共有 50,000 筆資料，則浪費 0.14MB 空間：((((4-1)*50,000)/1024)/1048)。

⑩ IEEE 754 規格提供四種四捨五入模式：四捨五入進位、無條件進位、無條件捨去進位與四捨五入到零。SQL Server 則使用無條件進位。所有運算都準確到保證的精確度，但可能產生稍微不同的浮點值。因為浮點數的二進位表示法可以使用任一種有效的四捨五入方式，所以不可能有效地限定浮點數值。

關於 decimal(numeric) 資料類型要提醒的一點是：該類型通常佔用磁碟固定資料長度。在 SQL Server 2005 Service Pack 2 版本後，decimal 資料類型可使用 Vardecimal 儲存格式來儲存成可變長度資料行，Vardecimal 是儲存方式的不同，而非資料類型[11]。該功能適用於 Enterprise、Developer 和 Evaluation Edition 等版本；但未來的 SQL Server 將移除 Vardecimal 功能。所以，應避免在新的開發專案中使用該功能，SQL 2008 後，建議改用新的 ROW 和 PAGE 壓縮功能。

浮點數不需指定小數點前後的位數，這與準確值資料類型剛好相反。浮點數之所以會歸類為近似值資料類型，是由於浮點數字的小數點位數不像準確數值是固定的。不過，浮點運算能處理的數字範圍較大，這是它的優點。

需要精確數值比較時，須小心使用 float 和 real 資料類型，因為會有數值運算誤差，例如，財務應用程式、牽涉到四捨五入的作業，或者進行相等比較時。應使用 integer、decimal、money 或 smallmoney 資料類型。另外，避免在 WHERE 子句的搜尋條件中使用 float 或 real 資料行，尤其是 = 與 <=、>= 運算子，最好限制為 > 或 < 比較。

money 和 smallmoney 資料類型的用法相當直觀，其精確度可達它們所代表之金融單位的萬分之一。不過有兩個部分需要特別注意。首先，不要為了呈現格式，自行替金額加上逗點。這會讓 INSERT 與 UPDATE 的指令發生錯誤。其次，預設的貨幣符號為美元 ($)，如果要使用不同的貨幣符號，只要在金額的前面加上指定的貨幣符號即可。程式碼如下所示：

```
INSERT Product (ProductName, Price) values( 'A-7079' ,'￥1000.00' )
```

不過，指定前面有貨幣符號的貨幣值時，SQL Server 不會儲存任何與符號相關聯的貨幣資訊，它只會儲存數值 ☹。

⑪ 啟動 Vardecimal 儲存格式的作法可參考線上叢書：「Database Engine」→「開發」→「查詢及變更資料」→「存取與變更資料庫資料」→「Transact-SQL 的元素」→「資料類型 (Database Engine) 」→「使用 decimal、float 與 real 資料」→「將十進位資料儲存成可變長度」。

4-2-4-1 進位與捨去

處理數值資料類型時，需要注意當運算使用到超過該類型資料可容許的小數位數時，SQL Server 將採用無條件進入、捨去，還是四捨五入。各類型說明如下：

- 整數類型：運算所產生的分數都會無條件捨去，不會四捨五入。例如，SELECT 5/3 會傳回數值 1，而非將分數的結果四捨五入得到 2。
- 浮點數 (float 和 real)：無條件進入到精確範圍內。
- 貨幣資料：小數點第四位後，四捨五入，範例如下：

```
select convert(money,0.12345) [0.12345],CONVERT(money,0.012345) [0.012345]
```

執行結果如圖 4-11 所示：

	0.12345	0.012345
1	0.1235	0.0123

圖 4-11：貨幣資料類型在小數點第四位後，四捨五入

- 十進位數值 (Decimal 或 Numeric)：四捨五入到有效的小數位數。其範例如下：

```
select convert(decimal(3,2),0.345) [0.345],CONVERT(decimal(3,2),0.2345) [0.2345]
```

其執行結果如圖 4-12 所示：

	0.345	0.2345
1	0.35	0.23

圖 4-12：Decimal 資料類型四捨五入到有效的小數位數

4-2-4-2 Round 函數

若要處理四捨五入，較常搭配使用 Round 函數，以四捨五入或捨去到指定長度的有效位數。其語法定義如下：

```
ROUND ( 數值運算式 , 長度 [ ,功能 ] )
```

Round 函數會傳回與「數值運算式」相同的類型結果，函數的引數定義如下：

- 數值運算式：精確數值或近似數值資料類型的運算式，但 bit 資料類型除外。
- 長度：「數值運算式」取四捨五入的有效位數。「長度」必須是 tinyint、smallint 或 int 等類型的運算式。當「長度」是正數時，「數值運算式」會四捨五入到「長度」所指定的小數位數。若是負數，則對小數點左側位數四捨五入。其範例如下：

```
select ROUND(45.45, -2) [-2],ROUND(45.45, -1) [-1],ROUND(45.45, 0)
[0],ROUND(45.45, 1) [1]
```

執行結果如圖 4-13 所示：

	-2	-1	0	1
1	0.00	50.00	45.00	45.50

圖 4-13：設定「長度」的正負值，以四捨五入不同的位數

圖 4-13 中需要注意的是第一欄，因為「長度」是負 2，所以取小數點左邊第二位，範例中是 4，被捨去後，變成整個數值為 0。若數值為負，且大於小數點前面的位數，ROUND 會傳回 0，例如：ROUND(45.45, -3) 會傳回 0。

- 功能：這是要執行的作業類型。「功能」引數的資料類型必須是 tinyint、smallint 或 int。當省略此參數，其預設值為 0，代表要四捨五入「數值運算式」。當指定為非 0值，則是無條件捨去。其範例如下：

```
select ROUND(45.45, -1,0) [-1,0],ROUND(45.45, -1,1) [-1,1],ROUND(45.45,
1,0) [1,0],ROUND(45.45, 1,1) [1,1]
```

執行結果如圖 4-14 所示：

	-1,0	-1,1	1,0	1,1
1	50.00	40.00	45.50	45.40

圖 4-14：設定「功能」引數的 0 值與否，決定四捨五入還是無條件捨去

4-2-5｜二進位資料類型

當需要儲存位元資料流 (stream)，例如，多媒體資料、特殊格式的文檔、加密過的資料…等等，就必須使用二進位資料類型。而 SQL Server 預設提供的一些類型，如 uniqueidentifier(UUID) 實際存放時，也就是二進位資料。

SQL Server 提供了如表 4-12 所呈現的二進位資料類型：

類型名稱	最大長度	說明
Binary	8000 位元組	固定長度的二進位資料
Varbinary	8000 位元組	可變動長度的二進位資料
Varbinary(max)*	2G 位元組	可變動長度的二進位資料
Image	2G 位元組	可變動長度的二進位資料，建議不要再使用，改以 Varbinary(Max) 取代

表 4-12：SQL Server 2008 預設支援的字元字串資料類型

* 代表 SQL 2005 後才新增的資料類型

Binary 與 Varbinary 資料最多可儲存 8,000 個位元組，但 Varbinary 宣告時若搭配 max，最多可儲存 2^31 個位元組。

二進位資料呈現時，會在常數前面以 0x (一個零和小寫字母 x) 開始，後面跟著以 16 進位表示的位元模式，也就是以兩個 16 進位數字代表一個位元組 (byte)，在使用上不需要替插入資料行的資料加上單引號。而 Binary 和 Varbinary 的差異如同先前所介紹的 Char 和 Varchar，存放固定或變動長度的資料。

我們簡單地示範將二進位資料存入資料表；並從資料表取出存回到檔案：

```
USE TempDB
GO
CREATE TABLE tbImg(
PK INT IDENTITY PRIMARY KEY,
FileName nvarchar(60),
FileType nvarchar(30),
ImageFile varbinary(max))
GO
```

```
INSERT INTO tbImg(FileName, FileType, ImageFile)
   SELECT 'Img0408.jpg' AS FileName,
      'JPG' AS FileType,
         * FROM OPENROWSET(BULK N'C:\Temp\IMG0408.JPG', SINGLE_BLOB) AS
ImageFile

--直接查詢二進位資料
SELECT * FROM tbImg

/*
在 Management Studio 可透過SQLCMD mode 執行外部程式
將二進位資料輸出到檔案
格式檔案(Format File)的定義
10.0
1
1 SQLBINARY 0 0 "" 1 ImageFile ""
*/
!!bcp "select ImageFile from tempdb.dbo.tbImg where PK=1" queryout
"C:\temp\myJPG.jpg" -T -f bcp.fmt
```

範例程式碼 4-18：將二進位資料存入資料表；並從資料表取出存回到檔案

上述查詢二進位資料的結果如圖 4-15 所示：

	PK	FileName	FileType	ImageFile
1	1	Img0408.jpg	JPG	0xFFD8FFE119FE4578696600004949 2A000800000009000F0...

圖 4-15：二進位資料以 16 進位數值的方式呈現

範例程式碼 4-18 中，利用 SQL Server 2005 後替 OpenRowset 系統函數增加的功能，以 Bulk 方式直接讀入存在單一檔案內的大型二進位資料 (SINGLE_BLOB)，搭配 Insert 語法載入到資料表中。最後再以 bcp 工具程式藉由 Format file 指定格式，將資料表內的大型二進位資料輸出到對應的檔案中。

SQL Server 2008 後，強化了 CONVERT 系統函數在二進位資料類型與 16 進位字串間的轉換。以 CONVERT 函數為例，其語法格式如下：

```
CONVERT ( 資料類型 [ (長度) ] , 運算式 [ , style ] )
```

其強化的部分是在第三個參數 style，使用說明如下：

■ 0 (預設值)：將 ASCII 字元轉譯成二進位位元組，或將二進位位元組轉譯成 ASCII 字元。每個字元或位元組都會以 1:1 的方式轉換。若「資料類型」是二進位類型，就會在結果的左邊加入字元 0x。

■ 1, 2 (SQL Server 2008 版後才提供)：

- 「資料類型」是二進位類型，此「運算式」的結果就必須是字元。「運算式」必須由偶數個十六進位代表數字 (如：0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,a,b,c,d,e,f) 所構成。如果為 style 1，字元 0x 就必須是運算式中的前兩個字元，若為 style 2 則否。

  如果此運算式含有奇數個字元，或者其中任何字元無效，會引發錯誤。若已轉換之運算式的長度超過「資料類型」的「長度」，結果就會向右截斷。固定長度的「資料類型」其定義若大於轉換之結果，會在結果的右邊加入零。

- 「資料類型」是字元類型，此「運算式」的結果就必須是二進位。每個二進位字元都會轉換成兩個十六進位字元。如果已轉換之運算式的長度超過「資料類型」的「長度」，就會向右截斷。若「資料類型」是固定大小的字元類型，而且已轉換之結果的長度小於「資料類型」的「長度」，就會在轉換結果的右邊加入空格，以便維持偶數個十六進位數字。

  若為 style 1，就會在已轉換之結果的左邊加入字元 0x，若為 style 2 則否。

```
SELECT CONVERT(VARCHAR(12) , 0x49747A696B   , 0) AS [0x49747A696B to
VARCHAR(12) Style 0]
SELECT CONVERT(VARBINARY(5), '0x49747A696B', 0) AS ['0x49747A696B' to
VARBINARY(5) Style 0]
SELECT CONVERT(VARCHAR(5),0x3078343937, 0) AS [0x3078343937 to VARCHAR(5)
Style 0]

SELECT CONVERT(VARCHAR(12) , 0x49747A696B   , 1) AS [0x49747A696B to
VARCHAR(12) Style 1]
SELECT CONVERT(BINARY(10), '0x49747A696B', 1) AS ['0x49747A696B' to
BINARY(10) Style 1 補0]

SELECT CONVERT(VARCHAR(5) , 0x49747A696B   , 2) AS [0x49747A696B to
VARCHAR(5) Style 2 截斷，需要偶數邊界]
SELECT CONVERT(VARCHAR(6) , 0x49747A696B   , 2) AS [0x49747A696B to
VARCHAR(6) Style 2 截斷，需要偶數邊界]
SELECT CONVERT(VARBINARY(5), '49747A696B'   , 2) AS ['49747A696B' to
VARBINARY(5) Style 2];
```

範例程式碼 4-19：比較 Convert 函數的 Style 參數與轉換資料長度的差異

其執行結果如圖 4-16 所示：

	0x49747A696B to VARCHAR(12) Style 0
1	Itzik

	'0x49747A696B' to VARBINARY(5) Style 0
1	0x3078343937

	0x3078343937 to VARCHAR(5) Style 0
1	0x497

	0x49747A696B to VARCHAR(12) Style 1
1	0x49747A696B

	'0x49747A696B' to BINARY(10) Style 1 補 0
1	0x49747A696B0000000000

	0x49747A696B to VARCHAR(5) Style 2 截斷，需要偶數邊界
1	4974

	0x49747A696B to VARCHAR(6) Style 2 截斷，需要偶數邊界
1	49747A

	'49747A696B' to VARBINARY(5) Style 2
1	0x49747A696B

圖 4-16：Convert 函數利用不同 Style 參數所做字串與二進位資料類型的轉換

有了 Convert 搭配 Style 1 或 2 的方式後，可以用來轉換透過雜湊 (Hash) 運算後的密碼值，由於 HashBytes 系統函數在使用 MD2, MD4, MD5 等演算法時，統一回傳 128 bits (16 bytes) 二進位資料，搭配 SHA 和 SHA1 演算法則回傳 160 bits (20 bytes) 二進位資料，所以經由如下的語法，就可以字串的方式存放或傳回給其他應用：

```
SELECT CONVERT(VARCHAR(40),HASHBYTES('sha1','Password'),2)
```

除了利用上述的 Convert 系統函數外，也可以叫用 sys.fn_VarBinToHexStr 系統函數，它傳入 varbinary(max) 類型的參數，一樣傳出 nvarchar(max) 類型資料，並將輸入的二進位值以 16 進位數值的文字描述法呈現。範例如下：

```
select sys.fn_VarBinToHexStr(HASHBYTES('sha1','Password'))
```

以上兩句語法依其在文章的先後順序，執行結果如圖 4-17 所示：

	(沒有資料行名稱)
1	8BE3C943B1609FFFBFC51AAD666D0A04ADF83C9D

	(沒有資料行名稱)
1	0x8be3c943b1609fffbfc51aad666d0a04adf83c9d

圖 4-17：將 2 進位資料直接以 16 進位文字呈現

二進位資料就介紹到此，接下來繼續討論大型資料類型。

4-2-6｜大型資料

當單一資料超過 8 K 位元組大小時，SQL Server 提供可存放在資料表內的大型資料類型，如表 4-13 所示：

類型名稱	說明
Varchar(max)*	可變動長度的非 unicode 字元資料
Text	可變動長度的非 unicode 字元資料，建議不要再使用，改以 Varchar(Max) 取代
NVarchar(max)*	可變動長度的 unicode 字元資料
NText	可變動長度的 unicode 字元資料，建議不要再使用，改以 NVarchar(Max) 取代
Varbinary(max)*	可變動長度的二進位資料
Image	可變動長度的二進位資料，建議不要再使用，改以 Varbinary(Max) 取代
XML*	以 Unicode (UTF-16) 來儲存的 XML 資料
CLR 自訂類型*	稱為使用者自訂資料類型 (UDT)，也就是將 .NET 撰寫的類別資料結構序列化存放

表 4-13：SQL Server 2008 預設支援的大型資料類型

* 代表 SQL 2005 後才新增的資料類型

表 4-13 中，MAX 關鍵字是SQL 2005 新增功能，當宣告了 MAX 關鍵字，varchar、nvarchar、varbinary 等類型欄位最多可以存放 2Giga (2^31-1) 位元組的資料。而不再受限於以往的 8K (也就是 SQL Server 資料存取基本單位的大小)。大型資料的最大長度都是 2G 位元組，而 CLR UDT 需要等到 SQL Server 2008 才可以存放到 2G 大小。

varchar(max)、nvarchar(max)、varbinary(max) 三種資料類型是 SQL Server 2005 提供的新資料類型，可用來取代 text、ntext 以及 image 等資料類型。由 SQL Server 自行決定存放大型資料的方式，而不需我們事先決定以指標結構存放 text ，還是一般資料頁內資料列的 char 等資料格式。

而以 VARCHAR(MAX) / NVARCHAR(MAX) 取代 text、ntext 最大的好處在於，一般的 T-SQL 字元處理函數都可以操作 VARCHAR(MAX) / NVARCHAR(MAX) 欄位內的資料，而不像 text、ntext 會讓很多字元處理函數無法使用。

SQL Server 2005 針對這三種資料類型在 T-SQL Update 語法上，也新增了 Write 方法，功能與以往 Text 和 Image 型態的文字指標功能近似，讓你可以局部更新大型資料，如範例程式碼 4-20 所示：

```
--NVARCHAR(MAX),VARCHAR(MAX),VARBINARY(MAX) 等資料類型
--新增的 .write 更新方式
CREATE TABLE tbTest(c1 NVARCHAR(MAX))

INSERT tbTest VALUES(REPLICATE(N'1234567890',1000))
UPDATE tbTest SET c1.WRITE('lmnop',10,3)
SELECT * FROM tbTest
```

範例程式碼 4-20：使用新增的 Write 語法對大型資料局部更新

範例程式碼 4-20 透過 REPLICATE 函數將 1234567890 字元重複 1000 次後加入資料表；藉以模擬大量文字當做 c1 欄位的值。然後透過 WRITE 語法將欄位中第 10 個字開始，以 lmnop 字元取代掉原始內容的 3 個字。

表 4-13 的大型資料可以儲存在個別資料列中，或是獨立於資料表之外的大型字元或二進位字串結構，這是由 sp_tableoption 系統預存程序中的兩個選項來控制：

- large value types out of row 選項：適用於 varchar(max)、nvarchar(max)、varbinary(max) 和 xml 資料類型。
- text in row 選項：適用於 text、ntext 和 image 資料類型。

除非將「text in row」選項設為 ON，否則 text、ntext 或 image 資料類型都是儲存在資料列之外的大型字元或二進位字串。資料列僅存放 16 位元組的指標，指向由內部指標所組成之樹狀結構的根節點。這些指標會對應用來儲存字串片段的分頁，其示意圖如圖 4-18 所示：

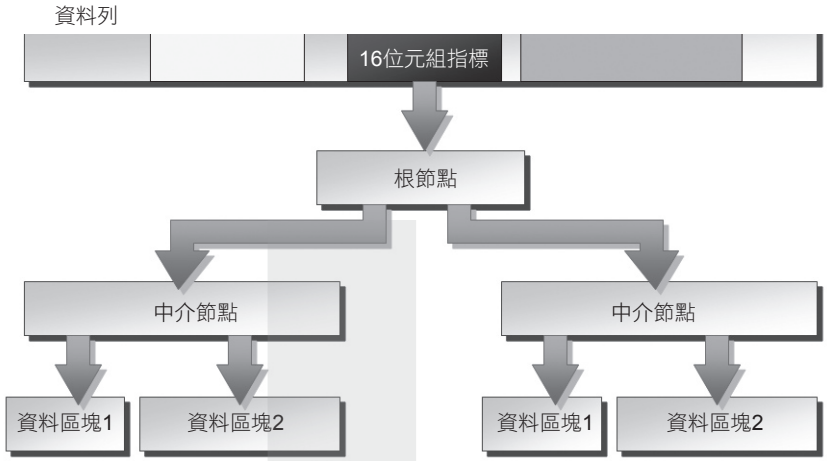


圖 4-18：SQL Server「儲存在資料列之外」的存放大型資料類型方式

可以針對包含大型資料類型資料行的資料表來設定「text in row」選項。也可以指定「text in row」選項限制資料大小，從 24 到 7,000 位元組。

在此需要強調的是：未來的 SQL Server 版本中，將會移除「text in row」選項。避免在新的開發工作中使用此選項，並請計畫修改目前使用「text in row」的應用程式。建議使用 varchar(max)、nvarchar(max) 或 varbinary(max) 資料類型來儲存大型資料。若要控制這些資料類型的儲存方式是同資料列或資料列外，可使用「large value types out of row」選項。在此，也就不細談 text、ntext 或 image 資料類型，若有需要，請參考線上說明：

```
Database Engine→開發→設計及實作結構化儲存體→資料表→設計資料表→指派資料行的資料類型→同資料列資料
```

同樣地，預設「large value types out of row」選項為 OFF，varchar(max)、nvarchar(max)、varbinary(max) 以及 xml 等類型的資料會儘可能地儲存在資料列中，大數值資料類型的最大同資料列儲存體設定為 8,000 位元組。與「text in row」 選項不同的是，不能為資料表中的資料行指定同資料列限制。若設為 ON，就會以非資料列方式來儲存值，只有 16 位元組的指標會儲存在記錄中。

將大型資料類型儲存在資料列，資料庫引擎讀寫時，不需存取其它分頁或分頁集。讓讀寫速度幾乎等同一般資料類型。反之，若是以非資料列來儲存值，資料庫引擎就會引發額外的分頁讀寫動作。

另外，也可能將指標從根節點移到資料列本身，資料庫引擎不使用根節點，以省略讀寫時的分頁存取，這可提高執行效能。當使用根節點時，會將其儲存成大型資料分頁中的字串片段，最多可容納五個內部指標。資料列內必須有 72 位元組的空間，才能儲存五個指標。

當「text in row」選項為 ON，或「large value types out of row」選項為 OFF，如果資料列中的空間不足，無法容納指標，資料庫引擎可能需要配置 8K 分頁來容納它們。如果該值的資料長度超過 40,200 位元組，則需要五個以上的同資料列指標，這時會只存 24 位元組在主要資料列中，並改在大型資料儲存空間中配置額外的資料頁。

當大型字串儲存於資料列時，其儲存方式和可變長度字串類似。資料庫引擎會以大小遞減的順序來將資料行排序，並將超過 8K 的資料存到非資料列，讓剩餘的資料行符合分頁大小。

經由 sp_tableoption 系統預存程序可啟用與停止資料表的「large value types out of row」選項，設定方式如下：

```
CREATE TABLE tbImg
(
PK INT IDENTITY PRIMARY KEY,
FileName NVARCHAR(60),
FileType NVARCHAR(30),
ImageFile VARBINARY(Max)
)
go

exec sp_tableoption N'tbImg', 'large value types out of row', 'ON'
```

範例程式碼 4-21：設定大型資料要放在資料列內或外

當改變「large value types out of row」設定值時，現存的 varchar(max)、nvarchar(max)、varbinary(max) 及 xml 值並不會立刻轉換。在後續更新時，才會變更儲存方式。新增的記錄，會依照實際的資料表選項來儲存。

若要檢查特定資料表的「large value types out of row」設定值，可查詢 sys.tables 系統檢視的「large_value_types_out_of_row」資料行。如果資料表未啟用「large value types out of row」，該資料行是 0，若啟用則為 1。範例如下：

```
select name,text_in_row_limit,large_value_types_out_of_row from sys.tables
where name LIKE 'tb%'
```

執行結果如圖 4-19 所示：

	name	text_in_row_limit	large_value_types_out_of_row
1	tblmg	0	1
2	tbTest	0	0

圖 4-19：查詢資料表對新舊兩種大型資料的存放方式設定

通常存取該資料列時，需同時存取大型資料類型的值，則以同資料列方式來儲存較有幫助。否則，由於大量資料儲存在資料列本身，將降低每一分頁可容納的資料列數量。而大部分存取資料表的陳述式都不存取大型資料類型的資料行，則減少分頁中的資料列，將增加處理查詢時必須讀取的分頁。換句話說，若多數對資料表的存取不需參考大型資料行，建議將此選項設為 ON。以資料列外方式來儲存大型資料，讓每個分頁可容納的資料列更多，以降低掃描資料表時所需的 I/O 數量。

在先前的範例程式碼 4-21 中，簡單地將大型資料與一般資料放在相同的資料表中，但若大型資料在一般應用時，不常用到，為避免使用者查詢時，未明確指定欄位 (例如，用 select * 取資料)，而涵蓋了大型資料欄位，雖然使用者用不到，但卻需要從硬碟載入記憶體，或在網路間傳遞，這將浪費許多資源。建議將大型資料獨立放在另一個資料表中，然後以 1 對 1 連結的方式對應存放相關資料的主資料表，當使用者明確需要時，再 Join 取得。

最後，當使用了大型資料類型後，透過 sp_help 系統預存程序；或 sys.columns 系統檢視分析資料表時，相關資料行的 Length 欄位將傳回 -1。範例如下所示：

```
exec sp_help 'tbImg'
```

執行結果如圖 4-20 所示：

	Name	Owner	Type	Created_datetime
1	tblImg	dbo	user table	2009-01-07 12:59:26.260

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	PK	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2	FileName	nvarchar	no	120			yes	(n/a)	(n/a)	Chinese_Taiwan_Stroke_CI_AS
3	FileType	nvarchar	no	60			yes	(n/a)	(n/a)	Chinese_Taiwan_Stroke_CI_AS
4	ImageFile	varbinary	no	-1			yes	no	yes	NULL

	Identity	Seed	Increment	Not For Replication
1	PK	1	1	0

	RowGuidCol
1	No rowguidcol column defined.

	Data_located_on_filegroup
1	PRIMARY

	index_name	index_description	index_keys
1	PK__tblImg__3215078720C1E124	clustered, unique, primary key located on PRIMARY	PK

	constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_keys
1	PRIMARY KEY (clustered)	PK__tblImg__3215078720C1E124	(n/a)	(n/a)	(n/a)	(n/a)	PK

圖 4-20：大型資料類型的長度以 -1 表示

4-2-7｜特殊資料類型

特殊資料類型是指不屬於在前幾節中曾經討論過的任何一種資料類型。在 SQL Server 2008 中，特殊資料類型包括 bit、hierarchyid、sql_variant、sysname、table、timestamp、CLR UDT、自訂資料表類型、別名資料…等諸多類型，各有其特殊應用。展望未來，SQL Server 期待將世界上所有資料都納入管理，相信在每個新版中，都會增加些新的資料類型 ☺，以因應更多特殊的用途。

各特殊資料類型的基本定義如表 4-14 所示：

類別名稱	說明
bit	儲存 0 或 1 的數值資料類型。
cursor^	資料指標 (cursor) 的參照。
HierarchyID**	存放具有階層式結構意義的資料。
Geography**	以 .NET Common Language Runtime (CLR) 實作的資料類型，儲存橢圓體 (圓形地球) 資料。

表 4-14：T-SQL 提供的特殊資料類型

類別名稱	說明
Geometry**	以 .NET Common Language Runtime (CLR) 實作的資料類型，儲存平面空間資料類型，一般稱之為 Euclidean (平面) 座標系統。
Rowversion(timestamp)	資料庫層級的唯一識別碼，用來表示資料庫中發生修改時的順序。之前被稱為 timestamp，但它僅是二進位數字，與時間或日期無關。
Sql_variant	允許單一資料欄位能夠儲存多種型態的資料 (但是 text, ntext, rowversion 與 sql_variant …等除外)。
sysname	可儲存資料庫的物件名稱。sysname 的確切定義與 SQL 物件識別碼的規則有關。因此，它可能會隨著不同的 SQL Server 執行個體而改變。sysname 在功能上與 nvarchar(128) 相同，不同的是，sysname 預設為 NOT NULL。在舊版 SQL Server 中，sysname 定義成 varchar(30)。 在區分大小寫或有二進位定序的資料庫中，只有當 sysname 以小寫出現時，系統才會將它辨識為 SQL Server 系統資料類型。
Table^	一種特別的資料型別，用來暫時儲存一些結果集。只能使用於區域變數，以及做為自訂函數傳回值的資料型別。
uniqueidentifier	128 bit (16 位元組大小) 的全域唯一識別碼 (Global unique identifier ；GUID，也稱為 Universal unique identifier)。
XML*	從 SQL Server 2005 開始，XML 值可用原生方式儲存在 xml 資料類型資料行中，依照 XML 結構描述的集合來設定類型，或維持不具 XML 結構描述的類型。
CLR UDT*	利用 .NET 程式語言所實作的類別 (class) 或結構 (structure)；來定義 SQL Server 內的資料類型。而 SQL Server 2008 的 HierarchyID、Geometry 和 Geography 都是此種用 .NET 實作的資料類型。
別名	針對特定用途自訂的描述名稱和格式，替 SQL Server 基底資料類型賦予不同的別名。
使用者定義資料表類型**^	使用者定義的資料表結構型別。可當作預存程序或函數的資料表值參數，或在批次、預存程序或函數的內文中使用的資料表變數。

表 4-14：T-SQL 提供的特殊資料類型（續）

表 4-14 中，各符號所代表的意義如下：

- *：代表該資料類型在 SQL Server 2005 版本後才提供。
- **：代表該資料類型在 SQL Server 2008 版本後才提供。
- ^：代表該資料類型不可用在資料表內資料行的類型宣告。

表 4-14 中有些資料類型在此僅簡單說明：

- **hierarchyid 資料類型**：用來管理具有階層式結構的階層式資料和資料表。若要在 T-SQL 程式碼中使用階層式資料，可考慮使用 hierarchyid 資料類型所提供的各類函數，以簡化開發。
- **Geography、Geometry 空間資料類型**：存放經緯度或 X、Y 座標資料。
- **XML 資料類型**：可以在資料表中建立 xml 類型的資料行，或是xml 類型的變數等，儲存 XML 執行個體。
- **CLR UDT 資料類型**：SQL Server 2005 後，可以利用 .NET Framework Common Language Runtime (CLR) 建立的組件來當作資料庫物件。藉由 CLR 所提供的豐富程式設計模型，建立觸發程序、預存程序、函數、彙總函數和使用者自訂類型 (CLR UDT) 等。

由於上述資料類型的用法複雜，需較多程式寫作的經驗，你可以參閱 DBWorld (http://www.dbworld.com.tw) 雜誌相關的專欄，我們將在《SQL Server 2008 資料庫開發》一書中再詳細介紹。

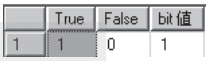
以下個別介紹不同資料類型的特徵。

4-2-7-1 bit

一般來說，若僅是「是 / 否」、「真 / 偽」兩種值的資料，可用 bit 資料類型來存。例如，性別、婚姻、兵役…等。而字串值 true 和 false 也可以轉換成 bit 值，如下列範例所示：

```
DECLARE @i BIT=1
SELECT CONVERT (bit, 'true') [True],CONVERT(bit, 'false') [False],@i [bit 值]
```

在上述範例中，true 會轉換成 1 而 false 會轉換成 0。執行結果如圖 4-21 所示：



	True	False	bit 值
1	1	0	1

圖 4-21：字串值 true 和 false 可以轉換成 bit 值的 1 與 0

若資料表內僅有一個資料行的類型為 bit，則該資料行仍占有 1 位元組 (byte)。但若資料表內有小於等於 8 個 bit 類型的資料行，這些資料行將共佔掉一個位元組，超過 8 但小於等於 16 個bit 類型資料行，則佔掉 2 個位元組，依此類推。因此，只要資料表內有多個資料行儲存「是 / 否」、「真 / 偽」兩種值的資料，宣告成 bit 資料類型是最省空間，並提升效率。

4-2-7-2 cursor

SQL Server 實作的 T-SQL 資料指標 (Cursor) 是以 DECLARE CURSOR 語法為基礎，主要用於 T-SQL 指令碼、預存程序和觸發程序。資料指標是在伺服器上實作，從用戶端傳送到伺服器的 T-SQL 陳述式來管理。

預存程序只能在 OUTPUT 參數使用 cursor 資料類型。如果參數指定 cursor 資料類型，就需要 VARYING 和 OUTPUT 參數。如果參數指定 VARYING 關鍵字，資料類型必須為 cursor，也必須指定 OUTPUT 關鍵字。在此利用預存程序簡單示範傳出資料指標的方式如下 (如何撰寫預存程序，請參考《第10章：T-SQL 程式設計概論》)：

```
--建立測試用資料
CREATE TABLE t(C1 INT,C2 CHAR(1))
GO
DECLARE @i INT=1
WHILE @i<10
BEGIN
   INSERT t VALUES(@i,CHAR(@i+64))
   SET @i+=1
END
GO
```

```
--建立在 t 資料表上；宣告並隨後開啟資料指標
CREATE PROCEDURE dbo.spCursor
    @Cursor CURSOR VARYING OUTPUT
AS
    SET @Cursor = CURSOR
    FORWARD_ONLY STATIC FOR
      SELECT *
      FROM t;
    OPEN @Cursor;
```

範例程式碼 4-22：建立透過參數傳出資料指標的預存程序

接下來，執行一個宣告區域資料指標變數的批次、執行此程序將資料指標指派給區域變數，再從資料指標擷取資料列：

```
DECLARE @MyCursor CURSOR;
EXEC dbo.spCursor @Cursor = @MyCursor OUTPUT;
WHILE (@@FETCH_STATUS = 0)
BEGIN;
     FETCH NEXT FROM @MyCursor;
END;
CLOSE @MyCursor;
DEALLOCATE @MyCursor;
```

範例程式碼 4-23：透過 T-SQL 批次語法使用預存程序傳出的資料指標

範例程式碼 4-23 中，透過資料指標與 Fetch Next 語法取回所有的記錄，而後關閉 / 釋放資料指標結構。在此僅作簡單的示範，關於如何在 T-SQL 語法中使用 Cursor，你可以參閱 DBWorld (http://www.dbworld.com.tw) 雜誌相關的專欄，我們將在《SQL Server 2008 資料庫開發》一書中詳細介紹。

4-2-7-3 sql_variant

除了表 4-15 所列的資料類型外，sql_variant 資料類型可讓單一資料行、參數或變數儲存不同資料類型的資料值：

varchar (max)	varbinary (max)	nvarchar (max)	xml	text	ntext
image	timestamp	sql_variant	使用者自訂類型	hierarchyid	cursor
table					

表 4-15：sql_variant 無法存放的資料類型

sql_variant 類型可以存放各種基底資料類型，這提供了存放未知資料的彈性，例如，「屬性 / 值」資料表，假設要存放賣場產品的屬性，但會有什麼性質的資料事前無法規劃，因此，用一個資料表擺放屬性與值的對應，而設定成 sql_variant 類型的資料行，則可以存放各式資料©。

sql_variant 資料行的每筆記錄 (執行個體) 都會同時存放資料值和描述此值的中繼資料。當資料行內包含了各式各樣的資料後，你可能想要判讀該資料的實際規格。透過 SQL_VARIANT_PROPERTY 系統函數，能取得有關 sql_variant 值的屬性資訊，如資料類型、有效位數或小數位數、總資料長度…等。範例程式碼 4-24 利用 SQL_VARIANT_PROPERTY 系統函數呈現相關的中繼資料：

```
CREATE    TABLE tbProperty(ProdID INT,Property NVARCHAR(50),Value sql_
variant)
go

DECLARE @i INT=1
INSERT tbProperty VALUES(1,N'重量',@i) --使用預設的類型
SET @i=NULL
INSERT tbProperty VALUES(1,N'重量',@i) --NULL 沒有類型，就算變數有預設的類型
INSERT tbProperty VALUES(1,N'長度',999999999999999999999999) --超過Bigint 範
圍，自動轉成numeric
INSERT tbProperty VALUES(2,N'重量',CAST(1234 AS FLOAT)) --透過cast 指定類型
INSERT tbProperty values(3,N'字碼','煊堃')  --最大長度定在8000
INSERT tbProperty values(3,N'字碼',N'煊堃')  --自動轉成unicode
INSERT tbProperty VALUES (3,N'字碼',CAST(N'煊堃' AS NVARCHAR(50)))

SELECT Value,
   SQL_VARIANT_PROPERTY(Value,'BaseType') AS [基礎資料類型],
   SQL_VARIANT_PROPERTY(Value,'Precision') AS [數值基底資料類型的位數],
   SQL_VARIANT_PROPERTY(Value,'Scale') AS [小數點右側的位數],
   SQL_VARIANT_PROPERTY(Value,'TotalBytes') AS [中繼資料和資料所需要的位元組數],
   SQL_VARIANT_PROPERTY(Value,'Collation') AS [定序],
   SQL_VARIANT_PROPERTY(Value,'MaxLength') AS [最大資料類型長度(位元組)]
FROM  tbProperty
```

範例程式碼 4-24：將不同類型值放入 sql_variant 類型資料行

執行結果如圖 4-22 所示：

	Value	基礎資料類型	數值基底資料類型的位數	小數點右側的位數	中繼資料和資料所需要的位元組數	定序	最大資料類型長度(位元組)
1	1	int	10	0	6	NULL	4
2	NULL	NULL	NULL	NULL	NULL	NULL	NULL
3	999999999999999999999999	numeric	24	0	17	NULL	13
4	1234	float	53	0	10	NULL	8
5	??	varchar	0	0	10	Chinese_Taiwan_Stroke_CI_AS	8000
6	煙筆	nvarchar	0	0	12	Chinese_Taiwan_Stroke_CI_AS	8000
7	煙筆	nvarchar	0	0	12	Chinese_Taiwan_Stroke_CI_AS	100

圖 4-22：資料庫引擎自行為 sql_variant 類型決定最適合的基底資料類型

由圖 4-22 可知，資料庫引擎會因為輸入資料的內容，自動判讀適合的類型。這讓我想起 VB 或 VBScript 的 variant 型別，或是 .NET 的 Object 基礎類別，要掌控得好非常不易，而這往往是潛藏 bug 的地方。若要使用它，最好先熟知基本類型的運作規則，你可以參考如下的線上說明：

```
Database Engine→開發→查詢及變更資料→存取與變更資料庫資料→Transact-SQL 的元素→資料類型 (Database Engine)→使用特殊資料類型→使用 sql_variant 資料
```

4-2-7-4 sysname

sysname 資料類型可用於儲存資料物件名稱的資料行、變數和預存程序參數。sysname 的確切定義與資料庫引擎識別碼的規則有關。因此，它可能會隨著不同的 SQL Server 版本而變。sysname 在功能上與 nvarchar(128) 相同，不同的是 sysname 預設為 NOT NULL。在 SQL Server 6.5 版或更舊版本僅支援較小的識別碼，換句話說，SQL Server 6.5 版以前 sysname 定義成 varchar(30)。

需要注意的是：在區分大小寫或有二進位定序的資料庫中，只有當 sysname 以小寫出現時，系統才會將它辨識為 SQL Server 系統資料類型。

簡單做一個測試如下：

```
CREATE TABLE t(c1 sysname)

SELECT name,system_type_id,TYPE_NAME(system_type_id) [資料類型],max_
length,is_nullable
FROM sys.columns WHERE object_id=object_id('t')
```

範例程式碼 4-25：檢視 sysname 資料類型在 SQL Server 2008 所等同的基底類型

⑫ 這只是想像的用途。其實，我不知道 sql_variant 適用在哪裡，因為不曉得如何搭配撰寫應用程式。

執行結果如圖 4-23 所示：

	name	system_type_id	資料類型	max_length	is_nullable
1	c1	231	nvarchar	256	0

圖 4-23：sysname 資料類型在 SQL Server 2008 所等同的基底類型

4-2-7-5 table

此種資料類型僅可用來定義 table 類型的本機變數，以及使用者自訂函數的傳回值，無法當作資料行的資料類型使用。建立 table 資料類型變數的功能就如同暫存資料表，它可用來儲存結果集，以便之後處理。

資料表變數或傳回值的定義包含資料行定義、資料類型、有效位數、每個資料行的小數位數，以及選擇性的 PRIMARY KEY、UNIQUE、NULL、CHECK 等條件約束。

table 變數中儲存的資料列格式，或使用者定義多重陳述式資料表值函數所傳回的資料列格式；必須在宣告此變數或建立此函數時定義 (如何撰寫使用者自訂函數，請參考《第10章：T-SQL程式設計概論》)。其語法是以 CREATE TABLE 語法為基礎，如範例程式碼 4-26 所示：

```
--透過四個參數測試Primary Key、Not Null 和check 等條件約束的功能
CREATE FUNCTION fn(@p1 int,@p2 int,@p3 int, @p4 int)
RETURNS @tb TABLE
(
  c1 int primary key,
  c2 int not null check(c2 > 10)
)
AS
BEGIN
  INSERT @tb VALUES(@p1,@p2)
  INSERT @tb VALUES(@p3,@p4)
  RETURN
END
GO

select * from dbo.fn(1,12,2,20)
select * from dbo.fn(1,12,2,NULL)
```

範例程式碼 4-26：簡單操作資料表類型

在範例程式碼 4-26 一開始，就以 Returns @tb TABLE 定義函數回傳的資料表變數結構。接著，在函數定義中，賦予該資料表變數相關記錄後回傳。

最下方，利用 select 語法呼叫自訂的資料表值函數，在此，可以透過不同的參數值，測試先前定義的「主鍵」，「不可 NULL」，和「Check」等條件約束的功能。

4-2-7-6 uniqueidentifier

在分散式資料處理時，想要維持散在各處多個資料表的「唯一」主鍵 (例如，各分店自行建立的客戶編號)，一般是設計成雙資料行主鍵，由 identity 數值搭配區域碼。而另一個作法是使用 uniqueidentifier 類型[13]。要產生一個 UniqueIdentifier 值，最簡單的方式是叫用 newid 系統函數，範例如下：

```
DECLARE @guid uniqueidentifier = NEWID()
PRINT @guid
```

執行結果如下：

```
9E902338-0FE9-47B0-8FFE-1D1D279248AF
```

另一個方式是呼叫 NEWSEQUENTIALID 系統函數，它會根據所在電腦建立 GUID，每次新產生出的 GUID 值大於之前由這個函數所產生的任何 GUID。

如果有資料安全的顧慮，請勿將 GUID 資料值暴露給使用者看，或不要使用這個函數。因為使用者不難猜出下一個 GUID 值，進而存取與該 GUID 相關聯的資料。

NEWSEQUENTIALID 系統函數只能用在 uniqueidentifier 類型之資料表資料行的 DEFAULT 條件約束。例如：

⑬ 筆者看過一個比喻，若要兩個符合 GUID / UUID 公式產生出來的值相等，其機率小於彗星撞地球，換句話說，若運氣不好，分處兩地的資料表產生出的 UniqueIdentifier 值相等，你也不用懊惱，因為地球已經不在了 ☺

```
CREATE TABLE tb (C1 uniqueidentifier DEFAULT NEWSEQUENTIALID())
GO
INSERT tb DEFAULT VALUES
INSERT tb DEFAULT VALUES
WAITFOR DELAY '00:01:30'
INSERT tb DEFAULT VALUES
INSERT tb DEFAULT VALUES

select c1 [uniqueidentifier],CONVERT(binary(16),c1) [uniqueidentifier 底層的 2
進位資料] from tb
```

範例程式碼 4-27：uniquedientifer 類型的基底類型為 2 進位資料

範例程式碼 4-27 的執行結果如圖 4-24 所示：

	uniqueidentifier	uniqueidentifier 底層的 2 進位資料
1	59024D4B-C3DC-DD11-8522-001C258B4E99	0x4B4D0259DCC311DD8522001C258B4E99
2	5A024D4B-C3DC-DD11-8522-001C258B4E99	0x4B4D025ADCC311DD8522001C258B4E99
3	399DF180-C3DC-DD11-8522-001C258B4E99	0x80F19D39DCC311DD8522001C258B4E99
4	3A9DF180-C3DC-DD11-8522-001C258B4E99	0x80F19D3ADCC311DD8522001C258B4E99

圖 4-24：比較 uniquedientifer 類型表面呈現與底層實際存放的 2 進位資料之差異

在範例程式碼 4-27 中，故意在利用 NEWSEQUENTIALID 系統函數產生 GUID 值時，延遲 1 分 30 秒，讓時間因素影響 NEWSEQUENTIALID 系統函數。當直接檢視 UniqueIdentifier 底層的 2 進位資料轉化成 GUID 表示法時，會覺得資料值並沒有越來越大，但若改以 2 進位資料直接檢視時，可以發現從 0x 開始的前四個位元組之順序與 GUID 表示法是顛倒的，而 GUID 實際的 2 進位值的確是越來越大。而利用 UniqueIdentifer 資料行排序時，是以 2 進位值為準。

當 NEWSEQUENTIALID 用於 DEFAULT 運算式時，不能與其他純量運算子結合。例如，不可執行下列作業：

```
CREATE TABLE t(c1 uniqueidentifier DEFAULT dbo.fn(NEWSEQUENTIALID()))
```

在上述範例中，即便 fn() 是個簡單的使用者自訂純量值函數，僅接受和傳回 uniqueidentifier 值，仍會有錯誤。我們直接以函數呼叫來證明 NEWSEQUENTIALID 不能在查詢中參考。其範例如下：

```
CREATE FUNCTION fn(@p UNIQUEIDENTIFIER)
RETURNS UNIQUEIDENTIFIER
AS
BEGIN
   RETURN @p
END
GO
SELECT dbo.fn(NEWID())
```

範例程式碼 4-28：簡單傳回 uniquedientifer 類型使用者自訂純量值函數

範例程式碼 4-28 中，雖然可以將 NEWID() 系統函數產生的 GUID 值賦予 fn 使用者自訂純量值函數當參數，但若改以如下的語法呼叫：

```
SELECT dbo.fn(NEWSEQUENTIALID())
```

則會出現如下的錯誤：

```
訊息302，層級16，狀態0，行1
在CREATE TABLE 或ALTER TABLE 陳述式中，newsequentialid() 內建函數只能用於類型為
'uniqueidentifier' 之資料行的DEFAULT 運算式中。它不可以和其他運算子結合，形成複雜的純量
運算式。
```

另一點要注意的是：每個使用 NEWSEQUENTIALID() 所產生的 GUID 在該電腦上都是唯一的。但只有當來源電腦具有網路卡時，使用 NEWSEQUENTIALID() 所產生的 GUID 在多部電腦上才是唯一的。

4-2-7-7 timestamp / rowversion

timestamp / rowversion 資料類型與時間或日期無關。timestamp 值是二進位數字，指出資料庫中修改資料記錄的相對先後順序，跨資料庫則沒有更新的相對順序。請勿在索引鍵中使用 timestamp 資料行，尤其是主索引鍵，因為每次修改資料列時，timestamp 值就會變更。簡單的測試程式碼如下：

```
CREATE TABLE t(PK INT IDENTITY,c1 rowversion,c2 INT DEFAULT 0)
CREATE TABLE t2(PK INT IDENTITY,c1 rowversion,c2 INT DEFAULT 0)
CREATE TABLE t3(PK INT IDENTITY,c1 int DEFAULT 0)

INSERT t DEFAULT VALUES
INSERT t2 DEFAULT VALUES
```

```
INSERT t3 DEFAULT VALUES

SELECT (SELECT c1 FROM t) t_c1,(SELECT c1 FROM t2) t2_c1,(SELECT c1 FROM
t3) t3_c1

--嘗試更新，實則根本沒有動 t,t2 資料表，
--但 rowversion 依然自動加一了
--且以資料庫為單位，資料庫內跨資料表的 rowversion 是唯一的
declare @i int=1
while @i<3
begin
             update t set @i+=1
             update t2 set @i+=1
             update t3 set @i+=1
end

SELECT (SELECT c1 FROM t) t_c1,(SELECT c1 FROM t2) t2_c1,(SELECT c1 FROM
t3) t3_c1
```

範例程式碼 4-29：資料庫引擎會自動維護該資料庫內更新的順序數值

執行結果如圖 4-25 所示：

	t_c1	t2_c1	t3_c1
1	0x0000000000000FD3	0x0000000000000FD4	0

	t_c1	t2_c1	t3_c1
1	0x0000000000000FD5	0x0000000000000FD6	0

圖 4-25：比較相同資料庫內，更新兩個資料表，其 rowversion 類型的資料會持續累加

只要是嘗試更動某一筆記錄，資料庫引擎會自動為該筆記錄的 rowversion 類型資料行加 1。接續上一段範例程式碼，更動 t 資料表類 PK 資料行為 1 的 c2 資料行記錄：

```
INSERT t DEFAULT VALUES
SELECT * FROM t
UPDATE t SET c2=1 WHERE PK=1
SELECT * FROM t
go

drop table t,t2,t3
```

範例程式碼 4-30：資料庫引擎會自動維護該資料庫內，各資料表的資料列更新順序數值

更動完某筆記錄後，重新查詢 t 資料表，執行結果如圖 4-26 所示：

	PK	c1	c2
1	1	0x0000000000000FD5	0
2	2	0x0000000000000FD7	0

	PK	c1	c2
1	1	0x0000000000000FD8	1
2	2	0x0000000000000FD7	0

圖 4-26：更新資料列時，該列 rowversion 類型的資料行會自動累加

從圖 4-26 記錄更動前後的 RowVersion 類型欄位比較，可以看到因為 FD7 被 PK=2 的資料行用掉，所以 PK=1 的記錄更動後，該筆記錄的 c1 欄位由 FD5 變成 FD8。

由於更動資料列時，資料庫引擎會自動會為該列的 rowversion 類型資料行遞增值。所以，在撰寫多人存取的前端程式，於先前取得資料表內容值，在使用者端離線運作，而後要再更新相同記錄時，可以根據存取前後rowversion資料行的值是否相等，來判讀期間是否有其他人更動過該筆記錄。

若要記錄資料表中發生資料修改的時間，請使用 datetime2 或 smalldatetime 資料類型來記錄，在修改時自行更新這些值。

4-2-7-8 別名資料型別

利用 Create Type 陳述式可以在目前資料庫中；建立「別名資料類型」、「使用者定義類型」或「使用者定義資料表類型」。別名資料型別的實作是以 SQL Server 原生系統類型為基礎。使用者自訂類型是利用 .NET Framework Common Language Runtime (CLR) 中之組件的類別或結構來實作，而使用者定義資料表類型，則是以宣告資料表的方式定義類型，其 T-SQL 語法定義如下：

```
CREATE TYPE [ schema_name. ] 類型名稱
{
    FROM base_type [ ( precision [ , scale ] ) ] [ NULL | NOT NULL ]
  | EXTERNAL NAME 組件名稱 [ .類別名稱 ]
  | AS TABLE ( { <資料行定義> | <計算資料行定義> }
        [ <資料表條件約束> ] [ ,...n ] )
} [ ; ]
```

別名資料型別讓你使用可描述特定用途的名稱和格式，來擴充 SQL Server 基底資料型別 (例如 varchar)。另外，除了 T-SQL 的 Create Type 語法外，也可以利用 sp_addtype 系統預存程序來建立別名資料類型。例如，下列陳述式會實作 birthday 使用者定義資料類型，此資料類型是以 datetime 資料類型為基礎，而且允許 NULL 值：

```
EXEC sp_addtype birthday, datetime, 'NULL'
```

或是透過 SQL Server Management Studio 的「物件總管」，展開「可程式性」→「類型」節點後，滑鼠右鍵點選「使用者定義資料類型」節點，選擇「新增使用者定義資料類型」選項，如圖 4-27 所示：

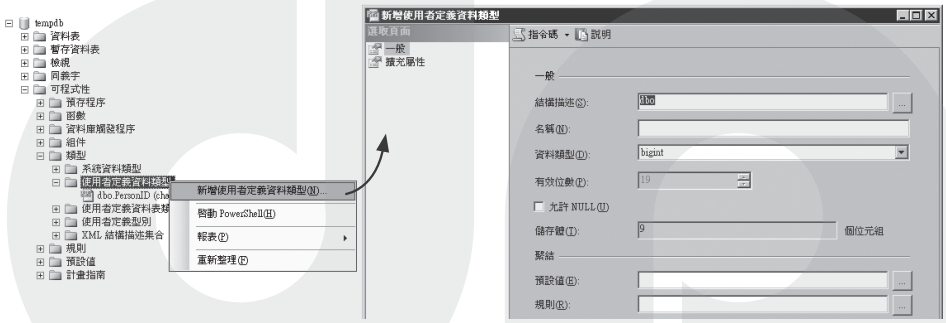


圖 4-27：新增使用者定義資料類型

建立完使用者定義資料類型後，便可以當作一般的資料類型使用。例如，我們的身分證格式是 char(10)。然後建立 CHECK 條件約束，以便強制將社會安全碼的格式儲存在資料表中，如下列範例所示：

```
CREATE TYPE PersonID FROM CHAR(10) NOT NULL
GO

CREATE TABLE t(EmployeeID int PRIMARY KEY, ID PersonID
CHECK (ID LIKE '[a-z][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]' )
)

INSERT t VALUES(1,'A123456789')
--驗證該 PersonID 別名資料類型不可以 NULL
INSERT t VALUES(2,NULL)

DROP TYPE PersonID
```

範例程式碼 4-31：建立並使用別名資料類型

4-2-7-9 使用者定義資料表類型

「使用者定義資料表類型」是依照資料表結構規範的使用者定義型別。可利用該類型宣告預存程序或函數的「資料表值參數」，或是在批次、預存程序或函數的內文中，宣告成該類型的資料表變數。

若要建立使用者定義資料表類型，需使用 CREATE TYPE 陳述式。為了確保在使用者定義資料表類型中的資料符合特定的需求，可在使用者定義資料表類型中建立「唯一」或「主鍵」條件約束。

首先，透過 T-SQL 語法定義自訂資料表類型：

```
CREATE TYPE tpTbl AS TABLE
(ID INT IDENTITY PRIMARY KEY, C1 INT NOT NULL UNIQUE CHECK(C1>10),C2 AS
C1*10)
```

在 SQL Server 2008 Management Studio 的「物件總管」視窗內，可以觀察到使用者自訂的資料表類型，如圖 4-28 所示：

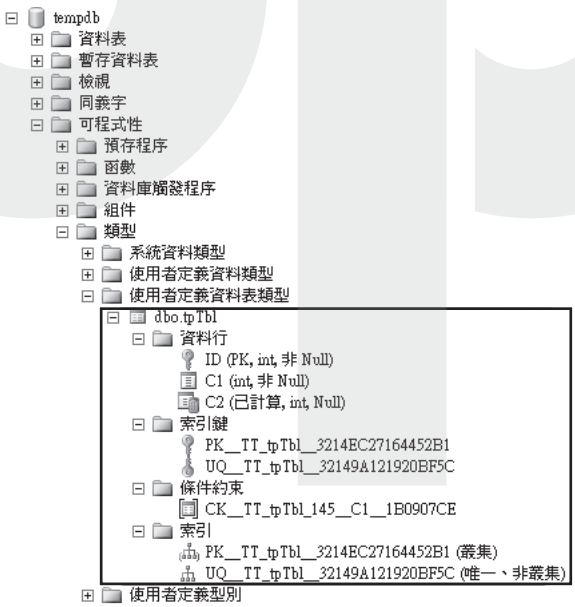


圖 4-28：建立使用者自訂資料表類型

在此，先用 T-SQL 簡單測試先前定義的自訂資料表類型：

```
DECLARE @t tpTbl
INSERT @t(C1) VALUES(11)
SELECT * FROM @t

DROP TYPE tpTbl
```

範例程式碼 4-32：在批次中存取使用者自訂資料表類型

使用者自訂資料表類型除了在批次中存取，使用方式如同宣告資料表變數外，也可以當作預存程序或函數的「資料表值參數」。「資料表值參數」是 SQL Server 2008 新增的參數類型。經由使用者定義的資料表類型所宣告。透過資料表值參數，可以將多筆資料列傳送到 T-SQL 陳述式、預存程序或函數。換句話說，在呼叫預存程序或函數時，若要傳遞多筆紀錄或陣列，可避免建立暫存資料表、符號分格字串或許多參數，且減少在網路上來回的批次數量。其範例語法如下：

```
--結構相同的資料表
SELECT * INTO tbTbl FROM @t WHERE 1=0

INSERT tbTbl(C1) VALUES(12)
GO
CREATE PROCEDURE spUpdateDataFromTVP
   @TVParam tpTbl READONLY
AS
   SET NOCOUNT ON
   UPDATE tbTbl SET C1=s.C1
   FROM tbTbl t JOIN @TVParam s
   ON t.ID=s.ID
GO

--呼叫使用資料表值參數的預存程序
DECLARE @t tpTbl
INSERT @t(C1) VALUES(11)
EXEC spUpdateDataFromTVP @t
SELECT * FROM tbTbl
```

範例程式碼 4-33：呼叫預存程序時，傳遞資料表值參數

然而，資料表值參數有下列限制：

- SQL Server 不會維護資料表值參數之資料行的統計資料。
- 資料表值參數必須是輸入 READONLY。換句話說，不能在函數或預存程序內，針對資料表值參數執行 DML 作業，例如，UPDATE、DELETE 或 INSERT⑭。
- 不能使用資料表值參數當作 SELECT INTO 或 INSERT EXEC 陳述式的目標。資料表值參數可以在 SELECT INTO 的 FROM 子句，或是 INSERT EXEC 字串或預存程序內。

4-2-7-10 其他注意事項

資料表若要更改既定的欄位類型，有些類型僅須更動中繼資料，但有些類型需要大量的 I/O，可以透過簡單的語法測試，如範例程式碼 4-34：

```
create table t(c1 int,c2 nvarchar(100))
go
declare @i int=0
while @i<1000
begin
   insert t values(@i,@i)
   set @i+=1
end
go

set statistics io on
--資料表't'。掃描計數1，邏輯讀取699，實體讀取0，讀取前讀取0，LOB 邏輯讀取0，LOB 實體讀取0，LOB 讀取前讀取0。
alter table t alter column c1 bigint

--僅回傳如下的訊息，沒有 I/O
--命令已順利完成。
alter table t alter column c2 nvarchar(200)
```

範例程式碼 4-34：更動既有資料表的資料行類型時，小心是否導致大量 I/O

⑭ 這一點有人在大聲疾呼，一定要讓其變成可讀寫，你可以參考如下的網址：https://connect.microsoft.com/SQLServer/feedback/ViewFeedback.aspx?FeedbackID=299296。

從執行範例程式碼 4-34 最後的兩個 ALTER 陳述式所傳回之訊息可以得知，若將 INT 類型的資料行改成 BIGINT 類型，因為各資料分頁要空出新的空間，因此需要大量的 I/O，但若是轉成 nvarchar(200)，則僅是更動中繼資料，而不需大量 I/O。

另外，當運算子結合的兩個運算式有不同的資料類型、定序、有效位數、小數位數或長度時，結果的性質取決於下列各點：

- 結果的資料類型：取決於輸入運算式的資料類型優先順序規則。
- 結果的定序：當結果資料類型是 char、varchar、text、nchar、nvarchar 或 ntext 時，取決於定序優先順序的規則。
- 結果的有效位數、小數位數和長度：隨著輸入運算式的有效位數、小數位數和長度而不同。

而這些定義可以參考線上圖書：

```
Database Engine→技術參考→Transact-SQL 參考→資料類型
```

最後，若想要一次查詢所有資料類型的屬性，可以透過 sys.types 系統檢視，範例語法如下：

```
select name,max_length,precision,scale,collation_name,is_nullable,is_
assembly_type from sys.types
```

執行結果如圖 4-29 所示：

	name	max_length	precision	scale	collation_name	is_nullable	is_assembly_type
1	image	16	0	0	NULL	1	0
2	text	16	0	0	Chinese_Taiwan_Stroke_CI_AS	1	0
3	uniqueidentifier	16	0	0	NULL	1	0
4	date	3	10	0	NULL	1	0
5	time	5	16	7	NULL	1	0
6	datetime2	8	27	7	NULL	1	0
7	datetimeoffset	10	34	7	NULL	1	0
8	tinyint	1	3	0	NULL	1	0
9	smallint	2	5	0	NULL	1	0
10	int	4	10	0	NULL	1	0
11	smalldatetime	4	16	0	NULL	1	0
12	real	4	24	0	NULL	1	0
13	money	8	19	4	NULL	1	0
14	datetime	8	23	3	NULL	1	0
15	float	8	53	0	NULL	1	0
16	sql_variant	8016	0	0	NULL	1	0
17	ntext	16	0	0	Chinese_Taiwan_Stroke_CI_AS	1	0
18	bit	1	1	0	NULL	1	0
19	decimal	17	38	38	NULL	1	0
20	numeric	17	38	38	NULL	1	0
21	smallmoney	4	10	4	NULL	1	0
22	bigint	8	19	0	NULL	1	0
23	hierarchyid	892	0	0	NULL	1	1
24	geometry	-1	0	0	NULL	1	1
25	geography	-1	0	0	NULL	1	1
26	varbinary	8000	0	0	NULL	1	0
27	varchar	8000	0	0	Chinese_Taiwan_Stroke_CI_AS	1	0
28	binary	8000	0	0	NULL	1	0
29	char	8000	0	0	Chinese_Taiwan_Stroke_CI_AS	1	0
30	timestamp	8	0	0	NULL	0	0
31	nvarchar	8000	0	0	Chinese_Taiwan_Stroke_CI_AS	1	0
32	nchar	8000	0	0	Chinese_Taiwan_Stroke_CI_AS	1	0
33	xml	-1	0	0	NULL	1	0
34	sysname	256	0	0	Chinese_Taiwan_Stroke_CI_AS	0	0

圖 4-29：SQL Server 2008 預設支援的資料類型

結語

本章介紹了 T-SQL 語法的基本組成元素，以及 SQL Server 2008 後所支援的各種基本資料類型，最後仍然要強調，熟悉基本資料類型的特徵，精確地使用資料類型，是善用 SQL Server 的基本。效率、穩定、擴充性都藏在這些細節裡。

學習評量

選擇題

1. 下列何者為 T-SQL 的註解？ (請選擇兩個答案)

( ) A. --
( ) B. /* */
( ) C. /##/
( ) D. ’

2. 下列何者為 SQL Server 中的「分隔識別碼」？ (請選擇兩個答案)

( ) A. []
( ) B. {}
( ) C. ()
( ) D. ” ”

3. 下列何者非邏輯運算子？

( ) A. BETWEEN
( ) B. EXISTS
( ) C. AND
( ) D. WHILE

4. 下列何者不是 T-SQL 用來處理字串的函數？

( ) A. SUBSTRING
( ) B. LEN
( ) C. LEFT
( ) D. CONVERT

5. 下列日期時間的資料型別，何者精確度最低？

( ) A. Datetime
( ) B. Datetime2
( ) C. Datetimeoffset
( ) D. Time

6. 下列哪個資料型別是在 SQL Server 2005 之後才新增的？

( ) A. varchar(max)
( ) B. text
( ) C. sysname
( ) D. datetime

7. 下列對 varchar 與 char 資料型別的比較，請選出最正確的答案。

( ) A. 兩者都可以儲存二進位資料
( ) B. varchar(10) 與 char(10) 都代表能儲存 10 bytes 的資料
( ) C. varchar 是固定資料型別，而 char 是可變資料型別
( ) D. varchar 與 char 之間要做資料型別轉換，需要使用 convert 函數

8. 下列何種資料型別，適合存放 true 與 false 的資料？

( ) A. int
( ) B. varchar
( ) C. bit
( ) D. tinyint

9. 以下對資料型別的描述，何者正確？

( ) A. 儘量選擇較大範圍的資料型別
( ) B. 儲存學生成績 (滿分 100 分)，可以選用 smallint 最適合
( ) C. 固定字串資料型別因為太浪費空間，使用機會不大
( ) D. 超過 8000 bytes 以上字串的資料型別，可以使用 varchar(max) 資料型別

10. 下列哪種資料型別，不一定只傳回單一筆資料結果？

( ) A. table
( ) B. int
( ) C. datetime
( ) D. varchar

問答題

1. 請問選擇資料類型時的考量為何？

2. 假設有一個系統需要記錄員工上班時間，上班時間超過 9 點鐘視為遲到，此欄位的資料類型應該如何設計較為適當？請說明理由。