Appendix

D

資料型別補充說明

在 4-5 節我們已經向各位介紹過 SQL Server 2016 中所有的系統內建資料型別, 但是因為有些資料型別的用法比較特殊, 所以我們另外再安排『**附錄 D**』為各位做補充說明。

本附錄將補充 timestamp (rowversion)、uniqueidentifier、sql_variant 以及 hierarchyid 這四種資料型別的用法, 另外還要說明 Text in Row 這個資料表選項。Text in Row 選項會影響 text、ntext、image 型別欄位儲存資料的地點。

請各位先將**練習 DD** 資料庫附加到您的 SQL Server 上, 以便操作。

D-1 timestamp (rowversion) 資料型別

每一個資料表中最多只能有一個欄位設為 timestamp 型別 (rowversion 和 timestamp 是同義字，二者都可以使用)，主要是用來儲存記錄的『版本編號』，當資料表中的一筆記錄新增或修改時，該筆記錄的 timestamp 欄位便會自動更新其版本編號。此版本編號是取自所屬資料庫的內部計數器，而該計數器的值會不斷累加，因此每一個版本編號在該資料庫中都是唯一的。

TIP timestamp 欄位的大小為 8 Bytes。

底下我們利用**練習 DD** 資料庫的**員工記錄**資料表來為您說明 timestamp 型別的特性。**員工記錄**資料表的結構如下：

```
CREATE TABLE 員工記錄
(
  記錄編號 int IDENTITY(1, 1) NOT NULL,
    異動日期 datetime,
    員工編號 int,
  薪資 smallmoney,
    備註 sql_variant,          ◄── 這是 D-3 節將介紹的 sql_variant 資料型別
  tmstmp timestamp,            ◄── tmstmp 欄位的資料型別為 timestamp
    guid uniqueidentifier      ◄── 這是 D-2 節將介紹的 uniqueidentifier 資料型別
)
```

Step1 首先，我們為**員工紀錄**資料表新增一筆記錄，其中 tmstmp 欄位不需設定欄位值：

```
INSERT 員工記錄 (異動日期, 員工編號, 薪資)
VALUES ( GETDATE(), 5, 30000 )

SELECT * FROM 員工記錄
```

	記錄編號	異動日期	員工編號	薪資	備註	tmstmp	guid
1	6	2016-12-18	5	30000.00	NULL	0x0000000000002EE1	NULL

自動產生 tmstmp 欄位的值

Step2 將剛才輸入的那筆記錄的內容修改一下：

```
UPDATE 員工記錄
SET 異動日期 = GETDATE()
WHERE 員工編號 = 5

SELECT * FROM 員工記錄
```

	記錄編號	異動日期	員工編號	薪資	備註	tmstmp	guid
1	6	2016-12-18	5	30000.00	NULL	0x0000000000002EE2	NULL

因為記錄被修改, tmstmp 欄位的值也自動改變了

由於每個資料庫中的每個 timestamp 欄位的值都是唯一的，亦即在同一個資料庫中 timestamp 欄位永遠不會有重複的值，所以我們也可以利用該欄位的值來識別記錄。但是要聲明一點，timestamp 欄位並不適合做索引，因為只要一修改記錄，timestamp 欄位的值就會改變。

不過也由於上述的特性，所以我們可以利用 timestamp 欄位的值來判斷記錄是否有被更動。例如當我們讀取一筆記錄後要做更改，可以在 UPDATE 敘述的 WHERE 子句中指定要以讀取當時的 timestamp 值為更新條件（例如『WHERE tmstmp = @前次讀取的 tmstmp 值』）；那麼如果該筆記錄在我們讀取後又被別人更改過，將會因 timestamp 值已不同而更新失敗，如此一來就可知道該記錄已被更改過，必須重新讀取一次再決定是否仍需更改。

D-2 uniqueidentifier 資料型別

uniqueidentifier 型別的欄位，通常是用來儲存作為**全域唯一識別碼**（GUID, Globally Unique Identifier）的資料，亦即該欄位的資料可用來在各資料庫之間識別出記錄的唯一性。其資料的格式是 16 bytes 的 16 進位值，如：'4C047CB3-B007-11D2-9C59-0080C846994D'。

通常，我們會在資料表中設定一個具有**識別**屬性的自動編號欄位，做為該資料表的識別欄位。但是若多個資料表中都以這種欄位來辨識記錄，那麼當要從這些資料表中選取出某筆記錄時，就可能會碰到兩邊數字相同的情況，此時這種欄位就不具有判別記錄唯一性的條件了。如果您有需要從整個資料庫 (或多個資料庫間) 判別記錄的唯一性，那就利用 uniqueidentifier 型別的欄位值吧！

uniqueidentifier 型別的欄位如何取得 GUID 的值呢？一個方法是比照 16 bytes 的 16 進位格式自行輸入，但是這種方法一來不方便，再則是我們無法確保其值是否唯一，所以很少用；一般我們會使用 NEWID() 函數來產生 GUID 值。怎麼使用 NEWID() 函數呢？底下同樣用**練習 DD** 資料庫內的**員工記錄**資料表來為您示範。

Step1 新增一筆記錄，暫不指定 guid 欄位的值：

```
INSERT 員工記錄 (異動日期, 員工編號, 薪資)
VALUES ( GETDATE(), 6, 32000 )

SELECT * FROM 員工記錄
```

	記錄編號	異動日期	員工編號	薪資	備註	tmstmp	guid
1	6	2016-12-18	5	30000.00	NULL	0x0000000000002EE2	NULL
2	7	2016-12-18	6	32000.00	NULL	0x0000000000002EE3	NULL

Step2 利用 NEWID() 函數為剛才那筆記錄的 guid 欄位產生 GUID 值：

```
UPDATE 員工記錄
SET guid = NEWID()          ← 使用 NEWID() 函數為 guid 欄位設定新值
WHERE 員工編號 = 6

SELECT * FROM 員工記錄
```

	記錄編號	異動日期	員工編號	薪資	備註	tmstmp	guid
1	6	2016-12-18	5	30000.00	NULL	0x0000000000002EE2	NULL
2	7	2016-12-18	6	32000.00	NULL	0x0000000000002EE4	887056BE-20B0-4EE8-9EEB-E550320CDBBF

讓 uniqueidentifier 欄位自動產生 GUID 值

其實有個方法可以讓 uniqueidentifier 欄位自動產生 GUID 值，那就是將 uniqueidentifier 欄位的**預設值**屬性設為 NEWID() 函數，如此一來，每當新增記錄時就會自動填入一個新的 GUID 值了。

最後，我們再強調一次 timestamp 型別與 uniqueidentifier 型別的用法。timestamp 型別在新增或修改記錄時會自動更新，可用來判斷記錄自從前次讀取後是否已被修改；而 uniqueidentifier 型別在我們以 NEWID() 設定 GUID 值後，就不會自己改變了，因此可以做為多資料表之間、多資料庫之間、甚至全世界中每一筆記錄的唯一識別值。

D-3 sql_variant 資料型別

sql_variant 資料型別最大的特色就是，可以讓 sql_variant 型別的欄位、參數或變數，得以儲存不同資料型別的資料值 (但是 text、ntext、image、timestamp、以及 sql_variant 除外)。以下仍然以**練習 DD** 資料庫內的**員工記錄**資料表來做個小小的實驗。

Step1 新增一筆記錄：

```
DECLARE @TT varchar(30)
SET @TT = ' 升官加薪'
INSERT 員工記錄 (異動日期, 員工編號, 薪資, 備註)
VALUES ( GETDATE(), 1, 52000, @TT)
```

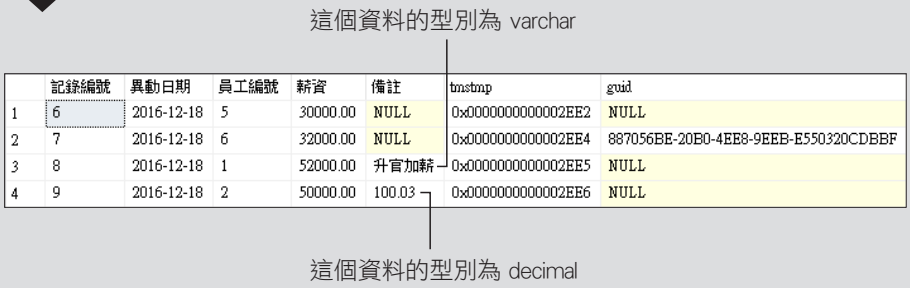
Step2 再新增一筆記錄：

```
DECLARE @DD decimal(5, 2)
SET @DD = 100.03
INSERT 員工記錄 (異動日期, 員工編號, 薪資, 備註)
```

接下頁

```
VALUES ( GETDATE(), 2, 50000, @DD)

SELECT * FROM 員工記錄
```



	記錄編號	異動日期	員工編號	薪資	備註	tmstmp	guid
1	6	2016-12-18	5	30000.00	NULL	0x0000000000002EE2	NULL
2	7	2016-12-18	6	32000.00	NULL	0x0000000000002EE4	887056BE-20B0-4EE8-9EEB-E550320CDBBF
3	8	2016-12-18	1	52000.00	升官加薪	0x0000000000002EE5	NULL
4	9	2016-12-18	2	50000.00	100.03	0x0000000000002EE6	NULL

每一個 sql_variant 型別的資料，都會記錄資料值以及 Meta Data (就是有關資料屬性的資訊，如基本資料型別、小數點位數、精確度 ... 等)，我們可以利用 SQL_VARIANT_PROPERTY 函數來取得這些資訊。SQL_VARIANT_PROPERTY 函數的語法如下：

```
SQL_VARIANT_PROPERTY(expression, property)
```

expression 是指 sql_variant 型別的欄位、變數或參數，至於 property 則是欲取得的屬性名稱，請看下表：

property	說明
BaseType	基本資料型別
Precision	精確度
Scale	小數點位數
TotalBytes	保存此 sql_variant 資料所需的位元數
MaxLength	資料的最大長度

```
SELECT SQL_VARIANT_PROPERTY(備註, 'BaseType' ) AS 基本資料型別,
       SQL_VARIANT_PROPERTY(備註, 'Precision' ) AS 精確度,
       SQL_VARIANT_PROPERTY(備註, 'Scale' ) AS 小數點位數,
       SQL_VARIANT_PROPERTY(備註, 'MaxLength' ) AS 最大長度
FROM 員工紀錄
```

	基本資料型別	精確度	小數點位數	最大長度
1	NULL	NULL	NULL	NULL
2	NULL	NULL	NULL	NULL
3	varchar	0	0	30
4	decimal	5	2	5

這 2 筆是我們剛才新增的

另外，若要將 sql_variant 型別的資料取出來運算，把握一個要點就是，先將 sql_variant 的資料轉換成您所需要的資料型別：

```
SELECT CAST(備註 AS int) + 30
FROM 員工記錄
WHERE 員工編號 = 2
```

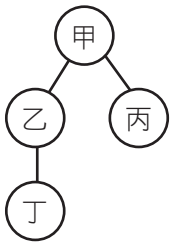
	(沒有資料行名稱)
1	130

有關資料型別的轉換可參考 10-5 節的說明，至於 CAST 函數的用法則可參閱 SQL Server 線上叢書。

D-4 hierarchyid 資料型別

hierarchyid 資料型別是 SQL Server 2008 新增加的型別，主要用來儲存『階層式』的資料，例如員工的主管階層、檔案系統的資料夾階層、網頁的連結階層等等。只要是具有上下層關係的階層資料，都適合用 hierarchyid 型別來儲存及應用。

hierarchyid 快速上手



hierarchyid 是可變長度的型別，其內存放的資料可用 toString() 轉換成『文字式』的階層關係，以方便閱讀，例如要表達右圖的階層：

```
DECLARE @甲 hierarchyid, @乙 hierarchyid, ◄── 宣告 4 個 hierarchyid 變數
        @丙 hierarchyid, @丁 hierarchyid

SET @甲= hierarchyid::GetRoot()           ◄── 存入根節點
SET @乙= @甲.GetDescendant(NULL, NULL)    ◄── 存入@甲的下一層節點
SET @丙= @甲.GetDescendant(@ 乙, NULL)    ◄── 存入@甲的下一層節點且在@乙的右邊
SET @丁= @乙.GetDescendant(NULL, NULL)    ◄── 存入@乙的下一層節點

PRINT @甲.ToString()
PRINT @乙.ToString()
PRINT @丙.ToString()
PRINT @丁.ToString()
```

```
/           ◄── 第 0 層的根節點 (甲)
/ 1 /       ◄── 第 1 層的左邊節點 (乙)
/ 2 /       ◄── 第 1 層的右邊節點 (丙)
/ 1 / 1 /   ◄── 第 1 層的左邊節點 (乙) 之下的第 2 層節點 (丁)
```

在上面程式中，hierarchyid::GetRoot() 可傳回根節點的值；而『A.GetDescendant(B, C)』則可傳回 A 節點之下的新節點，並依 B、C 的值而有左右之分：

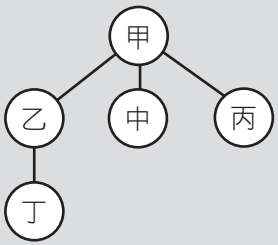
- ◉ 若 B、C 均為 NULL, 則傳回任意一個新的子節點。
- ◉ 若只有 C 為 NULL, 則傳回 B 右邊 (值比 B 大) 的新子節點。
- ◉ 若只有 B 為 NULL, 則傳回 C 左邊 (值比 C 小) 的新子節點。
- ◉ 若 B、C 都不是 NULL, 則傳回介於 B 和 C 之間的新子節點。

接著我們在前面程式的最後面，再加入以下程式看看：

```
DECLARE @甲 hierarchyid, . . .
:
:
DECLARE @中 hierarchyid = @甲.GetDescendant(@乙, @丙) ◄── 將中加在乙和丙之間
PRINT @中.ToString()
```

⬇

```
/
/ 1 /
/ 2 /
/ 1 / 1 /
/1.1/          ◄── 中(/1.1/) 介於乙(/1/) 和丙(/2/) 之間
```



『文字式』的階層字串，也可以轉換為 hierarchyid 型別的資料，例如 hierarchyid::Parse ('/1/1/') =0x5AC0。

hierarchyid 欄位的實作

有了 hierarchyid 型別，就不必再用**上層 ID** 之類的欄位來表達階層關係了，例如以前儲存主管階層的方法為：

以前都是使用此欄位來儲存員工的直屬主管，但在處理階層關係時並不方便！

	員工編號	姓名	性別	主管員工編號	職稱	區域
1	1	張瑾雲	女	0	經理	NULL
2	2	陳季暄	男	0	經理	NULL
3	3	趙飛燕	女	0	經理	NULL
4	4	李美麗	女	1	銷售員	北區
5	5	劉天王	男	3	銷售員	北區
6	6	黎國明	男	3	銷售員	中區
7	7	郭國璁	男	2	銷售員	南區
8	8	蘇涵蘊	女	1	銷售員	中區
9	9	孟庭亭	女	2	銷售員	北區
10	10	賴俊良	男	1	銷售員	南區
11	11	何大樓	男	3	銷售員	南區
12	12	王大德	男	2	銷售員	中區
13	13	楊大頭	男	NULL	NULL	NULL

底下為了避免複雜化，我們就以前面的甲、乙、丙...主管階層來示範，首先建立**員工階層**資料表，並加入一筆根節點 (甲) 的記錄：

```
CREATE TABLE 員工階層
(
    階層 hierarchyid,
    員工編號 int IDENTITY PRIMARY KEY,
    姓名 nvarchar(50)
)

INSERT 員工階層(姓名, 階層)
VALUES ('甲', hierarchyid :: GetRoot())      ←— 加入根節點
```

接下來為了方便加入各子節點的員工資料，我們先建立一個**加入子員工**預存程序：

```
CREATE PROC 加入子員工
(@姓名 nvarchar(8) ,
 @父名 nvarchar(8), @弟名 nvarchar=NULL, @兄名 nvarchar(8)=NULL) ←┐
AS                                                                 以參數指定要加入的階層位置,
BEGIN                                                              即：父.GetDescendant(弟, 兄)
    DECLARE @父節點 hierarchyid,
            @弟節點 hierar chyid, @兄節點 hierarchyid

    SELECT @父節點= 階層 FROM 員工階層 WHERE 姓名= @父名
    SELECT @弟節點= 階層 FROM 員工階層 WHERE 姓名= @弟名
    SELECT @兄節點= 階層 FROM 員工階層 WHERE 姓名= @兄名

    INSERT 員工階層(姓名, 階層)
    VALUES (@姓名, @父節點.GetDescendant(@弟節點, @兄節點))
END
```

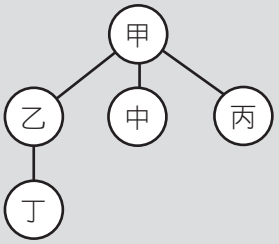
有了這個預存程序，我們就可以很方便地加入根節點 (甲) 之下的員工了：

```
EXEC 加入子員工 '乙', '甲'
EXEC 加入子員工 '丙', '甲', '乙'
EXEC 加入子員工 '丁', '乙'
EXEC 加入子員工 '中', '甲', '乙', '丙'
```

接下頁

```
SELECT 階層.ToString() 階層串, 階層.GetLevel() 層級, 姓名, 階層
FROM 員工階層
ORDER BY 階層
```

	階層串	層級	姓名	階層
1	/	0	甲	0x
2	/1/	1	乙	0x58
3	/1/1/	2	丁	0x5AC0
4	/1.1/	1	中	0x62C0
5	/2/	1	丙	0x68



可以用 GetLevel() 算出在第幾層

這是階層欄位的 hierarchyid 二進位內容

在查詢時如果以 hierarchyid 欄位來排序，則會優先顯示同一分支內的子節點 (深度優先)，例如在前面的結果中，丁 (第 2 層的節點) 會在乙後面。如果想要先顯示完同一層的節點，然後才顯示下一層的節點 (廣度優先)，則可改變排序方式：

```
SELECT 階層.ToString() 階層串, 階層.GetLevel() 層級, 姓名, 階層
FROM 員工階層
ORDER BY 階層.GetLevel(), 階層   ◄── 先依層級排, 層級相同時則由左到右排
```

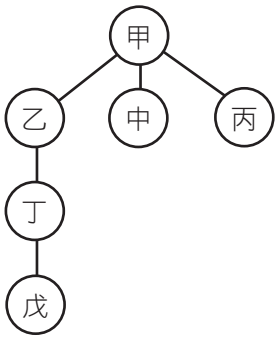
	階層串	層級	姓名	階層
1	/	0	甲	0x
2	/1/	1	乙	0x58
3	/1.1/	1	中	0x62C0
4	/2/	1	丙	0x68
5	/1/1/	2	丁	0x5AC0

第 2 層的丁跑到最後面了！

最後我們再來示範幾個有用的技巧：

1. 為了方便說明, 請在丁之下再加入一個**戊**節點：

```
EXEC 加入子員工 '戊', '丁'
```



2. 找出第 1 階層的所有主管：

```
SELECT 階層.ToString(), 姓名
FROM 員工階層
WHERE 階層.GetLevel() = 1
```

	(沒有資料行名稱)	姓名
1	/1/	乙
2	/2/	丙
3	/1.1/	中

3. 找出乙之下的所有部屬：

```
DECLARE @Str varchar(20)
SELECT @Str = 階層.ToString() FROM 員工階層 WHERE 姓名 = '乙'

SELECT 階層.ToString(), 姓名
FROM 員工階層
WHERE 階層.ToString() LIKE @Str + '%'
      AND @Str <> 階層.ToString()
```

	(沒有資料行名稱)	姓名
1	/1/1/	丁
2	/1/1/1/	戊

4. 找出**戊**之上的所有主管：

```
DECLARE @Str varchar(20)
SELECT @Str = 階層.ToString() FROM 員工階層 WHERE 姓名 = '戊'

SELECT 階層.ToString(), 姓名
FROM 員工階層
WHERE @Str LIKE (階層.ToString() + '%')
               AND @Str <> 階層.ToString()
```

	(沒有資料行名稱)	姓名
1	/	甲
2	/1/	乙
3	/1/1/	丁

5. 找出**戊**往上 2 層的主管：

程式中使用 A.GetAncestor(n) 找出由 A 往上 n 層的父節點。

```
DECLARE @node hierarchyid
SELECT @node = 階層 FROM 員工階層 WHERE 姓名 ='戊'

SELECT 階層.ToString(), 姓名
FROM 員工階層
WHERE 階層 = @node.GetAncestor(2)
```

	(沒有資料行名稱)	姓名
1	/1/	乙

6. 將**乙**及其部屬全部移到**丙**之下：

```
DECLARE @舊父 hierarchyid, @新父 hierarchyid,
        @本尊 hierarchyid

SELECT @舊父 = 階層 FROM 員工階層 WHERE 姓名='甲'
SELECT @新父 = 階層 FROM 員工階層 WHERE 姓名='丙'
SELECT @本尊 = 階層 FROM 員工階層 WHERE 姓名='乙'

UPDATE 員工階層
SET 階層 =
    階層.GetReparentedValue(@ 舊父, @新父)
WHERE 階層.IsDescendantOf(@本尊) = 1
```

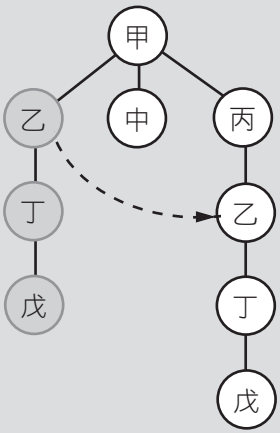
將乙及其部屬的階層全部更改到丙之下

```
SELECT 階層.ToString(), 姓名
FROM 員工階層
ORDER BY 階層
```

← 顯示結果

	(沒有資料行名稱)	姓名
1	/	甲
2	/1.1/	中
3	/2/	丙
4	/2/1/	乙
5	/2/1/1/	丁
6	/2/1/1/1/	戊

乙及其部屬全部移到丙之下了



D-5 Text in Row

Text in Row 選項是專為 text、ntext、以及 image 型別所設計的。text、ntext、和 image 這 3 種資料型別所能容納的資料長度都相當大 (最多可到 2GB), 為避免拖累資料表的存取效率, 以往 SQL Server 並不將這些型別的資料和資料表的其它資料存在一起, 而是另外儲存於個別的分頁中。這樣的做法使得存取 text、ntext、image 資料的效率, 要比存取資料表其它資料的效率來得差！

分頁 (page) 為 SQL Server 中資料儲存的基本單位, 每個分頁的大小為 8 KB。

有鑑於此, 自 SQL Server 2000 起新增了 text in row 選項, 這個選項可以讓中小型的 text、ntext、及 image 資料直接儲存於資料列中, 以提升存取效率。

SQL Server 線上叢書中說明, text、ntext、image 這 3 種資料型態, 及與之配合的 test in row 選項, 在未來的 SQL Server 版本中可能會取消, 故建議盡量改用 nvarchar(max)、varchar(max) 及 varbinary(max) 等資料型態替代。

我們可以利用 sp_tableoption 預存程序來開/關資料表的 Text in Row 選項, 或指定將 text、ntext、image 資料直接儲存在資料列的最大限制, 其語法如下：

```
sp_tableoption'table', 'option_name', 'value'
```

- **table：**欲設定的資料表名稱。
- **option_name：**指定資料表選項名稱。sp_tableoption 可設定的資料表選項不止一個, 不過在此我們僅介紹 text in row 選項, 所以此參數設為'text in row' 就對了。
- **value：**設定資料表選項的值。text in row 選項可設定的值請看下表：

value	說明
O N	啟用 text in row 選項, 此時預設的最大限制為 256 bytes
OFF 或 0	停用 text in row 選項, 此為資料表 text in row 選項的預設值
24 ~ 7000	啟用 text in row 選項, 並指定允許的最大限制, 單位是 byte

例如, 我們想啟用**練習 DD** 資料庫**書籍**資料表的 text in row 選項, 並將最大限制設為 200 bytes：

```
EXEC sp_tableoption'書籍', 'text in row', '200'
```

那麼以後只要**產品簡介**資料表中的 text (或 ntext、image) 資料符合下面的條件, 就會直接儲存在資料列中：

- 資料長度比 text in row 選項指定的最大限制還短。
- 資料列中有足夠的空間來容納這筆資料。

假如 text 資料不符合上述的條件呢？那麼該欄位實際上會存放一個 16 bytes 的文字指標, 這個指標將指引我們找到實際儲存 text 資料的地方。

查看資料表是否啟用 text in row 選項

我們可以利用 OBJECTPROPERTY 函數, 來查看資料表是否有啟用 text in row 選項, 例如：

```
SELECT OBJECTPROPERTY(OBJECT_ID('書籍'), 'TableTextInRowLimit')
```

	(沒有資料行名稱)
1	200

若 OBJECTPROPERTY 函數傳回 0, 表示停用 text in row 選項；若傳回其它數值, 則表示啟用 text in row 選項, 該數值即為 text in row 允許的最大限制。