

CHAP 13 T-SQL 程式設計

- 13-1 批次執行
- 13-2 使用註解 (Comment)
- 13-3 區域變數與全域變數
- 13-4 table 型別的變數
- 13-5 條件判斷與流程控制
- 13-6 特殊的程式控制
- 13-7 錯誤處理
- *13-8 偵錯：找出程式錯誤的地方

CHAP 13 T-SQL 程式設計

- **13-9** 使用 **CTE** 進行遞迴查詢
- **13-10** 使用 **MERGE** 來合併資料
- **13-11 SQL Script**
- **13-12** 自動產生 **SQL Script**
- **13-13** 使用不同資料庫或不同 **Server** 中的物件

13-1 批次執行

```
INSERT INTO 員工 (姓名, 性別)
VALUES ('楊大頭' , '男')
SELECT 員工編號, 姓名, 性別
FROM 員工
```



(影響 1 個資料列)

← 顯示 INSERT 的結果

員工編號	姓名	性別
-----	-----	----
1	張瑾雯	女
2	陳季暄	男
3	趙飛燕	女
...		
12	王大德	男
13	楊大頭	男

← 顯示 SELECT 的結果

批次執行

- 用 **GO** 分隔不同的批次

```
USE 練習 02      ← 第 1 個批次
GO
SELECT *
FROM 客戶
SELECT *
FROM 訂單
GO
```

第 2 個批次

放在一起的批次，其中一個指令錯誤，即會停止。**CREATE VIEW、CREATE DEFAULT、CREATE RULE、CREATE PROCEDURE & CREATE TRIGGER** 只能單獨放在一個批次中執行。

批次執行

- 批次錯誤時系統的處理方式
 - 編譯錯誤時：不會有任何敘述被執行
 - 執行中發生較大的錯誤時：之前的敘述不會影響，之後不會被執行。
 - 執行中發生輕微的錯誤時：只取消該錯誤的批次，其他不受影響。

13-2 使用註解 (COMMENT)

```
/ * 這是註解 * /
```

```
- - 這也是註解
```

```
SELECT *                /* 註解可以在這兒 */
```

```
FROM 員工              - - 當然也可以在這裡
```

```
SELECT 客戶編號, 聯絡人 /* , 地址 AS 送貨地址, 電話 */
```

```
FROM 客戶
```

```
WHERE 客戶編號 > 5 /* AND 聯絡人 LIKE "江*"
```

```
ORDER BY 電話 */
```

```
/ *
```

```
UPDATE 訂單細目
```

```
SET 數量 = 15
```

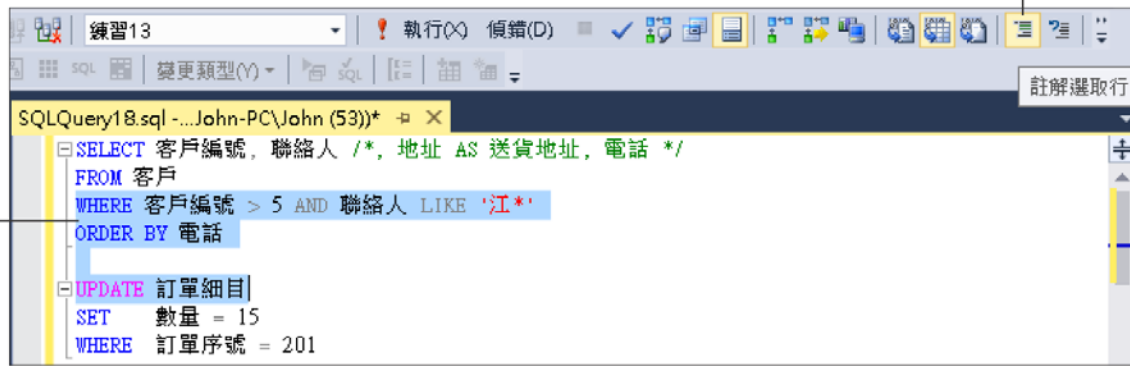
```
WHERE 訂單序號 = 201
```

```
* /
```


使用註解 (COMMENT)

1 選取要設為註解的多行程式

2 按註解選取行鈕

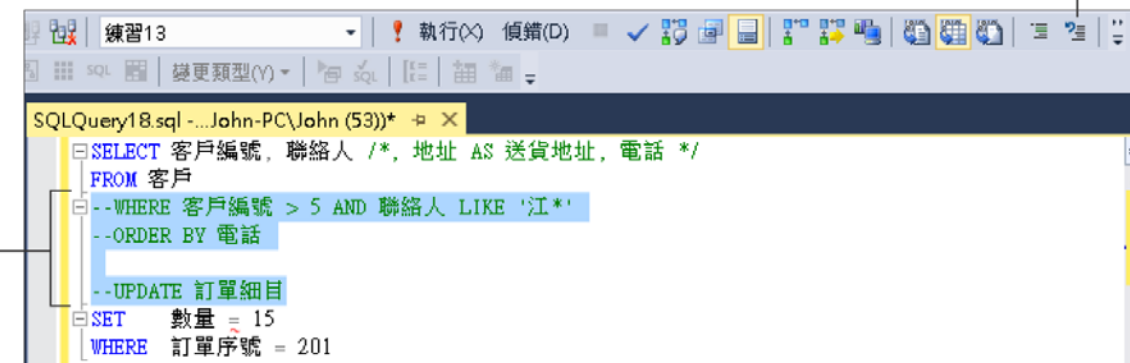


```
SQLQuery18.sql - ...John-PC\John (53)*
SELECT 客戶編號, 聯絡人 /*, 地址 AS 送貨地址, 電話 */
FROM 客戶
WHERE 客戶編號 > 5 AND 聯絡人 LIKE '江*'
ORDER BY 電話

UPDATE 訂單細目
SET 數量 = 15
WHERE 訂單序號 = 201
```

3 所有選取行都加上-- 註解了

4 按取消註解選取行鈕,
可取消選取行的 -- 註解



```
SQLQuery18.sql - ...John-PC\John (53)*
--SELECT 客戶編號, 聯絡人 /*, 地址 AS 送貨地址, 電話 */
--FROM 客戶
--WHERE 客戶編號 > 5 AND 聯絡人 LIKE '江*'
--ORDER BY 電話

--UPDATE 訂單細目
SET 數量 = 15
WHERE 訂單序號 = 201
```


13-3 區域變數與全域變數

```
DECLARE @variable_name [AS] data_type  
SET @variable_name = value
```

變數名稱前要加上@符號

```
DECLARE @customer AS varchar(30)  
DECLARE @counter int, @today datetime
```

宣告變數與資料型別。DECLARE 之後如果要定義多個變數, 那麼必須以逗號分開

```
SET @customer = '天天書局'  
SET @counter = 1  
SET @today = getdate()
```

設定變數的值

```
SELECT @customer  
SELECT @counter  
SELECT @today
```

顯示變數的內容



天天書局

1

2016-09-13 11:55:24.137

區域變數與全域變數

```
DECLARE @customer varchar(30)
```

```
SELECT @customer = 客戶名稱
```

← 將查詢結果指定給變數

```
FROM 客戶
```

```
WHERE 客戶編號 = 4
```

```
SELECT @customer
```



英雄書店

區域變數與全域變數

```
DECLARE @name char(10), @sex char (10) ← 定義 2 個變數
SELECT @name = 姓名, @sex = 性別 ← 一次設定多個變數
FROM 員工
WHERE 員工編號 = 3
```

```
SELECT @name AS '名字', @sex
```



```
名字      沒有資料行名稱      ← 性別欄的標題未指定, 所以標題
-----  -----      的部分會顯示沒有名稱的訊息
趙飛燕  女
```

```
DECLARE @customer varchar(30)
SELECT @customer = '大雄書局'
GO

SELECT @customer
```

分兩批次來執行, 則宣告的變數 @customer 只有在上面認得, 下面的就不認得了, 這就是區域變數的特性; 若將敘述中的 GO 刪除, 就會成為一個批次, 執行時就不會出現錯誤訊息了

13-4 TABLE 型別的變數

```
DECLARE @local_variable TABLE ( <table_definition> )
```

```
-- 宣告 table 變數
```

```
DECLARE @mybook TABLE ( 書籍編號 int PRIMARY KEY, 書籍名稱 varchar(50) )
```

```
INSERT @mybook ← 將 SELECT 的結果存入 table 變數中
```

```
SELECT 書籍編號, 書籍名稱
```

```
FROM 書籍
```

```
WHERE 單價 >= 460
```

```
SELECT * FROM @mybook
```

	書籍編號	書籍名稱
1	7	PhotoShop 細說從頭
2	15	計算機概論

```
UPDATE @mybook ← 修改 table 變數的內容
```

```
SET 書籍名稱 += '(附 CD)'
```


TABLE 型別的變數

```
DELETE @mybook      ← 刪除 table 變數中的記錄  
WHERE 書籍編號 = 7
```

```
- - 必須使用資料表別名來指示欄位  
SELECT m.書籍編號, m.書籍名稱, 單價  
FROM @mybook m JOIN 書籍  
    ON m.書籍編號 = 書籍.書籍編號  
GO
```



	書籍編號	書籍名稱	單價
1	15	計算機概論(附CD)	480.00

加入了 "(附CD)" 字樣

```
SELECT 書籍編號, 書籍名稱  
INTO @mybook      ← 語法錯誤！不可用在 INTO 中  
FROM 書籍
```


13-5 條件判斷與流程控制

- **BEGIN...END**
- **IF...ELSE** 條件判斷
- **WHILE** 迴圈
- **CASE** 函數
- **GOTO** 跳躍控制

BEGIN...END

- 是用來表示一個區塊，凡是在 BEGIN...END 之間的程式碼都是屬於同一個流程控制。

```
BEGIN
    expression_1
    expression_2
    ...
END
```

```
IF @id > 5
    BEGIN
        SET @count = 20
        SELECT @name = 姓名
        FROM 員工
        WHERE 編號 = @id
        PRINT @name
    END
```

如果 IF 條件成立，
要執行這些敘述

IF...ELSE 條件判斷

- 是“如果是...則..., 否則就...”。

```
IF boolean_expression ←— 如果 boolean_expression 為真 (true), 便會執行此敘述
    then_statement
[ELSE
    else_statement] ←— 如果 boolean_expression 為假 (false), 則會執行這個敘述
```

```
IF boolean_expression
    BEGIN
        then_statement1
        then_statement2
        ...
    END
[ELSE
    BEGIN
        else_statement1
        else_statement2
        ...
    END
]
```


IF...ELSE 條件判斷

```
IF boolean_expression  
    statement
```

```
statementB
```

```
IF (SELECT SUM(價格) FROM 標標公司) > 1100  
    PRINT '標標公司產品總價大於 1100 元'  
ELSE  
    PRINT '標標公司產品總價小於 1100 元'
```



標標公司產品總價大於 1100 元

IF...ELSE 條件判斷

```
DECLARE @avg_price int
SET @avg_price = (SELECT AVG(單價) FROM 書籍)

IF @avg_price > 600
    PRINT '書籍平均價格太高'
ELSE
    IF @avg_price > 400    ← 在 ELSE 中的 IF...ELSE
        PRINT '書籍平均價格適中'
    ELSE
        PRINT '書籍平均價格太低'
```


善用 **BEGIN...END** 避免錯誤 並增加可讀性

```
IF @avg_price > 600
    PRINT '書籍平均價格太高'
ELSE
    BEGIN
        IF @avg_price > 400
            PRINT '書籍平均價格適中'
        ELSE
            PRINT '書籍平均價格太低'
    END
END
```

在 ELSE 中的 IF...ELSE

IF...ELSE 條件判斷

- 邏輯運算式的應用

```
IF 'Windows 使用手冊' IN(SELECT 書籍名稱 FROM 書籍) ← 尋找是否有 Windows  
    PRINT '有 Windows 使用手冊' ← 使用手冊這本書  
ELSE  
    PRINT '無 Windows 使用手冊'
```

↓

有 Windows 使用手冊

```
IF 1000 > ALL (SELECT 單價 FROM 書籍)  
    PRINT '沒有任何書籍超過 1000 元'
```

↓

沒有任何書籍超過 1000 元

IF...ELSE 條件判斷

```
IF (SELECT 書籍名稱 FROM 書籍 WHERE 書籍編號 = '1001') IS NULL  
    PRINT '1001 的編號未輸入'  
ELSE  
    PRINT '1001 的編號已輸入'
```

```
IF EXISTS (SELECT 書籍名稱 FROM 書籍 WHERE 書籍編號 = '1001')  
    PRINT '1001 的編號已輸入'  
ELSE  
    PRINT '1001 的編號未輸入'
```


WHILE 迴圈

- 當判斷條件是 true 的時候，則進入迴圈執行。

```
WHILE boolean_expression
  BEGIN
    statement_1
    statement_2
    ...
    [BREAK]

    statement_a
    statement_b
    ...
    [CONTINUE]
  END
```


WHILE 迴圈

```
DECLARE @id int, @name varchar (50), @price int, @count int
SET @id = 0
SET @count =1
```

```
WHILE @id < 500
```

```
  BEGIN
```

```
    SET @id = @id + 1
```

```
    SELECT @name = 書籍名稱, @price = 單價
      FROM 書籍 WHERE 書籍編號 = @id
```

```
    IF @@ROWCOUNT = 0 /* @@ ROWCOUNT 中會儲存著 */
```

```
      BEGIN /* SELECT 傳回的記錄筆數 */
```

```
        PRINT '*** The End ***'
```

```
        BREAK
```

```
      END
```

```
    IF @price >= 400 CONTINUE
```

```
    PRINT CAST(@price AS CHAR(4)) + ' -- ' + @name
```

```
    IF @count % 3 = 0 PRINT '.....'
```

```
    SET @count = @count + 1
```

```
  END
```

遇到 CONTINUE 即
跳到 WHILE 開頭

遇到 BREAK
即跳出迴圈

WHILE 迴圈



350 -- AutoCAD 操作入門
320 -- Internet 上線實務
320 -- Internet 精緻之旅
.....
350 -- Visual Basic 學習手冊
350 -- WWW 實用寶典
350 -- 遊戲程式設計
.....
320 -- 電腦低階應用實務
*** The End ***

CASE 函數

```
IF @a = 1 PRINT "IS A"    ← @a 的值有多種情況要判斷
ELSE IF @a = 2 PRINT "IS B"
    ELSE IF @a = 3 PRINT "IS C"
        ELSE IF @a = 4 PRINT "IS D"
            ELSE PRINT "IS OTHERS"
```

```
IF @a > 500 PRINT ">500" ← 以 500、300 將 @a 的值分為 3 個範圍
    ELSE IF @a >300 PRINT ">300"
        ELSE PRINT "<=300"
```


CASE 函數

- 單一值的比對

```
CASE input_expr
  WHEN when_expr THEN result_expr
  [ ...n ]
  [ ELSE else_result_expr ]
END
```


CASE 函數

```
DECLARE @a INT, @answer CHAR(10)
SET @a = 3

SET @answer = CASE @a
                WHEN 1 THEN 'A'
                WHEN 2 THEN 'B'
                WHEN 3 THEN 'C'
                WHEN 4 THEN 'D'
                ELSE 'OTHERS'
            END
PRINT 'IS' + @answer
```



IS C

CASE 函數

```
SELECT '<' +
```

```
    CASE RIGHT(書籍名稱, 2) ← 取欄位最右邊 2 個字做比對
```

```
        WHEN '手冊' THEN '1 入門'
```

```
        WHEN '實務' THEN '2 實例'
```

```
        WHEN '應用' THEN '3 技巧'
```

```
        WHEN '秘笈' THEN '4 技術'
```

```
        ELSE '5 未分'
```

```
    END + '類>' AS 類別
```

```
    , 書籍名稱
```

```
FROM 書籍
```

```
ORDER BY 類別
```



CASE 函數

類別	書籍名稱
<1 入門類>	Linux 使用手冊
<1 入門類>	Excel 使用手冊
<1 入門類>	PowerPoint 使用手冊
<1 入門類>	Visual Basic 學習手冊
<1 入門類>	Windows 使用手冊
<1 入門類>	Word 使用手冊
<1 入門類>	Flash 學習手冊
<2 實例類>	電腦低階應用實務
<2 實例類>	Internet 上線實務
<5 未分類>	Internet 精緻之旅
<5 未分類>	WWW 實用寶典
<5 未分類>	遊戲程式設計
<5 未分類>	計算機概論
<5 未分類>	PhotoShop 細說從頭
<5 未分類>	AutoCAD 操作入門

CASE 函數

- 多種條件的判斷

```
CASE
  WHEN Boolean_expr THEN result_expr
  [ ...n ]
  [ ELSE else_result_expression ]
END
```

```
SET @answer =
CASE
  WHEN @a > 700 THEN 'A'
  WHEN @a > 500 THEN 'B'
  WHEN @a > 300 THEN 'C'
  ELSE 'D'
END
PRINT @answer
```


GOTO 跳躍控制

```
label:  
...  
GOTO label
```

```
DECLARE @number smallint  
SET @number = 99
```

```
IF (@number % 3) = 0
```

```
    GOTO Three
```

```
ELSE GOTO NotThree
```

```
Three: ←
```

```
    PRINT '三的倍數'
```

```
    GOTO TheEnd
```

```
NotThree: ←
```

```
    PRINT '不是三的倍數'
```

```
TheEnd: ←
```



三的倍數

GOTO 跳躍控制



13-6 特殊的程式控制

- **WAITFOR** 時間延遲
- **RETURN** 傳回值
- **EXECUTE** 執行預存程序或 **SQL** 字串
- **NULL** 值的處理

WAITFOR 時間延遲

```
WAITFOR { DELAY | TIME } 'time'
```

DELAY：暫停一段時間。

TIME：等到某一時間。

```
DECLARE @count INT
SET @count = 0
WHILE @count < 5          /* 此迴圈最多做 5 次*/
BEGIN
    INSERT 員工記錄(異動日期, 員工編號, 薪資)
    VALUES ( '2012/10/6' , 15, 30000)
    IF @@error = 0 BREAK    /* 如果成功即跳出迴圈*/
    SET @count = @count + 1
    WAITFOR DELAY '00:00:05' /* 等待 5 秒*/
END
```

▼ (執行後會等 25 秒，然後顯示以下訊息共 5 次)

訊息 547, 層級 16, 狀態 0 , 行 5

INSERT 陳述式與 FOREIGN KEY 條件約束 "FK_員工記錄_員工" 衝突。衝突發在資料庫 "練習 13" , 資料表 "dbo.員工" , column' 員工編號' 。

陳述式已經結束。

WAITFOR 時間延遲

```
WAITFOR TIME '23:50'  
SELECT * INTO 訂單備份  
FROM 訂單
```

等到半夜 **23:50** 之後再將訂單的所有資料放入
訂單備份資料表中，避免影響資料庫的效率。

RETURN 傳回值

```
CREATE PROCEDURE CheckOrder AS /* 建立自訂的預存程序 */  
  
IF EXISTS (SELECT * FROM 訂單 WHERE 客戶編號 = 2)  
    RETURN 1                /* 如果查詢到訂單，則傳回 1 */  
ELSE  
    RETURN 2                /* 沒有訂單就傳回 2 */  
  
GO  
  
DECLARE @value int  
EXEC @value = CheckOrder    /* 執行自訂預存程序 */  
PRINT @value
```

關於自訂預存程序，在第 14 章會有詳細介紹



EXECUTE 執行預存程序或 SQL 字串

- 執行系統的或自訂的預存程序：

```
EXEC[UTE]                                將預存程序的傳回值儲存  
[ @return_var      =]                    ← 於變數 @return_var 中  
  
{ proc_name | @proc_name_var }          ← 預存程序的名稱或儲存  
                                         了預存程序名稱的變數  
  
[ value | @var ] [, ...n]                ← 預存程序的參數, 可以有多个
```

```
-- 執行系統預存程序, 並指定 3 個參數  
EXEC sp_dboption '練習 13', 'ARITHABORT', 'ON'  
  
-- 作用同上一個範例, 但使用變數來指定預存程序名稱  
DECLARE @pname varchar(30)  
SET @pname = 'sp_dboption'  
EXEC @pname '練習 13', 'ARITHABORT', 'ON'
```


EXECUTE 執行預存程序或 SQL 字串

- 執行儲存於字串中的 SQL 批次敘述：

```
EXEC[UTE] (string_expression)
```


EXECUTE 執行預存程序或 SQL 字串

```
DECLARE @tablename varchar(20)
WHILE 1 = 1 ← 1=1 永遠為 True
  BEGIN
    SELECT @tablename = 暫存資料表名稱 ← 從『暫存資料表清單』資料表中
    FROM 暫存資料表清單               取得『暫存資料表名稱』欄位內的
    WHERE 建立日期 < getdate( ) -7     值，並且指定給 @tablename

    IF @@ROWCOUNT > 0 ← @@ROWCOUNT 儲存著傳回的記錄筆數
      BEGIN
        EXEC ( 'DROP TABLE ' + @tablename )
        DELETE 暫存資料表清單
        WHERE 暫存資料表名稱 = @tablename
      END
    ELSE
      BREAK ← 找不到時即跳出迴圈
  END
END
```


NULL 值的處理

```
IF (SELECT SUM(數量) FROM 訂單細目 WHERE 書籍編號 = 123) < 100  
    PRINT '訂購數量未達標準'  
ELSE  
    PRINT '訂購數量高於標準'
```

若書籍編號 = 123 並不存在，SELECT 會傳回 NULL 值，則 IF 會執行"訂購數量高於標準"。

NULL 值的處理

- **COALESCE** (**expr1**, **expr2**, **expr3**, ...) : 由左至右計算 **expr1** , **expr2** , ... , 當遇到非 **NULL** 的值即將之傳回
- **NULLIF** (**expr1**, **expr2**) : 當 **expr1** 等於 **expr2** 時會傳回 **NULL** 值, 否則傳回 **expr1**
- **ISNULL** (**expr1**, **expr2**) : 當 **expr1** 為 **NULL** 時, 即傳回 **expr2** 的值, 否則傳回 **expr1** 。

```
IF ISNULL((SELECT SUM(數量) FROM 訂單細目
           WHERE 書籍編號 = 123), 0) < 100
    PRINT '訂購數量未達標準'
ELSE
    PRINT '訂購數量高於標準'
```


13-7 錯誤處理

- 使用 **RAISERROR** 引發錯誤訊息
- 使用 **@@ERROR** 判斷是否發生錯誤
- 使用 **TRY...CATCH** 進行錯誤處理
- 使用 **THROW** 引發錯誤

使用 **RAISERROR** 引發錯誤訊息

```
EXEC sp_addmessage 66666, 7, 'Monsters!', @lang = 'us_english' ;
```

↑
訊息編號, 使用者自訂訊息
其編號的必須大於 50000

↑
英文的訊息

↑
指定語言種類為英文

↑
嚴重層級 (1~25), 但 19~25
只有系統管理者才可以設定

GO

```
EXEC sp_addmessage 66666, 7, '有怪獸!有怪獸!', @lang = '繁體中文' ;
```

↑ ↑
嚴重層級和訊息編號
必須和英文版本一樣

↑
中文的訊息

↑
指定語言種類為繁體中文 (若省略此
行則會指定現在使用的語言, 結果是
一樣的, 但前面的 ',' 也必須刪掉)

建立自訂訊息，首先要建立英文版本，才能建立其他的語言版本。

使用 **RAISERROR** 引發錯誤訊息

```
GO
```

```
RAISERROR (66666, 7, 1)
```

```
有怪獸!有怪獸!
```

```
訊息 66666 , 層級 7 , 狀態 1
```

```
RAISERROR ( {msg_id | msg_string }, severity, state )
```


使用 **RAISERROR** 引發錯誤訊息

- {msg_id | msg_string }

第一個參數可以是錯誤訊息的編號或字串。

```
RAISERROR ('失敗為成功之母' , 9, 1)
```



失敗為成功之母

訊息 50000 , 層級 9 , 狀態 1

此時訊息編號固定為 50000

使用 **RAISERROR** 引發錯誤訊息

- **Severity**

代表錯誤的嚴重層級，可由 **0** 到 **25**。

```
RAISERROR ('發生嚴重錯誤！', 20, 1) WITH LOG
```

- **state**：代表錯誤的狀態，可由 **0** 到 **255**。其意義可由我們自訂，例如可用 **state** 來代表錯誤所在的行號。

RAISERROR VS PRINT VS SELECT

RAISERROR ('此訊息由 RAISERROR 產生', 0, 1)

PRINT '此訊息由 PRINT 產生'



訊息

此訊息由 RAISERROR 產生

此訊息由 PRINT 產生

RAISERROR 更多功能，嚴重訊息可記錄於資料庫中。

SELECT '此訊息由 SELECT 產生'



(沒有資料行名稱)	
1	此訊息由 SELECT 產生

結果窗格

結果 訊息

(1 個資料列受到影響)

訊息窗格

使用 @@ERROR 判斷是否發生錯誤

```
BEGIN TRANSACTION      ← 開始交易
INSERT 旗旗公司(產品名稱, 價格)
VALUES ('PHP 程式語言', 500)

UPDATE 訂單
SET 是否付款= 1
WHERE 訂單序號= 3
IF @@ERROR != 0 OR @@ROWCOUNT = 0  ← 如果發生 UPDATE 執行錯誤
    ROLLBACK TRANSACTION           ← 取消 (回復) 交易
ELSE
    COMMIT TRANSACTION             ← 否則確認 (完成) 交易
```

@@ERROR 當交易發生錯誤，系統會將其設為非 0 的值
@@ROWCOUNT 期內存放受到影響的紀錄筆數

使用 **TRY...CATCH** 進行錯誤處理

- **TRY...CATCH** 的語法

```
BEGIN TRY  
    sql_statement  
END TRY
```

```
BEGIN CATCH  
    sql_statement  
END CATCH
```


使用 **TRY...CATCH** 進行錯誤處理

- 取得錯誤訊息、嚴重層級等資訊的函數
 - **ERROR_MESSAGE()**：會傳回完整的錯誤訊息。
 - **ERROR_NUMBER()**：傳回訊息編號。
 - **ERROR_SEVERITY()**：傳回嚴重層級。
 - **ERROR_STATE()**：傳回錯誤狀態代碼。
 - **ERROR_PROCEDURE()**：傳回發生錯誤的預存程序或觸發程序的名稱。
 - **ERROR_LINE()**：傳回批次或程序內造成錯誤的行號。

使用 **TRY...CATCH** 進行錯誤處理

```
BEGIN TRY
    RAISERROR (66666, 19, 1) WITH LOG
    PRINT '沒有發生重大錯誤'
END TRY
BEGIN CATCH
    IF ERROR_SEVERITY() > 16
        PRINT '發生嚴重錯誤！請通知管理員，錯誤編號為：' +
            CAST(ERROR_NUMBER() AS CHAR)
    ELSE
        PRINT '發生錯誤！錯誤訊息為：' + ERROR_MESSAGE()
END CATCH
```

訊息編號 66666 是 13-27 頁新增的訊息

使用 RAISERROR 產生嚴重層級 19 的錯誤，讓程式跳到 BEGIN CATCH...END CATCH 區塊

判斷嚴重層級是否大於 16



發生嚴重錯誤！請通知管理員，錯誤編號為：66666

使用 **TRY...CATCH** 進行錯誤處理

- **TRY...CATCH** 無法處理的錯誤：例如語法錯誤，或者物件不存在

```
BEGIN TRY
    SELECT * FROM 資料表 ABC
END TRY
BEGIN CATCH
    PRINT 'TRY...CATCH 發現一個錯誤：' + ERROR_MESSAGE()
END CATCH
```



訊息 208 ， 層級 16 ， 狀態 1 ， 行 2
無效的物件名稱 '資料表 ABC' 。

使用 **TRY...CATCH** 進行錯誤處理

```
CREATE PROCEDURE testTRYCATCHProc ← 建立預存程序
AS
    SELECT * FROM 資料表 ABC
GO

BEGIN TRY
    EXECUTE testTRYCATCHProc ← 執行預存程序, 當預存程序發生錯誤
                                時, 會由 TRY...CATCH 進行處理
END TRY
BEGIN CATCH
    PRINT 'TRY...CATCH 發現一個錯誤: ' + ERROR_MESSAGE()
END CATCH
```

↓

TRY...CATCH 發現一個錯誤: 無效的物件名稱 '資料表 ABC'。

改為預存程序的方式即可以使用 **CATCH** 抓到錯誤

使用 **THROW** 引發錯誤

```
THROW [error_number , message, state]
```

- **error_number** : **int** 型別的錯誤編號。
- **message** : **nvarchar(2048)** 型別的錯誤訊息字串。
- **state** : **tinyint** 型別的錯誤狀態。

THROW 一般用於 **TRY** 區塊中引發錯誤，若用於 **TRY** 區塊以外的地方，則在引發錯誤後立即中止目前批次。若將 **THROW** 參數省略，只能用於 **CATCH** 區塊中。

RAISERROR 與 THROW 的差異

項目	RAISERROR	THROW
錯誤編號和錯誤訊息	只能擇一指定 (若指定錯誤訊息, 則錯誤編號固定為 50000)	必須同時指定
錯誤編號	必須已定義在 sys.messages 中(包含系統內建及使用者自訂的錯誤)	必須大於等於 50000, 與 sys.messages 無關
嚴重層級	必須指定	不能指定 (固定為 16)

使用 **THROW** 引發錯誤

```
THROW 51000, '這是由 THROW 產生的自訂錯誤.', 1
```



訊息 51000，層級 16，狀態 1，行 1
這是由 THROW 產生的自訂錯誤。

```
BEGIN TRY
```

```
    THROW 51000, '這是由 THROW 產生的自訂錯誤.', 1
```

```
END TRY
```

```
BEGIN CATCH
```

```
    PRINT '●進入 CATCH 區塊';
```

```
        THROW 重新引發TRY中發生的錯誤
```

```
    PRINT '●正常結束 CATCH 區塊'
```

```
END CATCH
```

```
PRINT '●批次結束'
```

這兩個 PRINT 不會被執行



●進入 CATCH 區塊

訊息 51000，層級 16，狀態 1，行 3
這是由 THROW 產生的自訂錯誤。

****13-8 偵錯：找出程式錯誤的地方 (在 SQL SERVER 2016 才有)**

- 可逐步執行程式碼
- 可設定中斷點
- 在逐步執行到預存程序、自訂函數、或觸發程序時，可決定是否要進入程序中逐步執行，或將整個程序當成一個敘述來執行

SSMS v18之後已經將偵錯的功能移除，若要使用偵錯的功能 (VS)，參照：
https://www.dotblogs.com.tw/jamesfu/2022/01/29/Debug_SQL_With_SSMS

偵錯：找出程式錯誤的地方

-- 建立可傳回訂單筆數的預存程序

```
CREATE PROCEDURE CountOrder
AS
    DECLARE @cnt INT
    SELECT @cnt = COUNT(*) FROM 訂單
    RETURN @cnt -- 傳回訂單筆數
```

```
DECLARE @value int, @msg varchar(30)
SET @msg = '訂單筆數為：'

EXEC @value = CountOrder      /* 執行預存程序 */
IF @value > 0
    SET @msg += CAST(@value AS varchar)
ELSE
    SET @msg += '沒有訂單'
PRINT @msg
```


偵錯：找出程式錯誤的地方

黃色箭頭指出下一

個要執行的敘述

偵錯工具列

此處標示目前正在『偵錯模式』中

The screenshot shows the SQL Server Enterprise Manager interface in debug mode. The main window displays a T-SQL script with the following code:

```
DECLARE @value int, @msg varchar(30)
SET @msg = '訂單筆數為：'

EXEC @value = CountOrder /* 執行預存程序 */
IF @value > 0
    SET @msg += CAST(@value AS varchar)
```

The 'Variables' window shows the current values of the variables:

名稱	值	類型
@value		int
@msg		varchar

The 'Call Stack' window shows the current execution context:

名稱	語言
SQLQuery21.sql(0) 行 2	Trans.

The status bar at the bottom indicates the current execution context: 第 2 行, 第 1 欄, 字元 1, INS.

區域變數窗格會顯示目前批次、程序、或函數中的區域變數資訊

目前區域變數的值均為 NULL, 必要時可以手動更改其內容

呼叫堆疊窗格會顯示目前敘述的程序 (或函式) 呼叫歷程, 目前在最上層的批次中

偵錯：找出程式錯誤的地方

- 逐步執行
- 使用中斷點
- 監看各種變數的值

逐步執行

執行完『SET @msg = '訂單筆數為：'』後，即暫停在下一個敘述

```
SQLQuery21.sql -...John (53)) 偵錯中... * X SQLQuery17.sql -...John-PC\John (54))
DECLARE @value int, @msg varchar(30)
SET @msg = '訂單筆數為：'
EXEC @value = CountOrder /* 執行預存程序 */
IF @value > 0
    SET @msg += CAST(@value AS varchar)
ELSE
    SET @msg += '沒有訂單'
PRINT @msg
```

名稱	值	類型
@va		int
@m: 訂單筆數為：		varchar

名稱	語言
SQLQuery21.sql() 行 4	Tran:

@msg 的值已經改變了

當值欄右側出現放大鏡時，表示按一下放大鏡可開啟另一個視窗來檢視其內容。另外還可拉下列示窗來選擇顯示的格式（文字、XML、或 HTML）

逐步執行

```
SET @msg += CAST(@value AS varchar)
```

@msg | 查詢 | 訂單筆數為 :

按圖釘可讓快速監看框

固定顯示在程式右側 :

```
IF @value > 0  
    SET @msg += CAST(@value AS varchar)  
ELSE  
    SET @msg += '沒有訂單'
```

@msg | 查詢 | 訂單筆數為 :

切換是否要隨著程式捲動

關閉此框

切換是否顯示註解

@msg | 查詢 | 訂單筆數為 :

在此輸入註解

逐步執行

逐步執行到預
存程序中了

會開啟預存程序專用的查詢視窗，
並以程序的名稱為頁籤的名稱

```
CREATE PROCEDURE CountOrder
AS
    DECLARE @cnt INT
    SELECT @cnt = COUNT(*) FROM 訂單
    RETURN @cnt -- 傳回訂單筆數
```

名稱	值	類型
@cnt		int

名稱	語言
CountOrder(John-PC.練習13)0 行 4	Tran:
SQLQuery21.sql0 行 4	Tran:

這是預存程序
內的區域變數

目前是在 CountOrder
預存程序中

這是上一層的批次（越上層
的呼叫程式會顯示在越下面）

逐步執行

果然立即執行到程序結束, 然後返回
上一層, 停在下一個要執行的敘述上

```
SQLQuery21.sql -...John (53)) 偵錯中... * X SQLQuery17.sql -...John-PC\John (54)
DECLARE @value int, @msg varchar(30)
SET @msg = '訂單筆數為:'

EXEC @value = CountOrder /* 執行預存程序 */

IF @value > 0
    SET @msg += CAST(@value AS varchar)
ELSE
```

正在偵錯查詢... | John-PC (13.0 RTM) | John-PC\John (53) | 練習13 | 00:19:19 | 0 個資料列

區域變數			呼叫堆疊	
名稱	值	類型	名稱	語言
@value	28	int	SQLQuery21.sql 行 5	Tran:
@msg	訂單筆數為:	varchar		

@value 的值變成 28 了

已回到最上層的批次

逐步執行


一直往下執行到中斷點位置, 或程式結束為止(**Alt** + **F5**)

停止偵錯 (**Shift** + **F5**)

暫停執行(正在執行時才可按)

讓查詢視窗顯示出黃色箭頭所指的程式碼(如果目前沒有顯示出來的話)

十六進位



The image shows a horizontal toolbar with several icons. From left to right: a green play button, a grey pause button, a red stop button, a grey right-pointing arrow, a blue downward-pointing arrow, a blue circular refresh icon, a blue upward-pointing arrow, a vertical separator, the text '十六進位' (Hexadecimal), another vertical separator, a magnifying glass icon, and a small grey square icon.

逐步執行

The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows a query window with the following T-SQL code:

```
DECLARE @value int, @msg varchar(30)
SET @msg = '訂單筆數為：'

EXEC @value = CountOrder /* 執行預存程序 */

IF @value > 0
    SET @msg += CAST(@value AS varchar)
ELSE
    SET @msg += '沒有訂單'
PRINT @msg
```

The bottom pane shows the '訊息' (Messages) window with the output:

```
訊息
-----
訂單筆數為：28

已成功執行查詢。 | John-PC (13.0 RTM) | John-PC\John (53) | 練習13 | 00:21:50 | 0 個資料列
```

The '輸出' (Output) window is also visible, showing the execution of the stored procedure with return codes.

程式輸出的結果

會自動顯示輸出窗格，其內容為偵錯的一些輸出訊息，可不用管它

使用中斷點

有紅圈圈的地方即為中斷點。在敘述左側的灰色區域上按鈕 (或按 **F9** 鍵), 即可設定或刪除中斷點

```
SQLQuery21.sql -...John-PC\John (53)*  X  SQLQuery17.sql -...John-PC\John (54))
DECLARE @value int, @msg varchar(30)
SET @msg = '訂單筆數為：'

EXEC @value = CountOrder      /* 執行預存程序 */
IF @value > 0
    SET @msg += CAST(@value AS varchar)
ELSE
    SET @msg += '沒有訂單'
PRINT @msg
```

100 %

訊息

已成功執行查詢。 | John-PC (13.0 RTM) | John-PC\John (53) | 練習13 | 00:21:50 | 0 個資料列

使用中斷點

CountOrder 的偵錯視窗 (頁籤為程序的名稱)

```
John-PC.[練習13].[dbo].[CountOrder]  X  
  
CREATE PROCEDURE CountOrder  
AS  
    DECLARE @cnt INT  
    SELECT @cnt = COUNT(*) FROM 訂單  
    RETURN @cnt  -- 傳回訂單筆數
```

100 %

當偵錯結束時，程序的偵錯視窗並不會關閉，而中斷點在後續的偵錯中也仍然有效

使用中斷點

在中斷點圖示上按右鈕, 可刪除或停用中斷點

The screenshot shows the SQL Server Enterprise Manager interface. The main window displays the code for a stored procedure named 'CountOrder'. A red dot indicates a breakpoint is set on the line 'SELECT @cnt = COINT(*) FROM 訂單'. A context menu is open over this breakpoint, with '停用中斷點(D) Ctrl+F9' selected. Below the code, the '中斷點' (Breakpoints) window is open, showing a list of breakpoints. The first breakpoint is selected and highlighted in bold. The table below shows the details of the breakpoints:

名稱	標籤	條件	叫用次數
<input checked="" type="checkbox"/> =1394104007, 第 4 行第 2 字元		(無條件)	(目前 1)永遠中斷
<input checked="" type="checkbox"/> ~vsF135.sql, 第 6 行第 2 字元		(無條件)	(目前 0)永遠中斷
<input checked="" type="checkbox"/> ~vsF135.sql, 第 8 行第 2 字元		(無條件)	(目前 0)永遠中斷

在中斷點上雙按, 即可在查詢視窗中顯示該中斷點的程式碼

取消勾選可停用
該中斷點的作用

中斷點頁次

粗體表示目前執行到此中斷點

使用中斷點

- 條件

John-PC.[練習13].[dbo].[CountOrder] 中斷點設定 X

```
CREATE PROCEDURE CountOrder
AS
  DECLARE @cnt INT
  SELECT @cnt = COUNT(*) FROM 訂單
```

位置: =1394104007, 行: 4, 字元: 2, 必須符合來源

條件

條件運算式 為 true @cnt > 3

新增條件

動作

關閉

選此項則當運算式為 true 時才會生效

輸入一個運算式, 例如 '@cnt > 3'

```
RETURN @cnt -- 傳回訂單筆數
```

100 %

使用中斷點

- 編輯標籤

1 可新增或選取現有的標籤名稱

編輯中斷點標籤

輸入新標籤(T):

預存程序中 加入(A)

或選擇現有標籤(R):

預存程序中

確定 取消

使用中斷點

- 編輯標籤

The screenshot shows the '中斷點' (Breakpoints) window in Visual Studio. It contains a table with the following columns: '名稱' (Name), '標籤' (Label), '條件' (Condition), and '叫用次數' (Hit Count). Three breakpoints are listed, with the third one selected and highlighted in blue. Annotations with lines pointing to the table cells are present: a '2' points to the '名稱' column, and another line points to the '條件' and '叫用次數' columns.

名稱	標籤	條件	叫用次數
<input checked="" type="checkbox"/> + =1394104007, 第 4 行第 2 字元	預存程序中	當 '@cnt > 3' 為 True	(目前 1)永遠中斷
<input checked="" type="checkbox"/> ~vsF135.sql, 第 6 行第 2 字元		(無條件)	(目前 0)永遠中斷
<input checked="" type="checkbox"/> ~vsF135.sql, 第 8 行第 2 字元		(無條件)	(目前 0)永遠中斷

2 顯示標籤名稱 顯示已設定的條件、叫用次數

監看各種變數的值

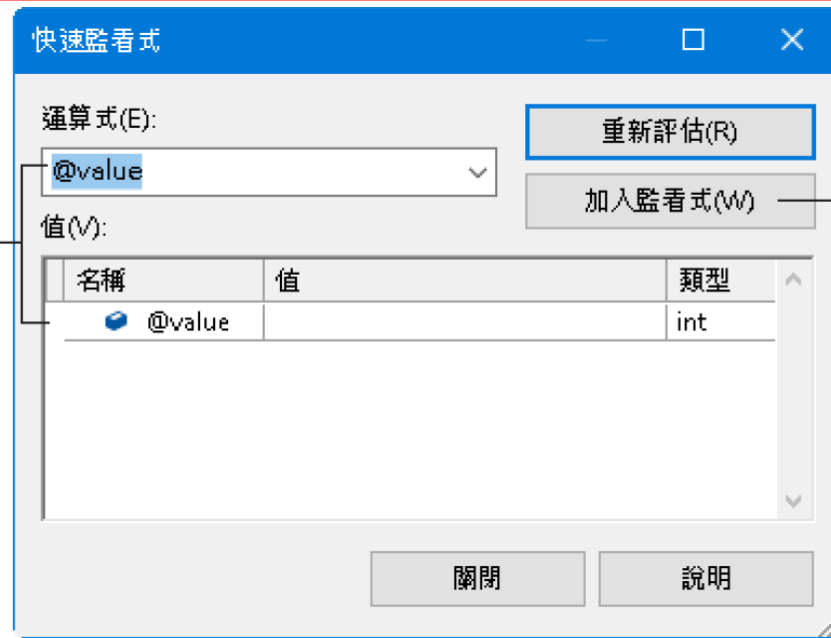


名稱	值	類型
@value		int

在**監看式**窗格中可直接輸入要監看的變數、參數、或運算式, 即可監看其值

監看各種變數的值

會自動填入程式中插入點所在或選取的字串，並顯示其值。也可手動修改，然後按**重新評估**鈕來顯示其值



按此鈕可將監看式加入前述的**監看式 1** (或 2、3、4) 窗格中

13-9 使用 CTE 進行遞迴查詢

- **CTE** 的作用和語法
- 使用 **CTE** 進行遞迴查詢的語法
- 使用 **CTE** 進行遞迴查詢的範例

CTE 可以看成是一個暫時的檢視表，其生命週期只存在於該批次的執行期間

CTE 的作用和語法

```
CREATE VIEW 下單記錄_VIEW ← 建立下單記錄_VIEW 檢視表
AS
SELECT 日期, 客戶名稱, 地址
FROM 訂單, 客戶
WHERE 訂單.客戶編號 = 客戶.客戶編號
GO
```

```
SELECT * FROM 下單記錄_VIEW ← 查詢下單記錄_VIEW 檢視表的所有內容
```



日期	客戶名稱	地址
-----	-----	-----
2016-12-27 00:00:00	愚人書店	台北市北平東路 24 號
2016-01-18 00:00:00	新新書店	台北市中山北路六段 88 號
2016-01-22 00:00:00	十全書店	台北市仁愛路二段 56 號
.....		

CTE 的作用和語法

```
WITH 下單記錄_CTE ← 建立名為下單記錄_CTE 的 CTE
AS (
  SELECT 日期, 客戶名稱, 地址
  FROM 訂單, 客戶
  WHERE 訂單.客戶編號 = 客戶.客戶編號
)
```

```
SELECT * FROM 下單記錄_CTE ← 查詢下單記錄_CTE 的所有內容
```



日期	客戶名稱	地址
-----	-----	-----
2016-12-27 00:00:00	愚人書店	台北市北平東路 24 號
2016-01-18 00:00:00	新新書店	台北市中山北路六段 88 號
2016-01-22 00:00:00	十全書店	台北市仁愛路二段 56 號
.....		

CTE 的作用和語法

```
WITH CTE_name [ ( column [ , ...n ] ) ]  
AS (  
    select_statement  
)
```


使用 **CTE** 進行遞迴查詢的語法

```
WITH CTE_name [ ( column [ , ...n ] ) ]  
AS (  
    select_statement1  
    UNION ALL  
    select_statement2  
)
```


使用 CTE 進行遞迴查詢的範例

	員工編號	姓名	性別	主管員工編號	職稱	區域
1	1	張瑾雯	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	郭國斌	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

表示該員工
沒有主管。

使用 **CTE** 進行遞迴查詢的範例

```
| 主管 1  
| - - 員工 A  
| - - 員工 B  
| 主管 2  
| - - 員工 C  
.....
```

階層是樹狀表，第 1 層為最高主管 (主管編號為 0)。

使用 CTE 進行遞迴查詢的範例

```
WITH 員工階層_CTE (員工編號, 姓名, 主管員工編號, level, sort)  
AS (
```

level 與 sort 欄位是我們自訂的新欄位, 將分別用來記錄層級與排序值。sort 欄中將儲存『主管姓名-員工姓名』的字串, 可同時做為排序及顯示之用

```
/* 錨點成員先找出第 1 層的主管 */
```

```
SELECT 員工編號,  
       姓名,  
       主管員工編號,
```

```
       1, 因為是第一層, 所以指定 level 欄位的值為 1
```

此為註解

```
       CONVERT(varchar (255), 姓名)
```

← 將查到的姓名轉為 varchar (255) 型別, 然後存入 sort 欄位

```
FROM 員工
```

```
WHERE 主管員工編號 = 0
```

```
UNION ALL
```


使用 CTE 進行遞迴查詢的範例

```
/* 遞迴成員接著以自我呼叫的方式，找出各主管的員工 */  
SELECT 員工.員工編號,  
       員工.姓名,           將原本的 level 欄位值加 1, 所以第一次  
       員工.主管員工編號, 遞迴查到的記錄, 其 level 欄位的值皆  
       level+1,           ← 為 2, 第二次遞迴的值則為 3, 依序增加       CONVERT (varchar(255), sort + '-' + 員工.姓名) ←
```

將原本的 sort 欄位值加上該次遞迴查到的員工姓名, 所以第一次遞迴查到的記錄, 其 sort 欄位將是 "主管名-員工名", 第二次遞迴則會是 "主管名-員工名-員工名", 其餘依此類推

```
FROM 員工  
JOIN 員工階層_CTE ON 員工.主管員工編號 = 員工階層_CTE.員工編號
```

進行自我查詢, 前面錨點成員找到主管的記錄後, 會存入員工階層_CTE。當第一次遞迴時, 會找出哪些員工的主管員工編號等於員工階層_CTE 內的員工編號, 再存入員工階層_CTE。之後, 第二次遞迴時再依照同樣方式尋找第 3 層的員工

)

使用 CTE 進行遞迴查詢的範例

```
SELECT '|' + REPLICATE('-', level*2) + 姓名 AS 員工層級,
```

依照 level 欄位值, 使用 REPLICATE 重複顯示 "-" 字元, level 欄位值越高, 表示位於樹狀圖越末端, 因此顯示的 "-" 也會越多

員工編號, 主管員工編號, level, sort

```
FROM 員工階層_CTE
```

```
ORDER BY sort ← 依照 sort 欄位進行排序
```



	員工層級	員工編號	主管員工編號	level	sort
1	張瑾雯	1	0	1	張瑾雯
2	--李美麗	4	1	2	張瑾雯-李美麗
3	--賴俊良	10	1	2	張瑾雯-賴俊良
4	--蘇涵蘊	8	1	2	張瑾雯-蘇涵蘊
5	陳季暄	2	0	1	陳季暄
6	--王大德	12	2	2	陳季暄-王大德
7	--孟庭亨	9	2	2	陳季暄-孟庭亨
8	--郭國斌	7	2	2	陳季暄-郭國斌
9	趙飛燕	3	0	1	趙飛燕
10	--何大樓	11	3	2	趙飛燕-何大樓
11	--劉天王	5	3	2	趙飛燕-劉天王
12	--黎國明	6	3	2	趙飛燕-黎國明

使用 CTE 進行遞迴查詢的範例

先將錨點成員產生的初始紀錄單獨顯示

```
SELECT 員工編號, 姓名, 主管員工編號, 1, CONVERT(varchar (255), 姓名)
FROM 員工
WHERE 主管員工編號 = 0
```



員工編號	姓名	主管員工編號	(level)	(sort)
1	張瑾雯	0	1	張瑾雯
2	陳季暄	0	1	陳季暄
3	趙飛燕	0	1	趙飛燕

使用 CTE 進行遞迴查詢的範例

後面的地回成員會根據這些資料，分別去找那些員工的主管員工編號是 1、2 或 3。找到之後，會將level的欄位的值加 1，表示這是第 2 層員工，sort 欄位則會再加上員工姓名，變成"張瑾雯-李美麗"。依序遞迴查詢所有階層，並經由 UNION ALL 合併結果，作為 CTE 的查詢結果。

請將前面的 CTE 範例程式改為查詢員工多層資料表，則會傳回多層的主管階層：

	員工編號	姓名	性別	主管員工編號	職稱	區域
1	1	張瑾雯	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	5	銷售員	中區
7	7	郭國誠	男	2	銷售員	南區
8	8	蘇涵慈	女	4	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	6	銷售員	南區
12	12	王大德	男	8	銷售員	中區

員工多層資料表



使用 CTE 進行遞迴查詢的範例

	員工層級	員工編號	主管員工編號	level	sort	
1	└張瑾雯	1	0	1	張瑾雯	
2	└─李美麗	4	1	2	張瑾雯-李美麗	
3	└──蘇涵蘊	8	4	3	張瑾雯-李美麗-蘇涵蘊	—— 第 3 層
4	└───王大德	12	8	4	張瑾雯-李美麗-蘇涵蘊-王大德	—— 第 4 層
5	└─賴俊良	10	1	2	張瑾雯-賴俊良	
6	└陳季暄	2	0	1	陳季暄	
7	└─孟庭亭	9	2	2	陳季暄-孟庭亭	
8	└─郭國斌	7	2	2	陳季暄-郭國斌	
9	└趙飛燕	3	0	1	趙飛燕	
10	└─劉天王	5	3	2	趙飛燕-劉天王	
11	└──黎國明	6	5	3	趙飛燕-劉天王-黎國明	
12	└───何大樓	11	6	4	趙飛燕-劉天王-黎國明-何大樓	

13-10 使用 MERGE 來合併資料

部門資料表

	ID	部門名稱	主管	成立日期	人數
1	BA	理髮部	李美麗	2012-09-19	1
2	FI	財務部	陳季暄	2011-06-21	3
3	MG	管理部	張瑾雯	2010-01-01	2
4	SA	業務部	趙飛燕	2010-10-10	9

部門草案資料表

	ID	部門名稱	主管
1	FI	財務部	劉天王
2	MG	管理部	何大王
3	PU	採購部	楊大頭
4	SA	業務部	趙飛燕

這二個部門要更換主管

加入一個新部門

另外，沒有列出的部門（理髮部）則要裁撤掉

使用 **MERGE** 來合併資料

```
MERGE 部門 t      ← 目的資料表 (要被更新的資料表)
USING 部門草案 s ← 來源資料表
ON t.ID = s.ID   ← 指定二個資料表的配對 (JOIN) 條件
WHEN MATCHED AND t.主管 <> s.主管 THEN ← 條件符合且主管不同時, 就修改主管
    UPDATE
    SET t.主管 = s.主管
WHEN NOT MATCHED BY TARGET THEN ← 不在目的資料中的(但在來源資料中),就新增
    INSERT ( ID, 部門名稱, 主管)
    VALUES ( s.ID, s .部門名稱, s.主管)
WHEN NOT MATCHED BY SOURCE THEN ← 不在來源資料中的(但在目的資料中), 就刪除
    DELETE;      ← 最後必須加上分號表示結束
```

可以省去分別撰寫新增、修改、刪除敘述的麻煩。

MERGE 語法

```
MERGE [INTO] 目標資料      ← 可以是資料表、或檢視表 (INTO 可有可無)
USING 來源資料          ← 可以是資料表、檢視表、或查詢 (須加上別名)
ON 配對撮合的條件
WHEN MATCHED [ AND <額外的篩選條件> ] THEN
    UPDATE SET 或 DELETE 子句
WHEN NOT MATCHED [BY TARGET] [ AND <額外的篩選條件> ] THEN
    INSERT 子句
WHEN NOT MATCHED BY SOURCE [ AND <額外的篩選條件> ] THEN
    UPDATE SET 或 DELETE 子句
[OUTPUT <output_clause> ] ; ← 最後要加上分號表示結束
```


使用 **MERGE** 來合併資料

```
SELECT * FROM 部門
```

```
MERGE 部門 d
```

```
USING 部門草案 s
```

```
ON t.ID = s.ID ← 指定二個資料表的配對 (JOIN) 條件
```

```
WHEN MATCHED AND t.主管 <> s.主管 THEN
```

```
UPDATE
```

```
SET t.主管 = s.主管
```

```
WHEN NOT MATCHED BY TARGET THEN
```

```
INSERT (ID, 部門名稱, 主管)
```

```
VALUES (s.ID, s.部門名稱, s.主管)
```

```
WHEN NOT MATCHED BY SOURCE THEN
```

```
DELETE
```

```
OUTPUT $action, ← 將異動前、後的資料傳回, $action 欄會傳回異動的種類
```

```
deleted.ID, deleted.部門名稱, deleted.主管,
```

```
inserted.ID, inserted.部門名稱, inserted.主管;
```

```
SELECT * FROM 部門
```


使用 MERGE 來合併資料



	ID	部門名稱	主管	成立日期	人數
1	BA	理髮部	李美麗	2012-09-19	1
2	FI	財務部	陳季暄	2011-06-21	3
3	MG	管理部	張瑾雯	2010-01-01	2
4	SA	業務部	趙飛燕	2010-10-10	9

MERGE 前的部門資料表

inserted

	\$action	ID	部門名稱	主管	ID	部門名稱	主管
1	INSERT	NULL	NULL	NULL	PU	採購部	楊大頭
2	DELETE	BA	理髮部	李美麗	NULL	NULL	NULL
3	UPDATE	FI	財務部	陳季暄	FI	財務部	劉天王
4	UPDATE	MG	管理部	張瑾雯	MG	管理部	何大王

OUTPUT 子句傳回的異動資料：
刪除 1 筆、修改 2 筆、新增 1 筆

	ID	部門名稱	主管	成立日期	人數
1	PU	採購部	楊大頭	NULL	NULL
2	FI	財務部	劉天王	2011-06-21	3
3	MG	管理部	何大王	2010-01-01	2
4	SA	業務部	趙飛燕	2010-10-10	9

MERGE 後的部門資料表

使用 MERGE 來合併資料

股票庫存		
股票名稱	張數	
中鋼	10	
台塑	5	
華碩	8	

(記錄目前擁有多少股票)

序號	股票名稱	購買張數	已處理	股票交易記錄
1	台塑	5	1	(買或賣了多少股票)
2	中鋼	10	1	
3	華碩	8	1	
4	統一	20	0	
5	台塑	3	0	
6	中鋼	-10	0	

只須處理『未處理』的記錄

正數為買入, 負數為賣出

```

MERGE 股票庫存 t
USING (SELECT * FROM 股票交易記錄 WHERE 已處理 = 0) s
ON t.股票名稱 = s.股票名稱
WHEN MATCHED AND t.張數 + s.購買張數 = 0 THEN ← 賣光現有股票時
    DELETE
WHEN MATCHED THEN ← 買、賣現有股票時
    UPDATE SET t.張數 = t.張數 + s.購買張數
WHEN NOT MATCHED THEN ← 買新股票時
    INSERT (股票名稱, 張數)
    VALUES (s.股票名稱, s.購買張數)
OUTPUT $action,
        deleted.股票名稱, deleted.張數,
        inserted.股票名稱, inserted.張數;

SELECT * FROM 股票庫存
    
```

\$action	股票名稱	張數	股票名稱	張數
DELETE	中鋼	10	NULL	NULL
UPDATE	台塑	5	台塑	8
INSERT	NULL	NULL	統一	20

OUTPUT 傳回的異動資料, 分別
新增、修改、刪除了一筆記錄

股票名稱	張數
台塑	8
統一	20
華碩	8

MERGE 處理後的
股票庫存資料表

使用 **MERGE** 來合併資料

	序號	股票名稱	購買張數	已處理
1	1	台塑	5	1
2	2	中鋼	10	1
3	3	華碩	8	1
4	4	統一	20	0
5	5	統一	3	0
6	6	中鋼	-10	0

買了 2 次相同的新股票，執行 MERGE 時就會在股票庫存中新增 2 筆同名的股票！

執行 **MERGE** 前先將紀錄以 **GROUP BY** 進行彙總

```
MERGE ...  
USING (SELECT 股票名稱, SUM(購買張數) AS 購買張數  
FROM 股票交易記錄  
WHERE 已處理 = 0  
GROUP BY 股票名稱) s  
ON...  
...
```


13-11 SQL SCRIPT

1 按此鈕

這裡出現 "*" 表示未命名

```
SELECT '<' +  
CASE RIGHT(書籍名稱, 2)  
WHEN '手冊' THEN '1入門'  
WHEN '實務' THEN '2實例'  
WHEN '應用' THEN '3技巧'  
WHEN '秘笈' THEN '4技術'  
ELSE '5未分'  
END + '類' AS 類別  
FROM 書籍  
ORDER BY 類別
```

一或多個 SQL 批次

已成功... | John-PC (13.0 RTM) | John-PC\John (54) | 練習13 | 00:00:00 | 15 個資料列

第 11 行 第 15 欄 字元 13 INS

2 選取要存放的資料夾

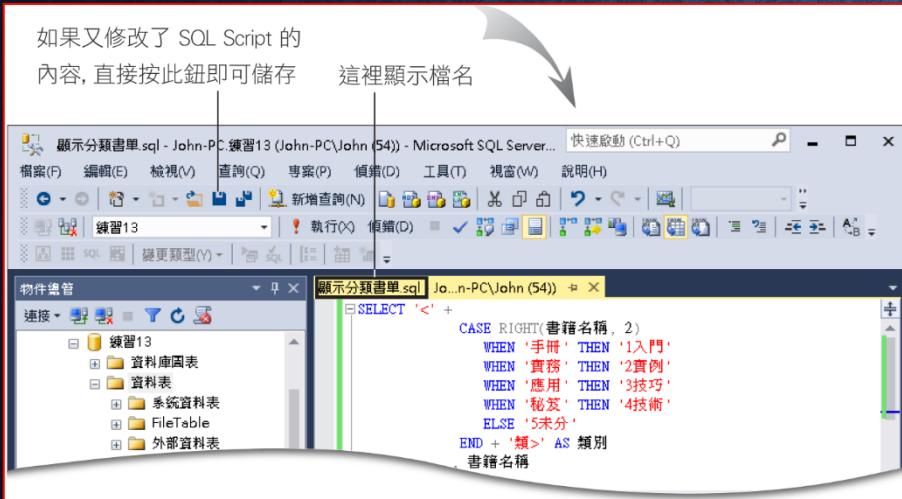
3 輸入 SQL Script 的檔名

建議使用 sql 為副檔名

4 按此鈕儲存

SQL SCRIPT

如果又修改了 SQL Script 的內容，直接按此鈕即可儲存 這裡顯示檔名



1 選取資料夾 2 選取要開啟的 SQL Script



3 按此鈕進行載入

SQL SCRIPT

4 選擇要連接的伺服器

伺服器類型(T): 資料庫引擎
伺服器名稱(S): John-PC
驗證(A): Windows 驗證
使用者名稱(U): John-PC\John
密碼(P):
 記住密碼(M)

連接(C) 取消 說明 選項(O) >>

如果尚未連接伺服器，會開啟此交談窗要求連接

5 按此鈕與伺服器連接

```
SELECT '<' +  
CASE RIGHT(書籍名稱, 2)  
WHEN '手冊' THEN '1入門'  
WHEN '實務' THEN '2實例'  
WHEN '應用' THEN '3技巧'  
WHEN '秘笈' THEN '4技術'  
ELSE '5未分'  
END + '類>' AS 類別  
FROM 書籍  
ORDER BY 類別
```

載入 SQL Script 的內容了

命令提示字元下的 **SQL** 執行工具

```
osql /S FLAG /d 練習13 /U sa /P abc /i 顯示分類書單.sql
```

- **/S KEN**
- **/d 練習 02**
- **/U sa**
- **/P pwd**
- **/i 顯示分類書單.sql**

參數大小寫是有區分的！

命令提示字元下的 **SQL** 執行工具

```
C: \Work>osql /S FLAG /d 練習 13 /U sa /P abc /i 顯示分類書單. sql
```

按 Enter 鍵

```
1> 2> 3> 4> 5> 6> 7> 8> 9> 10> 11> 12>
```

顯示執行的行號

類別	書籍名稱
<1 入門類>	Linux 使用手冊
<1 入門類>	Excel 使用手冊
<1 入門類>	PowerPoint 使用手冊
.....	
<5 未分類>	PhotoShop 細說從頭
<5 未分類>	AutoCAD 操作入門

(15 列受影響)

```
C:\Work>osql /S FLAG /d 練習 13 /U sa /P abc /i 顯示分類書單.sql /o result.txt
```


13-12 自動產生 SQL SCRIPT

- 使用 SQL Server Management Studio 自動產生 SQL Script

The screenshot shows the SQL Server Management Studio interface. The '物件總管' (Object Explorer) pane on the left shows a server instance with a database 'dbo' containing tables '訂單T6', '訂單總目', and '員工'. The '員工' table is selected, and a context menu is open over it. The menu items are: '新增資料表(B)...', '設計(G)', '選取前 1000 個資料列(W)', '編輯前 200 個資料列(P)', '編寫資料表的指令碼為(S)', '檢視相依性(V)', '記憶體最佳化建議程式(M)', '加密資料行...', '全文檢索引(T)', '儲存體(A)', '延展(S)', and '原則(O)'. The '編寫資料表的指令碼為(S)' item is highlighted, and its sub-menu is open, showing: 'CREATE 至(C)', 'ALTER 至(A)', 'DROP 至(D)', 'DROP 並 CREATE 至(R)', 'SELECT 至(S)', 'INSERT 至(I)', 'UPDATE 至(U)', 'DELETE 至(L)', and 'EXECUTE 至(E)'. The 'CREATE 至(C)' item is highlighted, and its sub-menu is open, showing: '新增查詢編輯器視窗', '檔案...', '剪貼簿', and '代理程式作業...'. The '新增查詢編輯器視窗' item is highlighted. A callout box points to the '新增查詢編輯器視窗' item with the text: '2 執行『編寫資料表的指令碼為/CREATE 至 新增查詢編輯器視窗』命令, 表示將 CREATE 此資料表的 SQL 敘述存到新的窗格中'. Another callout box points to the '檔案...' and '剪貼簿' items with the text: '這些命令可分別將結果存到檔案或剪貼簿中'. A third callout box points to the 'ALTER 至(A)', 'DROP 至(D)', 'DROP 並 CREATE 至(R)', 'SELECT 至(S)', 'INSERT 至(I)', 'UPDATE 至(U)', 'DELETE 至(L)', and 'EXECUTE 至(E)' items with the text: '不適用於目前資料表的命令會變成灰色而無法執行'. A fourth callout box points to the '編寫資料表的指令碼為(S)' item with the text: '1 在要產生 Script 的物件上按滑鼠右鈕, 這裡以員工資料表為例'. A grey arrow points from the '編寫資料表的指令碼為(S)' item to the '新增查詢編輯器視窗' item.

1 在要產生 Script 的物件上按滑鼠右鈕, 這裡以員工資料表為例

2 執行『編寫資料表的指令碼為/CREATE 至 新增查詢編輯器視窗』命令, 表示將 CREATE 此資料表的 SQL 敘述存到新的窗格中

這些命令可分別將結果存到檔案或剪貼簿中

不適用於目前資料表的命令會變成灰色而無法執行

自動產生 SQL SCRIPT

The screenshot displays the SQL Server Enterprise Manager interface on the left, showing a tree view of a database with various tables and views. On the right, the SQL Query window shows the generated T-SQL script for the '員工' table. A callout bubble points to the 'CREATE TABLE' statement, indicating that the script was automatically generated.

```
USE [練習13]
GO

/***** Object: Table [dbo].[員工]    Script Date: 20...
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[員工](
    [員工編號] [int] IDENTITY(1,1) NOT NULL,
    [姓名] [varchar](20) NOT NULL,
    [性別] [char](2) NULL,
    [主管員工編號] [int] NULL,
    [職稱] [varchar](10) NULL,
    [區域] [varchar](10) NULL,
    CONSTRAINT [PK_員工] PRIMARY KEY CLUSTERED
(
    [員工編號] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_K
) ON [PRIMARY]
GO
```

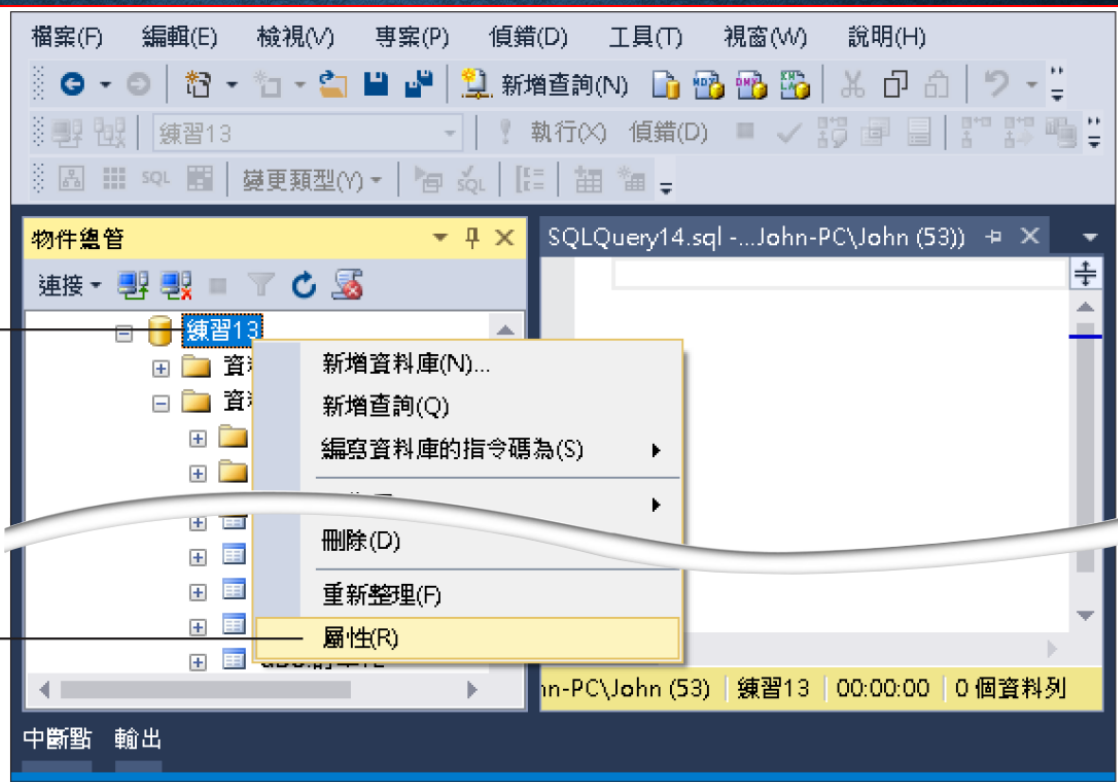
立即產生了 CREATE 資料表的敘述

自動產生 SQL SCRIPT

- 對物件屬性作修改時，也可以將修改的動作自動產生 **SQL** 敘述。

1 在要產生 Script 的物件上按滑鼠右鈕，這裡以練習 13 資料庫為例

2 執行此命令，我們要變更屬性的設定



自動產生 SQL SCRIPT

3 我們選擇變更此項的設定

5 按箭頭拉下列示窗，並選擇產生 SQL 敘述方式

4 將唯讀屬性變更為 True

6 由於我們只打算產生 SQL 敘述，並不打算真的變更屬性，所以按取消鈕不變更設定

資料庫屬性 - 練習13

選取頁面

- 一般
- 檔案
- 檔案群組
- 選項
- 變更追蹤
- 權限
- 擴充屬性
- 鏡像
- 交易記錄傳送
- 查詢存放區

連線

伺服器: John-PC

連接: John-PC\John

[檢視連接屬性](#)

進度

就緒

指令碼 說明

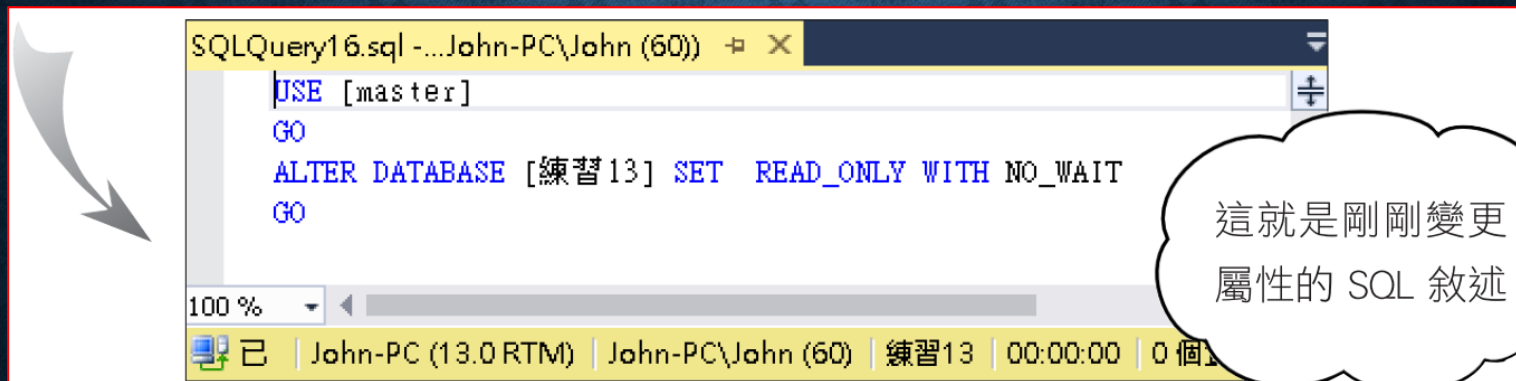
- 編寫動作的指令碼至新增查詢視窗 Ctrl+Shift+N
- 編寫動作的指令碼至檔案 Ctrl+Shift+F
- 編寫動作的指令碼至剪貼簿 Ctrl+Shift+C
- 編寫動作的指令碼至作業 Ctrl+Shift+M

其他選項(O):

遞迴觸發程序已啟用	False
數值捨入中止	False
狀態	
加密已啟用	False
限制存取	MULTI_USER
資料庫狀態	NORMAL
資料庫唯讀	True
復原	
目標復原時間 (秒)	0
頁面確認	CHECKSUM
資料指標	
預設資料指標	GLOBAL
認可時關閉資料指標已啟用	False
資料庫範圍設定	
資料庫唯讀	

確定 取消

自動產生 SQL SCRIPT



The screenshot shows a SQL query window titled "SQLQuery16.sql - ...John-PC\John (60)". The query text is:

```
USE [master]
GO
ALTER DATABASE [練習13] SET READ_ONLY WITH NO_WAIT
GO
```

A callout bubble on the right contains the text: "這就是剛剛變更屬性的 SQL 敘述" (This is the SQL statement for the property change just made).

The status bar at the bottom shows: "John-PC (13.0 RTM) | John-PC\John (60) | 練習13 | 00:00:00 | 0 個" (John-PC (13.0 RTM) | John-PC\John (60) | 練習13 | 00:00:00 | 0 objects).

13-13 使用不同資料庫或不同 **SERVER** 中的物件

- 物件的完整名稱
- 結構描述的意義與用途
- 預設結構描述
- 使用不同資料庫中的物件
- 使用不同 **Server** 中的物件

物件的完整名稱

伺服器名稱.資料庫名稱.結構描述.物件名稱

例如：

FLAG.練習 13.dbo.客

← FLAG 為 SQL Server 的名稱

伺服器名稱.資料庫名稱. 結構描述. 物件名稱

伺服器名稱.資料庫名稱. . 物件名稱

伺服器名稱..結構描述. 物件名稱

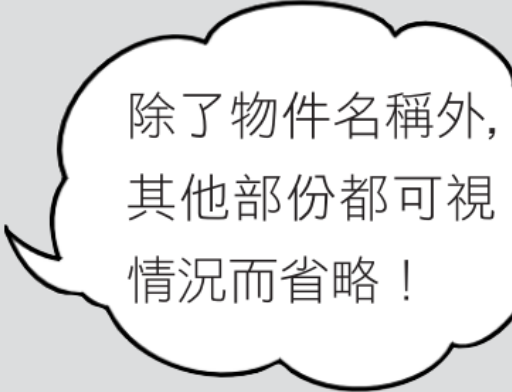
伺服器名稱...物件名稱

資料庫名稱.結構描述. 物件名稱

資料庫名稱..物件名稱

結構描述.物件名稱

物件名稱



除了物件名稱外,
其他部份都可視
情況而省略！

結構描述的意義與用途

- 在完整名稱中，伺服器名稱、資料庫名稱、物件名稱等都具有相當直覺的意義，一般人望文生義應該不會對這些名稱有所疑惑，不過對於結構描述 (**SCHEMA**) 可能就不太能瞭解其意義了。
- 資料庫名稱可以視情況省略，同樣地，結構描述也可以在許多情況下省略。
- 可以想像**資料庫名稱**就像磁碟機代號 (**C:**、**D:**、...)，**結構描述**是資料夾，**物件名稱**則為檔案。

預設結構描述

- 前面提到資料庫名稱可以視情況省略，同樣地，結構描述也可以在許多情況下省略。

使用不同資料庫中的物件

```
SELECT A.日期, B.客戶名稱 -- 在執行前請先確定目前伺服器中有練習 12 資料庫  
FROM 練習 13..訂單 AS A JOIN 練習 12..客戶 AS B  
ON A.客戶編號 = B.客戶編號
```



	日期	客戶名稱
1	2016-09-22	愚人書店
2	2016-10-14	新新書店
3	2016-10-18	十全書局

使用不同 **SERVER** 中的物件

- 連接到另一台 **SQL Server**

The screenshot shows the SQL Server Configuration Manager interface. The left pane shows the tree view with 'SQL Server 網路組態' expanded. The right pane shows a table of communication protocols:

通訊協定名稱	狀態
Shared Memory	已啟用
Named Pipes	已停用

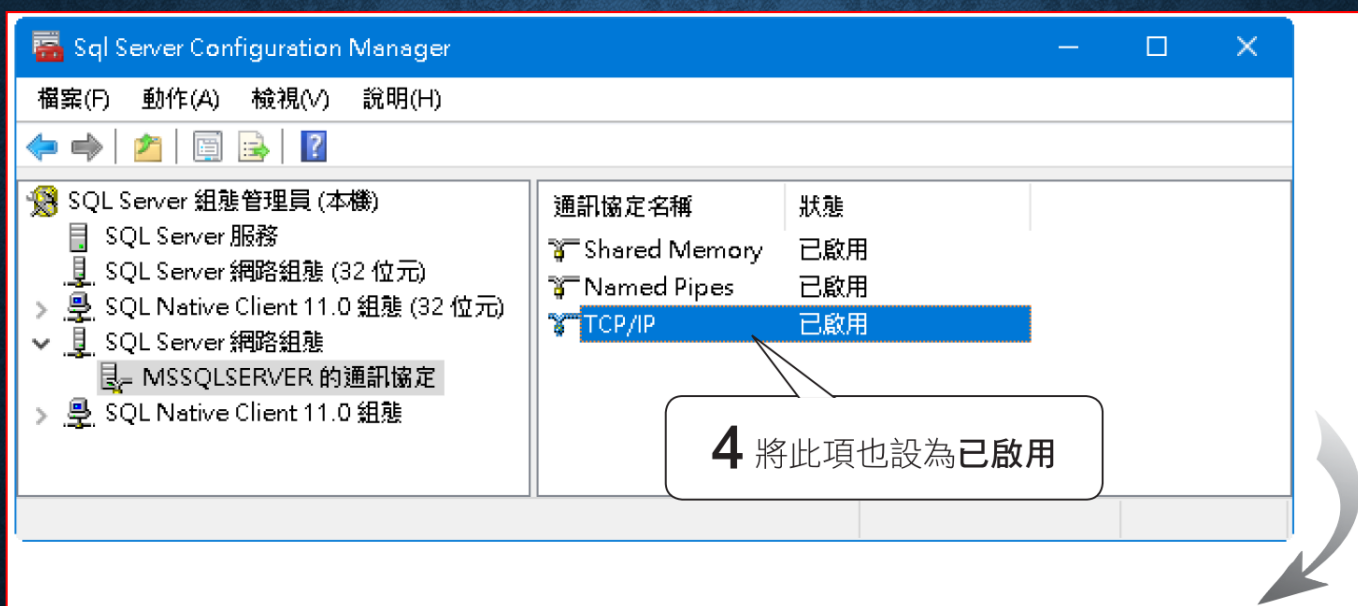
A context menu is open over 'Named Pipes', with '啟用(E)' selected. A callout box points to this menu item with the text: "2 按右鈕執行『啟用』命令".

A warning dialog box is open in the foreground. It contains the text: "將會儲存任何變更;但是,服務停止並重新啟動之前,變更將不會生效。" Below this text is a button labeled "確定". A callout box points to this button with the text: "3 按此鈕".

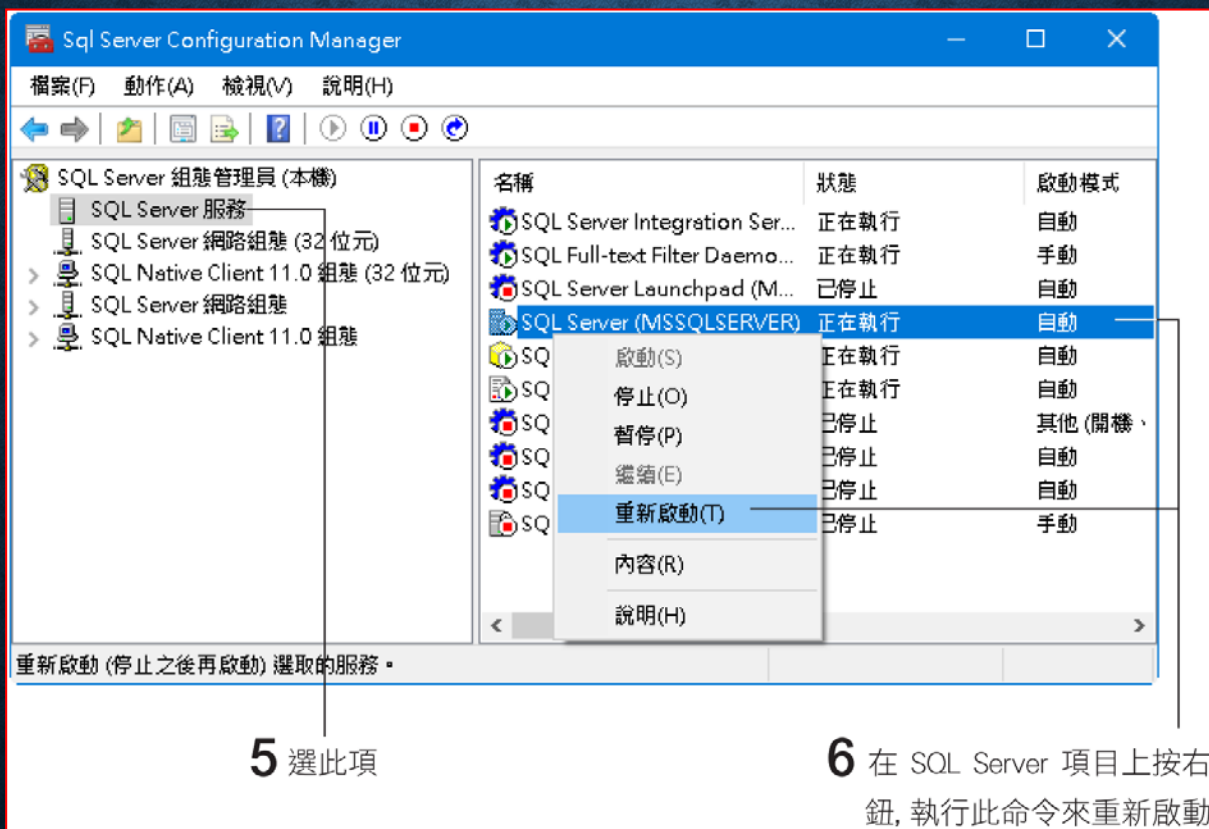
Another callout box points to the 'Named Pipes' item in the tree view with the text: "1 選此項".

A third callout box points to the warning dialog box with the text: "提示您必須重新啟動 SQL Server 服務後才會生效".

使用不同 **SERVER** 中的物件



使用不同 **SERVER** 中的物件



The screenshot shows the SQL Server Configuration Manager interface. The left pane shows the tree view with 'SQL Server (MSSQLSERVER)' selected. The right pane shows a list of services with their status and start mode. A context menu is open over the 'SQL Server (MSSQLSERVER)' service, with '重新啟動(T)' (Restart) selected. The status bar at the bottom of the window reads '重新啟動 (停止之後再啟動) 選取的服務。' (Restart (Restart after stop) selected service).

名稱	狀態	啟動模式
SQL Server Integration Ser...	正在執行	自動
SQL Full-text Filter Daemo...	正在執行	手動
SQL Server Launchpad (M...	已停止	自動
SQL Server (MSSQLSERVER)	正在執行	自動
SQL 啟動(S)	正在執行	自動
SQL 停止(O)	正在執行	自動
SQL 暫停(P)	已停止	其他(開機、
SQL 繼續(E)	已停止	自動
SQL 重新啟動(T)	已停止	手動
內容(R)		
說明(H)		

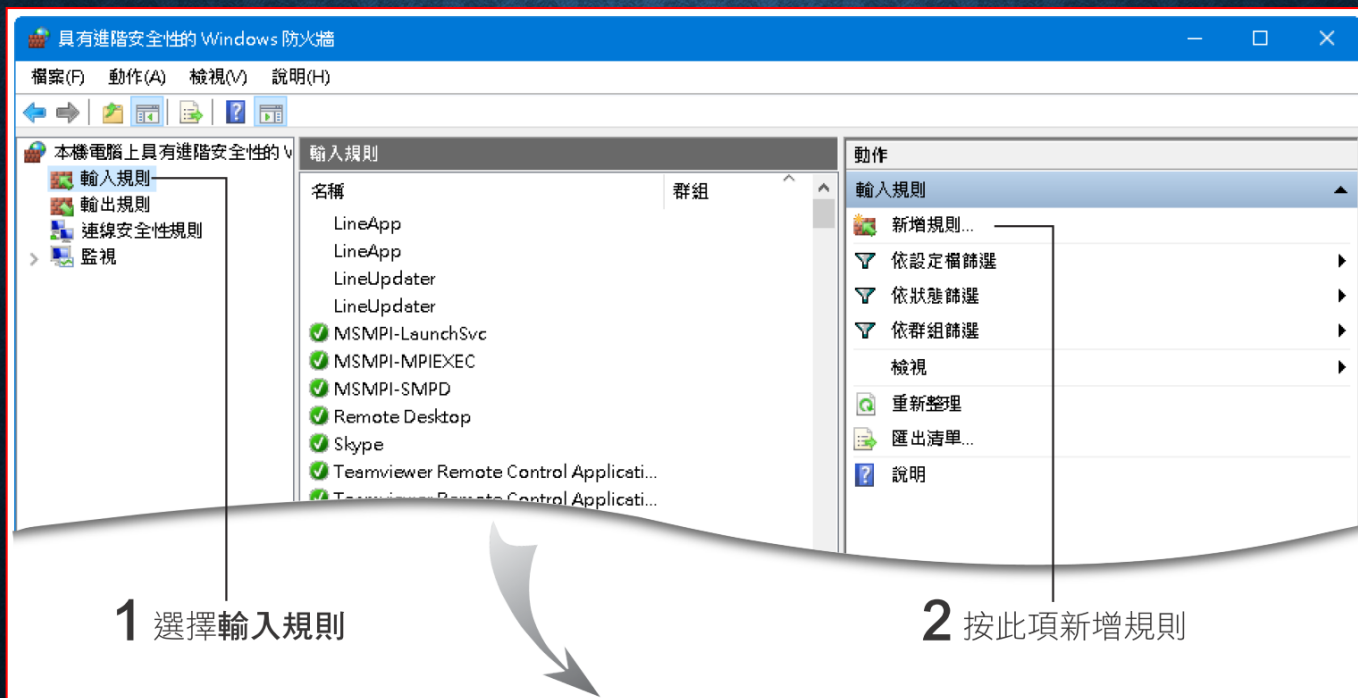
5 選此項

6 在 SQL Server 項目上按右鈕, 執行此命令來重新啟動

設定好之後, 如果有防火牆, 也要開啟 **TCP 1433** 連接埠

以 WIN 10 為例，開啟連接埠

- 開始按右鍵/設定，更新與安全性/Windows 安全性，按防火牆與網路防護，進階設定：



使用不同 **SERVER** 中的物件

3 選擇連接埠項目

新增輸入規則精靈

規則類型
選取要建立的防火牆規則類型。

步驟:

- 規則類型
- 通訊協定及連接埠
- 動作
- 設定檔
- 名稱

想要建立何種類型的規則？

程式 (P)
控制程式之連線的規則。

連接埠 (O)
控制 TCP 或 UDP 連接埠之連線的規則。

預先定義的 (E):
AllJoyn 路由器
控制 Windows 體驗之連線的規則。

自訂 (C)
自訂規則。

使用不同 **SERVER** 中的物件

按下一步繼續

4 選擇 TCP

新增輸入規則精靈

通訊協定及連接埠

指定套用這個規則的通訊協定與連接埠。

步驟:

- 規則類型
- 通訊協定及連接埠
- 動作
- 設定檔
- 名稱

此規則會套用到 TCP 或 UDP?

TCP (T)

UDP (U)

這個規則套用到所有本機連接埠或特定本機連接埠?

所有本機連接埠 (A)

特定本機連接埠 (S):

1433

範例: 80, 443, 5000-5010

5 選擇特定本機連接埠

6 輸入埠號 1433, 這是 SQL Server 預設使用的連接埠號 (若您更改了預設連接埠號, 則此處也要更改)

按下一步繼續

使用不同 **SERVER** 中的物件

新增輸入規則精靈

動作

指定要在連線符合規則中指定的條件時採取的動作。

步驟:

- 規則類型
- 通訊協定及連接埠
- 動作
- 設定檔
- 名稱

當連線符合指定的條件時，應採取哪些動作？

- 允許連線 (A)**
這包含使用 IPsec 保護的連線，以及未使用 IPsec 保護的連線。
- 僅允許安全連線 (C)**
這只包含已使用 IPsec 驗證的連線。會使用 [連線安全性規則] 節點中的 IPsec 內容和規則設定，來確保連線的安全。
- 封鎖連線 (K)**

7 選擇允許連線

按下一步繼續

使用不同 **SERVER** 中的物件

The screenshot shows the 'New Rule Wizard' window in Windows Firewall. The title bar reads '新增輸入規則精靈'. The main heading is '設定檔' (Profiles), with the instruction '指定要套用此規則的設定檔。' (Specify the profiles to which this rule will apply). On the left, a '步驟' (Steps) pane lists: '規則類型' (Rule type), '通訊協定及連接埠' (Protocol and ports), '動作' (Action), '設定檔' (Profiles), and '名稱' (Name). The '設定檔' step is currently selected. The main area asks '何時會套用此規則?' (When will this rule apply?). Three options are listed, each with a checked checkbox: '網域 (D)' (Domain) with the description '當電腦連線至其公司網域時套用。' (Apply when the computer is connected to its corporate network.); '私人 (P)' (Private) with the description '當電腦連線至私人網路位置時套用，例如住家或工作場所。' (Apply when the computer is connected to a private network location, such as home or work.); and '公用 (U)' (Public) with the description '當電腦連線至公用網路位置時套用。' (Apply when the computer is connected to a public network location.).

8 選擇要套用此規則的網路位置

按下一步繼續

使用不同 **SERVER** 中的物件

新增輸入規則精靈

名稱

指定此規則的名稱與描述。

步驟:

- 規則類型
- 通訊協定及連接埠
- 動作
- 設定檔
- 名稱**

名稱(N):
SQL Server

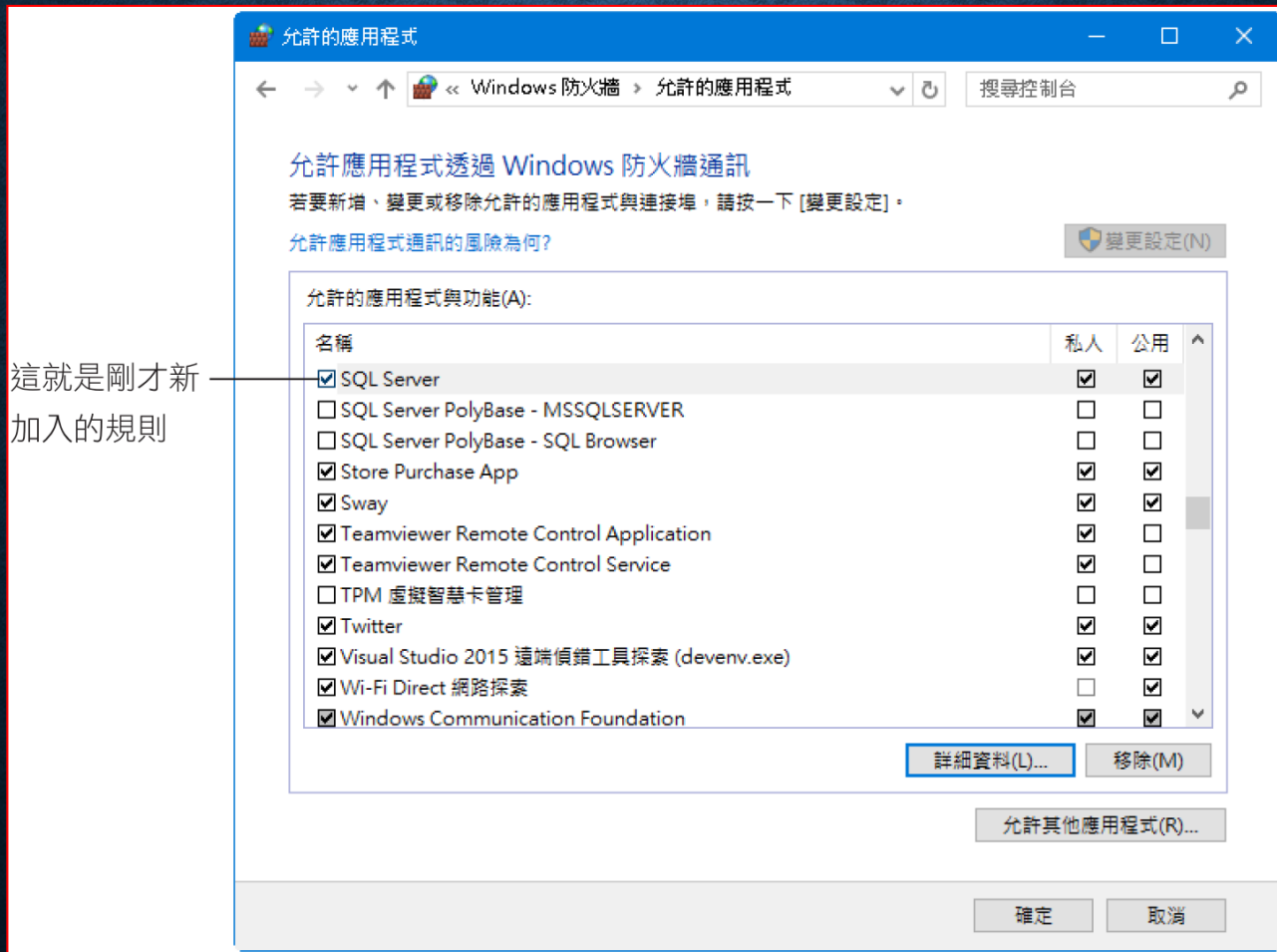
描述 (可省略)(D):

9 為此規則取一個名稱

10 按此鈕完成設定

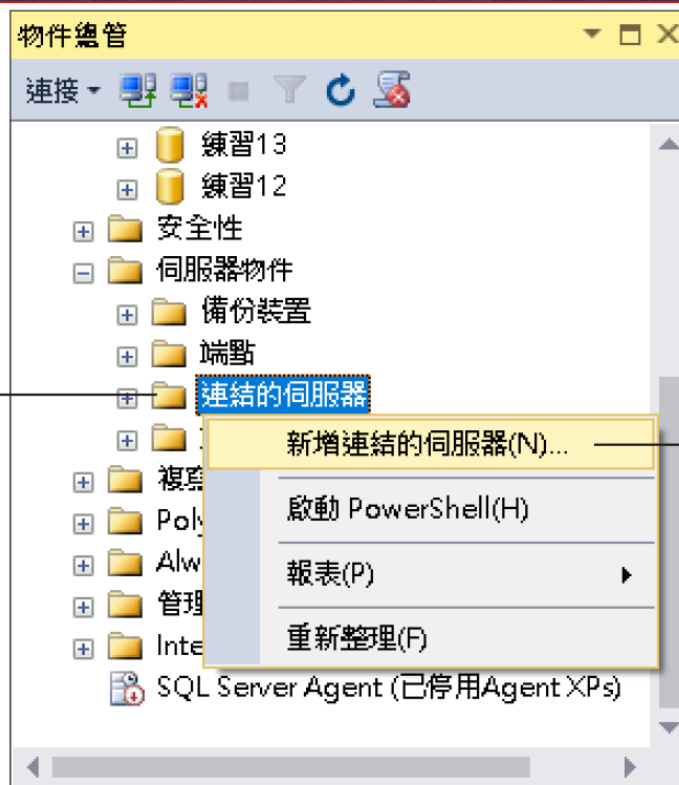
< 上一步(B) **完成(F)** 取消

使用不同 **SERVER** 中的物件



使用不同 **SERVER** 中的物件

1 選取伺服器物件下的
連結的伺服器項目



2 按滑鼠右鈕執行
『新增連結的伺
服器』命令



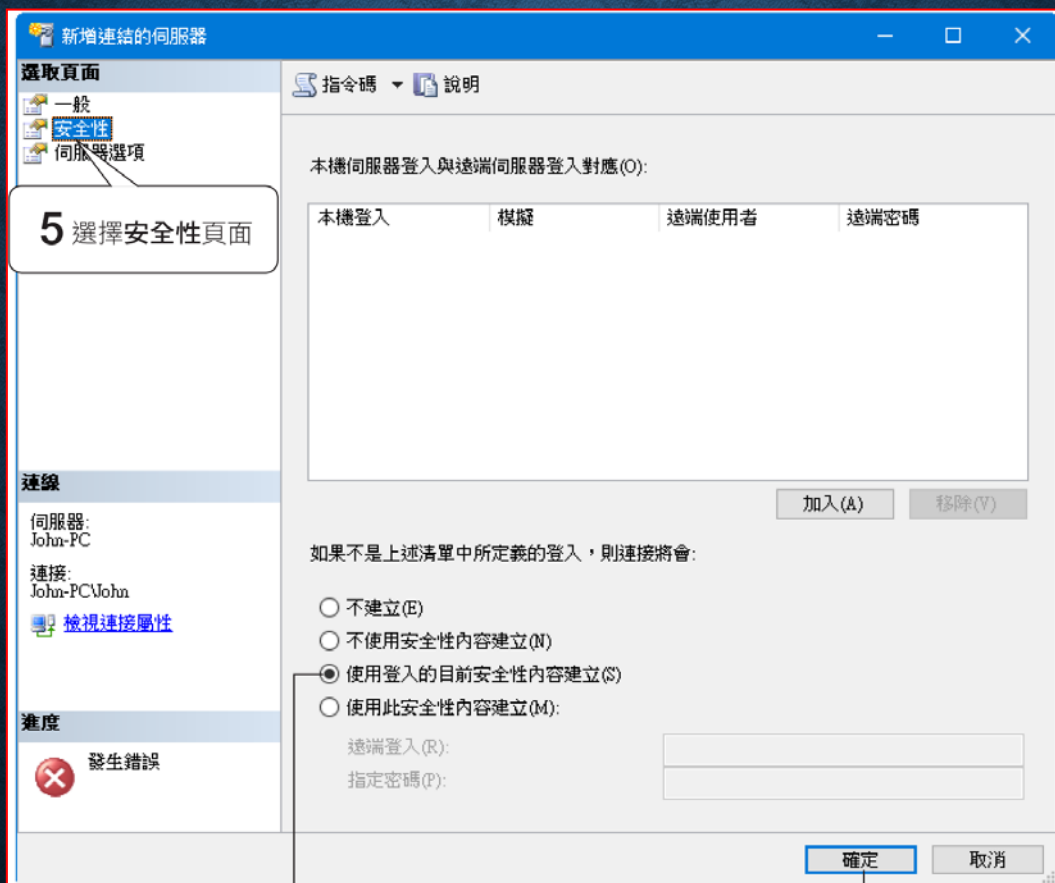
使用不同 **SERVER** 中的物件

4 選擇 SQL Server 項目

3 輸入要連結的伺服器名稱

也可連結到其他種類的 Server (例如 Oracle Server)

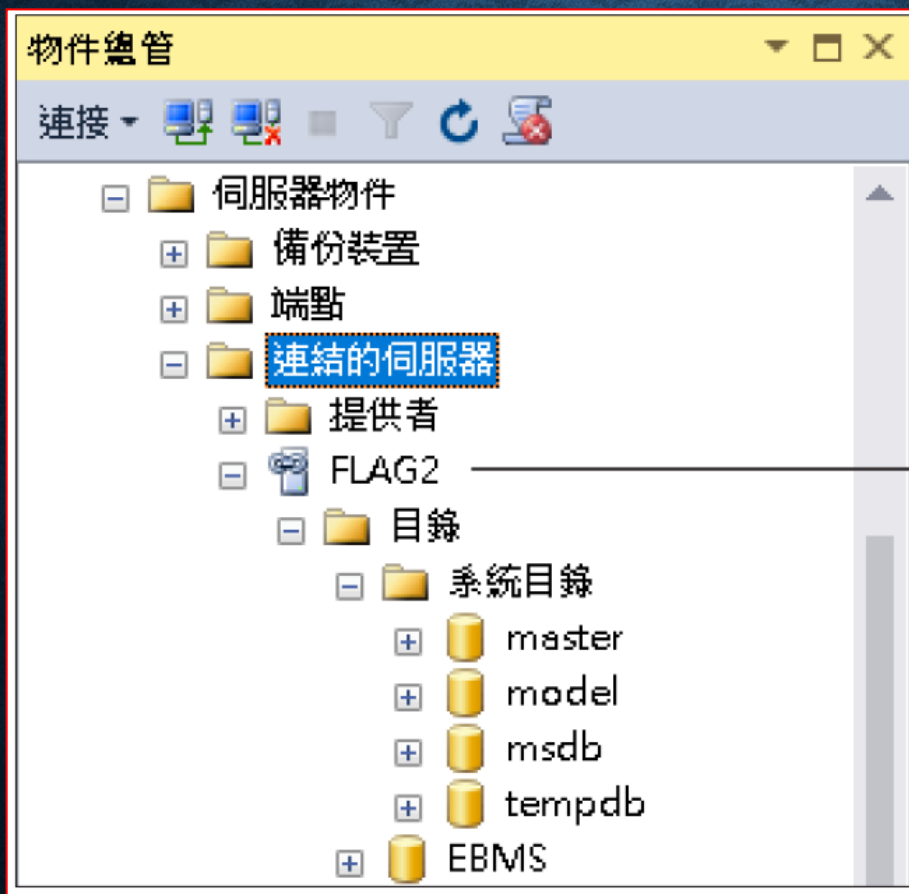
使用不同 **SERVER** 中的物件



6 若目前的伺服器和要連接的伺服器有相同的帳號及密碼，則可以選擇此項；否則請參見本節最後 2 段的說明

7 按確定鈕完成

使用不同 **SERVER** 中的物件



The screenshot shows the SQL Server Enterprise Manager interface. The tree view on the left is expanded to show the following structure:

- 物件總管
 - 伺服器物件
 - 備份裝置
 - 端點
 - 連結的伺服器
 - 提供者
 - FLAG2
 - 目錄
 - 系統目錄
 - master
 - model
 - msdb
 - tempdb
 - EBMS

A horizontal line points from the text on the right to the 'FLAG2' server instance in the tree view.

這是新加入的
連結伺服器

使用不同 **SERVER** 中的物件

```
SELECT *
```

```
FROM FLAG2.練習 13.dbo.書籍
```

← 此時記得指明其伺服器名稱是 FLAG2

使用不同 **SERVER** 中的物件

本機的登入帳號

遠端連結伺服器中的登入帳號及密碼

遠端連結伺服器屬性 - FLAG2

本機伺服器登入與遠端伺服器登入對應(O):

本機登入	模擬	遠端使用者	遠端密碼
John-PC\John	<input type="checkbox"/>	John	*****
Ken	<input checked="" type="checkbox"/>		

加入(A) 移除(V)

如果不是上述清單中所定義的登入，則連接將會:

- 不建立(E)
- 不使用安全性內容建立(N)
- 使用登入的目前安全性內容建立(S)
- 使用此安全性內容建立(M):

遠端登入(R): sa

指定密碼(P): *****

確定 取消

若勾選此項，表示用本機的登入帳號及密碼來存取遠端伺服器

當目前登入的帳號不在上表中時，才會依此處的選項來決定如何存取

要注意安全性的問題