

TRANSACT-SQL Stored Procedure & Function

朱克剛



變數宣告

■ 變數形式

- 區域變數 (使用者自訂變數)
 - 變數以 @ 開頭
 - 例如 : @n
- 全域變數 (系統變數)
 - 變數以 @@ 開頭
 - 例如 : @@ERROR

■ 宣告方式

- `DECLARE @n Int`
- `DECLARE @cname nvarchar(10)`

一定要給大小

變數的設定與顯示

- 設定變數

```
SET @n = 100
```

```
SELECT @n = 100
```

- 顯示變數的內容

```
SELECT @n
```

```
PRINT(@n)
```

顯示在訊息視窗

將 SELECT 結果放入變數

- 將A01的姓名放入變數中

```
DECLARE @cname NVARCHAR(50)
```

```
SELECT @cname = cname  
FROM UserInfo  
WHERE uid = 'A01'
```

IF 判斷式

- 如果 BEGIN END 中只有一行程式碼, 可省略
- 沒有 ELSEIF 之類的語法, 要用 ELSE IF 巢狀判斷

```
IF @i > 10
BEGIN
    -- 判斷式成立
END
ELSE
BEGIN
    -- 判斷式不成立
END
```

```
IF @i > 10
    -- 判斷式成立
ELSE
    -- 判斷式不成立
```

CASE 判斷式

```
declare @cname nvarchar(50)
declare @lastname nvarchar(10)

select @cname = cname from userinfo where uid = 'A02'

set @lastname = case @cname
    when '王大明' then '王'
    when '李大媽' then '李'
    else '不知道姓啥'
end

select @lastname as lastname
```

While 迴圈

- T-SQL 只有 while 迴圈，沒其他的

初始化

```
DECLARE @i INT = 0
```

```
SET @i = 0
```

```
WHILE @i < 10
```

```
BEGIN
```

```
    IF @i = 5
```

```
    BEGIN
```

```
        SET @i = 7
```

```
        Continue
```

```
    END
```

```
    IF @i = 9
```

```
        Break
```

```
    print @i
```

```
    SET @i = @i + 1
```

```
END
```

結果：

0
1
2
3
4
7
8

GOTO

- 程式直接跳到 GOTO 所指定的標籤執行
- GOTO 會破壞程式可讀性，盡量精簡使用，不要來回跳

```
GOTO Label
```



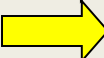

```
...
```

```
Label:
```

```
...
```


@@ERROR




- 每一個SQL Command 執行完, 系統都會將錯誤編號放到@@ERROR 變數中, 如果沒有錯誤發生, @@ERROR 等於 0

成功   `begin transaction`
失敗   `update UserInfo set cname = '吳小弟' where uid = 'A05'`
`insert into UserInfo values ('A01', '沈月月')`

```
if @@error <> 0
    rollback
else
    commit
```

錯誤控制 – 小心！！

- 執行完, A05資料已被更改

失敗   `begin transaction`
成功   `insert into UserInfo values ('A01', '沈月月')`
`update UserInfo set cname = '吳小弟' where uid = 'A05'`

```
if @@error <> 0
    rollback
else
    commit
```

XACT_ABORT

- 設定為 ON 時，只要產生任何一個錯誤就會導致整個交易失敗，然後自動 rollback
- 預設為 OFF，指令失敗並不會導致交易結束，會繼續往下執行

```
set xact_abort on
```

```
begin tran
```

```
insert into UserInfo (uid) values ('Z02')
```

```
insert into UserInfo (uid) values ('A01')
```

```
commit
```



@@ROWCOUNT

- 紀錄 SQL Command 執行後所影響的資料筆數，此為系統變數

```
update UserInfo set cname = NULL  
print @@rowcount
```

若6筆資料被update,
則@@rowcount等於6

Try...Cache

- 類似其他語言的 try catch 機制，用來攔截錯誤，例如下面的 SQL 指令會造成 PK 重複的錯誤，然後被 catch 攔截並處理

```
begin try
  insert into UserInfo values ('A01', '吳小弟')
end try
begin catch
  print(error_message())
end catch
```

預存程序

預存程序 – Stored Procedure

- 將T-SQL程式碼儲存在資料庫中並給予一個名字，需要時呼叫即可
- 資料庫內建的預存程序以sp_開頭命名
 - 例如 *sp_databases* 、 *sp_tables*
- 執行方式
 - *exec 名字 參數*
 - 例如 *exec sp_databases*

建立預存程序

```
create procedure procedure_name  
as  
begin  
    -- code here  
end
```

■ 執行

```
exec procedure_name
```


以結果集方式傳回資料

- 可在預存程序中執行查詢指令，查詢結果會直接傳至 client 端

```
create procedure p_proc  
as  
begin  
    select * from UserInfo  
end
```

有參數的預存程序

- 輸入UID，傳回該 UID 資料

```
create procedure p_proc  
    @uid varchar(50)  
as  
begin  
    select * from UserInfo where uid = @uid  
end
```

- 執行

```
exec p_proc 'A01'
```

透過 OUTPUT 參數傳回資料

- 此參數可以輸入資料，也可以輸出資料

```
create procedure p_proc
    @uid varchar(50),
    @cname varchar(50) output
as
begin
    select @cname = cname from UserInfo where uid = @uid
end
```

- 執行

```
declare @data varchar(50)
exec p_proc 'A01', @data output
select @data
```

透過 return 傳回資料

- 雖然不是函數，但也可以使用 return 方式傳回資料，但僅限 int 型態

```
create procedure p_proc
    @uid varchar(50), @cname varchar(50)
as
begin
    declare @status int = 1
    begin try
        insert into UserInfo (uid, cname) values (@uid, @cname)
        set @status = 0
    end try
    begin catch end catch
    return @status
end
```

- 執行

```
declare @status int
exec @status = p_proc 'C01', 'David'
select @status
```

指定參數名稱

```
create procedure p_proc  
    @first int,  
    @second int,  
    @third int  
as  
begin  
    select @first as first, @second as second, @third as third  
end
```

- 呼叫時可以指定參數名稱

```
exec p_proc @first = 20, @third = 5, @second = 7
```

	first ▼	second ▼	third ▼
1	20	7	5

參數預設值

```
CREATE PROCEDURE p_userinfo
    @uid NVARCHAR(50) = null
AS
BEGIN
    if @uid is null
        select * from UserInfo
    else
        select * from UserInfo where uid = @uid
END
```

■ 執行看看

```
exec p_userinfo
exec p_userinfo 'A01'
```

提前結束

- Return 可以提前結束預存程序

```
create procedure p_proc
    @a float,
    @b float
as
begin
    if @b = 0
    begin
        print('分母不可為零')
        return
    end

    select @a / @b as status
end
```

練習

- 建立一個使用者註冊的 Stored Procedure，接受四個參數
 - *uid*
 - *cname*
 - *address*
 - *tel*
- 呼叫完後相關資料表填入適當資料
- 非主索引欄位可以不填資料

練習

- 根據上題，請讓後端呼叫者知道目前執行結果，並且如果使用者的 ID 已經被註冊了，需傳回錯誤代碼
- 將此錯誤代碼與錯誤訊息表關連，一氣呵成取得錯誤碼與錯誤訊息

自訂函數

自訂函數

- 有時也稱為 Stored Function (預存函數)

- 語法：

```
CREATE FUNCTION f_name (PARAMETER LIST)
RETURNS (return_type) AS
BEGIN

END
```

- 例如:

```
CREATE FUNCTION f_add (@v1 FLOAT, @v2 FLOAT)
RETURNS FLOAT
AS
BEGIN
    RETURN @v1 + @v2
END
```

函數呼叫

- 自訂函數執行時一定要加 schema 名字

```
SELECT dbo.f_add(4, 3)
```

Function 的傳回型態

- 純量型別
 - 如 *int, char, nvarchar...* 等
- 資料表型別
 - 傳回 *recordset* , 型態為 *TABLE*
 - 呼叫方式: *select * from dbo.f1()*

這裡 *schema* 名字可以省略

行內資料集函數

- Inline Table-valued Function
- 內容只有一行指令，因此沒有 BEGIN END
- 類似 View，但與 View 的差別在於可以輸入參數

```
CREATE FUNCTION showLastname(@lastname varchar(10))  
RETURNS TABLE  
AS  
RETURN (  
    select *  
    from UserInfo  
    where cname like @lastname + '%'  
)
```

多敘述資料集函數

■ Multi-statement Table-valued Function

```
CREATE FUNCTION customUserInfo() RETURNS @t TABLE
(
    uid nvarchar(50),
    cname nvarchar(50)
)
AS
BEGIN
    insert @t select uid, isnull(cname, '') from UserInfo
    insert @t select 'C01', '丁小雨'
    return
END
```

補充 - 暫存表

- SQL Server 有三種類型的暫存表，儲存在 tempdb 中

- #Table
- ##Table
- @table

- #Table

- 建立者擁有，連線中斷後會消失

- ##Table

- 全域，所有使用者斷線後會消失

- @table

- 資料型態為 Table 的變數，指令執行完就消失

```
declare @table Table
(
    product varchar(50),
    price int
)
insert into @table values ('鉛筆', 10)
insert into @table values ('原子筆', 20)
select * from @table
```


CURSOR

用途

- 一筆一筆的處理資料，可以對每一筆資料作最細微的控制
- 例如：將欄位中的阿拉伯數字轉成大寫國字
 - 1 -> 壹元
 - 203 -> 貳佰零叁元

建立、開啟與關閉 Cursor

建立 → `DECLARE c CURSOR FOR
select * from Bill`

開啟 → `OPEN c`

關閉 → `...`
`CLOSE c`

移除 → `DEALLOCATE c`

fetch

■ 列出每一筆電話費用

BEGIN

declare c cursor for select fee from Bill

declare @fee int

open c

fetch c into @fee

while (@@fetch_status = 0)

begin

print @fee

fetch c into @fee

end

close c

deallocate c

END

cursor

fee

cursor

250

cursor

300

100

0 : 成功

-1 : 失敗

可改變資料內容的 Cursor

- 將沒有名字的會員設定名字為「NONAME」

```
BEGIN
  declare c cursor for
  select uid from UserInfo
    where cname = '' or cname is null for update
  open c

  fetch c
  while (@@fetch_status = 0)
  begin
    update userinfo set cname = 'NONAME' where current of c
    fetch c
  end

  close c
  deallocate c
END
```

應用

- 在 UserInfo 中加上 id 欄位，型態為 Int
- 將將第一筆資料的 id 欄位填入 10000，第二筆為 10001 以此類推到最後一筆
- 全部填完後將 id 欄位設定為不可為 NULL，並且加上 unique index

練習

- 修改之前新增UserInfo資料的 trigger，使用 Cursor 將每筆資料異動都會記錄在 Log 中，而不僅僅只是最後一筆被紀錄而已

```
CREATE TRIGGER tr_log_userinfo
ON UserInfo
FOR INSERT
AS
    DECLARE @uid NVARCHAR(50)
    DECLARE @cname NVARCHAR(50)

    -- 停止計算 SQL 影響的資料筆數
    SET NOCOUNT ON

    SELECT @uid = uid, @cname = cname FROM INSERTED
    INSERT INTO Log (body) VALUES
        ('在資料表 UserInfo 中新增 ' + @uid + ', ' + @cname + ' 資料')
```