

第七章 結構化查詢語言 SQL





本章內容

- 7.1 前言
- 7.2 資料定義語言 (DDL)
- 7.3 資料處理語言 (DML) 的查詢部份
- 7.4 資料處理語言 (DML) 的異動部份
- 7.5 資料控制語言 (DCL)
- 7.6 SQL與關聯式代數之間的對應
- 7.7 嵌入式 SQL (Embedded SQL)
- 7.8 動態 SQL 與 SQL Injection攻擊
- 7.9 SQL Server 2008 的程式化功能
- *7.10 SQL Server 2008 進階的 SQL 功能
- 7.11 系統目錄的查詢與異動

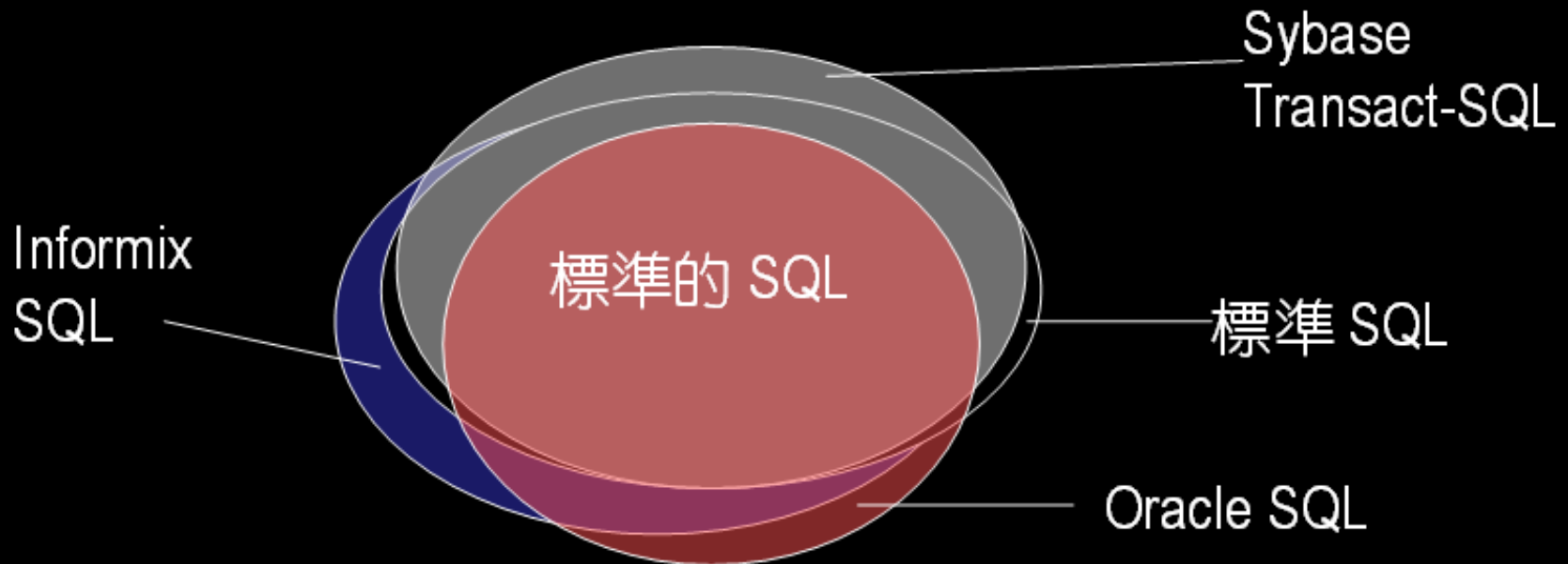



SQL (Structured Query Language) 簡述

- 由 D.D. Chamberlin 與 R.F. Boyce 在 IBM 研究中心所發展出來
- 可看成是關聯式代數與關聯式計算的綜合體
- 已經是標準的查詢語言
- 1986 與 1987 年分別被 ANSI (American National Standards Institute) 與 ISO (International Standards Organization) 採用為關聯式資料庫查詢語言的標準 (稱為 SQL-86)
- 後續針對功能與特性強化，訂有 SQL-89 和 SQL-92 (也稱 SQL2) 等版本。
- 目前大部份資料庫產品都遵循 SQL-89 或 SQL-92 的規範

SQL 簡述

- SQL-86 → SQL-89 → SQL-92 → SQL 3 (SQL:1999~SQL:2003)
- 廠商所提供的大多是 Superset of a Subset
- SQL Server 2008 的 SQL 稱為 **Transact-SQL**



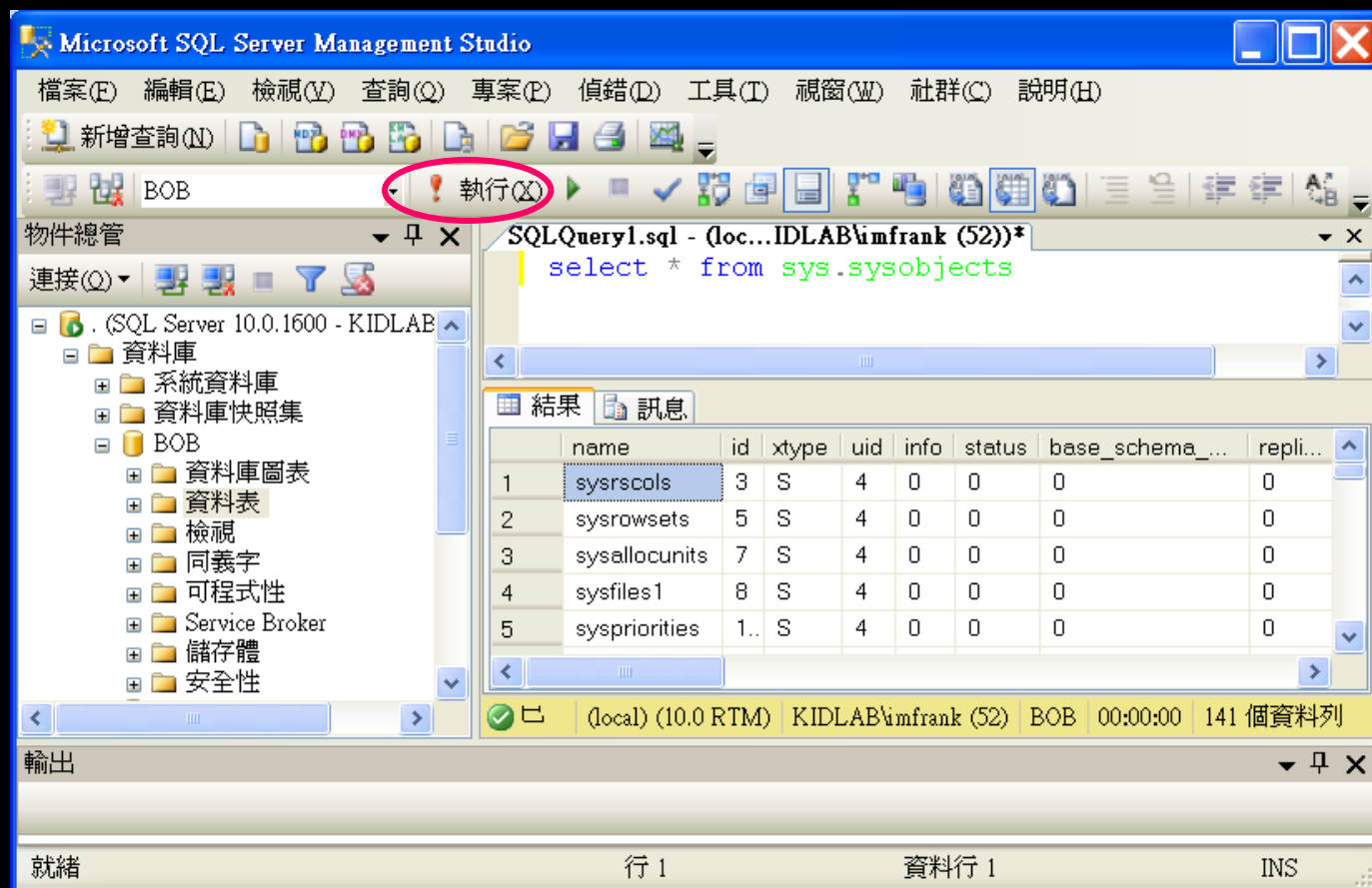


SQL 的三類命令

- 資料定義語言 (Data Definition Language, DDL) : 定義資料庫的物件。
 - SQL Server 2008 將所有 DDL 指令統一以 **Create**、**Alter** 與 **Drop** 來進行物件的新增、修改與刪除動作。
- 資料操縱語言 (Data Manipulation Language, DML) : 對關聯表內容進行新增 (**Insert**)、刪除 (**Delete**)、修改 (**Update**) 與選擇 (**Select**) 等運算
- 資料控制語言 (Data Control Language, DCL) : 控制資料庫內物件的使用權限與安全設定
 - SQL Server 2008 透過 **Grant**、**Revoke** 與 **Deny** 進行授與、撤銷與拒絕使用者或各類角色的使用權限

SQL Server 2008 的查詢畫面

■ Transact-SQL 指令間以 go 隔開





Transact-SQL 的註解寫法

- 單列註解 (Single-Line Comments)：是以兩個 '--' 引導，後面接著寫上註解。
- 跨列註解：其語法為 */* text-of-comments */*，與 C 語言的註解寫法相同。也可以使用巢狀註解。

```
-- This is a legal comment in SQL Server 2008.
```

```
/*
```

```
The following SQL command is also a legal command in SQL Server 2008.
```

```
*/
```

```
/* In SQL Server 2008, You can use nested /* comments */ as comments. */
```



資料定義語言 (DDL)

- 分成以下幾大類 (有打 * 的將於本書中探討)

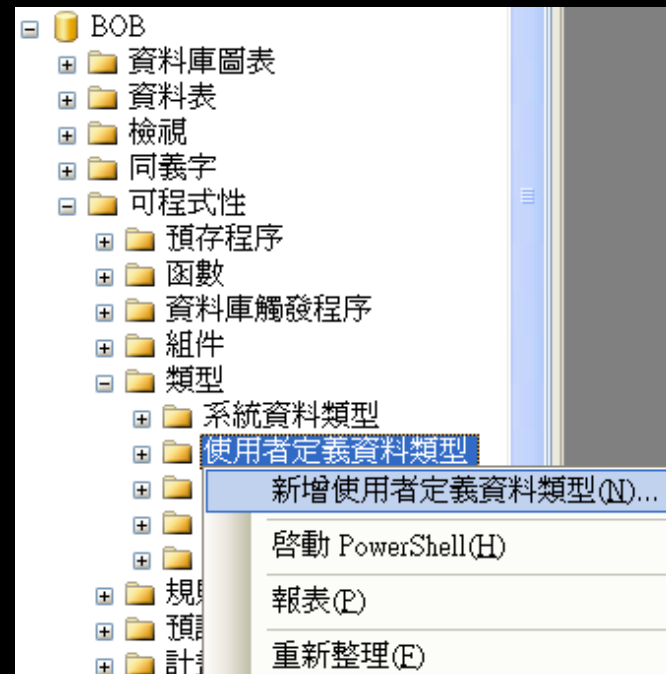
CLR 整合的 DDL	Service Broker 方面的 DDL
資料庫方面的 DDL*	視界方面的 DDL*
基底關聯表方面的 DDL*	資料分割 (Partition) 方面的 DDL
定義同義字 (Synonym) 的 DDL*	全文檢索 (Full Text Indexing) 方面的 DDL
索引 (Index) 方面的 DDL*	觸發程序*和事件通知 (Event Notification) 的 DDL
安全性方面的 DDL*	預儲程序與使用者自訂函數的 DDL*

使用者自訂資料型態

- SQL Server 2008 不建議使用以前的 sp_addxxx 指令

- 建議透過下述自訂資料型態指令來為之

- `create type type_of_age from tinyint not null`
`go`
`create type type_of_color from char(12) not null`
`go`
`create type type_of_price from tinyint null`
`go`



使用者自訂資料型態的建立畫面

編寫限制規則 Rule

- 使用 Create Rule 編寫限制規則 (Rule) 以限制屬性值域之合法值

```
create rule rule_of_age as @age between 18 and 65
```

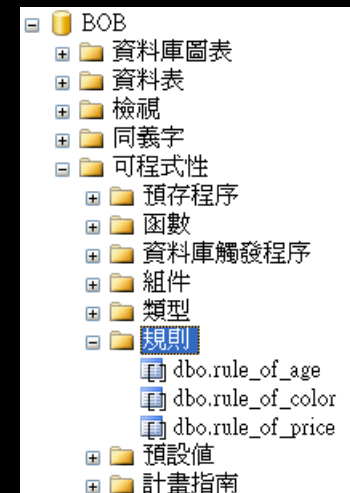
```
go
```

```
create rule rule_of_color as @color in ('Red', 'Yellow', 'Blue', 'White', 'Black')
```

```
go
```

```
create rule rule_of_price as @price between 80 and 250
```

```
go
```



編寫預設值規則 Default

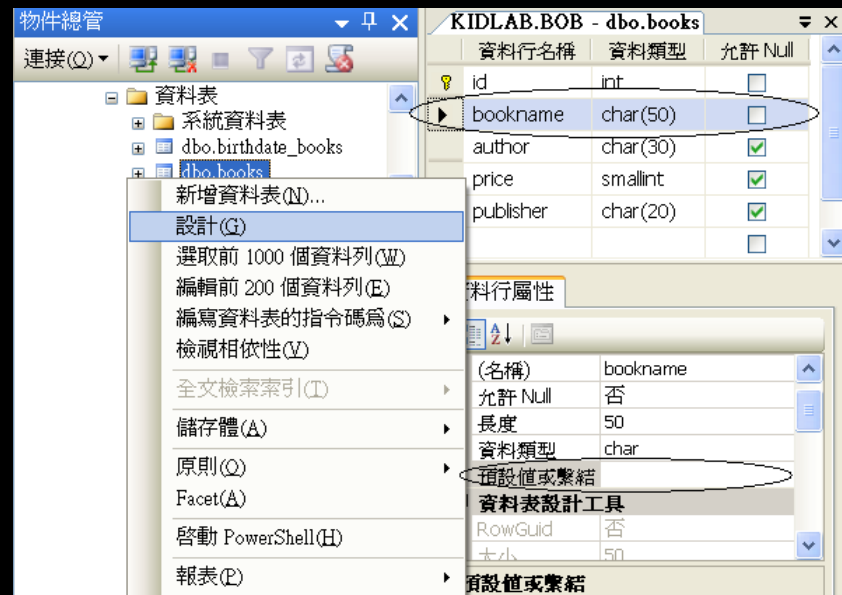
- 使用 Create Default 編寫預設規則 (Default) 以預設屬性值

create default *default_of_age* as 25

create default *default_of_color* as 'Red'

create default *default_of_price* as 100

go





將屬性與 Rule/Default 繫結

- 將屬性與 Rule/Default 結合 (每一個指令後面都要加 go)

```
1> sp_bindrule 'rule_of_price', 'Books.price'
```

```
2> sp_bindefault 'default_of_price', 'Books.price'
```

- 取消屬性的 Rule/Default (每一個指令後面都要加 go)

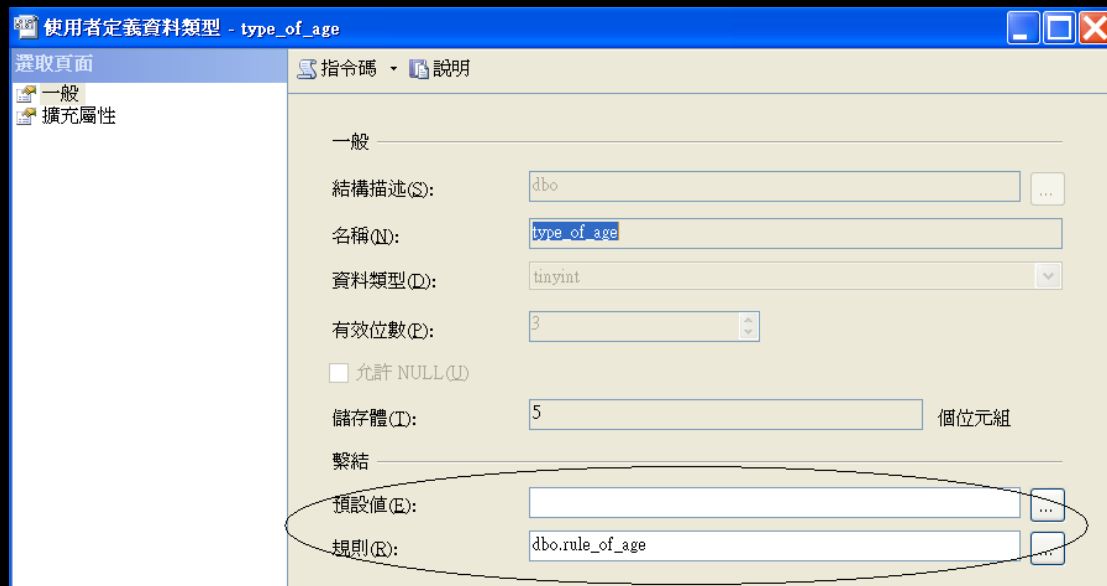
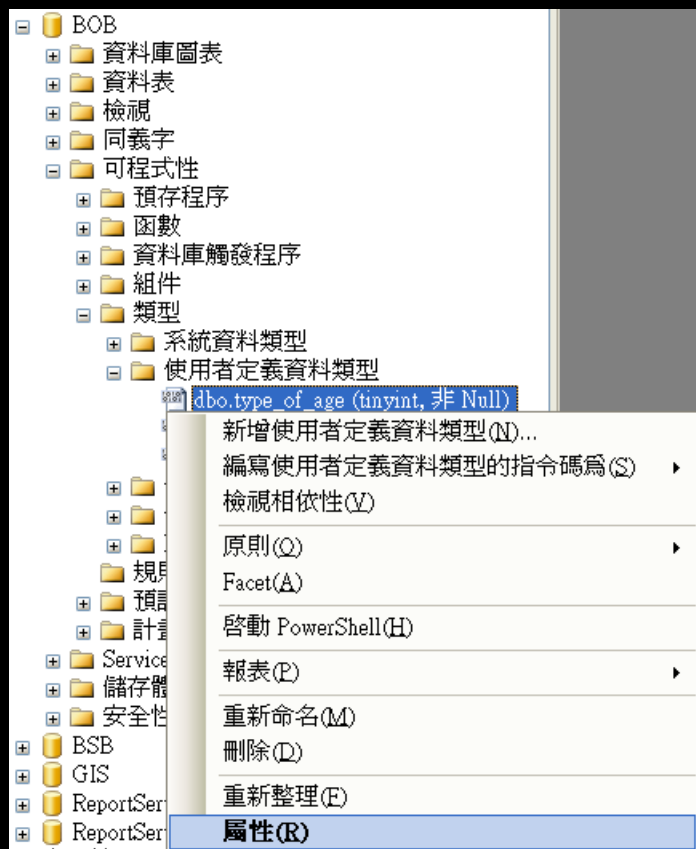
```
1> sp_unbindrule 'Books.price'
```

```
2> sp_unbindefault 'Books.price'
```

- 刪除自訂資料型態 (每一個指令後面都要加 go)

```
1> sp_droptype type_of_age
```

也可以透過畫面設定





資料庫的 DDL

- Create/Drop Database—新增/刪除資料庫

If Exists (Select * From master.sys.sysdatabases Where name = 'BOB')

Drop Database *BOB*

go

Create Database *BOB*

go

- 建立 *BOB* 資料庫快照集 (Database Snapshot)

Create Database *BOB_snapshot_1* on (Name = *BOB*,
Filename = 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\DATA\BOB_snapshot_1.ss')
As Snapshot of *BOB*

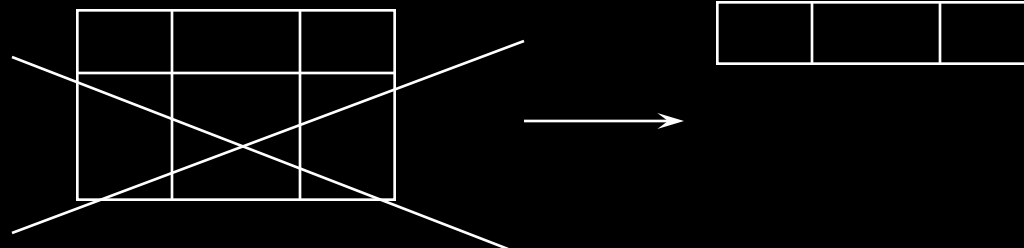
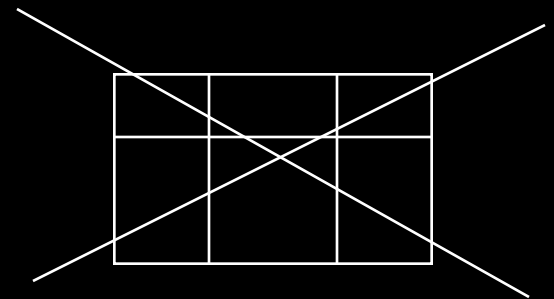


Alter Database—修改資料庫定義

- 更改 BOB 名稱為 New_BOB :
`Alter Database BOB Modify Name = New_BOB`
`go`
- 擴充存放 BOB 資料庫的檔案數量 :
`Alter Database BOB Add File (`
`Name = BOBdat1, Filename = 'c:\BOB_1.ndf', Size =`
`10MB, MAXSIZE = 50MB, FILEGROWTH = 2MB)`
`go`
- 建議用手動的方式透過 SQL Server Management Studio 建立、變更或刪除資料庫較方便

基底關聯表的 DDL

- 定義、修改，以及去除基底關聯表的綱要結構
- create table
- alter table
- drop table (資料與綱要都刪除)
- 清除基底關聯表中的資料
- truncate table (綱要仍然保留)





Create Table: 新增關聯表

- create table *base-table-name* (
 columns-name data-type [null | not null], ...
 [, primary key (*column-commalist*)]
 [, foreign key (*column-commalist*)
 references *base-table* [(*column-commalist*)]
 [on delete *option*] (*option* 有 { cascade | no action | set null | set default })
 [on update *option*] ,...]
 [, check (*condition*)]
)



宣告關聯表後再另外指定限制條件

- 事後指明主鍵的 SQL 指令：
`alter table table-name add constraint constraint-name primary key (column-commalist)`
- 要加入外來鍵的話，則是使用：
`alter table table-name add constraint constraint-name foreign key (column-commalist) references table-name (column-commalist)`
- 刪除限制條件：
`alter table table-name drop constraint constraint-name`
- 注意：XML 資料型態的屬性不可以宣告為主鍵 (Primary Key)、外來鍵 (Foreign Key)，或 Unique

常用資料型態

分類	資料型態		說明	備註	
精確數值	Bigint		存放數值資料	固定佔 8 個位元組	
	Int		存放數值資料	佔 4 個位元組	
	Smallint		存放數值資料	佔 2 個位元組	
	Tinyint		存放數值資料	佔 1 個位元組	
	Bit		有 1、0 或 NULL 幾種值。 字串 True 和 False 可以轉換為 bit 值	True 轉換為 1，False 轉換為 0。	
	decimal[(p[, s])] numeric[(p[, s])]		存放固定精準度的數值資料	佔位元組視精準度 (p 與 s 的值) 而定	
	Money		存放代表金錢數值的資料	佔 8 個位元組	
	Smallmoney		存放代表金錢數值的資料	佔 4 個位元組	
近似值	Float		存放具小數點的數值資料	佔 8 個位元組	
	Real		存放具小數點的數值資料	佔 4 個位元組	
日期	Datetime	Date/Time	存放日期及時間資料	佔 8 個位元組	佔 3~5 個位元組
	Datetime2		datetime2 結合了以 24 小時制為基礎的當日時間，為 datetime 類型的延伸模組。	較大的日期範圍、較大的有效位數。	
	datetimeoffset		記錄與時區有關的時間資料		
	smalldatetime		存放日期及時間資料	佔 4 個位元組	
字元字串	char(n)		存放字串資料，最多 n 個字元	固定佔 n 位元組	
	varchar(n max)		存放變動長度字串資料，最多 n 個字元或 max = 2 ³¹ -1	如所需佔的位元組數目	
	Text		存放本文資料	如：一篇文章	
萬國碼字串	nchar(n)		存放 Unicode 字串資料，最多 n 個字元	固定佔 2*n 位元組	
	nvarchar(n max)		存放變動長度的 Unicode 字串，最多 n 個字元或 max = 2 ³¹ -1	佔所需的位元組數目 * 2	
	Ntext		存放 Unicode 本文資料	如：一篇文章	
二進位字串	Binary[(n)]		固定長度的二進位資料	n 最大可達 8000	
	varbinary(n max)		可變長度的二進位資料，最多 n 個位元組或 max = 2 ³¹ -1。	儲存大小是資料實際長度再加 2 位元組	
	Image		存放影像資料	如：bitmap, jpg	

其它 SQL Server 2008 資料型態

分類	資料型態	說 明	備 註
空間資料 型態	(Spatial data type)	可結合地理資訊，搭配空間階層式索引結構 (Spatial Index)，存取地理資料。	需要較大的篇幅說明，詳細請參考 SQL Server 2008 使用手冊的說明
Hierarchyid	Hierarchyid	儲存有階層性 (Hierarchical) 的資料， 便利維護樹狀結構	詳細請參考 SQL Server 2008 使用手冊 的說明
其他類型	Cursor	不能用於 Create Table 中宣告資料型態	用法請參考 7.7.2 節
	sql_variant	可存放，除了 text, ntext, timestamp 及 sql_variant 之外，的各種資料型態	佔記憶空間視情況而定
	Table	用於暫存的表格宣告	佔記憶空間視情況而定
	Timestamp	在資料庫自動產生唯一的 Binary 數字	佔 8 位元組
	uniqueidentifier	確保值組可在多份關聯表中唯一識別	透過 newid() 產生值
	xml [(xml_schema)]	存放符合 xml_schema 規範的 xml 資料	配合 query()、value()、exist()、modify()、 nodes() 等方法進行資料萃取

產生 BOB 資料庫綱要的指令

產生 Bookstores 的綱要	產生 Orders 的綱要	產生 Books 的綱要
<pre>create table Bookstores (<i>no</i> int not null, <i>name</i> char(20), <i>rank</i> tinyint, <i>city</i> char(16) [, primary key (<i>no</i>)]);</pre>	<pre>create table Orders (<i>no</i> int not null, <i>id</i> int not null, <i>quantity</i> int [, primary key (<i>no</i>, <i>id</i>), foreign key (<i>no</i>) references Bookstores on update cascade on delete cascade, foreign key (<i>id</i>) references Books on update cascade on delete cascade, check (quantity > 0 and quantity < 5001)])</pre>	<pre>create table Books (<i>id</i> int not null, <i>bookname</i> char(50), <i>author</i> char(30), <i>price</i> smallint, <i>publisher</i> char(20) [, primary key (<i>id</i>)]);</pre>

Collation 的設定

- 想知道系統中有多少 Collation，還可以使用指令來查詢：
`select * from ::fn_helpcollations()`
- 假設：書籍作者要以中文筆劃順序來排序；英文字的大、小寫字母要視為不同；碰到抑、重音符號則予以忽略。那麼就宣告：
 - `author char(30) Collate Chinese_Taiwan_Stroke_CS_AI`
 - 表示使用繁體中文字的筆劃順序來排序；
 - 若用 `author char(30) Collate Chinese_Taiwan_Bopomofo_CS_AI` 則表示以ㄅㄆㄇㄉ注音來排序
 - CS 表示 Case Sensitive 的縮寫，大、小寫字母視為不同；
 - AI 是 Accent Insensitive 的縮寫，表示忽略抑、重音符號
- 注意：具有 XML 資料型態的屬性是不可以配合 Collate 關鍵字宣告進行大小寫區別、或全形/半形的區別等，因為XML本身有其內部的Encoding方式。

Alter Table—更改綱要

- 加入一個新的欄位：
`alter table base-table-name add column-name data-type null`
- 刪除一個欄位：
`alter table base-table-name drop column column-name`
- 定義一個已存在欄位之內定值：
`alter table base-table-name add constraint constraint-name default constant-expression for column-name`
- 加入一條新的檢查條件到該關聯表上：
`alter table base-table-name add constraint constraint-name check (expression)`，*expression* 所算出來的結果必須是一個布林代數值 (True/False)
- 改變現有欄位的資料型態：
`alter table table-name alter column column-name new_data_type`
- 將現有的欄位名稱改名：用 `sp_rename` 可以達成，例如：
`sp_rename 'Books.bookname', 'name', 'column'`



Alter Table—更改綱要 (續)

- alter table *base-table-name*
add *columns-name data-type* null
- 例如：要在 Books 關聯表上新加一個 “打折” 欄位 discount
alter table **Books** add discount real null
- 要注意到：除非原本的表格內容是空的，否則新增欄位是不可以用 not null 的 (Why ?)
- 新增欄位當然也不能是主鍵的一部份 (Why ?)



Alter Table—更改綱要 (續)

- 不論「主鍵」(primary key)、「外來鍵」(foreign key)、「內定值」(default)或是「檢查條件」(check)等，在 SQL Server 上看來都是一種限制條件。
- 在 SQL Server 2008 版的綱要修正，幾乎都已經物件導向化了，只要使用 SQL Server Management Studio，在進入資料庫後於表格上按滑鼠右鍵，點選 [修改(Y)] 即可更改。



Drop/Truncate Table

- 刪除一個基底關聯表會連定義於其上的索引、視界等相關物件也一併刪除
drop table *base-table-name*
- 也可以只刪去關聯表內容，並保留其綱要以及定義於其上的索引、視界等相關物件
truncate table *base-table-name*

視界的 DDL

- create view *view-name*
[(*columns-name* [, *column-name*], ...)]
as *SQL-subquery* (DML 將在稍後討論)
- drop view *view-name*

Bookstores

<i>no</i>	<i>name</i>	<i>rank</i>	<i>city</i>
1	巨蟹書局	20	臺北市
2	射手書局	10	高雄市
3	水瓶書店	30	新竹市
4	天秤書局	20	臺中市
5	獅子書局	30	臺南市

產生視界
→

Odd_No_Bookstores

<i>no</i>	<i>name</i>	<i>rank</i>	<i>city</i>
1	巨蟹書局	20	臺北市
3	水瓶書店	30	新竹市
5	獅子書局	30	臺南市



索引的 DDL

- 由於 SQL Server 2008 所增加的 XML 資料型態，其索引方面的 DDL 不同於一般屬性的，因此，我們以 7.2.3 節、7.2.4 節分別來介紹：一般屬性索引的 DDL 與 XML 屬性索引的 DDL。

一般屬性索引的 DDL

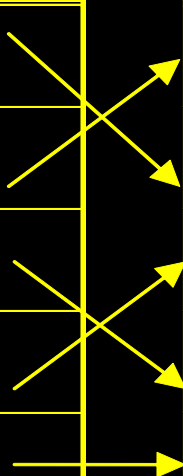
- Primary index : 建立在主鍵上的索引
- Secondary index 如下例

rank 索引

rank	指標
10	
20	
20	
30	
30	

Bookstores

no	name	rank	city
1	巨蟹書局	20	臺北市
2	射手書局	10	高雄市
3	水瓶書店	30	新竹市
4	天秤書局	20	臺中市
5	獅子書局	30	臺南市



非密集索引 (Nondense index)

no 索引

Bookstores

no	指標
1	
3	
5	

no	name	rank	city
1	巨蟹書局	20	臺北市
2	射手書局	10	高雄市
3	水瓶書店	30	新竹市
4	天秤書局	20	臺中市
5	獅子書局	30	臺南市

放在
第一頁

放在
第二頁

放在
第三頁

- 一個關聯表的非密集索引必須與叢聚索引配合, 所以最多只能有一個
- 優點：空間節省。缺點：無法由索引測試是否存在



Create Index—建立索引

- create [unique] index *index-name* on *base-table-name* (*columns-name* [*order*], [, *column-name* [*order*]]...) [*cluster*]

← 加了 cluster 表示要建成「叢聚索引」

- 若要刪除索引，則使用

drop index *table-name.index-name*

「索引」主要到次要的順序

- *order* 若不是「asc」(遞增) 就是「desc」(遞減)，若未註明，則預設為 asc。
- create index *Xrn* on Bookstores (*rank*, *no* desc)

<i>Xrn</i>			Bookstores			
<i>rank</i>	<i>no</i>	指標	<u><i>no</i></u>	<i>name</i>	<i>rank</i>	<i>city</i>
10	2		1	巨蟹書局	20	臺北市
20	4		2	射手書局	10	高雄市
20	1		3	水瓶書店	30	新竹市
30	5		4	天秤書局	20	臺中市
30	3		5	獅子書局	30	臺南市

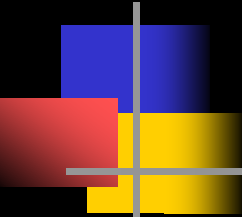


unique：表示關聯表中不允許有兩筆值組具有相同的索引值

- create **unique** index X_n on Bookstores (*no*)
意義就像宣告：被索引的屬性集為「候選鍵」(Candidate Key)。
- 在建立索引時，若該關聯表的內容包含有**重複**的索引屬性值，那麼 create unique index 就會失敗！

XML 屬性索引的 DDL

- XML 資料型態的屬性可以建立「索引」(Index)，但是作用不在加速 SQL 運算，而在加速 XML 的函數或方法 (Method)。
- 不可以是任意組合而成的複合屬性 (Composite Attribute)
- 假設 XMLBooks 關聯表定義如下：
create table XMLBooks(
 id int primary key, *bookname* char(50) not null,
 author xml)
- 對 author 建立「XML主索引」(Primary XML Index)：
create primary XML index XML_IX_author on XMLBooks(*author*)
- 建立「XML副索引」(Secondary XML Index) 加速查詢的速度：
create XML index XML_2nd_IX_author on XMLBooks(*author*)
using XML index XML_IX_author for [*path* | *value* | *property*]



Drop XML Index—刪除 XML 索引

- 將已經存在的 XML 索引從資料庫中刪除：
`drop index XML_2nd_IX_author ON dbo.XMLBooks`
`drop index XML_IX_author ON dbo.XMLBooks`
- 副索引 `XML_2nd_IX_author` 是依附在主索引 `XML_IX_author` 之上
- 如果只是刪除主索引，那麼其實也會將依附其上的副索引一併刪除。



安全方面的 DDL

- 系統在安全機制的規劃上，主要考慮兩點：
 - 辨識主體的身份 (Authentication)：除了以密碼來驗證之外，還可以透過強制執行密碼原則 (Password Policies) 與 Windows 的憑證名稱 (Credential) 來存取外部檔案或網路上的分享，提高安全性。
 - 權限授與 (Authorization)：當主體經過身份驗證後，系統會透過所賦予之權限，決定該主體可以在安全實體 (Securables) 上執行何種運算。

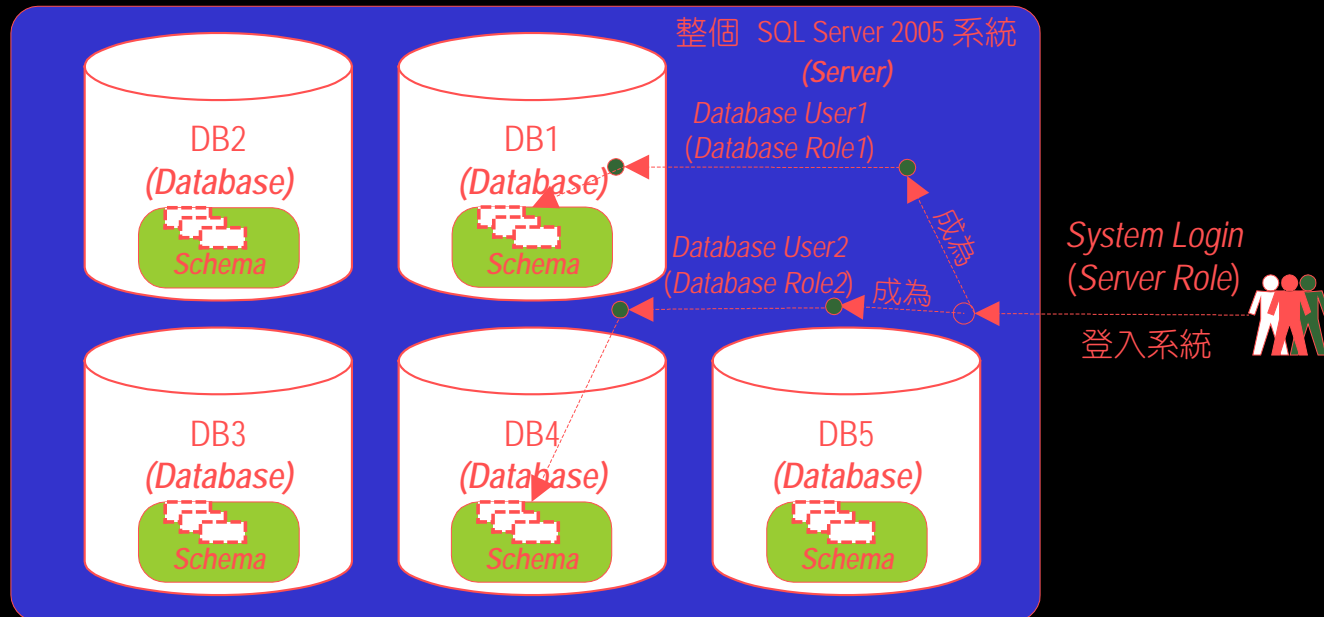
主體的 DDL

- 要使用 SQL Server 2008 資料庫，必須先取得一個登入帳號 (Login)：

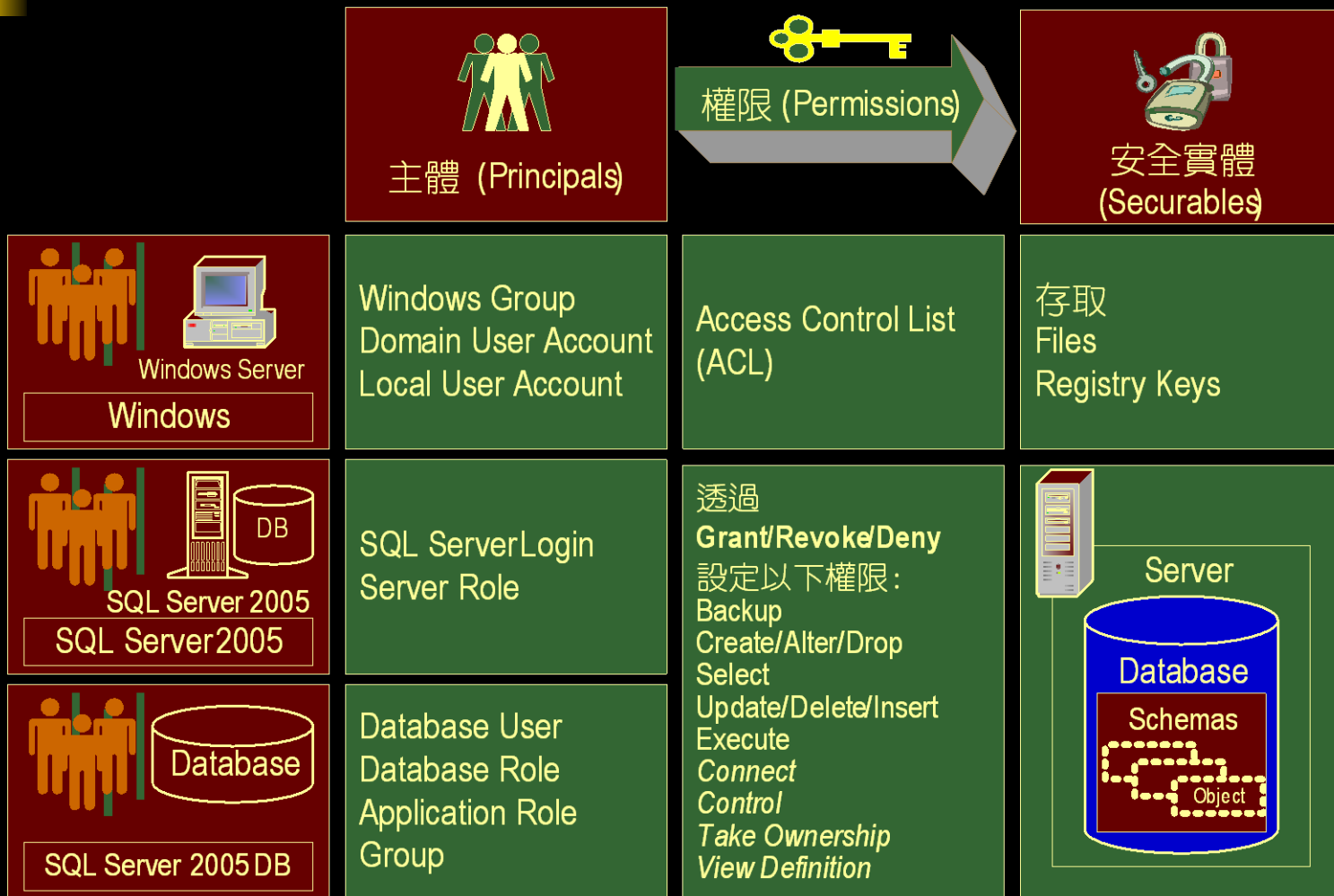
Use BOB

go

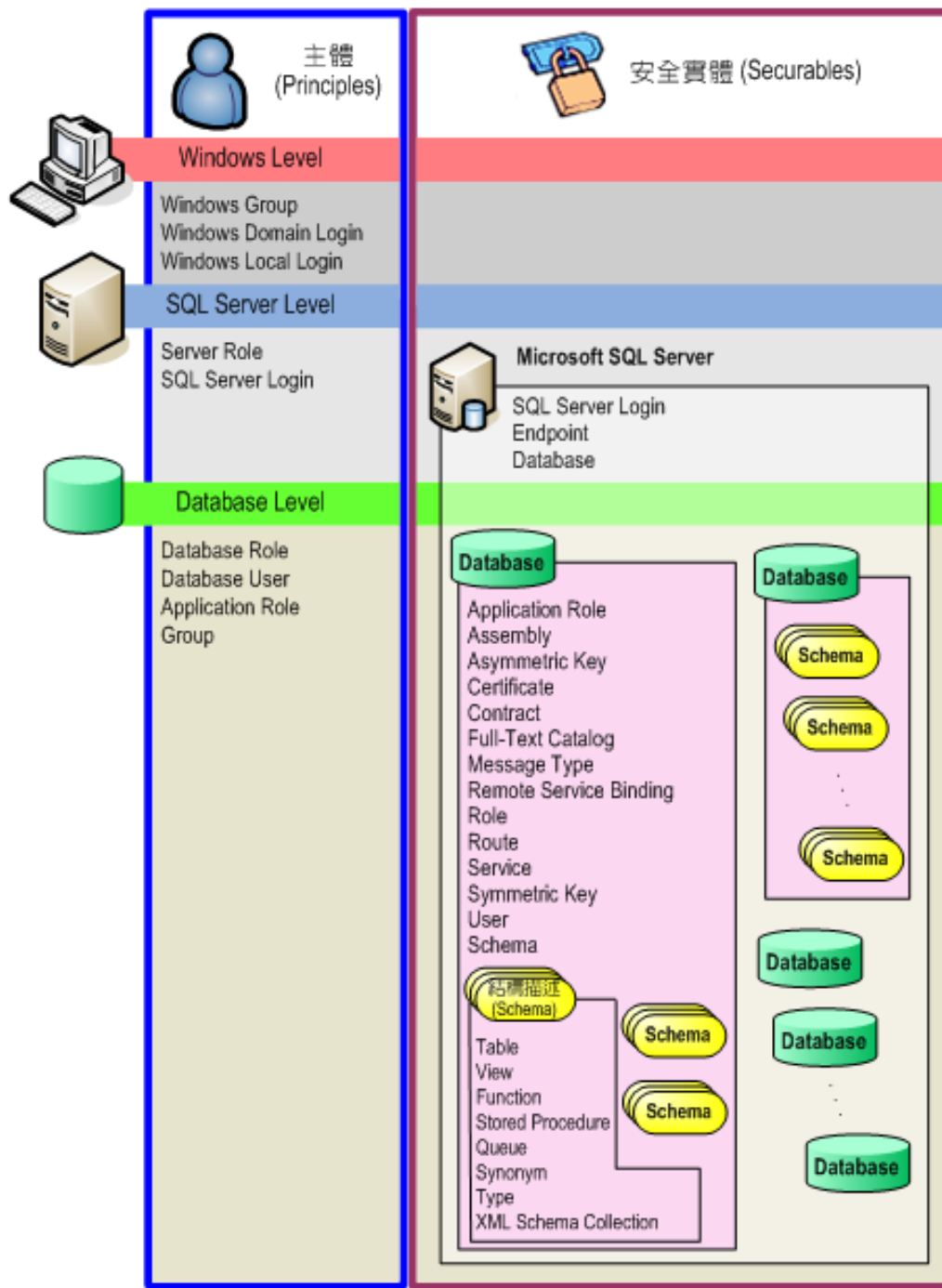
Create login *frank* with password = 'K@0hs1un91stUn1v'



主體的 DDL



主體方面的 DDL





搭配 Windows Server 2008 可更嚴謹

- 密碼驗證上不會將密碼送上網路；
- 可以設定成要求第一次登入時即更改密碼 (Must_Change)；
- 可以強制使用者選用不易猜測的密碼原則 (Check_Policy)
- 監督使用者在某個使用期間過後，就要求更換密碼 (Check_expiration)，
- 例如，前述的登入帳號建立指令可以改成：
Create login *frank* with password = 'K@0hs1un91stUn1v'
Must_Change, Default_database = *BOB*, Check_expiration = on,
Check_policy = on



更改登入帳號

- 更改登入帳號名稱、密碼、Check_expiration/Check_policy，或預設資料庫的話，則使用 alter login：

```
alter login frank with password = 'Ka0hs1un91stUn1v' -- 更改密碼
alter login frank with name = john -- 更改 frank 帳戶名稱為 john
alter login john with name = frank -- 將 john 帳戶名稱改回 frank
alter login frank with check_expiration = off -- 關閉 check_expiration
alter login frank with check_policy = off -- 關閉 check_policy
```



伺服器角色 (Server Role)

- bulkadmin : 可進行大量資料匯入與匯出的管理動作
- dbcreator : 具有自行建立資料庫的權限 (可授予一般使用者)
- diskadmin : 可以對資料庫所存放的硬碟檔案空間進行管理
- processadmin : 可以終止執行中的程序
- securityadmin : 具有管理登入及其屬性的權限。
- serveradmin : 可以變更整個伺服器的組態選項 (Configuration) 與停止伺服器運轉
- setupadmin : 可以加入和移除分散式資料庫運作環境中的 SQL Server 連結伺服器 (Linked Server)
- sysadmin : 可以執行 Database Engine 中的所有活動
- 越下方的權限越大



資料庫角色 (Database Role)

- **db_accessadmin** : 可新增/移除 Windows 登入、Windows 群組以及 SQL Server 登入成為資料庫使用者，並擁有某些存取權
- **db_backupoperator** : 具有備份資料庫的權限
- **db_datareader** : 可以讀取資料表的所有資料
- **db_datawriter** : 可以新增、刪除或變更資料表中的資料
- **db_ddladmin** : 可在資料庫中執行資料定義語言 (DDL)
- **db_denydatareader** : 不能讀取資料庫中任何使用者資料表的資料
- **db_denydatawriter** : 不能新增、修改或刪除資料庫中任何使用者資料表的資料
- **db_owner** : 簡稱dbo，這類角色的成員可以在資料庫上執行所有的組態和維護活動
- **db_securityadmin** : 可管理資料庫角色成員資格與修改其使用權限
- **public** : 每個資料庫使用者都屬於 public 角色

將登入帳號對應到結構描述 (Schema)

- 系統除了將登入帳號對應到某個資料庫使用者之外，還會對應到其所屬的預設 Schema，
- 如果設定時沒有指定 Schema，系統會指定 dbo 為預設 Schema

```
Create user user_name [ { for | from }  
    { login login_name | certificate cert_name  
      | asymmetric key asym_key_name }  
    | without login ]  
[ with default_schema = schema_name ]
```



將 Schema 授權給資料庫使用者

- 可將 Schema 授權由某個 *database_user*，或某個 *database_role* 擁有
Create Schema *schema_name* Authorization {*database_user* | *database_role*}
- E.g.,
Use *BOB*
go
Create User *frank* for login *frank* with default_schema = *Chapter7*
go
exec sp_addrolemember 'db_owner', 'frank' -- 將 frank 加入 dbo 角色
go
Create Schema *Chapter7* Authorization *frank* -- frank 擁有了 Chapter7
go



新建表格將會放在預設 Schema 中

- 上頁指令完成後，若接著下：
`Setuser 'frank' -- 改變資料庫使用者身份為 frank`
`go`
`Create table new_orders(no int, id int, quantity int,`
`primary key (no, id))`
`go`
`Setuser -- 將資料庫使用者身份回復原來身份`
`go`
- 則 `new_orders` 表格將會放在 Chapter7 Schema 下



沒有指定預設 Schema 的情況

- 在建立資料庫使用者 frank 的時候，若未指定 default_schema 為 Chapter7 的話，則其預設 Schema 為 dbo
- 但如果 frank 也不屬於 dbo 成員的話 (也就是假設指令
`exec sp_addrolemember 'db_owner', 'frank'`
沒有被執行的話)，那會出現錯誤訊息！
- 因為 frank 沒有權限在 dbo 這個 Schema 上建立表格。

上述情況的解決方法

- 如果 frank 不屬於 dbo 成員的話，而且 Chapter7 也不是其預設 Schema 的話，那麼原先的

Create table new_orders(no int, id int, quantity int,
primary key (no, id))

指令只要換成

改成這樣即可！

Create table Chapter7.new_orders(no int, id int,
quantity int, primary key (no, id))


就可以了



注意：Create Role 不能指定 Default_Schema

- 原因是：因為一個資料庫角色中會包含許多的資料庫使用者，
- 每一個使用者都可能指定不同default_schema
- 因此在 create role 指令中指定 default_schema 就沒什麼意義了。
- 指令範例：
`Create role general_user -- 也可加上 [authorization {db_user | db_role}]`
`exec sp_addrolemember 'general_user', 'frank'`

學習安全設定指令的最快方法

- 若不用指令完成安全設定動作，也可以使用 SQL Server Management Studio 從 [資料庫 | BOB | 安全性 | 使用者] 上按滑鼠右鍵，選擇 [新增使用者(N)...] 來加入，
- 而且也可以請 SQL Server 2008 幫我們產生相對指令。
- 做法是做完各種勾選動作後，按下  指令碼，系統就會將指令產生在 SQL Server Management Studio 的右半邊視窗中，對於學習上的助益實在很大

請 SQL Server 2008 產生相對指令

點此按鈕後
會產生下列
指令碼於
右半視窗中

```
USE [BOB]
GO
CREATE USER [frank] FOR LOGIN [frank]
GO
USE [BOB]
GO
EXEC sp_addrolemember N'db_owner', N'frank'
GO
```

資料庫使用者 - 新增

選取頁面

- 一般
- 安全性實體
- 擴充屬性

指令碼 說明

使用者名稱(U): frank

登入名稱(L): frank

憑證名稱(C):

索引鍵名稱(K):

不使用登入(W)

預設結構描述(D): dbo

這個使用者擁有的結構描述(O):

擁有的結構描述

連接

伺服器: KIDLAB

連接: KIDLAB\Administrator

檢視連接屬性

進度

指令碼已順利完成。

資料庫角色成員資格(M):

角色成員

- ☐ db_datareader
- ☐ db_datawriter
- ☐ db_ddladmin
- ☐ db_denydatareader
- ☐ db_denydatawriter
- ☒ db_owner
- ☐ db_securityadmin

確定 取消



也可以只授權某指令的執行權限

- SQL Server 2008 不僅可以將資料庫物件的使用權限授與資料庫使用者或角色，也可以將指令授權出去 (例如：下面的 Shutdown 指令執行權限)，E.g.,

```
use master
```

```
go
```

```
GRANT ADMINISTER BULK OPERATIONS TO [frank]
```

```
GRANT ALTER ANY LOGIN TO [frank]
```

```
GRANT CONNECT SQL TO [frank]
```

```
GRANT CREATE ANY DATABASE TO [frank]
```

```
GRANT CREATE ENDPOINT TO [frank]
```

```
GRANT SHUTDOWN TO [frank]
```

```
GRANT VIEW ANY DATABASE TO [frank]
```

```
go
```



登入帳號的狀態設定

- 連接到 Database Engine 的權限：可以是“授與”或“拒絕”。
 - 登入 (Login)：可以是“已啟用”或“已停用”。
- ```
alter login frank disable -- 停用登入帳號frank
alter login frank enable -- 啟用登入帳號 frank
go
```
- 也可以解除鎖定登入帳號
- ```
alter login frank with password = 'Ka0hs1un91stUn1v' unlock  
go
```



刪除主體的做法

- 注意：因為 Login、Role、User、Schema 之間有相依性，所以要記住下列原則，並依序做完才行：
 1. Schema 中若含有資料庫物件，那必須先將裡面的物件**轉移**到其他 Schema，或是刪除這些物件。
`alter schema Chapter6 transfer Chapter7.new_orders go` -- 將 Chapter7.new_orders 轉到 Chapter6 中
 2. 資料庫使用者 (Database User) 若擁有 Schema，那必須先將 Schema 擁有權轉給另一位使用者，否則無法刪除原使用者。
`alter authorization on schema::Chapter7 to dbo go` -- 將 Chapter7 轉給 dbo 角色



刪除主體的做法 (續)

3. 資料庫使用者 (Database User) 若擁有資料庫角色 (Database Role) 的話，那麼也必須先將這些角色擁有權轉給另一位使用者，否則無法刪除。

```
alter authorization on role::general_user to frank  
go -- 將 general_user 轉給 frank
```

4. 自行建立的資料庫角色 (Database Role) 若含有資料庫使用者，則必須先將它們從此角色中移除，否則無法刪除原角色：

```
exec sp_droprolemember 'database_role', 'database_user'  
go
```




刪除主體的做法 (續)

5. 若想刪除登入帳號 (Login)，那麼系統會提出警告訊息：
 - 要求你自行將該登入帳號對應到的所有資料庫使用者帳號刪除，
 - 同時先將這些資料庫使用者所擁有的 Schema 先行轉給其他使用者擁有。
6. 如果上述的幾點步驟都已經照順序完成的話，就可以不予理會此警告訊息。

「最小使用權原則」

(The Principle of Least Privilege)

- 主體還包含了「應用程式角色」(Application Role)：
Create application role *application_role_name* with password = '*password*' [, default_schema = *schema_name*]
- 專門提供應用程式呼叫某個 Schema 中的預儲程序，或使用裡面的資料庫物件之用，
- 目的是要達成「最小使用權原則」(The Principle of Least Privilege) 之理念。
- 它與資料庫角色 (Database Role) 的差異：
 - 應用程式角色並不含任何資料庫使用者，而且預設是非使用中狀態。
 - 但是資料庫角色中會包含許多的使用者。



應用程式角色 Application Role

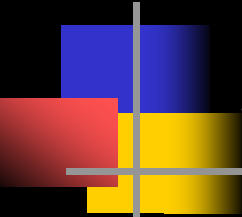
- 讓應用程式以其獨有，且類似於資料庫使用者的權限來執行 SQL 指令。
- 對於特定應用程式，系統只允許透過應用程式角色來連線，以存取指定的資料。
- 應用程式角色還必須使用 `sp_setapprole` 並提供密碼來啟動。
- 應用程式角色是資料庫層級的主體，它們只能透過各個資料庫中授與 guest 使用者帳戶的權限來存取資料庫內容。



使用應用程式角色 Application Role

■ 步驟：

- 使用者 A 執行用戶端應用程式。
- 應用程式以使用者身分連接到 SQL Server 2008 執行個體
- 應用程式用自己的密碼執行 sp_setapprole 預儲程序，其中含有 @Cookie Output 選項，可在啟動應用程式角色前建立包含內容資訊的 Cookie。
- 若應用程式角色名稱與密碼驗證通過，則 SQL Server 2008 便啟動該應用程式角色
- 接下來，使用者 A 本身在資料庫中的權限將失效，SQL Server 2008 從此以應用程式角色的權限來存取資料庫
- 後續可利用 sp_unsetapprole @cookies 來中止該應用程式角色，並透過上述 Cookie 將工作階段還原為原始內容。



安全實體的 DDL

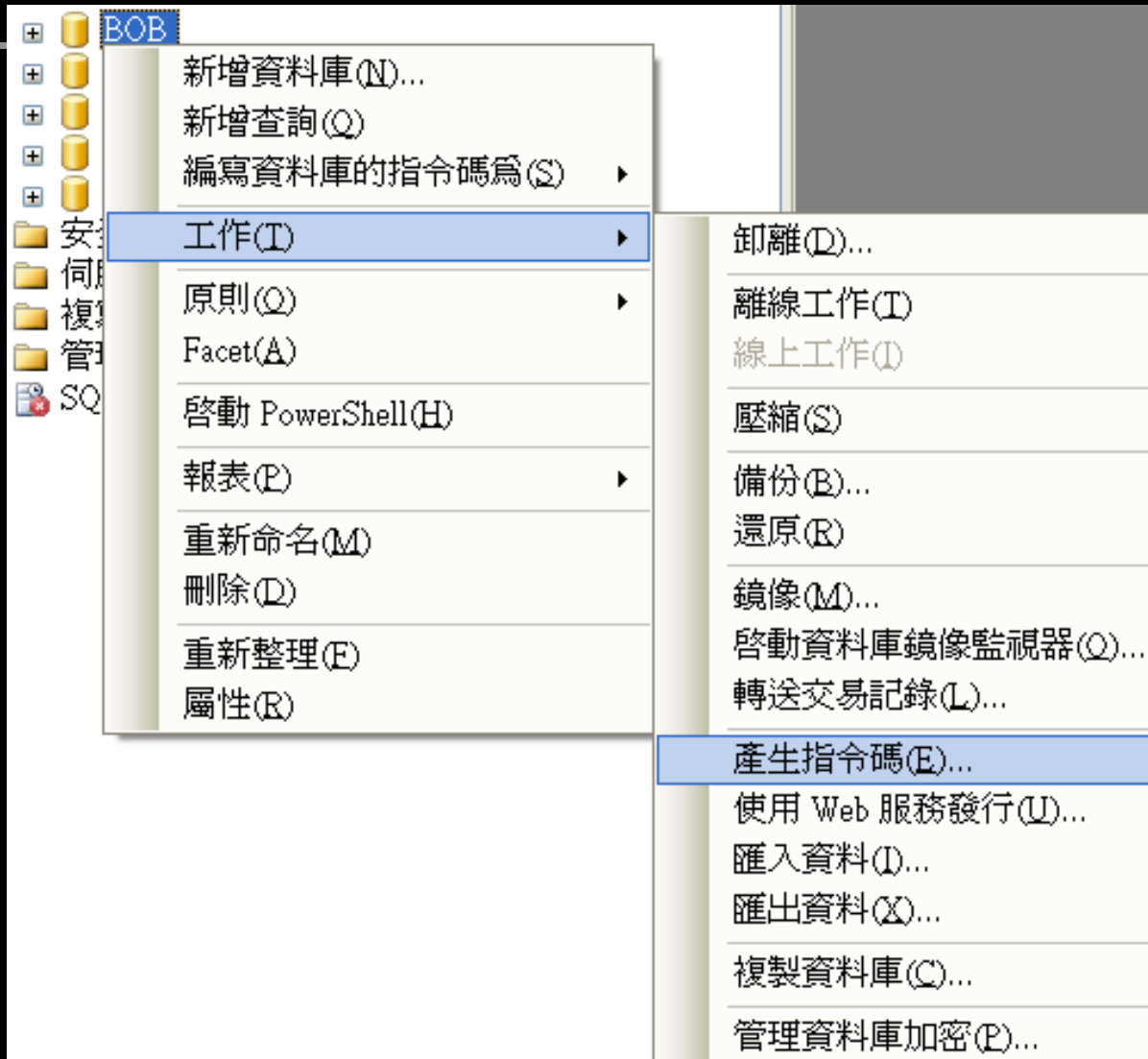
- 資料庫層級的安全實體有四個層層包覆的階層結構(如圖 3.2 與 3.4 所示)：
 - 伺服器層次 (Server Scope)：指的是整個 SQL Server 2008 的安裝與設定
 - 資料庫層次 (Database Scope)：資料庫方面的 DDL 已經在 7.2.1 節說明
 - 結構描述層次 (Schema Scope)：在第三章已經說明
 - 物件層次 (Object Scope)：包含了所有以 DDL 定義的常用資料庫物件
 - 以上引發了：物件在 SQL 中常常須以 4 部命名方式 (Four-Part Naming Convention) 來指定
Server.Database.Schema.Object 的寫法

「4 部命名方式」

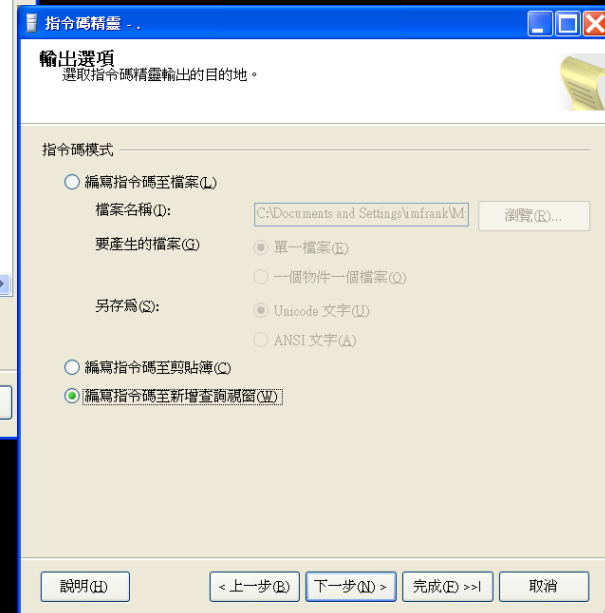
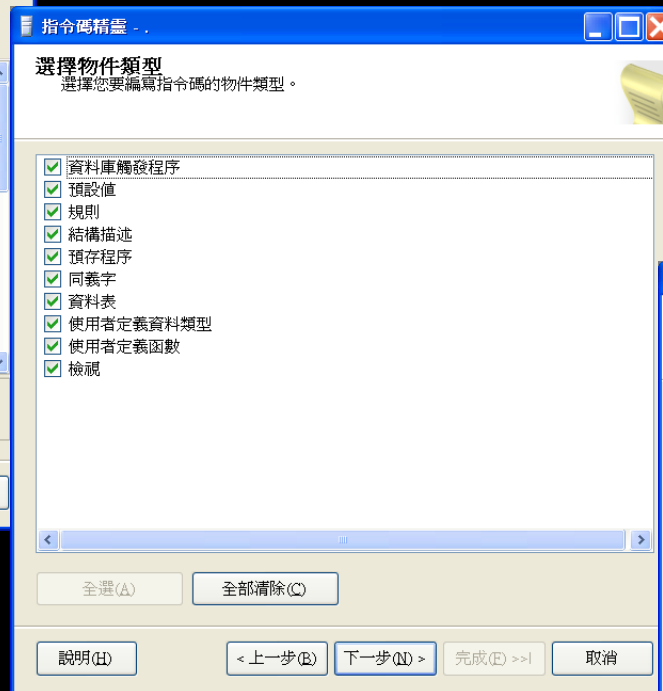
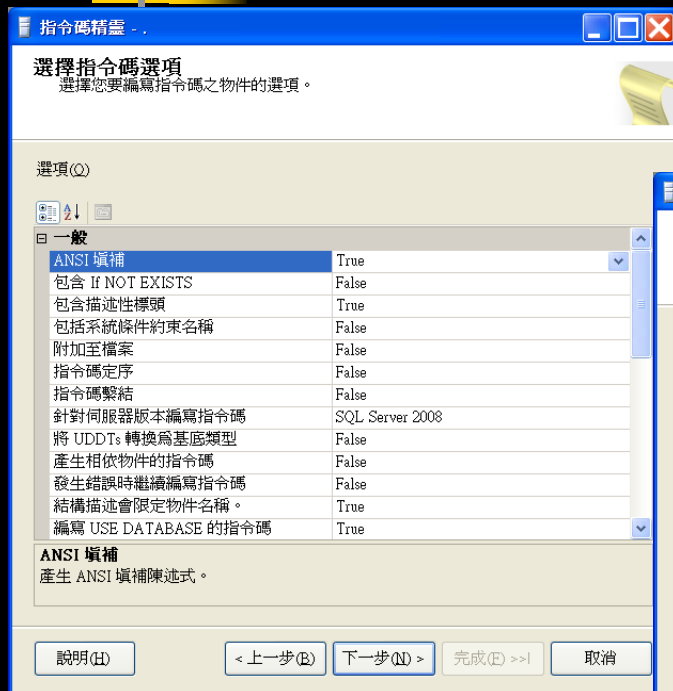
(Four-Part Naming Convention)

- 在 *Server* 與 *Database* 都明確的情況下允許用簡寫 *Schema.Object*，甚至只寫 *Object* 也可
- 系統在遇上 *Object* 縮寫時，會以下順序尋找物件：
 - 先找尋該使用者的預設 *Schema* 下的 *Object*，若找不到則接著往下找
 - 找尋 *dbo* 這個 *Schema* 下的 *Object*，若找不到則接著往下找
 - 找尋 *sys* 這個 *Schema* 下的 *Object*，若找不到則顯示錯誤訊息

在 SQL Server 2008 上自動產生 DDL



在 SQL Server 2008 上自動產生 DDL



資料處理語言 (DML)

- *[with common-table-expression]* -- 將在 8.4 節詳細討論

Select [distinct] *a-list-of-attribute-names* (可運算式)

[Into new_relation_name] (有的系統放在最後面)

From *a-list-of-relation-names* (可跟著 *alias name*)

[Where a-true/false-condition]

[Group by [all] group-by-expression [with {cube | rollup}]

[Having a-true/false group-condition]] -- 一定要有 Group by

[Order by a-list-of-attribute-name]

(查詢結果可用 *[Into 子句]* 建立另一關聯表)

Select 子句

- `select * from Books` (* 表示所有欄位)
- `select bookname, author, price, publisher from Books`

Books

<u>id</u>	<i>bookname</i>	<i>author</i>	<i>price</i>	<i>publisher</i>
1	三國演義	羅貫中	120	古文出版社
2	水滸傳	施耐庵	170	中庸出版社
3	紅樓夢	曹雪芹	170	春秋出版社
4	西遊記	吳承恩	140	聊齋出版社
5	水經注	酈道元	120	易經出版社
6	道德經	老子	190	大唐出版社

Select 子句 (續)

- select **distinct** price from Books -- 刪除重覆部份
- select bookname, '打八折後的價格 =', price * 0.8 as new_price from Books

<i>price</i>
120
170
140
190

表示所有的書籍
總共有四種不同的
價格

<i>bookname</i>		<i>new_price</i>
三國演義	打八折後的價格 =	96.0
水滸傳	打八折後的價格 =	136.0
紅樓夢	打八折後的價格 =	136.0
西遊記	打八折後的價格 =	112.0
水經注	打八折後的價格 =	96.0
道德經	打八折後的價格 =	152.0



From 子句與合併運算

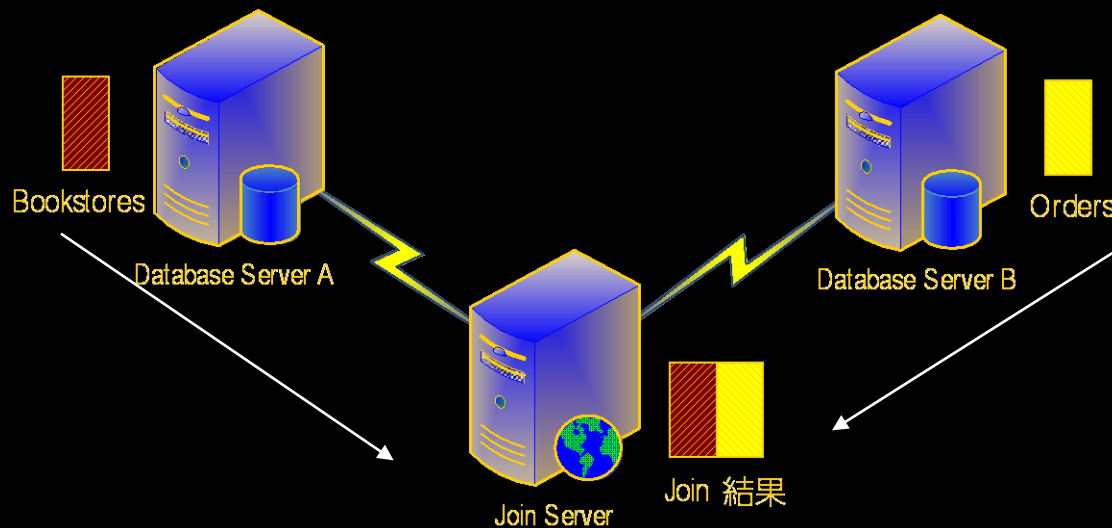
- $A \bowtie_p B = \sigma_p(A \times B)$
- From 子句後的所有關聯表，要做乘積運算
- Where 子句後的條件可以是
 - 選擇條件—有一邊是常數 (e.g. price > 100) (選擇)
 - 合併條件—兩邊都是屬性 (e.g. B.no = O.no) (合併)
- ```
select Bookstores.*, Orders.*
from Bookstores, Orders -- 做乘積運算
where Bookstores.no = Orders.no -- 合併條件
```

# 從不同資料庫進行合併運算

- 假設 Bookstores 在資料庫 A，Orders 在資料庫 B，則兩種 SQL 寫法如下：

```
select Bookstores.*, Orders.*
from A..Bookstores, B..Orders
where Bookstores.no = Orders.no
```

```
select Bookstores.*, Orders.*
from A.dbo.Bookstores, B.dbo.Orders
where Bookstores.no = Orders.no
```



# 自然合併運算 (Natural Join)

- 自然合併—把重複的欄位資料刪除

```
select Bookstores.no, name, rank, city, id, quantity
from Bookstores, Orders
where Bookstores.no = Orders.no
```

| <i>no</i> | <i>name</i> | <i>rank</i> | <i>city</i> | <i>no</i> | <i>id</i> | <i>quantity</i> |
|-----------|-------------|-------------|-------------|-----------|-----------|-----------------|
| 1         | 巨蟹書局        | 20          | 臺北市         | 1         | 1         | 30              |
| 1         | 巨蟹書局        | 20          | 臺北市         | 1         | 2         | 20              |
| 1         | 巨蟹書局        | 20          | 臺北市         | 1         | 3         | 40              |
| ⋮         | ⋮           | ⋮           | ⋮           | ⋮         | ⋮         | ⋮               |
| 4         | 天秤書局        | 20          | 臺中市         | 4         | 2         | 20              |
| 4         | 天秤書局        | 20          | 臺中市         | 4         | 4         | 30              |
| 4         | 天秤書局        | 20          | 臺中市         | 4         | 5         | 40              |



# 自身合併運算 (Self-Join)

- `select First.name, Second.name  
from Bookstores First, Bookstores Second  
where First.rank = Second.rank  
and First.no < Second.no` -- 過濾多餘的部份

| <i>First.name</i> | <i>Second.name</i> |
|-------------------|--------------------|
| 巨蟹書局              | 天秤書局               |
| 水瓶書局              | 獅子書局               |

# 自身合併運算 (續)

## 加入 $\text{First.no} < \text{Second.no}$ 的用意

Bookstores (First)

| no | name | rank | city |
|----|------|------|------|
| 1  | 巨蟹書局 | 20   | 臺北市  |
| 2  | 射手書局 | 10   | 高雄市  |
| 3  | 水瓶書店 | 30   | 新竹市  |
| 4  | 天秤書局 | 20   | 臺中市  |
| 5  | 獅子書局 | 30   | 臺南市  |

Bookstores (Second)

| no | name | rank | city |
|----|------|------|------|
| 1  | 巨蟹書局 | 20   | 臺北市  |
| 2  | 射手書局 | 10   | 高雄市  |
| 3  | 水瓶書店 | 30   | 新竹市  |
| 4  | 天秤書局 | 20   | 臺中市  |
| 5  | 獅子書局 | 30   | 臺南市  |

| First.name | Second.name | First.name | Second.name |
|------------|-------------|------------|-------------|
| 巨蟹書局       | 巨蟹書局        | 巨蟹書局       | 天秤書局        |
| 射手書局       | 射手書局        | 天秤書局       | 巨蟹書局        |
| 水瓶書店       | 水瓶書店        | 水瓶書店       | 獅子書局        |
| 天秤書局       | 天秤書局        | 獅子書局       | 水瓶書店        |
| 獅子書局       | 獅子書局        |            |             |

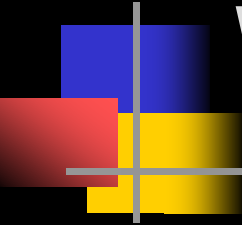
刪除  
多餘  
部份



# Where 子句—查詢條件

- 可以是邏輯運算式 (包含比較運算式)  
select no, name, rank, city  
from Bookstores  
where rank > 10 and no > 2

| <i>no</i> | <i>name</i> | <i>rank</i> | <i>city</i> |
|-----------|-------------|-------------|-------------|
| 3         | 水瓶書店        | 30          | 新竹市         |
| 4         | 天秤書局        | 20          | 臺中市         |
| 5         | 獅子書局        | 30          | 臺南市         |

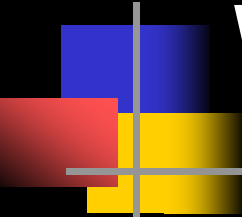


# Where 子句—查詢條件(續)

---

- 判斷欄位值是否 null，不可以用比較運算子!
- SQL Server 2008 雖然允許使用 = 來比較，但是其意義不同，所以我們建議還是用以下的語法

```
select *
from Bookstores
where city is null /* 不是 city = null */
```



# Where 子句—查詢條件 (續)

- 可以做部份吻合查詢  
select \*  
from Bookstores  
where city like '臺\_市'

| <i>no</i> | <i>name</i> | <i>rank</i> | <i>city</i> |
|-----------|-------------|-------------|-------------|
| 1         | 巨蟹書店        | 20          | 臺北市         |
| 4         | 天秤書局        | 20          | 臺中市         |
| 5         | 獅子書局        | 30          | 臺南市         |

# Where 子句—查詢條件 (續)

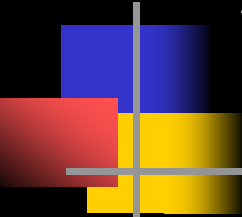
- 可以是複雜的混合運算式

```
select no, name, rank, city
```

```
from Bookstores
```

```
where not (rank >= 20 and city like '臺%市')
```

| <i>no</i> | <i>name</i> | <i>rank</i> | <i>city</i> |
|-----------|-------------|-------------|-------------|
| 2         | 射手書局        | 10          | 高雄市         |
| 3         | 水瓶書局        | 30          | 新竹市         |



# 簡易的字串處理技巧

---

- 在 SQL Server 2008 中，字串是用單引號 (') 括起來的。
- 如果字串中本身就有單引號的話，那只要將字串中的每個單引號都以連續兩個單引號來表示即可。
- 例如：'Let's go!' 所代表的便是「Let's go!」這個字串。

# Where 子句—查詢條件 (續)

可以使用集合運算

```
select no, name, rank, city
from Bookstores
where rank in (20, 30) and
city in ('臺北市', '臺中市', '新竹市')
```

| <i>no</i> | <i>name</i> | <i>rank</i> | <i>city</i> |
|-----------|-------------|-------------|-------------|
| 1         | 巨蟹書店        | 20          | 臺北市         |
| 3         | 水瓶書局        | 30          | 新竹市         |
| 4         | 天秤書局        | 20          | 臺中市         |

- 注意：具有 XML 資料型態的變數或屬性是不可以出現在Where 子句裡。

# 使用外部合併運算取代 Where 子句

- 做「等位合併」時，若合併欄位中含有 **null**，則合併欄位值為 **null** 的值組會被捨棄。
- 例如：以下表格存放研究所入學考的考生資料

Students

| <u>no</u> | name | class | zip | address      |
|-----------|------|-------|-----|--------------|
| 1         | 瑪莉   | A     | 100 | 中華路一段 x 號    |
| 2         | 志明   | B     | 104 | 中正路二段 x 號    |
| 3         | 阿倫   | A     | 105 | 中山北路一段 x 號   |
| 4         | 春嬌   | B     | —   | 桃園市民生路 x 號   |
| 5         | 大衛   | B     | —   | 台北市仁愛路三段 x 號 |

Zipdata

| <u>Zip</u> | district |
|------------|----------|
| 100        | 台北市中正區   |
| 103        | 台北市大同區   |
| 104        | 台北市中山區   |
| 105        | 台北市松山區   |
| 106        | 台北市大安區   |



# 狀況說明

---

- 4 號與 5 號考生沒有在報名表上填入通訊處的郵遞區號 (*zip* 欄位放 null)。
- 教務處聘請工讀生鍵入考生資料時，會明確將兩位考生地址資料中的詳細資料輸入到 *address* 欄位中。
- 這樣的資料組合，很容易出現錯誤情況：當最後統計完成績準備寄發成績單時，電腦系統可能會將兩個表格進行合併動作，如下頁所示：



# 做合併查詢時的差別

- select        Students.\*, *district*  
from           Students, Zipdata  
where          Students.zip = Zipdata.zip

| <i>no</i> | <i>name</i> | <i>class</i> | <i>address</i> | <i>zip</i> | <i>District</i> |
|-----------|-------------|--------------|----------------|------------|-----------------|
| 1         | 瑪莉          | A            | 中華路一段 x 號      | 100        | 台北市中正區          |
| 2         | 志明          | B            | 中正路二段 x 號      | 104        | 台北市中山區          |
| 3         | 阿倫          | A            | 中山北路一段 x 號     | 105        | 台北市松山區          |

- 4 號與 5 號考生的 *zip* 欄位放 null，會被捨棄，因此其成績單不會被列印出來。

# 解決方式: 使用 Left Outer Join

- 使用「左方外部合併」(Left Outer Join) 運算來取代 Where 中的等位合併 (將左方的關聯表未併成的部份撈回，右方表格欄位補 Null 值)

```
select no, name, class, address, Zipdata.zip, district
from Students left outer join Zipdata
on Students.zip = Zipdata.zip
```

| <u>no</u> | Name | class | address      | zip | district |
|-----------|------|-------|--------------|-----|----------|
| 1         | 瑪莉   | A     | 中華路一段 x 號    | 100 | 台北市中正區   |
| 2         | 志明   | B     | 中正路二段 x 號    | 104 | 台北市中山區   |
| 3         | 阿倫   | A     | 中山北路一段 x 號   | 105 | 台北市松山區   |
| 4         | 春嬌   | B     | 桃園市民生路 x 號   | —   | —        |
| 5         | 大衛   | B     | 台北市仁愛路三段 x 號 | —   | —        |

# 相關運算: Right Outer Join

- 使用「右方外部合併」(Right Outer Join) 運算的查詢結果 (將右方的關聯表未併成的部份撈回，左方表格欄位補 Null 值)

```
select no, name, class, address, Zipdata.zip, district
from Students right outer join Zipdata
on Students.zip = Zipdata.zip
```

| <u><i>no</i></u> | <i>Name</i> | <i>class</i> | <i>address</i> | <i>zip</i> | <i>district</i> |
|------------------|-------------|--------------|----------------|------------|-----------------|
| 1                | 瑪莉          | A            | 中華路一段 x 號      | 100        | 台北市中正區          |
| 2                | 志明          | B            | 中正路二段 x 號      | 104        | 台北市中山區          |
| 3                | 阿倫          | A            | 中山北路一段 x 號     | 105        | 台北市松山區          |
| —                | —           | —            | —              | 103        | 台北市大同區          |
| —                | —           | —            | —              | 106        | 台北市大安區          |

# 相關運算: Full Outer Join

- 使用「完整外部合併」(Full Outer Join) 運算的查詢結果 (將雙方的關聯表未併成的部份撈回，結果相當於 Left Outer Join 與 Right Outer Join 的聯集)

```
select no, name, class, address, Zipdata.zip, district
from Students Full outer join Zipdata
on Students.zip = Zipdata.zip
```

| <i>no</i> | <i>Name</i> | <i>class</i> | <i>address</i> | <i>zip</i> | <i>district</i> |
|-----------|-------------|--------------|----------------|------------|-----------------|
| 1         | 瑪莉          | A            | 中華路一段 x 號      | 100        | 台北市中正區          |
| 2         | 志明          | B            | 中正路二段 x 號      | 104        | 台北市中山區          |
| 3         | 阿倫          | A            | 中山北路一段 x 號     | 105        | 台北市松山區          |
| —         | —           | —            | —              | 103        | 台北市大同區          |
| —         | —           | —            | —              | 106        | 台北市大安區          |
| 4         | 春嬌          | B            | 桃園市民生路 x 號     | —          | —               |
| 5         | 大衛          | B            | 台北市仁愛路三段 x 號   | —          | —               |

# Order by—對結果做排序

- 可以指定主要排序欄位與次要排序欄位

```
select no, name, rank, city
from Bookstores
```

order by 3 desc, no asc (第三欄為主，第一欄為次)

| <u>no</u> | name | rank | city |
|-----------|------|------|------|
| 3         | 水瓶書店 | 30   | 新竹市  |
| 5         | 獅子書局 | 30   | 臺南市  |
| 1         | 巨蟹書局 | 20   | 臺北市  |
| 4         | 天秤書局 | 20   | 臺中市  |
| 2         | 射手書局 | 10   | 高雄市  |

- 注意：具有 XML 資料型態的變數或屬性不可直接使用 Order by 子句，除非先利用 Cast() 或 Convert() 函數轉換為字串資料。

# 聚合函數 (Aggregates)

- SQL有五個聚合函數：COUNT、SUM、AVG、MIN、MAX，查詢所得的結果為一統計值

- `select COUNT(*)  
from Bookstores`

- 答案是

|   |
|---|
|   |
| 5 |

Bookstores

| <i>no</i> | <i>name</i> | <i>rank</i> | <i>city</i> |
|-----------|-------------|-------------|-------------|
| 1         | 巨蟹書局        | 20          | 臺北市         |
| 2         | 射手書局        | 10          | 高雄市         |
| 3         | 水瓶書店        | 30          | 新竹市         |
| 4         | 天秤書局        | 20          | 臺中市         |
| 5         | 獅子書局        | 30          | 臺南市         |

# 聚合函數 (續)

- select COUNT(distinct no)  
from Orders
- select SUM(quantity)  
from Orders where no = 2
- select AVG(quantity)  
from Orders where no = 2
- select MAX(quantity)  
from Orders where no = 1
- select MIN(quantity)  
from Orders where no = 1

|   |
|---|
|   |
| 4 |

|    |
|----|
|    |
| 70 |

|    |
|----|
|    |
| 35 |

|    |
|----|
|    |
| 40 |

|    |
|----|
|    |
| 10 |

Orders

| no | id | quantity |
|----|----|----------|
| 1  | 1  | 30       |
| 1  | 2  | 20       |
| 1  | 3  | 40       |
| 1  | 4  | 20       |
| 1  | 5  | 10       |
| 1  | 6  | 10       |
| 2  | 1  | 30       |
| 2  | 2  | 40       |
| 3  | 2  | 20       |
| 4  | 2  | 20       |
| 4  | 4  | 30       |
| 4  | 5  | 40       |



# SQL Server 2008 進階的聚合函數

- 很適合用在輸出報表的分頁處理上：
  - **ROW\_NUMBER() OVER (order by *attr*) as *alias***：依照 *attr* 屬性排序，傳回每一值組的編號。
  - **RANK() OVER (order by *attr*) as *alias***：依照 *attr* 屬性排序後，傳回每一筆值組的排名，若遇有同名次的值組，則全部使用同一個排名順序，後續的排名要跳過同名次的值組數量。
  - **DENSE\_RANK() OVER (order by *attr*) as *alias***：依照 *attr* 屬性排序後，傳回每一筆值組的排名，若遇有同名次的值組，則全部使用同一個排名順序，但是接下來的排名不跳過同名次的值組數量
  - **NTILE(*n*) OVER (order by *attr*) as *alias***：依照 *attr* 屬性排序後，將所有值組分成 *n* 群，傳回每一筆值組的群組編號



# SQL Server 2008 進階的聚合函數

select id, bookname, price,  
ROW\_NUMBER() OVER (order by price) as 列編號,  
RANK() OVER (order by price) as 價格排名由小到大,  
DENSE\_RANK() OVER (order by price) as 緊密價格排名,  
NTILE(3) OVER (order by price) as 依排名分成 3 區段  
from Books

| <i>id</i> | <i>bookname</i> | <i>price</i> | 列編號 | 價格排名由小到大 | 緊密價格排名 | 依排名分成 3 區段 |
|-----------|-----------------|--------------|-----|----------|--------|------------|
| 1         | 三國演義            | 120          | 1   | 1        | 1      | 1          |
| 5         | 水經注             | 120          | 2   | 1        | 1      | 1          |
| 4         | 西遊記             | 140          | 3   | 3        | 2      | 2          |
| 2         | 水滸傳             | 170          | 4   | 4        | 3      | 2          |
| 3         | 紅樓夢             | 170          | 5   | 4        | 3      | 3          |
| 6         | 道德經             | 190          | 6   | 6        | 4      | 3          |



# SQL Server 2008 進階的聚合函數

- 上述後面三個函數還可以加上 `partition by attr` 指示：
  - `RANK() OVER (partition by attr1 order by attr2) as alias`：先依照 `attr1` 分成數個類別，再依照 `attr2` 屬性排序後，傳回每一筆值組的排名，若遇有同名次的值組，則全部使用同一個排名順序，後續的排名要跳過同名次的值組數量。
  - `DENSE_RANK() OVER (partition by attr1 order by attr2) as alias`：先依照 `attr1` 分成數個類別，再依照 `attr2` 屬性排序後，傳回每一筆值組的排名，若遇有同名次的值組，則全部使用同一個排名順序，但是接下來的排名不跳過同名次的值組數量
  - `NTILE(n) OVER (partition by attr1 order by attr2) as alias`：先依照 `attr1` 分成數個類別，再依照 `attr2` 屬性排序後，將所有值組分成  $n$  群，傳回每一筆值組的群組編號

# SQL Server 2008 進階的聚合函數

- 為了說明加上 *partition by attr* 的指示，我們假設 Books 關聯表中有一欄 *category* 如下：

| <i>id</i> | <i>bookname</i> | <i>category</i> | <i>price</i> | <i>author</i> |
|-----------|-----------------|-----------------|--------------|---------------|
| 1         | 三國演義            | 文學類             | 120          | 羅貫中           |
| 2         | 水滸傳             | 文學類             | 170          | 施耐庵           |
| 3         | 紅樓夢             | 文學類             | 170          | 曹雪芹           |
| 4         | 西遊記             | 文學類             | 140          | 吳承恩           |
| 5         | 水經注             | 文學類             | 120          | 酈道元           |
| 6         | UNIX 系統         | 電腦類             | 300          | 周韻寰           |
| 7         | 資料庫系統           | 電腦類             | 360          | 曾守正           |
| 8         | Web 架站實務        | 電腦類             | 450          | 張三            |
| 9         | 英美文學讀本          | 語言類             | 250          | 李四            |
| 10        | 日文讀本            | 語言類             | 280          | 王五            |
| 11        | 德語入門            | 語言類             | 320          | 毛六            |
| 12        | 應用法語入門          | 語言類             | 220          | 李四            |

# SQL Server 2008 進階的聚合函數

select id, bookname, price,

RANK() OVER (partition by category order by price) as 同類中排名,

DENSE\_RANK() OVER (partition by category order by price) as 同類  
緊密排名,

NTILE(3) OVER (partition by category order by price) as 依排名分  
成3區段

from Books

| id | bookname | price | category | 同類中排名 | 同類緊密排名 | 依排名分 3 區段 |  |
|----|----------|-------|----------|-------|--------|-----------|--|
| 1  | 三國演義     | 120   | 文學類      | 1     | 1      | 1         |  |
| 5  | 水經注      | 120   | 文學類      | 1     | 1      | 1         |  |
| 4  | 西遊記      | 140   | 文學類      | 3     | 2      | 2         |  |
| 2  | 水滸傳      | 170   | 文學類      | 4     | 3      | 2         |  |
| 3  | 紅樓夢      | 170   | 文學類      | 4     | 3      | 3         |  |
| 6  | UNIX 系統  | 190   | 電腦類      | 1     | 1      | 1         |  |
| 8  | Web 架站實務 | 250   | 電腦類      | 2     | 2      | 2         |  |
| 7  | 資料庫系統    | 300   | 電腦類      | 3     | 3      | 3         |  |
| 11 | 德語入門     | 240   | 語言類      | 1     | 1      | 1         |  |
| 9  | 英美文學讀本   | 250   | 語言類      | 2     | 2      | 1         |  |
| 12 | 應用法語入門   | 280   | 語言類      | 3     | 3      | 2         |  |
| 10 | 日文讀本     | 320   | 語言類      | 4     | 4      | 3         |  |



# SQL Server 2008 其它系統函數

---

- 日期函數：用來計算日期，或取出日期的某些部份。
- 數學函數：用來做數字資料的數學運算。
- 虛值補充函數 (Nulladic Functions)
- 字串函數 (String Functions)：用來做字串的處理運算。
- 系統函數 (System Functions)：系統函數可以傳回資料庫中的各種特殊訊息。
- 文字與影像類型的處理函數：用來處理 text 與 image 類型的資料。
- 型態轉換的函數：用來將一種型態的資料轉換成另一種型態的資料。
- 產生唯一編號的函數：用來產生 uniqueidentifier 型態的唯一資料值。
- 詳細請看書上 7-53 ~ 7-56 頁，以及 SQL Server 2008 手冊



# 聚合函數的類型 [Han & Kamber (2005)]

若函數  $f$  所處理的對象為集合  $S$ ，且以  $f(S)$  表示  $f$  對  $S$  的運算結果，則有以下幾種類型：

- **可分散型 (Distributive) 函數**：若將集合  $S$  任意拆成  $n$  個子集合，也就是說： $S = S_1 \cup S_2 \cup \dots \cup S_n$ ，則此類的函數會符合  $f(S) = f(\{f(S_1), f(S_2), \dots, f(S_n)\})$  的特性。如： $\text{sum}()$ ,  $\text{min}()$ ,  $\text{max}()$ ,  $\text{count}()$  等都是。
- **代數型 (Algebraic)**：此類函數可以透過
  - 先以  $m$  個可分散型 (Distributive) 函數  $g_1(S), g_2(S), \dots, g_m(S)$  針對  $S$  求得部份解
  - 再將上述  $m$  個結果當成參數，運用某個代數運算  $\oplus$  求得最終解
  - 例如： $\text{avg}(S) = \text{sum}(S) / \text{count}(S)$ ，或計算標準差  $\text{standard\_deviation}()$ 。可運用「各個擊破法」(Divide-and-Conquer) 的技巧來快速求解
- **整體計算型 (Holistic)**：不具備前面兩類函數的特性，如：求取排名的  $\text{rank}()$ 、計算「中數」的  $\text{median}()$ ，或是找出「眾數」的  $\text{mode}()$

# Group by 子句

- select no, COUNT(distinct id)  
from Orders  
group by no

| <i>no</i> |   |
|-----------|---|
| 1         | 6 |
| 2         | 2 |
| 3         | 1 |
| 4         | 3 |

- 將 no 換成 id  
select id, COUNT(distinct no)  
from Orders  
group by id  
則答案為何? (做做看!)

Orders

| <i>no</i> | <i>id</i> | <i>quantity</i> |
|-----------|-----------|-----------------|
| 1         | 1         | 30              |
| 1         | 2         | 20              |
| 1         | 3         | 40              |
| 1         | 4         | 20              |
| 1         | 5         | 10              |
| 1         | 6         | 10              |
| 2         | 1         | 30              |
| 2         | 2         | 40              |
| 3         | 2         | 20              |
| 4         | 2         | 20              |
| 4         | 4         | 30              |
| 4         | 5         | 40              |

# Group by 子句 (續)

- select no, SUM(quantity)  
from Orders  
group by no

| <i>no</i> |     |
|-----------|-----|
| 1         | 130 |
| 2         | 70  |
| 3         | 20  |
| 4         | 90  |

Orders

| <i>no</i> | <i>id</i> | <i>quantity</i> |
|-----------|-----------|-----------------|
| 1         | 1         | 30              |
| 1         | 2         | 20              |
| 1         | 3         | 40              |
| 1         | 4         | 20              |
| 1         | 5         | 10              |
| 1         | 6         | 10              |
| 2         | 1         | 30              |
| 2         | 2         | 40              |
| 3         | 2         | 20              |
| 4         | 2         | 20              |
| 4         | 4         | 30              |
| 4         | 5         | 40              |



# Group by 子句 (續)

- select no, AVG(cast (quantity as real))  
from Orders  
group by no

| <i>no</i> |       |
|-----------|-------|
| 1         | 21.66 |
| 2         | 35    |
| 3         | 20    |
| 4         | 30    |

- 注意：具有 XML 資料型態的變數或屬性不可直接使用 group by 子句，除非先利用 Cast() 或 Convert() 函數轉換為字串資料。

## Orders

| <i>no</i> | <i>id</i> | <i>quantity</i> |
|-----------|-----------|-----------------|
| 1         | 1         | 30              |
| 1         | 2         | 20              |
| 1         | 3         | 40              |
| 1         | 4         | 20              |
| 1         | 5         | 10              |
| 1         | 6         | 10              |
| 2         | 1         | 30              |
| 2         | 2         | 40              |
| 3         | 2         | 20              |
| 4         | 2         | 20              |
| 4         | 4         | 30              |
| 4         | 5         | 40              |

# Group by 中的 with cube 選項

- 使用 **with cube** 運算：產生多維度的小計與總計結果，包括所有 group by 屬性的可能組合之交叉列表，以及它們小計與總計結果。

```
select no, id, SUM(quantity) as [訂購量]
from Orders
group by no, id with cube
```

| no | id   | 訂購量 | 2 | 1    | 30 | 4    | NULL | 90  |
|----|------|-----|---|------|----|------|------|-----|
| 1  | 1    | 30  | 2 | 2    | 40 | NULL | NULL | 310 |
| 1  | 2    | 20  | 2 | NULL | 70 | NULL | 1    | 60  |
| 1  | 3    | 40  | 3 | 2    | 20 | NULL | 2    | 100 |
| 1  | 4    | 20  | 3 | NULL | 20 | NULL | 3    | 40  |
| 1  | 5    | 10  | 4 | 2    | 20 | NULL | 4    | 50  |
| 1  | 6    | 10  | 4 | 4    | 30 | NULL | 5    | 50  |
| 1  | NULL | 130 | 4 | 5    | 40 | NULL | 6    | 10  |

# Group by 中的 with rollup 選項

- 使用 **with rollup** 運算：產生 *group-by* 中的群組屬性之間，由左至右的階層式小計與總計結果
- 以上例來說，結果比 *with cube* 少了第三組最後六筆，也就是只看 *no* 這個維度的小計與總計。

```
select no, id, SUM(quantity) as [訂購量]
from Orders
group by no, id with rollup
```

|    |      |     |   |      |    |      |      |     |
|----|------|-----|---|------|----|------|------|-----|
| no | id   | 訂購量 | 2 | 1    | 30 | 4    | NULL | 90  |
| 1  | 1    | 30  | 2 | 2    | 40 | NULL | NULL | 310 |
| 1  | 2    | 20  | 2 | NULL | 70 | NULL | 1    | 60  |
| 1  | 3    | 40  | 3 | 2    | 20 | NULL | 2    | 100 |
| 1  | 4    | 20  | 3 | NULL | 20 | NULL | 3    | 40  |
| 1  | 5    | 10  | 4 | 2    | 20 | NULL | 4    | 50  |
| 1  | 6    | 10  | 4 | 4    | 30 | NULL | 5    | 50  |
| 1  | NULL | 130 | 4 | 5    | 40 | NULL | 6    | 10  |



# 解決虛值所造成的混淆情況

---

- With {cube | rollup} 利用 Null 來指示小計與總計的資料，但有時候關聯表本身就有虛值，可能會造成混淆。
- 系統提供 **Grouping(*group-attribute*)** 函數可供利用：
  - 若查詢結果中 *group-attribute* 那一欄的 Null 是原來的虛值，則該函數會回傳 0，否則回傳 1。
  - 下頁的例子利用 **Grouping()** 函數，配合合併運算讓前述查詢結果顯示書局名稱與書籍名稱...

# 使用 Grouping() 的 SQL 指令範例

```
select case when (grouping(name) = 1) and (grouping(bookname) = 0) then '小計'
 when (grouping(name) = 1) and (grouping(bookname) = 1) then '總計'
 else isnull(name, '<NULL>')
end as [name],
case when (grouping(bookname) = 1) and (grouping(name) = 0) then '小計'
 when (grouping(bookname) = 1) and (grouping(name) = 1) then '總計'
 else isnull(bookname, '<NULL>')
end as [bookname], sum(quantity) as [訂購量]
from Orders O, Bookstores B, Books K
where O.no = B.no and O.id = K.id
group by name, bookname with cube
order by name
```

|      |          |     |      |      |    |      |      |     |
|------|----------|-----|------|------|----|------|------|-----|
| name | bookname | 訂購量 | 天秤書局 | 水滸傳  | 20 | 巨蟹書局 | 西遊記  | 20  |
| 小計   | 三國演義     | 60  | 天秤書局 | 西遊記  | 30 | 巨蟹書局 | 紅樓夢  | 40  |
| 小計   | 水經注      | 50  | 天秤書局 | 小計   | 90 | 巨蟹書局 | 道德經  | 10  |
| 小計   | 水滸傳      | 100 | 水瓶書局 | 水滸傳  | 20 | 巨蟹書局 | 小計   | 130 |
| 小計   | 西遊記      | 50  | 水瓶書局 | 小計   | 20 | 射手書局 | 三國演義 | 30  |
| 小計   | 紅樓夢      | 40  | 巨蟹書局 | 三國演義 | 30 | 射手書局 | 水滸傳  | 40  |
| 小計   | 道德經      | 10  | 巨蟹書局 | 水經注  | 10 | 射手書局 | 小計   | 70  |
| 天秤書局 | 水經注      | 40  | 巨蟹書局 | 水滸傳  | 20 | 總計   | 總計   | 310 |

# Having 子句

- 用來過濾不符條件的 group
- 正如 Where 子句是過濾不符條件的值組
- `select no, SUM(quantity)`  
`from Orders`  
`group by no`  
`having SUM(quantity) > 80`

| <i>no</i> |     |
|-----------|-----|
| 1         | 130 |
| 4         | 90  |



| <i>no</i> |     |
|-----------|-----|
| 1         | 130 |
| 2         | 70  |
| 3         | 20  |
| 4         | 90  |

Orders

| <i>no</i> | <i>id</i> | <i>quantity</i> |
|-----------|-----------|-----------------|
| 1         | 1         | 30              |
| 1         | 2         | 20              |
| 1         | 3         | 40              |
| 1         | 4         | 20              |
| 1         | 5         | 10              |
| 1         | 6         | 10              |
| 2         | 1         | 30              |
| 2         | 2         | 40              |
| 3         | 2         | 20              |
| 4         | 2         | 20              |
| 4         | 4         | 30              |
| 4         | 5         | 40              |

# 巢狀式查詢 (Nested Query)

- 條件中可以再出現另一個 SQL

```
select name, rank
from Bookstores
where rank < (
 select AVG(rank)
 from Bookstores)
```

- 先算出子查詢  
select AVG(rank)  
from Bookstores

|    |
|----|
| 22 |
|----|



- 再算出  
select name, rank  
from Bookstores  
where rank < 22

Bookstores

| <i>no</i> | <i>name</i> | <i>rank</i> | <i>city</i> |
|-----------|-------------|-------------|-------------|
| 1         | 巨蟹書局        | 20          | 臺北市         |
| 2         | 射手書局        | 10          | 高雄市         |
| 3         | 水瓶書店        | 30          | 新竹市         |
| 4         | 天秤書局        | 20          | 臺中市         |
| 5         | 獅子書局        | 30          | 臺南市         |



# Transact-SQL 的區域變數

---

- 上述巢狀查詢可用 Transact-SQL 變數來模擬
- declare @avg\_rank float  
select @avg\_rank = AVG(rank)  
from Bookstores
- select name, rank  
from Bookstores  
where rank < @avg\_rank



# 巢狀式查詢 (續)

- 條件中另一個 SQL  
只傳回一個值

```
select name
from Bookstores
where rank = (
```

```
select rank
from Bookstores
where no = 1)
```

- 再算出  
select name  
from Bookstores  
where rank = 20

Bookstores

| <i>no</i> | <i>name</i> | <i>rank</i> | <i>city</i> |
|-----------|-------------|-------------|-------------|
| 1         | 巨蟹書局        | 20          | 臺北市         |
| 2         | 射手書局        | 10          | 高雄市         |
| 3         | 水瓶書店        | 30          | 新竹市         |
| 4         | 天秤書局        | 20          | 臺中市         |
| 5         | 獅子書局        | 30          | 臺南市         |

| <i>name</i> |
|-------------|
| 巨蟹書局        |
| 天秤書局        |

# 傳回一個集合的巢狀子查詢

- 用 exists 或 not exists 來測試該集合的成員是否為空集合

- select distinct name  
from Bookstores  
where exists

(select \*  
from Orders

where Orders.no = Bookstores.no  
and Orders.id = 2)

Bookstores

| <i>no</i> | <i>name</i> | <i>rank</i> | <i>city</i> |
|-----------|-------------|-------------|-------------|
| 1         | 巨蟹書局        | 20          | 臺北市         |
| 2         | 射手書局        | 10          | 高雄市         |
| 3         | 水瓶書店        | 30          | 新竹市         |
| 4         | 天秤書局        | 20          | 臺中市         |
| 5         | 獅子書局        | 30          | 臺南市         |

# 傳回集合的巢狀子查詢 (續)

- 「求出訂購編號為 2 這本書的書局名稱」
- 先將第一筆 

|   |      |    |     |
|---|------|----|-----|
| 1 | 巨蟹書局 | 20 | 臺北市 |
|---|------|----|-----|

 取出後，以 Bookstores.no = 1 去執行  
`select *  
from Orders  
where Orders.no = 1  
and Orders.id = 2`
- 有符合的值組再將其 name 印出

Orders

| no | id | quantity |
|----|----|----------|
| 1  | 1  | 30       |
| 1  | 2  | 20       |
| 1  | 3  | 40       |
| 1  | 4  | 20       |
| 1  | 5  | 10       |
| 1  | 6  | 10       |
| 2  | 1  | 30       |
| 2  | 2  | 40       |
| 3  | 2  | 20       |
| 4  | 2  | 20       |
| 4  | 4  | 30       |
| 4  | 5  | 40       |



# 傳回集合的巢狀子查詢 (續)

---

■ select distinct name  
from Bookstores  
where exists  
    (select \*  
    from Orders  
    where Orders.no  
    = Bookstores.no  
    and Orders.id = 2)

■ select distinct name  
from Bookstores  
where no in  
    (select no  
    from Orders  
    where id = 2)



# 傳回集合的巢狀子查詢 (續)

---

- 任何含有 in 類型的查詢都可以換成以 exists 來取代，但**反之則否**！
- 前述的查詢中將 in 變成 not in 還是可以換成 not exists
- 但某些 exists (或 not exists) 的子查詢就不見得可以換成 in (或 not in)
- **如下頁的範例**



# 範例：找出那些沒有一本書他不訂購 的書局名稱 = 訂購所有書的書局名稱

```
■ select name -- 書局名稱
 from Bookstores
 where not exists -- 沒有一本書
 (select *
 from Books
 where not exists -- 不訂購
 (select *
 from Orders
 where Bookstores.no = Orders.no
 and Orders.id = Books.id))
```

其實  
就是  
除法  
運算

## Bookstores

| <i>no</i> | <i>name</i> | <i>rank</i> | <i>city</i> |
|-----------|-------------|-------------|-------------|
| 1         | 巨蟹書局        | 20          | 臺北市         |
| 2         | 射手書局        | 10          | 高雄市         |
| 3         | 水瓶書店        | 30          | 新竹市         |
| 4         | 天秤書局        | 20          | 臺中市         |
| 5         | 獅子書局        | 30          | 臺南市         |

## Books

| <u><i>id</i></u> | <i>bookname</i> | <i>author</i> | <i>price</i> | <i>publisher</i> |
|------------------|-----------------|---------------|--------------|------------------|
| 1                | 三國演義            | 羅貫中           | 120          | 古文出版社            |
| 2                | 水滸傳             | 施耐庵           | 170          | 中庸出版社            |
| 3                | 紅樓夢             | 曹雪芹           | 170          | 春秋出版社            |
| 4                | 西遊記             | 吳承恩           | 140          | 聊齋出版社            |
| 5                | 水經注             | 酈道元           | 120          | 易經出版社            |
| 6                | 道德經             | 老子            | 190          | 大唐出版社            |

## Orders

| <i>no</i> | <i>id</i> | <i>quantity</i> |
|-----------|-----------|-----------------|
| 1         | 1         | 30              |
| 1         | 2         | 20              |
| 1         | 3         | 40              |
| 1         | 4         | 20              |
| 1         | 5         | 10              |
| 1         | 6         | 10              |
| 2         | 1         | 30              |
| 2         | 2         | 40              |
| 3         | 2         | 20              |
| 4         | 2         | 20              |
| 4         | 4         | 30              |
| 4         | 5         | 40              |



# 傳回集合的巢狀子查詢 (續)

---

- 有些含有 in 類型的查詢事實上就是相等合併運算 (Equijoin)

```
select distinct name
from Bookstores
where no in (
 select no
 from Orders)
```

```
== select distinct name
 from Bookstores, Orders
 where Bookstores.no = Orders.no
```





# 含有 in 的巢狀查詢

---

- 在含有 in 的巢狀查詢，可以再加上額外的條件，其轉換方式仍然不變。(見下例)
- ```
select distinct name
from Bookstores
where no in
      (select no
       from Orders
       where id = 2)
```
- ```
select distinct name
from Bookstores, Orders
where Bookstores.no=
 Orders.no and
 Orders.id = 2
```



# 含有 in 的巢狀查詢(續)

---

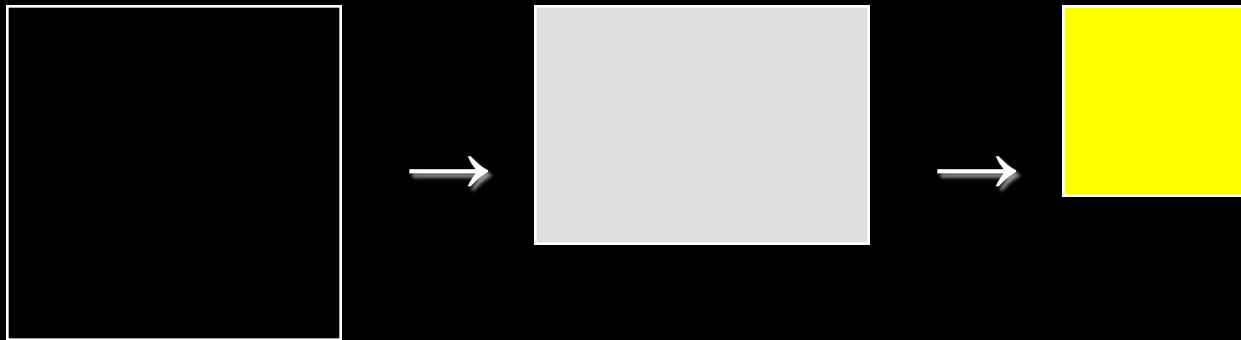
- 也可以在子查詢外面加上額外的條件，其轉換方式仍然一樣。(見下例)
- ```
select distinct name
from Bookstores
where rank = 30 and no
in (select no
    from Orders
    where id = 2)
```
- ```
select distinct name
from Bookstores, Orders
where Bookstores.no =
 Orders.no and
 Bookstores.rank = 30
 and Orders.id = 2
```



- select distinct name from Bookstores  
where no in (select no from Orders  
where id in (select id from Books))  
||
- select distinct name  
from Bookstores, Orders, Books  
where Bookstores.no = Orders.no and  
Orders.id = Books.id

# 複雜查詢的拆解

- 有些問題無法一次轉成一個 SQL 查詢
- 跟 SQL 的功力高、低有關
- 可以先將問題拆成數個小查詢，並暫存中間結果，再對中間結果繼續查詢，...





# 拆解查詢的範例

---

[範例] 找出至少訂購所有編號為 3 之書局所訂購書籍的書局，只列出其編號。

- **第一步驟**：將編號 3 之書局所訂購的書找出。

```
select id
into temp
from Orders
where no = 3
```



# 拆解查詢的範例 (續)

---

- **第二步驟**：將原查詢看成  
“找出訂購所有 temp 中所列書籍編號的書局編號”
  - 也就是說：“找出沒有一本 temp 中所列書籍編號它不訂購的書局編號”
  - 再套前面的三層巢狀結構 (除法運算)



# 拆解查詢的範例 (續)

---

- select no  
from Bookstores  
where not exists  
    (select \*  
    from temp  
    where not exists  
        (select \*  
        from Orders  
        where no = Bookstores.no  
        and id = temp.id))



# 拆解查詢的範例 (續)

---

- `select distinct no`    -- 只用一個關聯表即可  
from **Orders** OX  
where not exists  
    (select \*  
    from **Orders** OY  
    where no = 3  
    and not exists  
        (select \*  
        from **Orders** OZ  
        where OZ.no = OX.no  
        and OZ.id = OY.id))



# 聯集 (Union) 的查詢

- 兩個關聯表都要符合「聯集相容」(Union-Compatible) 的條件, 結果會去除重複值組

- ```
select id
from Books
where price > 160
union [all]
select id
from Orders
where no = 2
```

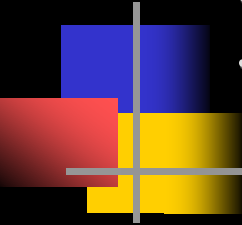
<i>id</i>	∪	<i>id</i>	=	<i>id</i>
2		1		1
3		2		2
6				3
				6

- 如果希望保留重複的值組則將 **union** 換成 **union all** 即可



一個複雜的例子

```
■ select bookname, '八折價=', price * 0.8, '銷售量 = ', SUM(quantity)
from Books, Orders          /* 合併 Books 與 Orders */
where Books.id = Orders.id  /* 合併條件 */
and (price >160 or price <150) /* 選擇條件 */
and quantity > 10          /* 選擇條件 */
group by bookname, price /* 以 1, 3 欄做為組成群組的依據 */
having SUM(quantity) > 45  /* 過濾各群組 */
order by 5 desc            /* 按 sum(quantity) 排序 */
```



執行順序

(5) select bookname, '八折價=', price * 0.8, '銷售量 = ', SUM(quantity)

(7) into Ad_hoc_Query /* 在此先省略不談 */

(1) from Books, Orders /* 合併 Books 與 Orders */

(2) where Books.id = Orders.id /* 合併條件 */

and (price > 160 or price < 150) /* 選擇條件 */

and quantity > 10 /* 選擇條件 */

(3) group by bookname, price /* 以 1, 3 欄做為組成群組的依據 */

(4) having SUM(quantity) > 45 /* 過濾各群組 */

(6) order by 5 desc /* 按 SUM(quantity) 排序 */



一個複雜的例子 (續)

- 第一步驟：先看 **From 子句**—將 Books, Orders 做乘積，得到 $6 * 12 = 72$ 筆
- 第二步驟：再看 **Where 子句**—將條件拿來過濾乘積的結果
where Books.id = Orders.id and
(price > 160 or price < 150) and quantity > 10
- 這兩步驟合起來就是「合併運算」(Join) 與「選擇運算」

第三步驟：看 Group by 子句，分成群組

■ Group by bookname, price

Books.id	bookname	author	price	publisher	no	Orders.id	quantity
1	三國演義	羅貫中	120	古文出版社	1	1	30
1	三國演義	羅貫中	120	古文出版社	2	1	30
2	水滸傳	施耐庵	170	中庸出版社	1	2	20
2	水滸傳	施耐庵	170	中庸出版社	2	2	40
2	水滸傳	施耐庵	170	中庸出版社	3	2	20
2	水滸傳	施耐庵	170	中庸出版社	4	2	20
3	紅樓夢	曹雪芹	170	春秋出版社	1	3	40
4	西遊記	吳承恩	140	聊齋出版社	1	4	20
4	西遊記	吳承恩	140	聊齋出版社	4	4	30
5	水經注	酈道元	120	易經出版社	4	5	40

第四步驟：看 **Having** 子句，過濾群組

- **Having** SUM(quantity) > 45

Books.id	<i>bookname</i>	<i>author</i>	<i>price</i>	<i>publisher</i>	<i>no</i>	Orders.id	<i>quantity</i>
1	三國演義	羅貫中	120	古文出版社	1	1	30
1	三國演義	羅貫中	120	古文出版社	2	1	30
2	水滸傳	施耐庵	170	中庸出版社	1	2	20
2	水滸傳	施耐庵	170	中庸出版社	2	2	40
2	水滸傳	施耐庵	170	中庸出版社	3	2	20
2	水滸傳	施耐庵	170	中庸出版社	4	2	20
4	西遊記	吳承恩	140	聊齋出版社	1	4	20
4	西遊記	吳承恩	140	聊齋出版社	4	4	30

第五步驟：看 **Select** 子句，投影欄位

- `select bookname, '八折價= ', price * 0.8, '銷售量 = ', SUM(quantity)`

<i>bookname</i>				
三國演義	'八折價= '	96	'銷售量 = '	60
水滸傳	'八折價= '	136	'銷售量 = ',	100
西遊記	'八折價= '	112	'銷售量 = ',	50

第六步驟：看 Order by 子句排序欄位

- order by 5 desc

<i>bookname</i>				
水滸傳	'八折價= '	136	'銷售量 = '	100
三國演義	'八折價= '	96	'銷售量 = ',	60
西遊記	'八折價= '	112	'銷售量 = ',	50

- 最後若要 into temp 的話, 那麼上面的每一個欄位都要有名稱, 也就是 select 子句要改成下面方式才行

```
select bookname, '八折價= ' as discount, price * 0.8 as real_price, '銷售量 = ' as sold, SUM(quantity) as total_quantity
```


將子查詢結果當成暫存表格的 複雜巢狀查詢

- 任何 SQL 查詢結果只要回傳表格，就可放在 From 之後加上別名 (Alias)，當成一般關聯表使用，以形成複雜的巢狀查詢
- **不過要注意**: 使用這種做法時，除非在子查詢的 Select 子句中加上 top n 選項—否則子查詢中不能出現 Order by 子句的)

```
select bookname, real_price, totalquantity_sold  
from (select top 5  bookname, '八折價=' as discount_title, price * 0.8 as real_price,  
                  '銷售量 =' as quantity_title, SUM(quantity) as totalquantity_sold
```

```
      from      Books, Orders  
      where     Books.id = Orders.id  
      and       (price > 160 or price < 150)  
      and       quantity > 10  
      group by  bookname, price  
      having    SUM(quantity) > 45  
      order by  5 desc) Tmp /* 若沒有前述的 top 5，則 order by 子句需去除 */
```

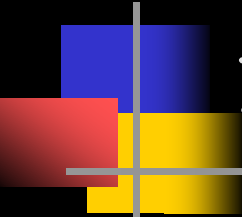
bookname	real_price	totalquantity_sold
水滸傳	136	100

```
where totalquantity_sold > 80 /* 將黃色子查詢的結果當成關聯表 Tmp */
```



影響查詢指令的選項與參數

- 透過動態指令限制查詢結果回傳的筆數
Set rowcount 10 -- 設定最多傳回 10 筆
 /* 此處放置 SQL 指令 */
Set rowcount 0 -- 設定回預設狀態
- 直接寫在 SQL 中的指令
top (expression) [percent] [with ties]
- 例如：
select top 10 with ties quantity
from Orders
order by quantity



查詢介於某個區段值的資料

- 以下指令可以查出 *no* 介於第 4 與第 5 名的書局資料
(作法是用前 5 名資料減去前 3 名的資料)

```
select * from (select top 5 * from Bookstores order by no) b1  
except  
select * from (select top 3 * from Bookstores order by no) b2
```

- 或使用
select *

```
from (select top 5 * from Bookstores order by no) b1  
where (not exists (select *  
                    from (select top 3 *  
                          from Bookstores order by no) b2  
                    where b1.no = b2.no))
```

DML 的異動部份

- insert into *tablename* [(*field1*, *field2*, ..., *fieldk*)]
values (*exp1*, *exp2*, ..., *expk*) | *subquery*

- 加入一筆新的值組

insert into Books (id, bookname, author, publisher)
values (7, '孫子兵法', '孫子', '大唐出版社')

7	孫子兵法	孫子	(Null)	大唐出版社
---	------	----	--------	-------

- insert into Books
values (7, '孫子兵法', '孫子', 200, '大唐出版社')



將查詢所得的結果新增到另一個關聯表中

- create table total_books_sold (
id char(6) not null,
name char(20) not null,
total_qty integer not null)
- insert into total_books_sold
select Books.id, bookname, SUM(quantity)
from Books, Orders
where Books.id = Orders.id
group by Books.id, bookname

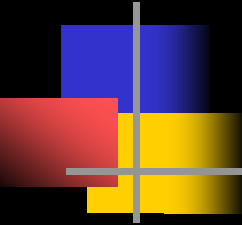
total_books_sold

<u>id</u>	name	total_qty
1	三國演義	60
2	水滸傳	100
3	紅樓夢	40
4	西遊記	50
5	水經注	50
6	道德經	10



Insert into ... select vs. select ... into

- insert into R ... select 指令與 select ... into New_R 指令很類似
- 都是將某個關聯表內容選擇出來之後，存入另一個關聯表中。
- 兩者的差別在於：select ... into New_R 中的 New_R 不需要事先存在 (如果事先存在會覆蓋過去)，但是 insert into R ... select 中的 R 卻必須要事先存在才行



Delete—刪除值組

- delete from *tablename*
where *predicate*
- 刪除單一值組
delete from Books
where id = 7 (以主鍵來做為刪除條件)
- 刪除多筆值組
delete from Books
where price = 120 (以非主鍵來做刪除條件)

依查詢結果做為刪除條件

- delete from Orders
where 120 =
(select price
from Books
where Books.id = Orders.id)

Books

<i>id</i>	<i>bookname</i>	<i>author</i>	<i>price</i>	<i>publisher</i>
1	三國演義	羅貫中	120	古文出版社
2	水滸傳	施耐庵	170	中庸出版社
3	紅樓夢	曹雪芹	170	春秋出版社
4	西遊記	吳承恩	140	聊齋出版社
5	水經注	酈道元	120	易經出版社
6	道德經	老子	190	大唐出版社

Orders

<i>no</i>	<i>id</i>	<i>quantity</i>
1	1	30
1	2	20
1	3	40
1	4	20
1	5	10
1	6	10
2	1	30
2	2	40
3	2	20
4	2	20
4	4	30
4	5	40



刪除條件的另一種寫法

■ delete from Orders where
120 =

(select price =
from Books
where Books.id =
Orders.id)

■ delete from Orders
where id in
(select id
from Books
where price = 120)

較易理解

速度較快



一次刪除多個關聯表

- **begin transaction**

- delete from Orders

- where no = 2

- delete from Bookstores

- where no = 2

- commit**

- 連鎖反應地刪除「外來鍵參考」要包成一個「異動」

Update—更改值組內容

- update *tablename*
set *field1* = *exp1*, *field2* = *exp2*, ...
[from *table1*, *table2*,...] where *predicate*
- update Bookstores
set name = '元智書坊', rank = rank + 10, city = '中壢市' where no
= 2

Bookstores

<i>no</i>	<i>name</i>	<i>rank</i>	<i>city</i>
1	巨蟹書局	20	臺北市
2	元智書坊	20	中壢市
3	水瓶書店	30	新竹市
4	天秤書局	20	臺中市
5	獅子書局	30	臺南市

依查詢結果來更改內容

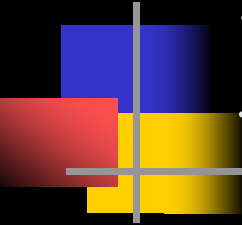
- update Orders set quantity = 0
where 120 =
(select price
from Books
where Books.id = Orders.id)

Books

<u>id</u>	bookname	author	price	publisher
1	三國演義	羅貫中	120	古文出版社
2	水滸傳	施耐庵	170	中庸出版社
3	紅樓夢	曹雪芹	170	春秋出版社
4	西遊記	吳承恩	140	聊齋出版社
5	水經注	酈道元	120	易經出版社
6	道德經	老子	190	大唐出版社

Orders

<u>no</u>	<u>id</u>	quantity
1	1	0
1	2	20
1	3	40
1	4	20
1	5	0
1	6	10
2	1	0
2	2	40
3	2	20
4	2	20
4	4	30
4	5	0



更改條件的另一種寫法

■ update Orders
set quantity = 0
where 120 =
(select price
from Books
where Books.id =
Orders.id)

==

■ update Orders
set quantity = 0
where id in
(select id
from Books
where price = 120)

較易理解

速度較快



一次更改多個關聯表

- **begin transaction**

- update Orders set no = 9

- where no = 2

- Update Bookstores set no = 9

- where no = 2

- commit**

- 連鎖反應地更改「外來鍵參考」要包成一個「異動」

依據某表格的資料更改對應欄位

- 假設關聯表 **Members** 用來存放公司網站中數萬會員資料 (上圖)
- 由於每天都會有數十位會員要求更改其行動電話號碼 (或是地址) 資料
- 業務助理每天都會將這些異動資料建成更新檔 **ToBeUpdated** (下圖)
- 如何找到 **Members** 中的相對記錄並更新?...

Members

<u>no</u>	...	Address	Cell_phone
A0001		中壢市	093x112233
A0002	...	鳳山市	093x223344
...
A1788	...	豐原市	093x777766
B0001	...	蘆洲市	093x556688
...
B8899	...	彰化市	095x008120

ToBeUpdated

<u>no</u>	Address	Cell_phone
A0038	花蓮市	093x123355
...
A1008	新店市	092x787866
B0035	中和市	093x566788
...
B0096	高雄市	091x002820



依據某表格的資料更改對應欄位



ToBeUpdated

<u>no</u>	Address	Cell_phone
A0038	花蓮市	093x123355
...
A1008	新店市	092x787866
B0035	中和市	093x566788
...
B0096	高雄市	091x002820

Members

<u>no</u>	...	Address	Cell_phone
A0001	...	中壢市	093x112233
A0002	...	鳳山市	093x223344
...
A0038	...	花蓮市	093x123355
...
A1008	...	新店市	092x787866
...
A1788	...	豐原市	093x777766
B0001	...	蘆洲市	093x556688
B0035	...	中和市	093x566788
...
B0096	...	高雄市	091x002820
...
B8899	...	彰化市	095x008120

update Members
set Address = ToBeUpdated.Address,
Cell_Phone = ToBeUpdated.Cell_Phone
from Members, ToBeUpdated
where Members.no = ToBeUpdated.no



資料控制語言 (DCL)

- DCL 用來設定資料物件 (包括：關聯表、視界、預儲程序等) 的使用權限主要的指令有三個：Grant、Revoke 與 Deny 指令
- 主要指令有：Grant、Revoke 與 Deny 指令：
 - **Grant** 授予權限給 Database User、Database Role 或 Application Role，
 - **Revoke** 取回使用權限，
 - **Deny** 明確對主體 (Principal) 拒絕權限，防止主體透過 Windows 群組或角色成員資格繼承權限。
- 也可以直接利用 SQL Server Management Studio 建立：
 - 可由 Server 的項目下建立 Login 帳號
 - 在資料庫的 [使用者] 中則可以建立 Database User 帳號
 - ... 如第三章所述



Grant—授予使用權限

```
grant {all [privileges] | privilege_list [(column_list)]}  
[on [class::] securable] to principal_list  
[with grant option] [as principal]
```

- 如果沒有寫 on 選項，表示要授與伺服器範圍的權限 (如：shutdown、create any database 等)。這種情況必須處於 master 資料庫的情況下，才能授與。
- 沒有寫 on 選項的另一種情況是授與資料庫範圍的權限，如：create table 等：
use BOB
grant create table to frank
go
- *privilege_list* 中可包含 select、insert、update、delete、execute、references、create、alter、control、view definition 等



Grant—授予使用權限 (續)

- *column_list* 只能配合 select 與 update 的權限授與關聯表中的屬性名稱
- *class* 指定安全實體的類別，包含 object (關聯表、視界、使用者預儲程序、等)、login、role、application role、user、schema 等
- *Securable* 指的是某個安全實體，該安全實體前若加上 *class*，則表示為某個 *class* 類別中的安全實體
- *principal_list* 表示可以逗號 “,” 隔開，表列多個主體
- 標明 *with grant option*，則代表被授權的使用者可再將權限授予其他使用者，後續可配合 Revoke 的 cascade 回收使用權限



privilege_list 的進一步說明

- Select 與 update：允許使用者對資料庫物件進行選擇與修改動作，也可以配合 (*column_list*) 指定關聯表的某幾個屬性讓使用者 select 與 update
- Insert 與 delete：允許使用者對資料庫物件進行新增與刪除動作
 - 都是以一整列值組為最小單位，所以不可以配合使用 (*column_list*) 指定關聯表中的某幾個屬性
- Execute 授與預儲程序 (Stored Procedure) 或函數 (User-Defined Function, UDF) 的執行授權



privilege_list 的進一步說明 (續)

- References 用在是否允許外來鍵 (Foreign Key) 參考
- create : 授與建立安全實體的權限，被建立的資料庫物件都是由該使用者所擁有
- control : 授與控制安全實體的任何權限
- view definition : 授與查看資料庫物件定義 (例如：預儲程序的原始碼) 的權限

Grant view definition on *schema::Chapter7* to public



使用範例

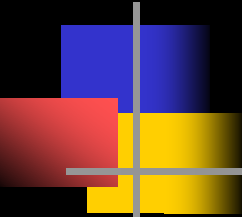
- 希望讓 Books 中的 (bookname, price, author, publisher) 讓 jesse, chris, annie, frank, mike 可以做選擇以及更新的動作時：

Grant select, update on Books(bookname, price, author, publisher) to jesse, chris, annie, frank, mike

- 注意：上述指令不能夠配合使用 insert 或 delete，因為這兩個權限不能指定某些屬性，除非將 (*bookname, author, price, publisher*) 刪除才行

- 希望 Books 可以讓所有人查詢的話，使用：

Grant select on Books to public with grant option



Revoke—取回使用權限

revoke [grant option for]

{ [all [privileges]] |

privilege_list [(*column_list*)] }

[on [*class* ::] *securable*]

{ to | from } *principal_list*

[**cascade**] [as *principal*]

- *privilege_list*、*column_list*、*class* 與上述 grant 說明相同
- **Cascade** 用於撤銷所有因 grant 配合 with grant option 所傳遞出去的授權狀態



使用範例

- 希望取回 jesse, chris, annie 對 Books 中的 (bookname, price, author, publisher) 做選擇以及更新的使用權時，則可以下達如下指令：

Revoke select, update on Books(bookname, price, author, publisher) from jesse, chris, annie

- 如果不再希望 *Books* 讓所有 public 角色查詢的話，則使用
revoke select on *Books* from public



運用技巧 (後令壓前令)

- 如果要讓大多數的使用者 (可能有200人) 都有完整的 Books 使用權限，但只有少數一、二位使用者 (假設是 tom 與 john) 不想賦予權限時，則只要：
Grant all on Books to public
Revoke all on Books from tom, john
- **注意：** 上述指令順序不可以相反



Deny—拒絕使用權限

```
deny {all [privileges]} | privilege_list [(column_list)] [, ...n]  
[ on [ class :: ] securable ] to principal_list [, ...n ]  
[ cascade ] [ as principal ]
```

- 明確對主體 (Principal) 拒絕權限，有點像防止某些主體鑽法律漏洞的感覺，畢竟法網恢恢，但 deny 可讓你疏而不漏
- 如果不希望 Books 讓所有人查詢的話
`deny all privileges on Books to public`
- 不希望讓 Chapter7 的 Schema 定義被查閱：
`deny view definition on schema::Chapter7 to public`



SQL vs. 關聯式代數

- $\sigma_p(R)$

select *
from R
where p

- $R \times S$

select *
from R, S

- $\pi_{a, b, c, d, \dots, z}(R)$

select distinct a, b, c, \dots, z
from R



SQL vs. 關聯式代數 (續)

■ R - S

select * from R

except

select * from S

-- SQL Server 2008 有提供

select * from R -- 若 SQL 沒提供上述 except 則只能如此
where not exists
 (select * from S
 where r1 = s1 and r2 = s2 and ... rn = sn)



SQL vs. 關聯式代數 (續)

- $R \cup S$

select * from R
union
select * from S

-- 標準 SQL 就有了

- $R \bowtie_p S = \sigma_p(R \times S)$

select *
from R, S
where p (合併條件)

SQL vs. 關聯式代數 (續)

■ $R \cap S$

select * from R

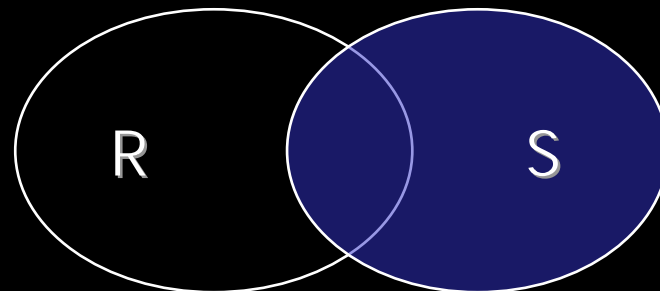
intersect -- SQL Server 2008 有提供

select * from S

若沒有提供 intersect 指令，則可以用 $R \bowtie S$ 的退化情況來模擬

或使用 $R \cap S = R - (R - S) = S - (S - R)$

見 6.2.2.4 節





用 $R \bowtie S$ 的退化情況來模擬

- $R(r1, r2, \dots, rn)$ 交集 $S(s1, s2, \dots, sn)$

```
select  R.*  
from    R, S  
where  $r1 = s1$  and  $r2 = s2$  and ... and  $rn = sn$ 
```

或

```
select  S.*  
from    R, S  
where  $r1 = s1$  and  $r2 = s2$  and ... and  $rn = sn$ 
```



用 $R \cap S = R - (R - S)$ 來模擬

- (1)

```
select *          /* R - S 的結果放入 tmp */
into Tmp
from R
where not exists (select *
                  from S
                  where r1 = s1 and r2 = s2 and ... and rn = sn)
```
- (2)

```
select *          /* R - tmp = R - (R - S) */
from R
where not exists (select *
                  from Tmp
                  where r1 = s1 and r2 = s2 and ... and rn = sn)
```




SQL vs. 關聯式代數 (續)

- $R[X, Y] \div S[Y]$

select distinct X
from R R1

where not exists “所有” = “沒有一個不”

(select *
from S

where not exists

(select *

from R R2

where R2.X = R1.X

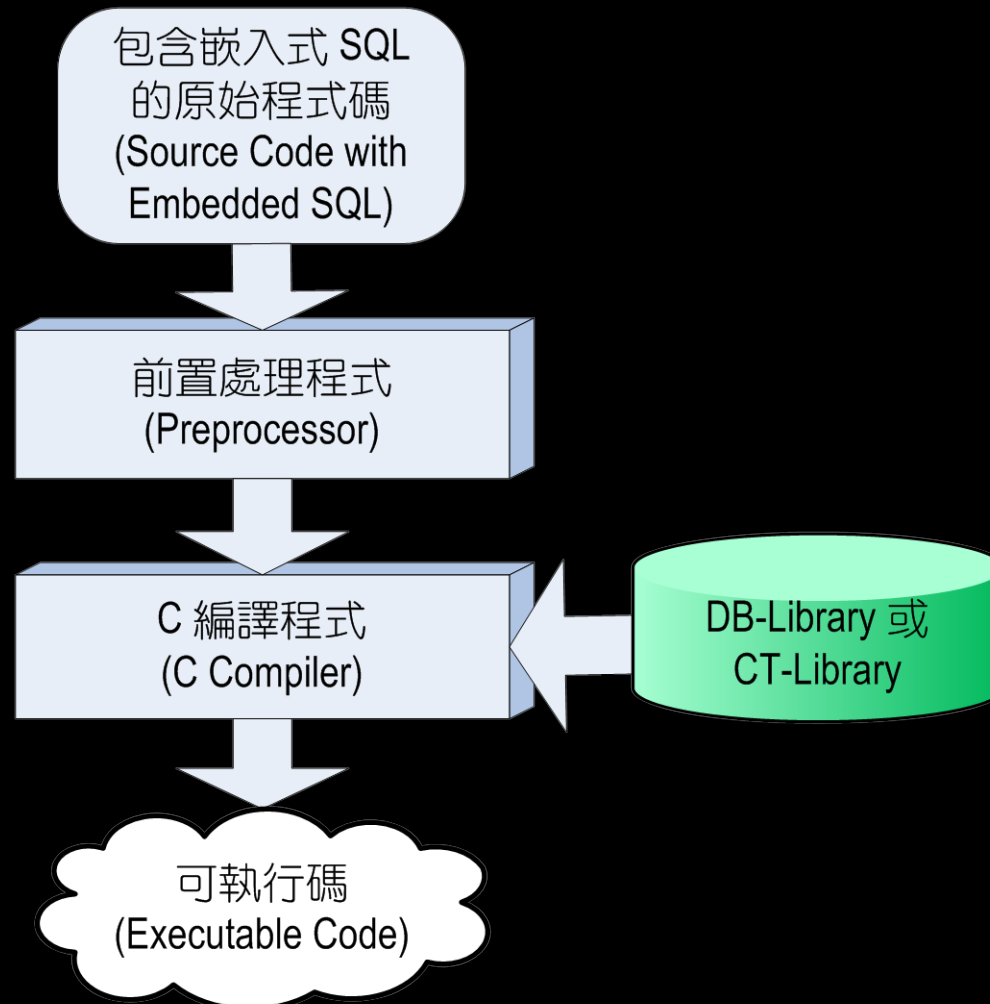
and R2.Y = S.Y))



嵌入式 SQL (Embedded SQL)

- 交談式 SQL 查詢僅能夠在線上做查詢與修改而已，所能發揮的影響有限
- 具備程式開發之能力，才能讓其影響發揮最大效力
- SQL 也可以嵌入高階的程式語言中
- 高階語言專注於輸出、入與流程之控制
- SQL 則專注於資料庫的存取、加入、刪除等 (Stored Procedures 也是)

SQL 嵌入高階語言的編譯程序





SQL 嵌入高階語言中的方式

- 將 SQL 嵌入高階語言中的方式：
 - 鬆散耦合 (Loosely-Coupled)，如：C 與 COBOL 內嵌 SQL。如圖 7.3 所示。
 - 緊密耦合 (Tightly-Coupled)，如：Microsoft Visual Basic



Transact-SQL 嵌入 C 語言範例

```
EXEC SQL include sqlca;
EXEC SQL begin declare section;
    char    bookname[20];
    char    author[12];
    int     price;
EXEC SQL end declare section;

.....

if (...) {
    EXEC SQL select bookname, author, price
        into :bookname, :author, :price
        from Books
        where id = 3;

    .....
}

printf("the price of %s is %d\n", bookname, price);
```



SQLCA (SQL Communication Area)

- 記錄每一個嵌入式 SQL 的執行狀態
- 成功與否的狀態則會放入 SQLCODE 中
- SQLCODE = 0 表示成功
- SQLCODE > 0 則表示有異常狀況發生
- SQLCODE < 0 則表示有錯誤狀況發生
- 讓程式設計師完全掌握在資料庫端執行時，所發生的各種狀況



Cursors 的說明

- 是一種新的資料型態 (Data Type)
- 包含一種指標 (Pointer) 可指向各筆值組
- 不需用到 Cursors 的情況計有：
 - 單一Select — 即只產生一筆記錄的 Select 指令
 - Update 指令
 - Insert 指令
 - Delete 指令



Cursors 的說明 (續)

- Indicator Variable (指示變數) 指示是否為 Null
- EXEC SQL select bookname, author, price
 into :bookname, :author, :price:ind
 from Books
 where id = 3;
if (ind == -1) { /* 取到虛值了 */

 }



Cursors 的宣告

- EXEC SQL declare *cursor-name* cursor for *select-statement*
[for update of *field1* [, *field2*, ...]]
- 宣告一個資料指標 X
EXEC SQL declare X cursor for
 select bookname, price, publisher
 from Books
 where price < 150;

Cursors 的運算

- 開啟資料指標

EXEC SQL open X → BOF

三國演義	120	古文出版社
西遊記	140	聊齋出版社
水經注	120	易經出版社

- 擷取資料並移動指標

EXEC SQL fetch X
into :bookname, :price, :publisher

三國演義	120	古文出版社
西遊記	140	聊齋出版社
水經注	120	易經出版社

- 關閉資料指標並釋放空間

EOF

EXEC SQL close X



Cursor 應用範例 (當作習題)

- 請見本章習題問答題第 30 題之習題解答

```
declare X cursor read_only for  
    select id, startdate, enddate  
    from Log  
    order by id
```

```
declare @id int, @startdate smalldatetime, @enddate smalldatetime
```

```
open X /* 接下來要將 cursor 由 BOF 移往第一筆 */
```

```
fetch next from X into @id, @startdate, @enddate
```

```
... (續下頁)
```



Cursor 應用範例 (續)

```
while (@@fetch_status = 0)  /* Cursor 已經指到 EOF 了 */
begin
    while (@startdate <= @enddate)
    begin
        insert into @DateTable(id, date) values (@id, @startdate)
        select @startdate = Dateadd(day, 1, @startdate)
    end
    fetch next from X into @id, @startdate, @enddate  /* 往下移 */
end
close X
```



Call-level Method

- 也可以直接在 C 語言中呼叫 DB-Library 或 ODBC API，那就不用經過 PreProcessor 的處理，但是這樣做就不能在程式中內嵌 Embedded SQL 了
- 範例可查 SQL Server 2008 on-Line help



資料指標 vs. 檔案指標

- 使用檔案時要先宣告一個檔案指標：`FILE *fp`；接著作業系統會在開啟檔案時，執行以下步驟：
 - 檢查使用者是否具有存取權限。若否，則駁回。
 - 配置一塊記憶空間給該檔案做為緩衝區 (Buffer)——如 DOS 下的 `config.sys` 中設定：`BUFFER = ??`
 - 指定檔案指標給 `*fp`，做為後續存取檔案的依據。
 - 鎖住該檔案，不許其它的程式或使用者對它做更新的動作。



資料指標 vs. 檔案指標

- 開啟資料指標時，資料庫管理系統也會做以下幾個動作：
 - 檢查使用者是否有存取表格的權限。若否，駁回。
 - 配置一塊記憶空間給該查詢結果做為緩衝區 (Buffer)。
 - 指定資料指標指向查詢結果，做為後續存取依據。
 - 鎖住該查詢中所用到的關聯表，不許其它的程式或使用者對它做更新的動作。



關閉檔案或資料指標

- 先將記憶體中的資料頁 (Dirty Pages) 寫回硬碟中，再解開 (Unlock) 檔案或資料指標鎖定狀態，
- 接著釋放緩衝區以及檔案或資料指標，
- 最後關閉檔案或資料指標 (Close)。

動態 SQL

■ 以交談方式來下達各種 SQL 的指令

1. 先宣告一個可變長度的字串變數（在此為 @SQLstring）

```
declare @SQLstring char(256)
```

2. 將變數 SQLstring 內容用字串相加運算，組成 SQL 指令，此步驟創造了最大的彈性運用空間

```
Select @SQLstring = 'select * from ' + 'Books'
```

3. 呼叫 Execute 函數，並將 @SQLString 當參數傳入，使得組合的 SQL 指令得以運作：

```
Execute(@SQLString)
```



SQL 隱碼 (SQL Injection) 攻擊

- 動態 SQL 創造了最大的彈性運用空間，但背後卻隱藏著危機，因為會有被 SQL Injection 攻擊的隱憂：
 - 在系統開發過程中，常常有接收 Username 與 Password，然後利用「動態 SQL」組合檢查帳號/密碼是否合法的 SQL 指令
 - 例如：假設帳號、密碼存放在下列關聯表 Userlist 中

<i>UserName</i>	<i>Password</i>	<i>CreateDate</i>
Frank	nkfustmis	2007/1/1
Annie	cmacis	2006/12/16



SQL 隱碼 (SQL Injection) 攻擊

- 在開發系統時 (以ASP為例)，便常常有如下的動作，用來檢驗使用者身份：
- 使用者名稱：
密碼：
- 並配合下列的程式碼 (以ASP為例) 來驗證：
`SQLString="Select * From UserList Where UserName=" & _
Request("UserName") & " AND Password=" & Request("Password") & ""
Set rec=conn.Execute(SQLString)`
- 請接續下頁...

SQL 隱碼 (SQL Injection) 攻擊

- 「SQL隱碼」(SQL Injection) 的攻擊原理很簡單，它在字串變數 @SQLString 的內容組合上動手腳，讓邏輯運算式永遠為 True，就可達成：E.g.,
- 使用者名稱：' or '' ='
密 碼：' or '' ='
- 則因為
SQLString="Select * From UserList Where UserName="" & "" or ""="" &_
"" and Password="" & "" or ""="" & """
- 就會組成
Select * from UserList where UserName = " or " = "
and Password = " or " = "

因為空字串 "" = ""
此條件永遠是 True



SQL 隱碼 (SQL Injection) 攻擊

- 想想看，如果輸入的使用者帳號跟密碼如下時，將會如何？
- 使用者名稱：sa' --
密碼：xxxxxx （隨便打都可以）
- 若想要更進一步了解 SQL Injection 攻擊，可參考
<http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf>
以及微軟在官方網站所公佈的文章，
<http://www.microsoft.com/taiwan/sql/>
目錄下的 SQL_Injection.htm、SQL_Injection_G1.htm，以及 SQL_Injection_G2.htm文章)



SQL Server 的程式化功能

- Transact-SQL :
- 基本的 SQL-92 指令，
 - 流程控制的指令，讓使用者可以像使用程式語言一樣
 - 副程式依性質又可區分成：
 - 「預儲程序」(Stored Procedures) 、
 - 「觸發程序」(Triggers) 、
 - 「使用者自訂函數」(User-Defined Functions) 等



Transact-SQL 流程控制指令

- `if ... else` : 為一條條件式指令，可以用 `exists` 或 `not exists` 來檢查某個查詢指令是否有傳回任何資料。
- `begin ... end` : 將一群 SQL 指令群組起來。
- `goto ...` : 跳到某個使用者所定義的標籤上執行。
- `while ...` : 為一迴圈指令。
- `break` : 中斷迴圈執行。
- `continue` : 回到迴圈的第一句繼續執行。
- `waitfor {delay time | time time}` : 延遲 *time* 後再執行或等待某個時間到達後繼續執行，如：
`waitfor delay '00:00:05'`
- `return` : 結束副程式回到原呼叫處。



Transact-SQL 流程控制指令

可搭配流程控制指令的其他Transact-SQL指令有：

- `print message`：列印 *message*。
- `declare @var type`：宣告變數 *var* 為型態 *type*。
- `select @var = constant`：設定 *@var* 變數的值。
- `case ... when ... then ... else ... end`：常用於 Select 指令中，讓資料可依據評估條件的內容展示不同的回應值，而這些回應值並不會取代資料庫內原來的資料。



程式範例

- 例1: 檢查 Bookstores 中是否有 *rank* = 35 的書局資料？如果沒有，則印出訊息。

```
if not exists (select * from bookstores where rank = 35)
    print '目前沒有 rank 為 35 的書局。'
```

- 例2: 寫程式刪除關聯表 Books。

```
if exists (select * from INFORMATION_SCHEMA.TABLES
           where table_name = 'Books' )
    drop table Books
```



程式範例 (續)

- 例3: 將 **Books** 關聯表中的每一本書籍價格漸次調高 10%，直到超過 200 元為止，但是調整的次數若超過三次的话，不論是否價格已經超過 200 元都必須要停止。

```
declare @count int
select @count = 0
while (exists (select * from books where price < 200)) and
    (@count < 3)
begin
    update books set price = price * 1.1 where price < 200
    select @count = @count + 1
end
```



程式範例 (續)

- 例4: 列出 Bookstores 的「書局名稱」與「星數」，其中「星數」的內容是以星星(*) 的圖示來表示，每一顆星代表 *rank* 值 10 分。

```
select name as '書局名稱', case rank
                                when 10 then '*'
                                when 20 then '**'
                                when 30 then '***'
                                else '-'
                                end as '星數'
from Bookstores
```



表格變數的使用

- 可以宣告表格變數當作暫存表格來使用

```
declare @t table (id int, name char(30), salary int)
```

```
insert @t values (1, 'Frank S.C. Tseng', 50000)
```

```
select id, name, salary from @t
```



撰寫 SQL 程式與應用程式的差別

- 資料庫的所有存取動作都會動到硬碟的存取，這與撰寫應用程式存取記憶體，在效能上有相當大的區別
- 建議：SQL 程式儘量以整個集合來處理，避免一筆一筆在資料庫上進行存取
- 例如：想在一個僅含一個欄位的關聯表 Numbers(n) 上產生 1 ~ 10000 的數字，傳統撰寫應用程式的習慣應該都是透過迴圈，一筆筆將資料存入關聯表中，但是概念上 (雖然 SQL Server 會協助我們做效能最佳化) 這會讓硬碟來回存取 10000 次 (速度慢可想而知)



傳統以迴圈 10000 次來存取的寫法

```
Create table Nums(n int not null primary key)
```

```
Declare @max as int, @rc as int
```

```
Set @max = 10000
```

```
Set @rc = 1
```

```
Begin transaction
```

```
While @rc <= @max
```

```
Begin
```

```
Insert into Nums values (@rc) -- 概念上要存取
```

```
Set @rc = @rc + 1 -- 10000 次硬碟
```

```
End
```

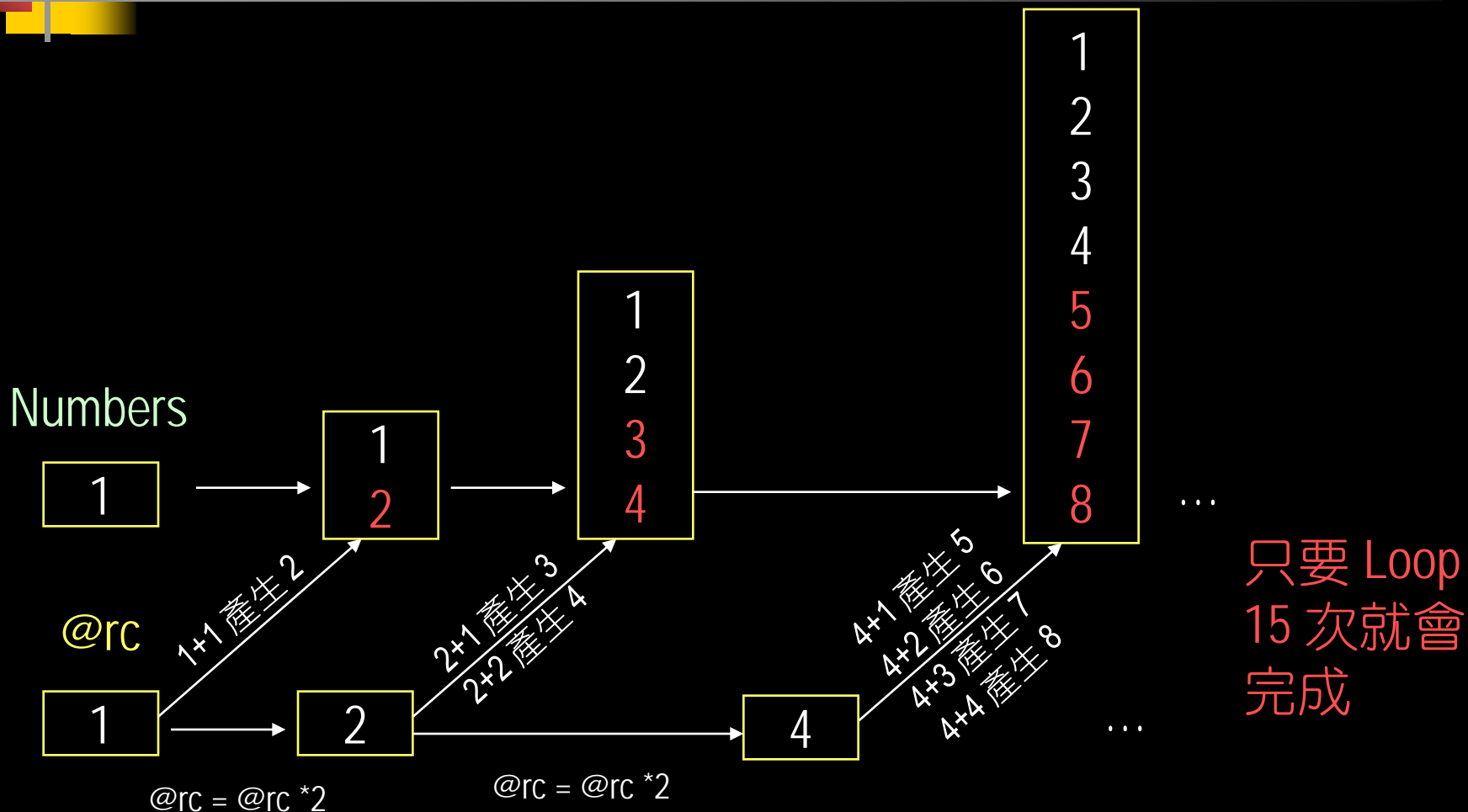
```
Commit Transaction
```



撰寫 SQL 程式與應用程式的差別

- 建議：換成盡量以集合來處理：
 - 讓 Numbers 中的資料以 2 的倍數成長，
 - 利用變數 @rc (也是每次 Loop 後就加倍) 儲存每次要由關聯表中取出之資料的相加對象
 - 這樣在 14 個迴圈次數中就可以完成任務
 - 等於將硬碟的存取次數從 10000 次降為 15 次
 - 讀者不妨將 @max = 10000 改成 100000 或 1000000 比比看兩者的效能差異 (雖然有時候不是很明顯，但那是 SQL Server 2008 有幫我們最佳化)
- 程式改寫如下...

程式改寫的觀念





撰寫 SQL 程式與應用程式的差別

```
Create table Numbers(n int not null primary key)
```

```
Declare @max as int, @rc as int
```

```
Set @max = 10000 -- 可以改成 100000 或 1000000 試看看
```

```
Set @rc = 1
```

```
Begin transaction
```

```
Insert into Numbers values (1)
```

```
While @rc * 2 <= @max -- 只要 Loop 15 次 (存取硬碟 15 次) 即可
```

```
Begin
```

```
Insert into Numbers select n + @rc from Numbers
```

```
Set @rc = @rc * 2
```

```
End
```

```
Insert into Numbers Select n + @rc from Numbers where n + @rc <= @max
```

```
Commit Transaction
```



SQL Server 的預儲程序

- 完全控制異動的完整性使資料維持一致性
- 達成某種程度的綱要資料與異動細節之隱密性與安全性
- 達成邏輯上的資料獨立
- 加強應用程式開發時的模組化程度
- 降低網路上的流量
- 降低人為錯誤的可能性

SQL Server 的預儲程序

if ...
call A
else
call B



Proc A
select *
from R

Proc B
select no
from S

與資料庫有關的程式應該寫在 Server 端, 避免寫在 Client 端



SQL Server 的預儲程序 (續)

- create proc *procedure_name* [(@*parameter1* *type1* [= *value*][, @*parameter2* *type2* [= *value*]...)]
as *SQL_statements*
- create proc joined_bob_tables
as select Bookstores.*, Orders.*, Books.*
from Bookstores, Orders, Books
where Bookstores.no = Orders.no
and Orders.id = Books.id
return



SQL Server 的預儲程序 (續)

- create proc bookstores_in_city (@city varchar(40) = '臺北市')
as
 if (@city is null)
 begin
 select name, city
 from Bookstores
 where city is null
 end
 else
 begin
 select name, city
 from Bookstores
 where city = @city
 end
- 呼叫時使用: exec bookstores_in_city



系統內定的整體變數

- 在任何預儲程序中都可以取得其值，
- 都是以 “@@” 做為引導符號。
- 某些值只是一個近似值而已，
- 可以透過 sp_configure 這個預儲程序在取得列表
- 所有的整體變數列舉如下：



整體變數列表

整體變數	說 明
@@connections	自 SQL Server 啟動後，登入及試圖登入的連線數目。
@@cpu_busy	自 SQL Server 啟動後所耗費的 CPU 時間。
@@dbts	資料庫目前的 timestamp 值，是一個唯一的值。
@@idle	自 SQL Server 啟動後，閒置的時間。
@@io_busy	自 SQL Server 啟動後，花在輸出、入上的時間。
@@max_connections	自 SQL Server 啟動後，同時連線的最大數目。
@@max_precision	目前在 decimal 與 numeric 資料型態上的精準度設定
@@microsoftversion	微軟公司內部用來追蹤目前版本所用。
@@nestlevel	目前查詢的巢狀層級。
@@pack_received	自 SQL Server 啟動後，讀取的輸入封包數目。



整體變數列表 (續)

@@pack_sent	自 SQL Server 啟動後，送出的封包數目。
@@packet_errors	自 SQL Server 啟動後，讀取或送出的錯誤封包數目。
@@rowcount	受到上一個 SQL 指令影響的記錄筆數。
@@servername	該伺服器的名稱。
@@servicename	代表執行中的服務名稱。
@@timeticks	每個 tick 的微秒 (microsecond) 數目。
@@total_errors	自 SQL Server 啟動後，讀取或寫入錯誤的數目。
@@total_read	自 SQL Server 啟動後，讀取磁碟機的次數。
@@total_write	自 SQL Server 啟動後，寫入磁碟機的次數。
@@version	目前的 SQL Server 版本。



使用回傳參數

- 可以使用 `output` 關鍵字來定義回傳參數 (return parameter)
- 要注意的是：呼叫該預儲程序時也必須要宣告該參數為回傳參數，見下例：
- ```
create proc book_sold(@id int, @total int output) as
 select @total = sum(quantity)
 from Orders where id = @id
return
```
- 呼叫時使用：

```
declare @total int
exec book_sold 1, @total output
print @total
```



# 回傳狀態值

---

- create proc KL\_Bookstores (@name char(10)) as  
if not exists(select \* from bookstores where name = @name)  
return -1 /\* error status \*/  
select no, name  
from bookstores  
where city = '基隆市' and name = @name  
return 0
- 可以用下面的方式來取得回傳的狀態值：
- declare @status int  
exec @status = KL\_Bookstores '水瓶書局'



# 預儲程序的撰寫原則

---

- 撰寫預儲程序的幾個原則：
  - 檢查是否有 Null 的參數傳入，若有則印出訊息告訴使用者如何呼叫此一預儲程序。
  - 檢查所有參數的合法性。
  - 不論是否執行成功，都要傳回執行狀態。
  - 印出具意義的訊息並在程序中加上適當的註解。
- 不再需要預儲程序時，其刪除指令為：  
`drop proc procedure_name`



# 內建的預儲程序

- 大部分都是以前綴 “sp\_” 或 “xp\_” 開頭。
- 例如：sp\_password 讓我們更改使用者密碼：  
sp\_password null, goodman, sa  
go -- 將原本為空字串 (null) 的 sa 密碼改為 goodman
- 不過要注意：SQL Server 2008 已提供新的指令  
Alter Login 用來取代 sp\_password  
Alter Login sa with password = 'I@mS@##'
- 內建預儲程序的程式內容可以透過  
sp\_helptext proc\_name 指令來查看
- sp\_help proc\_name 指令可以讓我們得知預儲程序的參數為何。

# 內建的預儲程序 (續)

- 有些指令因安全上的顧慮，SQL Server 2008 預設會關閉其使用許可
- 例如：xp\_cmdshell 'dos\_command' 接受 DOS 命令，然後予以執行，但因駭客可能以動態 SQL 組合成為：  
`xp_cmdshell 'del *.*'` 造成嚴重後果，所以 xp\_cmdshell 在預設上是 Off 的，
- 可以下面的指令開啟 (若要恢復，請將 1 改成 0，且執行順序要相反)：  
`Exec sp_configure 'show advanced options', 1`  
`reconfigure`  
`go`  
`EXEC sp_configure 'xp_cmdshell', 1`  
`reconfigure`  
`go`



# SQL Server 的觸發程序

---

- 撰寫觸發程序時，常常要用到兩個關聯表：
  - inserted (存放剛新增的值組內容)
  - deleted (存放剛刪除的值組內容)
  - 以及 if update (*column\_name*) 這個測試條件
- create trigger *trigger\_name* on *table\_name*  
for {insert | update | delete}, {...}  
as *SQL\_statements*
- 注意：由於觸發程序本身的主動特性，所以它並不接受參數傳遞，也不回傳值。



# 維繫資料一致的觸發程序

---

- Create trigger monitor\_Bookstores on Bookstores for delete as  
delete from Orders where Orders.no in (select no from deleted)
- Create trigger monitor\_Books on Books for delete as  
delete from Orders where Orders.id in (select id from deleted)

# 實際上的作法

放到 deleted 關聯表中

刪除

|   |      |    |     |
|---|------|----|-----|
| 1 | 巨蟹書局 | 20 | 臺北市 |
|---|------|----|-----|



Bookstores

| <u>no</u> | name | rank | city |
|-----------|------|------|------|
| 1         | 巨蟹書局 | 20   | 臺北市  |
| 2         | 射手書局 | 10   | 高雄市  |
| 3         | 水瓶書店 | 30   | 新竹市  |
| 4         | 天秤書局 | 20   | 臺中市  |
| 5         | 獅子書局 | 30   | 臺南市  |

Orders

| no | id | quantity |
|----|----|----------|
| 1  | 1  | 30       |
| 1  | 2  | 20       |
| 1  | 3  | 40       |
| 1  | 4  | 20       |
| 1  | 5  | 10       |
| 1  | 6  | 10       |
| 2  | 1  | 30       |
| 2  | 2  | 40       |
| 3  | 2  | 20       |
| 4  | 2  | 20       |
| 4  | 4  | 30       |
| 4  | 5  | 40       |





# 即時更新資料的觸發程序

---

- 假定我們在 Books 中加入了一個新的欄位 *discount\_price*，它存放打八折後的價錢，則我們可以由 *price* 來導出 *discount\_price* 的值如下：

update Books

set *discount\_price* = 0.8 \* *price*

- 再加上下列的觸發程序，就可以在更新 *price* 值時確保 *discount\_price* 的正確性：

create trigger insert\_discount\_price on Books for update

as if update (*price*)

update Books set *discount\_price* = *price* \* 0.8



# 自行製作稽核追蹤資料

---

- 建立一個 `audit_trail_log(loginame, database_used, username, date_used)` 的表格
- 在目標表格 `R` 上建立一個觸發程序  
`create trigger monitor_R on R for insert, update, delete  
as  
insert into audit_trail_log  
values (suser_sname(), db_name(), user_name(), getdate())`



# 觸發規則的刪除與優點

---

- 不再需要觸發程序時，其刪除指令為：  
drop trigger *trigger\_name*
- 觸發程序的優點：
  - 用來維持各種整合限制條件
  - 在分散式的資料庫系統上維持各資料站之間的資料一致性
  - 做一些例行性的檢查與補償措施 (存貨掌握)
  - 保持加總欄位，或是導出欄位的即時正確性
  - 自行做自己的異動記錄 (Customized Transaction Log)



# 觸發規則的缺點

---

- 觸發程序一多的時候，不容易維護，
- 可能會有彼此衝突的情形發生，
- 甚至於使得觸發程序無止境地呼叫自己或其它程序，造成資料庫的資料流失，或者失去控制
- 要以維持整個資料庫的一致性為考量
- 非針對應用程式各別寫一堆觸發程序

# 用 Instead of Trigger 模擬 Before Trigger

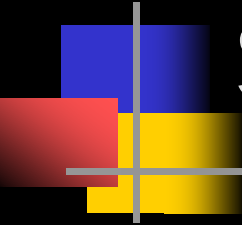
create trigger *instead\_of\_INSERT\_Books* on Books instead of INSERT  
as

if not exists(select B.*id* from Books B, inserted I where B.*id* = I.*id*)  
insert into Books select \* from inserted

else

update Books set *bookname* = I.*bookname*, *author* = I.*author*,  
*price* = I.*price*, *publisher* = I.*publisher*  
from Books B, inserted I  
where B.*id* = I.*id*

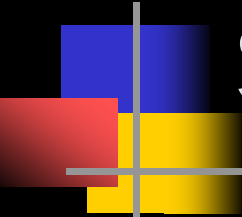
- 在使用者新增一筆資料到 Books 表時，由 *Instead\_of\_INSERT\_Books* 檢查看看該筆值組的主鍵 (也就是 *Id* 欄位) 是否已經存在 Books 中了？
- 若不存在，則將值組新增到 Books 中，否則就將新增指令改成依照主鍵來將所有非主鍵欄位更新為新資料的動作。



# SQL Server 2008 的使用者自訂函數

---

- 使用者自訂函數 (User-Defined Functions) 讓使用者可以函數傳回值來取代其名稱，故可用於 SQL 指令中
- 使用者自訂函數可以回傳一個表格變數，此一功能可彌補「**視界**」(Views) 在定義時無法傳遞參數的缺憾。
- SQL Server 2008 使用者自訂函數依傳回值分為**數值型**和**表格型**兩種，其完整名稱 (*database\_name.owner\_name.function\_name*) 都必須是唯一的。



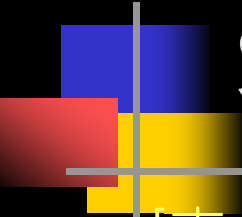
# SQL Server 2008 的使用者自訂函數

[數值型使用者自訂函數]

```
create function Bookstores_order_quantity (@name char(10))
returns int as
begin
declare @count int
select @count =SUM(quantity)
from Bookstores, Orders
where Bookstores.name=@name and Bookstores.no= Orders.no
return @count
end
```

注意：在呼叫數值型使用者自訂函數時，必須要註明函數的使用者名稱(owner\_name) 才行。例如：

```
select dbo.Bookstores_order_quantity ('巨蟹書局') as '巨蟹書局的總
訂購量'
```



# SQL Server 2008 的使用者自訂函數

## [表格型使用者自訂函數]

```
create function [日期區間](@startdate datetime, @enddate datetime)
returns @DateTable table ([日期] datetime) as
begin
 declare @datevar datetime
 select @datevar = @startdate
 while @datevar <= @enddate
 begin
 insert into @DateTable ([日期]) values (@datevar)
 select @datevar = Dateadd(day, 1, @datevar)
 end
 return
end
```

可以使用 `select * from [日期區間] ('2000/1/1', '2001/1/1')` 來呼叫  
不一定要提供函數的使用者名稱 (*owner\_name*)





# SQL Server 2008 的使用者自訂函數

---

## [表格型使用者自訂函數]

```
create function Books_eighty_percent_off (@author char(10))
returns table as
return (select bookname, author, price = case author
 when @author then price * 0.8
 else price
 end
 from Books)
```

使用 `select * from Books_eighty_percent_off('老子')` 來呼叫看看

# 擁有權串接 (Ownership Chaining) 的概念

- 如果想像某個關聯表 R 的使用權只能透過我們所撰寫的預儲程序來使用，以釐清責任，這就引發了所謂「擁有權串接」(Ownership Chaining) 問題。
- 範例：
  - Frank 是 Books 關聯表的擁有者
  - 為了要讓 John 僅能夠透過預儲程序存取 Books，Frank 撰寫了 Update\_Books\_for\_John 如下：  

```
create procedure Update_Books_for_John(@id int, @price int)
as
update Books set price = @price where id = @id
```



# Ownership Chaining 的概念

- 由於 Frank 擁有 Update\_Books\_for\_John 與 Books，因此 John 的執行過程不會引發系統檢查使用權，這種情形稱為 Ownership Chaining。
- 但若預儲程序中用了非 Frank 擁有的物件時，情況就不一樣了
- 假設 Orders 的擁有者是 Mary，且上述 Update\_Books\_for\_John 中增加一列將被修正價格的書籍訂單 Orders 的 quantity 設為 0：  
create procedure Update\_Books\_for\_John(@id int, @price int)  
as  
    update Books set price = @price where id = @id -- 系統不檢查權限  
    update Orders set quantity = 0 where id = @id -- 系統會檢查權限
- 則執行過程，系統會檢查 John 是否擁有 Orders 的 Update 權限
- 若應用了動態 SQL，那情況也不一樣，請接續下頁...



# 動態 SQL vs. Ownership Chaining

- 例如：Frank 想利用傳入的參數當作查詢的關聯表名稱，建立 Select\_Tables\_for\_John 供 John 使用：  
create procedure Select\_Tables\_for\_John(@Table\_Name Char(50))  
as -- 不能這樣寫，因參數不能用來直接當作表格名稱  
select count(\*) from @Table\_Name
- 變通的寫法是使用動態 SQL 如下：  
create procedure Select\_Tables\_for\_John(@Table\_Name Char(50))  
as -- 以下這樣寫語法是對的，但是會檢查使用權  
**execute**('select count(\*) from ' + @Table\_Name)



# 動態 SQL vs. Ownership Chaining

- 改寫後，在 John 沒有擁有任何資料庫物件的情況下，即使傳入的表格與Select\_Tables\_for\_John 的擁有者都是Frank，John 還是無法進行查詢
- 因為：透過動態 SQL 的指令都要接受權限檢查！不再適用「擁有權串接」概念
- 原因是：在預儲程序中使用動態建立SQL指令是非常危險的動作，因為可能引發 7.8 節所討論的「SQL 隱碼」(SQL Injection) 問題。
- 如：John 設定 @Table\_Name 如下 (想看看後果如何？)  
`select @Table_Name = 'Books; truncate table Books'`



# 執行脈絡 (Execution Context) 的設定

---

- SQL Server 2008 為了讓預儲程序及使用者自訂函數 (但不包含「數值型使用者自訂函數」) 更具彈性，提供三種「執行脈絡」(Execution Context) 的設定：
  - Execute as caller：權限檢查對象為呼叫該預儲程序 (或使用者自訂函數) 的資料庫使用者。
  - Execute as self：權限檢查對象為該預儲程序或使用者自訂函數的擁有者。
  - Execute as 'user'：權限檢查對象為 'user'。



# Execute as caller 的說明

---

create procedure Select\_Tables\_for\_John(@Table\_Name Char(50))  
**with execute as caller** -- 檢查使用權時，以呼叫者為對象  
as

execute('select count(\*) from ' + @Table\_Name)

- 會在 John 呼叫該程序時，檢查 John 的使用權限是否合法。
- 其執行結果等同於由 John 直接執行 execute() 指令的結果。



# Execute as self 的說明

---

```
create procedure Select_Tables_for_John(@Table_Name Char(50))
with execute as self -- 檢查使用權時，以此程序擁有者為對象
as
```

```
execute('select count(*) from ' + @Table_Name)
```

- 在 John 呼叫該程序時，會檢查 Frank 的使用權。
- 其執行結果等同於由 Frank 直接執行 execute() 指令的結果。





# Execute as '*user*' 的說明

---

```
create procedure Select_Tables_for_John(@Table_Name Char(50))
with execute as 'Mary' -- 檢查使用權時，以 Mary 為對象
as execute('select count(*) from ' + @Table_Name)
```

- 效果等同於直接執行下列指令：  
setuser Mary -- 改變使用者為 Mary 身份  
go  
execute('select count(\*) from ' + @Table\_Name)  
go  
setuser -- 設回原來的使用者身份  
go



# \* SQL Server 2008 進階的 SQL 功能

---

- SQL Server 2008 進階的 SQL 功能
  - APPLY 運算子、
  - OUTPUT、
  - PIVOT/UNPIVOT 指令、
  - 掌握與處理例外狀況的 TRY ... CATCH 區塊，
  - DDL Trigger 等功能。
- 如果資料庫是以 SQL Server 2000 建立，則必須設定如下才能執行這些指令：

`alter database [BOB] set compatibility_level = 100`  
或使用舊指令 `exec sp_dbcmtlevel 'BOB', '100'`



# Apply 運算子

- 分成 Cross Apply 與 Outer Apply 兩種
  - Cross Apply 有點像 Join 運算 (本書僅討論此項)
  - Outer Apply 則像 7.3.4 節中所提到的 Outer join。
- Cross Apply 最好的使用時機在於：
  - 運算元中有一個是一般的關聯表  $R$ ，
  - 另一個則是可接受參數  $a$  的表格型使用者自訂函數  $F(a)$ ，而參數  $a$  恰好就是  $R$  的某一個屬性  $R.a$ 。
  - 例子：假設週年慶時，希望任何一本書只要被訂購的當日與作者生日相同，就打八折優惠，那麼就可使用 Cross Apply 運算來求解。

# Apply 運算子 (先修正表格)

Date\_Orders

| <u>order_date</u> | no | id | quantity |
|-------------------|----|----|----------|
| 2005/3/7          | 1  | 1  | 30       |
| 2005/3/8          | 1  | 2  | 20       |
| 2005/4/1          | 1  | 3  | 40       |
| 2005/5/9          | 1  | 4  | 20       |
| 2005/6/8          | 1  | 5  | 10       |
| 2005/7/3          | 1  | 6  | 10       |
| 2005/8/1          | 2  | 1  | 30       |
| 2005/9/3          | 2  | 5  | 40       |
| 2005/10/10        | 3  | 4  | 20       |
| 2005/11/11        | 4  | 1  | 20       |
| 2005/11/16        | 4  | 2  | 30       |
| 2006/6/30         | 4  | 5  | 40       |

Birthdate\_Books

| <u>id</u> | bookname | author | birthdate  | price | publisher |
|-----------|----------|--------|------------|-------|-----------|
| 1         | 三國演義     | 羅貫中    | 1983/11/11 | 120   | 古文出版社     |
| 2         | 水滸傳      | 施耐庵    | 1964/11/16 | 170   | 中庸出版社     |
| 3         | 紅樓夢      | 曹雪芹    | 1952/3/12  | 170   | 春秋出版社     |
| 4         | 西遊記      | 吳承恩    | 1938/10/10 | 140   | 聊齋出版社     |
| 5         | 水經注      | 酈道元    | 1955/9/3   | 120   | 易經出版社     |
| 6         | 道德經      | 老子     | 1968/8/2   | 190   | 大唐出版社     |



# Apply 運算子

---

- 先撰寫使用者自訂函數如下：

```
create function birthday_eighty_percent_off (@order_date smalldatetime)
returns table as
return (select id, bookname, author, birthdate, -- 以下只比較月份與日
 price = case (abs(datepart(day, @order_date) - datepart(day, birthdate)) +
 abs(datepart(month, @order_date) - datepart(month, birthdate)))
 when 0 then price * 0.8 -- 0 表示訂購日與作者生日一樣
 else price end
from Birthdate_Books)
```

# Apply 運算子

- 用下述查詢找出符合打折條件的訂單與書籍

Select R.Order\_date, F.bookname, F.author, F.birthdate, F.price, R.quantity  
from Date\_Orders as R

**Cross Apply** birthday\_eighty\_percent\_off(R.order\_date) as F  
where (datepart(day, R.order\_date) = datepart(day, F.birthdate) and  
datepart(month, R.order\_date) = datepart(month, F.birthdate) and R.id = F.id)

| <u>Order_date</u> | bookname | author | birthdate  | price | quantity |
|-------------------|----------|--------|------------|-------|----------|
| 2005/9/3          | 水經注      | 酈道元    | 1955/9/3   | 96    | 40       |
| 2005/10/10        | 西遊記      | 吳承恩    | 1938/10/10 | 112   | 20       |
| 2005/11/11        | 三國演義     | 羅貫中    | 1983/11/11 | 96    | 20       |
| 2005/11/16        | 水滸傳      | 施耐庵    | 1964/11/16 | 136   | 30       |



# Output 指令

---

- Output 指令讓我們可以將 inserted 與 deleted 的內容轉存到表格變數中：

```
[Output {column_name | scalar_exp} [as] alias [, ... n]
Into { @table_var | output_table } [(column_list)]]
```

或

```
[Output {Deleted | Inserted | from_table_name} . {* | column_name}]
```



# Output 指令

---

- 舉一個例子同時說明這兩種用法：

```
declare @DeletedData as table (no int, id int, quantity int)
```

```
delete from Orders
```

```
output deleted.no, deleted.id, deleted.quantity into @DeletedData
```

```
where quantity < 20
```

```
select * from @DeletedData -- 印出被刪除的內容
```

```
insert into Orders(no, id, quantity) -- 再將被刪資料加回Orders
```

```
output inserted.no, inserted.id, inserted.quantity -- 回傳加回的內容
```

```
select no, id, quantity
```

```
from @DeletedData
```





# Pivot 指令

---

- 主要的兩種效果：
  - 簡化的線上分析處理 (OLAP)：用來對關聯表進行屬性的轉置 (Transpose)，其效果可看成簡化的「線上分析處理」(On-Line Analytical Processing, OLAP)，在 SQL:1999 的標準制定中已經將它納入 OLAP 的規範了
  - 用於產生「開放式表格綱要」(Open Schema)：目的是讓任何實體表格都能擁有任意數目的屬性個數。相較於傳統的做法，使用 Pivot 會讓這樣的表格綱要管理具備更高彈性。



# 簡化的線上分析處理 (OLAP)

```
select *
from Orders
pivot (sum(quantity) for no in ([1], [2], [3], [4], [5]))
as P
```

| <i>id</i> | 1  | 2    | 3    | 4    | 5    |
|-----------|----|------|------|------|------|
| 1         | 30 | 30   | NULL | NULL | NULL |
| 2         | 20 | 40   | 20   | 20   | NULL |
| 3         | 40 | NULL | NULL | NULL | NULL |
| 4         | 20 | NULL | NULL | 30   | NULL |
| 5         | 10 | NULL | NULL | 40   | NULL |
| 6         | 10 | NULL | NULL | NULL | NULL |

# 簡化的線上分析處理 (OLAP)

select \*      請同學上臺練習求出: 書籍在 1 ~ 6 月份的銷售量 (P134)

from ( select s.name, b.bookname, o.quantity  
      from Bookstores s, Orders o, Books b  
      where s.no = o.no and o.id = b.id) T

pivot (sum(quantity) for [name]  
in ([巨蟹書局], [射手書局], [水瓶書局], [天秤書局], [獅子書局]))  
as P

| <i>bookname</i> | 巨蟹書局 | 射手書局 | 水瓶書局 | 天秤書局 | 獅子書局 |
|-----------------|------|------|------|------|------|
| 三國演義            | 30   | 30   | NULL | NULL | NULL |
| 水經注             | 10   | NULL | NULL | 40   | NULL |
| 水滸傳             | 20   | 40   | 20   | 20   | NULL |
| 西遊記             | 20   | NULL | NULL | 30   | NULL |
| 紅樓夢             | 40   | NULL | NULL | NULL | NULL |
| 道德經             | 10   | NULL | NULL | NULL | NULL |

# 簡化的線上分析處理 (OLAP)

```
select * from (
 select datename(month, o.order_date) as '月份',
 b.bookname, o.quantity
 from date_orders o, Books b
 where o.id = b.id) T
pivot (sum(quantity) for [月份]
in ([一月], [二月], [三月], [四月], [五月], [六月]))
as P
```

| <i>bookname</i> | 一月   | 二月   | 三月   | 四月   | 五月   | 六月   |
|-----------------|------|------|------|------|------|------|
| 三國演義            | NULL | NULL | 30   | NULL | NULL | NULL |
| 水經注             | NULL | NULL | NULL | NULL | NULL | 50   |
| 水滸傳             | NULL | NULL | 20   | NULL | NULL | NULL |
| 西遊記             | NULL | NULL | NULL | NULL | 20   | NULL |
| 紅樓夢             | NULL | NULL | NULL | 40   | NULL | NULL |
| 道德經             | NULL | NULL | NULL | NULL | NULL | NULL |



# 簡化的線上分析處理 (OLAP)

---

- `select * from (`  
    `select no, datename(month, o.order_date) as '月份',`  
        `b.bookname, o.quantity`  
    `from date_orders o, Books b`  
    `where o.id = b.id) T`  
    `pivot (sum(quantity) for [月份]`  
    `in ([一月], [二月], [三月], [四月], [五月], [六月]))`  
    `as P`
- 執行看看, 會以三維方式展現...

# 輸出 0 或空白來取代輸出 NULL

- 可以藉助 `isnull()` 函數來達成

```
select A.id, isnull(A.[1], 0) 巨蟹書局, isnull(A.[2], 0)
射手書局, isnull(A.[3], 0) 水瓶書局,
isnull(A.[4], 0) 天秤書局, isnull(A.[5], 0) 獅子書局
from Orders
pivot (sum(quantity) for no in ([1], [2], [3], [4], [5])) as A
```

| Id | 巨蟹書局 | 射手書局 | 水瓶書局 | 天秤書局 | 獅子書局 |
|----|------|------|------|------|------|
| 1  | 30   | 30   | 0    | 0    | 0    |
| 2  | 20   | 40   | 20   | 20   | 0    |
| 3  | 40   | 0    | 0    | 0    | 0    |
| 4  | 20   | 0    | 0    | 30   | 0    |
| 5  | 10   | 0    | 0    | 40   | 0    |
| 6  | 10   | 0    | 0    | 0    | 0    |



# 多維度 Pivot 查詢

---

- 以上查詢結果，可以看成是以縱、橫二個維度來看第三個維度 (*quantity*) 的呈現方式，
- 整個查詢結果中總共含有三個維度。
- 如果想產生多一個維度的呈現結果，在 SQL Server 2008 中也不是難事，
- 只要最外層的 from 子句中的資料來源表格比前述的例子多一個欄位即可
- 請見下例...



# 多維度 Pivot 查詢

---

- 讓 from 後面的來源資料表多一個欄位即可:

```
select * from
 (select no, datename(month, o.order_date) as '月份', b.bookname,
 o.quantity -- 總共四個欄位
 from date_orders o, Books b
 where o.id = b.id) T
pivot (sum(quantity) for [月份] in
 ([一月], [二月], [三月], [四月], [五月], [六月])) as P
order by no
```



# 產生四維表格

| no | bookname | 一月   | 二月   | 三月   | 四月   | 五月   | 六月   |
|----|----------|------|------|------|------|------|------|
| 1  | 三國演義     | NULL | NULL | 30   | NULL | NULL | NULL |
| 1  | 水經注      | NULL | NULL | NULL | NULL | NULL | 10   |
| 1  | 水滸傳      | NULL | NULL | 20   | NULL | NULL | NULL |
| 1  | 西遊記      | NULL | NULL | NULL | NULL | 20   | NULL |
| 1  | 紅樓夢      | NULL | NULL | NULL | 40   | NULL | NULL |
| 1  | 道德經      | NULL | NULL | NULL | NULL | NULL | NULL |
| 2  | 三國演義     | NULL | NULL | NULL | NULL | NULL | NULL |
| 2  | 水經注      | NULL | NULL | NULL | NULL | NULL | NULL |
| 3  | 西遊記      | NULL | NULL | NULL | NULL | NULL | NULL |
| 4  | 三國演義     | NULL | NULL | NULL | NULL | NULL | NULL |
| 4  | 水經注      | NULL | NULL | NULL | NULL | NULL | 40   |
| 4  | 水滸傳      | NULL | NULL | NULL | NULL | NULL | NULL |

可以看成如下的四維表格



| no | bookname | 一月   | 二月   | 三月   | 四月   | 五月   | 六月   |
|----|----------|------|------|------|------|------|------|
| 1  | 三國演義     | NULL | NULL | 30   | NULL | NULL | NULL |
|    | 水經注      | NULL | NULL | NULL | NULL | NULL | 10   |
|    | 水滸傳      | NULL | NULL | 20   | NULL | NULL | NULL |
|    | 西遊記      | NULL | NULL | NULL | NULL | 20   | NULL |
|    | 紅樓夢      | NULL | NULL | NULL | 40   | NULL | NULL |
|    | 道德經      | NULL | NULL | NULL | NULL | NULL | NULL |
| 2  | 三國演義     | NULL | NULL | NULL | NULL | NULL | NULL |
|    | 水經注      | NULL | NULL | NULL | NULL | NULL | NULL |
| 3  | 西遊記      | NULL | NULL | NULL | NULL | NULL | NULL |
| 4  | 三國演義     | NULL | NULL | NULL | NULL | NULL | NULL |
|    | 水經注      | NULL | NULL | NULL | NULL | NULL | 40   |
|    | 水滸傳      | NULL | NULL | NULL | NULL | NULL | NULL |



# 「開放式表格綱要」(Open Schema)

- 假設書店除了書之外，還兼賣文具、CD 音樂、電影光碟等數十樣產品
- 每一樣產品都有不同數目的屬性。
- 傳統做法上，通常會採用兩種做法：
  - 每種產品各設計一個表格來存放：「土法煉鋼」
  - 使用兩個具有固定欄位數目的表格來存放資料
    - 一個用來存放產品名稱 (Product\_Name)，
    - 一個用來存放各產品的屬性與資料 (Product\_Property)，
    - 兩者以外來鍵 *id* 互相關聯，藉以找到產品各有何屬性？
    - 利用 Tuple\_id 的值若相同，則表示屬於同一筆資料

# 「開放式表格綱要」 (Open Schema)

| Product_Name |                  | Product_Property |                 |                  |              |
|--------------|------------------|------------------|-----------------|------------------|--------------|
| <u>id</u>    | <u>Prod_name</u> | <u>id</u>        | <u>Tuple_id</u> | <u>Attr_Name</u> | <u>Value</u> |
| 1            | 書籍               | 1                | 1               | Bookname         | 三國演義         |
| 2            | CD 音樂            | 1                | 1               | Author           | 羅貫中          |
| 3            | 電影光碟             | 1                | 1               | Price            | 120          |
| 4            | 文具               | 2                | 1               | Title            | 伍佰精選         |
| 5            | 海報               | 2                | 1               | Type             | 搖滾樂          |
| 6            | ...              | ...              | ...             | ...              | ...          |

| 書籍              |               |              | CD 音樂        |             | ... |
|-----------------|---------------|--------------|--------------|-------------|-----|
| <u>Bookname</u> | <u>Author</u> | <u>Price</u> | <u>Title</u> | <u>Type</u> | ... |
| 三國演義            | 羅貫中           | 120          | 伍佰精選         | 搖滾樂         | ... |
| ...             | ...           | ...          | ...          | ...         | ... |



# 用以下 SQL 指令產生

---

```
select Bookname, Author, Price
from Product_Property
pivot (max(Value) for Attr_Name in ([Bookname],[Author],[Price])) as P
where id in (select id from Product_Name where Prod_Name = '書籍')
```

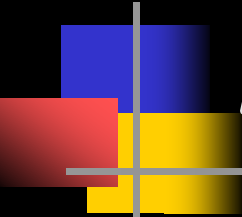
---

```
select Title, Type
from Product_Property
pivot (max(Value) for Attr_Name in ([Title],[Type])) as P
where id in (select id from Product_Name where Prod_Name = 'CD音樂')
```

- 將上面的指令廣義化後，可將 *Prod\_Name* 當參數傳入，並透過下頁的預儲程序產生動態 SQL 的效果

# 廣義化寫成 open\_schema

```
create procedure open_schema(@Prod_name nchar(30)) as
begin
declare X cursor for
 select Attr_Name from Product_Name n, Product_Property p
 where n.id = p.id and n.Prod_Name = @Prod_Name and Tuple_id = 1
declare @Attr_List nvarchar(max), @Attr_Name nchar(30)
select @attr_List = ""
open X
fetch next from X into @Attr_Name -- 任何表格至少要有一個attribute
while (@@fetch_status = 0)
begin
 if (@attr_List = "") select @Attr_List = @Attr_list + '[' + rtrim(@Attr_Name)
 else select @attr_list = @attr_list + '],[' + rtrim(@attr_Name)
 fetch next from X into @Attr_Name
end
if (@attr_List <> "") select @attr_list = @attr_list + ']'
close X; deallocate X
declare @pivot_statement nvarchar(max)
if (@attr_List <> "") select @pivot_statement =
'select ' + @attr_list + ' into [' + rtrim(@Prod_Name) +
'] from Product_Property pivot (max(value) for Attr_Name in (' + @attr_list + ')) ' +
'as P where id in (select id from Product_Name where Prod_Name = "' +
rtrim(@Prod_Name) + '")'
if exists(select * from sys.objects where name = @Prod_Name and type = 'U')
begin
 declare @drop_command char(100);
 select @drop_command = 'drop table ' + @Prod_Name; execute(@drop_command)
end
execute(@pivot_statement)
end
```



# 廣義化寫成 open\_schema

---

- 有了 Open\_Schema 之後，後續就可以呼叫  
`exec open_schema 'CD音樂'; exec open_schema '書籍';`
- 如果在 Product\_Name 與 Product\_Property 表格中有存放「電影光碟」「文具」「海報」的相關資料時，那就可以用以下指令：  
`exec open_schema '電影光碟';`  
`exec open_schema '文具';`  
`exec open_schema '海報';`  
產生各種產品的表格結構了



# Unpivot 指令

---

- 與 Pivot 指令動作相反，將 Pivot 轉置後的表格還原成原來的關聯表結構
- 以前面討論：哪一家書局訂購那一本書籍之數量之查詢指令，我們加入 into Pv 子句將該結果儲存到 Pv 表格，如下：

```
select *
```

```
Into Pv
```

```
from (select s.name, b.bookname, o.quantity
 from Bookstores s, Orders o, Books b
 where s.no = o.no and o.id = b.id) T
```

```
pivot (sum(quantity) for [name]
```

```
in ([巨蟹書局], [射手書局], [水瓶書局], [天秤書局], [獅子書局]))
```

```
as P
```



# Unpivot 指令

---

- 可以用下面的指令將 Pv 表格透過 Unpivot 動作轉回原來的結構：

```
select name, bookname, quantity
```

```
from (select bookname, [巨蟹書局], [射手書局], [水瓶書局], [天秤書局], [獅子書局]
 from Pv) P
```

```
unpivot (quantity for name
```

```
in ([巨蟹書局], [射手書局], [水瓶書局], [天秤書局], [獅子書局]))
```

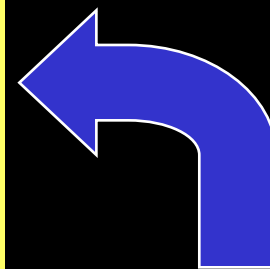
```
as UnPv
```

- 只要原結果集 Pv 裡面有  $n$  筆非 NULL 的交會方格，那麼在 UnPv 中就會有  $n$  筆資料出現



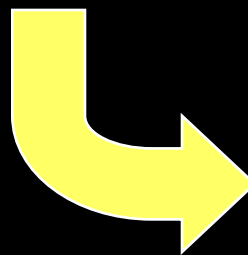
# Unpivot 指令

| <i>bookname</i> | 巨蟹書局 | 射手書局 | 水瓶書局 | 天秤書局 | 獅子書局 |
|-----------------|------|------|------|------|------|
| 三國演義            | 30   | 30   | NULL | NULL | NULL |
| 水經注             | 10   | NULL | NULL | 40   | NULL |
| 水滸傳             | 20   | 40   | 20   | 20   | NULL |
| 西遊記             | 20   | NULL | NULL | 30   | NULL |
| 紅樓夢             | 40   | NULL | NULL | NULL | NULL |
| 道德經             | 10   | NULL | NULL | NULL | NULL |



Pivot

Unpivot



| <i>name</i> | <i>Bookname</i> | <i>quantity</i> |
|-------------|-----------------|-----------------|
| 巨蟹書局        | 三國演義            | 30              |
| 射手書局        | 三國演義            | 30              |
| 巨蟹書局        | 水經注             | 10              |
| 天秤書局        | 水經注             | 40              |
| 巨蟹書局        | 水滸傳             | 20              |
| 射手書局        | 水滸傳             | 40              |
| 水瓶書局        | 水滸傳             | 20              |
| 天秤書局        | 水滸傳             | 20              |
| 巨蟹書局        | 西遊記             | 20              |
| 天秤書局        | 西遊記             | 30              |
| 巨蟹書局        | 紅樓夢             | 40              |
| 巨蟹書局        | 道德經             | 10              |



# Try ... Catch 區塊

---

- 新的程式語言大多已提供 Try ... Catch 區塊，讓整體的例外錯誤處理能夠集中管理。
- SQL Server 2008 也加上了類似的功能：

## Begin Try

*{sql\_statement | statement\_block }*

## End Try

## Begin Catch

*{sql\_statement | statement\_block }*

## End Catch



# Try ... Catch 區塊的寫法

- 程式碼放在 Begin Try ... End Try 區塊中，而且要緊跟著 Begin Catch ... End Catch 區塊，中間不能夾著 go 或其它指令。
- 在 TRY 區塊內的 SQL 指令被偵測到錯誤狀況時，控制權會轉移到 Catch 區塊來處理
- Catch 區塊處理完例外狀況之後，控制權會接著傳遞給 End Catch 後面的第一個 SQL 指令。
- 在 Try 區塊中，一旦發生錯誤，則該錯誤 SQL 指令後面的指令都不會被執行。
- 如果 Try 區塊在執行過程中沒有錯誤，則控制權會跳過緊接著的 End Catch，並從其後面的 SQL 指令繼續執行。



# Try ... Catch 區塊的錯誤代碼

---

- 錯誤代碼相當多，無法一一列舉，僅列出常用的：
  - `Error_Number()`：傳回錯誤號碼。
  - `Error_Message()`：傳回錯誤訊息的完整文字，例如：物件名稱與時間。
  - `Error_Severity()`：傳回錯誤嚴重性。
  - `Error_State()`：傳回錯誤狀態代碼。
  - `Error_Line()`：傳回導致錯誤的列號。
  - `Error_Procedure()`：傳回發生錯誤的預儲程序或觸發程序之名稱。



# Try ... Catch 區塊的範例

---

If Object\_ID('GetErrorInformation', 'P') is not null

Drop Procedure GetErrorInformation

Go

Create Procedure GetErrorInformation

As Select Error\_Number() as 錯誤號碼, Error\_Severity() as 錯誤嚴重性,  
Error\_State() as 錯誤狀態代碼, Error\_Procedure() as 錯誤程序,  
Error\_Line() as 錯誤列號, Error\_Message() as 錯誤訊息

Go

Begin Try

Select \* into Books from Bookstores

End Try

Begin Catch

Exec GetErrorInformation

End Catch



# DDL 觸發程序 (DDL Trigger)

---

- 可以監督 DDL 動作
- 監督的範圍可以包含整個伺服器 (在語法中指定 **on All Server**)，或整個資料庫 (在語法中指定 **on Database**)
- 沒有支援 **instead of** 的寫法
- 主要是用於控管資料庫物件本身的管理工作，監督的是 **Create**、**Alter** 和 **Drop** 指令，不是 **Insert**、**Delete** 和 **Update** 指令。



# DDL 觸發程序 (DDL Trigger)

---

- 建立 DDL Trigger 的基本語法如下：

```
Create Trigger trigger_name
 on {All Server | Database}
 {for | after} {event_type [... n] | DDL_Database_Level_Events }
as
 {sql_statement | statement_block}
```

- *event\_type* [... *n*] 定義了會觸發此程序的事件列表 (例如：Drop\_Table，其餘如下頁的表所列，其中打上「\*」的不能用在 on Database 選項上)

# 可用來設定觸發 DDL Trigger 的事件類型

|                               |                              |                             |
|-------------------------------|------------------------------|-----------------------------|
| Create_Database*              | Alter_Database               | Deny_Database               |
| Create_Application_Role       | Alter_Application_Role       | Drop_Application_Role       |
| Create_Assembly               | Alter_Assembly               | Drop_Assembly               |
| Create_Certificate            | Alter_Certificate            | Drop_Certificate            |
| Create_Function               | Alter_Function               | Drop_Function               |
| Create_Index                  | Alter_Index                  | Drop_Index                  |
| Create_Login*                 | Alter_Login*                 | Drop_Login*                 |
| Create_Message_Type           | Alter_Message_Type           | Drop_Message_Type           |
| Create_Partition_Function     | Alter_Partition_Function     | Drop_Partition_Function     |
| Create_Partition_Scheme       | Alter_Partition_Scheme       | Drop_Partition_Scheme       |
| Create_Procedure              | Alter_Procedure              | Drop_Procedure              |
| Create_Queue                  | Alter_Queue                  | Drop_Queue                  |
| Create_Remote_Service_Binding | Alter_Remote_Service_Binding | Drop_Remote_Service_Binding |
| Create_Role                   | Alter_Role                   | Drop_Role                   |
| Create_Route                  | Alter_Route                  | Drop_Route                  |
| Create_Schema                 | Alter_Schema                 | Drop_Schema                 |
| Create_Service                | Alter_Service                | Drop_Service                |
| Create_Table                  | Alter_Table                  | Drop_Table                  |
| Create_Trigger                | Alter_Trigger                | Drop_Trigger                |
| Create_User                   | Alter_User                   | Drop_User                   |
| Create_View                   | Alter_View                   | Drop_View                   |
| Create_Xml_Schema_Collection  | Alter_Xml_Schema_Collection  | Drop_Xml_Schema_Collection  |
| Create_Endpoint*              | Alter_Authorization_Database | Drop_Endpoint*              |
| Create_Event_Notification     | Alter_Authorization_Server*  | Drop_Event_Notification     |
| Create_Statistics             | Update_Statistics            | Drop_Statistics             |
| Revoke_Server*                | Grant_Server*                | Deny_Server*                |
| Revoke_Database               | Grant_Database               | Drop_Database*              |
| Create_Synonym                | Create_Type                  | Drop_Synonym                |
| Create_Contract               | Create_Xml_Index             | Drop_Contract               |





# DDL 觸發程序 (DDL Trigger)

---

- 不會使用 inserted/deleted 關聯表，而是函數 eventdata()，以回傳引發事件的相關資料，
- 因為回傳資料內容豐富，所以 eventdata() 以層層包覆的 XML 資料型態來回傳。
- 以下用一個 DDL Trigger 來印出 eventdata() 回傳的資料樣式
- 測試前請先執行下面指令將 Orders 資料複製到 NewOrders：  

```
select * into NewOrders from Orders
```



# DDL 觸發程序 (DDL Trigger)

---

- 撰寫 DDL Trigger 如下：

```
create trigger Print_EventData on database for
drop_table as select eventdata()
```

- 接著，請執行下面指令將 NewOrders 刪除

```
drop table NewOrders
```

- 指令會正確刪除 NewOrders，而且系統會回傳 EventData() 的值如下頁所示...



# 回傳EventData() 的內容

---

```
<EVENT_INSTANCE>
<EventType>DROP_TABLE</EventType>
<PostTime>2006-09-06T17:17:41.827</PostTime>
<SPID>62</SPID>
<ServerName>KIDLAB</ServerName>
<LoginName>KIDLAB\Administrator</LoginName>
<UserName>dbo</UserName>
<DatabaseName>BOB</DatabaseName>
<SchemaName>dbo</SchemaName>
<ObjectName>NewOrders</ObjectName>
<ObjectType>TABLE</ObjectType>
<TSQLCommand>
 <SetOptions ANSI_NULLS="ON" ANSI_NULL_DEFAULT = "ON"
 ANSI_PADDING = "ON" QUOTED_IDENTIFIER="ON" ENCRYPTED="FALSE" />
 <CommandText>drop table NewOrders</CommandText>
</TSQLCommand>
</EVENT_INSTANCE>
```



## 取出 EventData() 回傳的個別資料

---

- 利用 XML 資料型態上的 query() 方法，再配合 XQuery 中的 Xpath 路徑指引就可以取出表格的名稱
- 做法上可以宣告 EventLog 關聯表用來存放上述 EventData() 回傳資料的元素內容：

```
Create Table EventLog (
PostTime datetime, EventType char(30), ServerName char(50),
LoginName char(100), UserName char(100), DatabaseName char(50),
SchemaName char(30), ObjectName char(30), ObjectType char(30),
TSQLCommand nvarchar(max), CommandText nvarchar(max),
Primary key (PostTime))
```



# 取出 eventdata() 回傳的個別資料

- 再寫 DDL Trigger 如下，將內容存入 EventLog：

```
create trigger Get_EventData on database for drop_table
```

```
As
```

```
declare @Eventdata xml
```

```
select @EventData = eventdata()
```

```
truncate table EventLog
```

```
insert into EventLog(PostTime, EventType, ServerName, LoginName, UserName,
DatabaseName, SchemaName, ObjectName, ObjectType, TSQLCommand, CommandText)
Values (getdate(), convert(char(30), @EventData.query('data(//EventType)'),
convert(char(50), @EventData.query('data(//ServerName)'),
convert(char(100), @EventData.query('data(//LoginName)'),
CURRENT_USER,
convert(char(50), @EventData.query('data(//DatabaseName)'),
convert(char(30), @EventData.query('data(//SchemaName)'),
convert(char(30), @EventData.query('data(//ObjectName)'),
convert(char(30), @EventData.query('data(//ObjectType)'),
convert(nvarchar(max), @EventData.query('data(//TSQLCommand)'),
convert(nvarchar(max), @EventData.query('data(//CommandText)'))
```



# @EventData.query(XQuery) 的說明

- query(XQuery) 是 SQL Server 2008 為了因應 XML 資料類型新增的方法 (Method) :
  - XQuery 參數是字串類型的 XQuery 運算式 (被 W3C 認可的 XML 查詢語言)
  - query(XQuery) 方法會針對傳入參數依照 XQuery 的標準來查詢 XML 執行個體中的元素、屬性等節點，並回傳不具型態的 XML 資料
  - @EventData.query('data(//EventType)') 裡面所包含的 //EventType 為 XPath 路徑表示法
    - 雙斜線 (//) 表示從根節點 (EVENT\_INSTANCE) 往下找，只要找到匹配元素節點名稱 (例如：ObjectName) 即表示定位完成；
    - data() 用來取出被定位到的元素之中所夾起來的資料
    - 所以 @EventData.query('data(//ObjectName)') 會回傳 NewOrders。
    - XPath 中也可以用單斜線 (/) 像檔案路徑一樣定位到所要的元素
    - 上述的 //EventType 也可以換成 /EVENT\_INSTANCE/ EventType，取得的答案是一樣的



# 其它的 XML 資料型態方法 (Method)

- **exist(XQuery) 方法**：用來判斷查詢是否傳回非空的結果：1 代表 True；0 代表 False，即查詢傳回空的結果。
- 例如：
  - 新增以下關聯表  
create table XMLBooks(id int primary key,bookname char(50) not null,author xml)
  - 加入以下兩筆值組  
Insert into XMLBooks values(1, '資料庫系統理論與實務', '<作者群>  
<作者 姓名="曾守正" 性別="男"/>  
<作者 姓名="周韻寰" 性別="女"/></作者群>')  
Insert into XMLBooks values(2, 'SQL Server 2008使用指引', '<作者群>  
<作者 姓名="王小華" 性別="男"/>  
<作者 姓名="林大明" 性別="男"/></作者群>')
  - 使用下述查詢，就可知 XMLBooks 中是否有書籍含有女性作者  
Select bookname, author.exist('/作者群/作者[@性別="女"]') as 含女性作者  
From XMLBooks



# 其它的 XML 資料型態方法 (Method)

---

- `value('XQuery', 'SQLType')` 方法：用來擷取 XML 執行個體的 SQL 類型值。
  - 此方法的傳回類型符合 `SQLType` 參數，
  - 它與 `query(XQuery)` 最大的不同點在於：其回傳值只能是單一的純量值 (Single Scalar)，而 `query(XQuery)` 則是回傳 XML 型式的資料
- 例如：用下列 SQL 查詢書籍的第一作者：  

```
Select bookname, author.value('/作者群[1]/作者[1]/@姓名',
'varchar(20)') as 第一作者
From XMLBooks
```





# 其它的 XML 資料型態方法 (Method)

---

- **nodes (XQuery) as Table(Column) 方法**：此方法可以將 XML 資料中結構一樣的多個節點當成關聯表 *Table(Column)* 來看待
  - 做法是利用 XQuery 將 XML 中的元素資料萃取出來，並將一個節點看成一筆值組，形成表格 *Table(Column)* 中所含的多筆值組。
  - 回傳結果是只含單一屬性的 *Table(Column)* 關聯表
  - 整個動作也稱為將 XML 資料「碎片化」(Shredding)。
- 例如：請接下頁...

# nodes (XQuery) as *Table(Column)* 方法的查詢範例

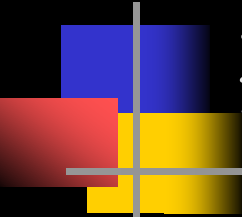
```
declare @books xml
set @books = '<Book><name>資料庫系統理論與實務</name>
 <作者群><作者 姓名="曾守正" 性別="男"/>
 <作者 姓名="周韻寰" 性別="女"/>
</作者群>
</Book>
<Book><name>SQL Server 2008使用指引</name>
 <作者群><作者 姓名="王小華" 性別="男"/>
 <作者 姓名="林大明" 性別="男"/>
</作者群>
</Book>'
```

```
select Books.Book.query('.') as result
from @books.nodes('/Book') Books(Book)
```

| result                                                                                                |
|-------------------------------------------------------------------------------------------------------|
| <Book><name>資料庫系統理論與實務</name><作者群><作者 姓名="曾守正" 性別="男" /><作者 姓名="周韻寰" 性別="女" /></作者群></Book>           |
| <Book><name>SQL Server 2008 使用指引</name><作者群><作者 姓名="王小華" 性別="男" /><作者 姓名="林大明" 性別="男" /></作者群></Book> |

```
select Books.Book.value('text()[1]', 'varchar(30)') as result
from @books.nodes('/Book/name') Books(Book)
go
```

| result               |
|----------------------|
| 資料庫系統理論與實務           |
| SQL Server 2008 使用指引 |



# 拆解 <作者群> 中的資料也可以

- 進一步將內部的 <作者群> 中的資料也拆解出來的寫法：

```
Select bookname, a.value('@姓名', 'varchar(20)') as author_name
from xmlbooks cross apply author.nodes('/作者群/作者') as Result(a)
```

| <i>bookname</i>      | <i>Author_name</i> |
|----------------------|--------------------|
| 資料庫系統理論與實務           | 曾守正                |
| 資料庫系統理論與實務           | 周韻寰                |
| SQL Server 2008 使用指引 | 王小華                |
| SQL Server 2008 使用指引 | 林大明                |



# 其它的 XML 資料型態方法 (Method)

- `xml.modify('insert expression1 {as first | as last} {into | after | before} expression2')`：用於直接將某些有順序的 XML 元素加入 `xml` 資料中的某個元素節點，當作其 Children 或 Siblings。
- 下列指令將一個新作者加到作者群中，排在最後面 (如果將 `as last` 換成 `as first`，那就是要加在最前面)：

```
declare @x xml
set @x = '<作者群><作者 姓名="王小華" 性別="男"/>
 <作者 姓名="林大明" 性別="男"/></作者群>'
set @x.modify('insert <作者 姓名="曾守正" 性別="男"/> as last into /作者群[1]')
select @x
```



## 其它的 XML 資料型態方法 (Method)

---

- `xml.modify('delete expression')`：可以用來直接將某個 XML 元素從 `xml` 中刪除。
- 例如，下列指令會將第一位作者刪除：

```
declare @x xml
set @x = '<作者群><作者 姓名="王小華" 性別="男"/><
作者 姓名="林大明" 性別="男"/></作者群>'
set @x.modify('delete (//作者)[1]')
select @x
```



# 其它的 XML 資料型態方法 (Method)

- `xml.modify('replace value of expression1 with expression2')` : 可以用來更改某個元素的本文值。
- 例如：下列指令會將第一位作者(“王小華”)的姓名改成“王小明”：

```
declare @x xml
```

```
set @x = '<作者群><作者 姓名="王小華" 性別="男" /><作者 姓名="林大明" 性別="男" /></作者群>'
```

```
set @x.modify('replace value of (/作者群/作者/@姓名)[1] with "王小明"')
```

```
select @x
```



## 另一種改法

---

- 下列指令會將第一位作者(“王小華”)的姓名本文改成 “王小明”：

```
declare @x xml
```

```
set @x = '<作者群><作者 性別="男">王小華</作者>
<作者 性別="男">林大明</作者></作者群>'
```

```
set @x.modify('replace value of (/作者群/作者/text())[1] with
"王小明"')
```

```
select @x
```



# DDL Trigger 與 DML Trigger 的比較

| DDL Trigger                       | DML Trigger                          |
|-----------------------------------|--------------------------------------|
| 1. 監督範圍包含整個伺服器，或整個資料庫             | 1. 監督範圍為單一表格                         |
| 2. 沒有支援 instead of 的寫法            | 2. 支援 instead of 的寫法                 |
| 3. 監督以 Create、Alter 和 Drop 為開頭的指令 | 3. 監督以 Insert、Delete 和 Update 為開頭的指令 |
| 4. 從 eventdata() 函數取得事件相關資料       | 4. 從 inserted 與 deleted 取得事件相關資料     |





# 系統目錄的查詢與異動

---

- 系統目錄的時機是：使用者 (非建立者) 對關聯表的綱要不熟悉時，則可透過 SQL 去查看
  - 系統中有那些資料庫？
  - 資料庫中有那些關聯表？
  - 關聯表中有那些欄位？
  - 資料型別 (值域) 為何？
  - 關聯表使用那個欄位當主鍵？
  - 關聯表是否有建索引？
  - 可以直接查詢 [系統檢視] `sys.objects`、`sys.database`、`sys.tables`、`sys.columns`、`sys.types` 或甚至是 `sys.indexes` 等。



# 查看系統目錄的靜態資料

---

- `select * from sys.databases`

- Use BOB

`go`

`SELECT * from sys.objects where type = 'U'`

`SELECT * from sys.tables`      `-- 結果與上面指令一樣`



# 查看系統目錄的靜態資料

---

- 想知道 BOB 資料庫中，各關聯表含有哪些欄位？使用什麼資料型態？只要將sys.tables、sys.columns 與 sys.types 合併起來，即可：

```
select c.name as name, t.name as Tbname, p.name as coltype
from sys.types p, sys.columns c, sys.tables t
where c.object_id = t.object_id
and c.system_type_id = p.system_type_id
order by Tbname, name
```



# 查看系統目錄的靜態資料

---

- 想知道關聯表建有哪些索引，就合併 `sys.tables` 與 `sys.indexes`：

```
select t.name as tbname, i.name as name
from sys.indexes i, sys.tables t
where i.object_id = t.object_id
order by tbname, name
```



# 查看系統目錄的靜態資料

---

- Select \* from INFORMATION\_SCHEMA.TABLES
- Select \* from INFORMATION\_SCHEMA.VIEWS
- Select \* from INFORMATION\_SCHEMA.COLUMNS
- Use BOB  
go  
exec sp\_help  
exec sp\_columns @Table\_Name = 'Books'  
exec sp\_helpindex @ObjName = 'Books'  
exec sp\_statistics @table\_name= 'Books'



# 查看系統目錄的動態資料

- 還可以查詢資料庫管理系統與作業系統 (Operating System) 在執行過程的相關動態資訊 (Dynamic Message) 。
  - 如：記憶體的使用量、目前共有連線、哪些執行緒 (Thread) 等，
  - 通常會使用到下列幾個以dm\_ 為開頭的 [系統檢視]：  
select \* from sys.dm\_exec\_connections -- 找出目前的連線  
Select \* from sys.dm\_os\_threads -- 找出作業系統下的執行緒  
Select \* from sys.dm\_os\_memory\_pools  
Select \* from sys.dm\_tran\_locks  
Select \* from sys.dm\_exec\_sessions



# 系統目錄的異動

---

- 不可以用 SQL 直接對系統目錄下達新增、刪除或更改的動作
- **系統目錄的新增**：透過 Create Table, Create Index, Create View, sp\_adduser、sp\_addlogin、...間接對系統目錄新增
- **系統目錄的刪除**：透過 Drop Table, Drop Index, Drop View, sp\_dropuser、sp\_droplogin 間接對系統目錄刪除
- **系統目錄的更新**：透過 Alter Table, sp\_rename、sp\_renamedb、sp\_password、...間接對系統目錄更新

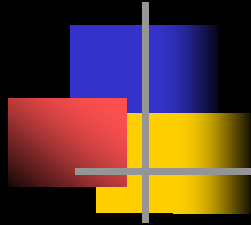


# 加速 SQL 查詢的技巧

---

- DBMS 會主動做查詢最佳化的工作
- 但人腦總是比電腦清楚所要的是什麼
- 請參考 “Optimizing SQL” by P. Gultzan and T. Pelzer (1995)
- 沒有絕對的鐵律可以告訴我們：如何查詢最好？因為那與資料庫當時的內容大小有關
- 建設性的作法是採取正確的資料庫效能調校策略，請見 <http://www.sql-server-performance.com>





本章結束  
The End.