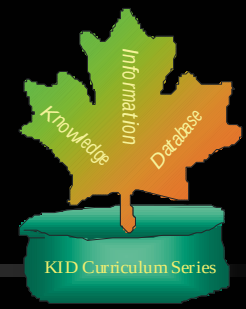
第八章



結構化查詢語言 SQL

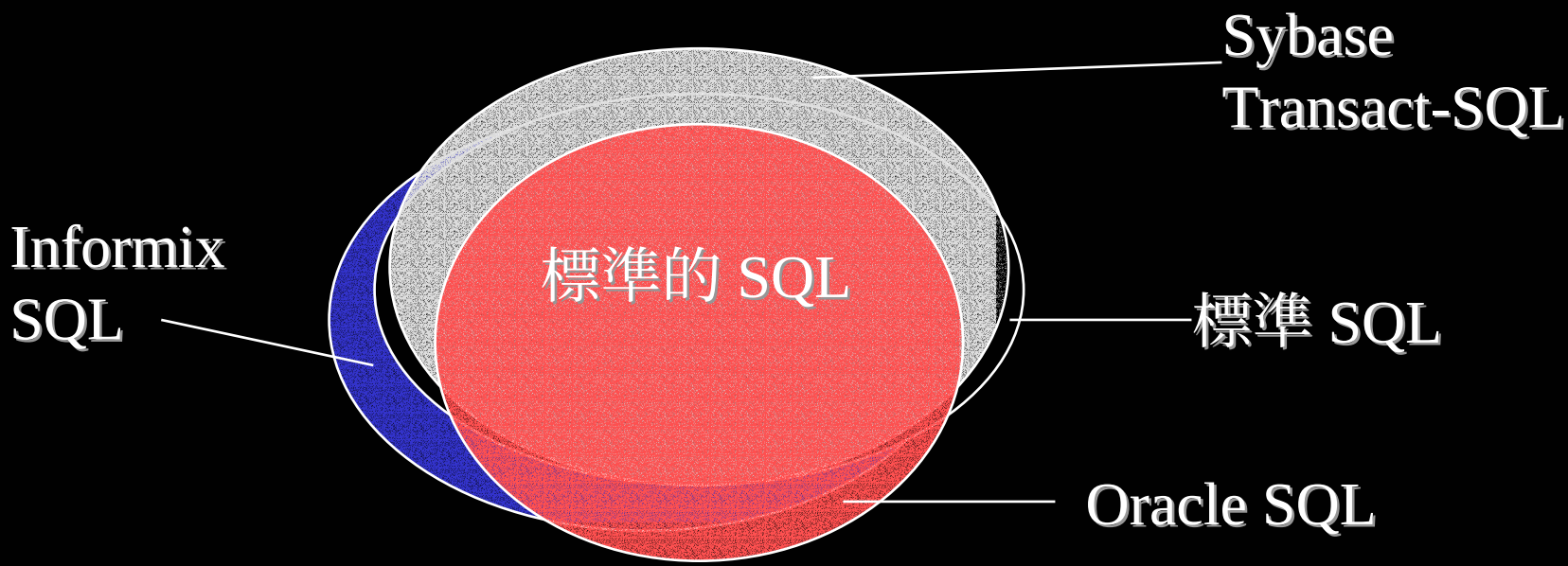
Frank S.C. Tseng
(http://www2.nkfust.edu.tw/~imfrank)

本章內容

- 8.1 簡介
- 8.2 資料定義語言 (DDL)
- 8.3 資料處理語言 (DML) 的查詢部份
- 8.4 資料處理語言 (DML) 的異動部份
- 8.5 資料控制語言 (DCL)
- 8.6 SQL 與關聯式代數之間的對應
- 8.7 嵌入式 SQL (Embedded SQL)
- 8.8 動態 SQL
- 8.9 SQL Server 的預儲程序與觸發程序
- 8.10 系統目錄的查詢與異動
- 8.11 加速查詢的一些原則與技巧

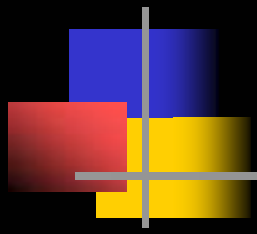
簡介

- SQL-86→SQL-89→SQL-92→SQL 3 (1999)
- 廠商所提供的大多是 Superset of a Subset



SQL 的三類命令

- 資料定義語言 (Data Definition Language, DDL)：定義資料庫的綱要
- 資料操縱語言 (Data Manipulation Language, DML)：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



資料定義語言 (DDL)

- 基底關聯表 (Base Table) 的 DDL
- 視界關聯表 (View) 的 DDL
- 資料索引 (Index) 的 DDL
- 新增資料型態
  - 1> sp_addtype type_of_age, 'tinyint', 'not null'
  - 2> sp_addtype type_of_color, 'char(12)', 'not null'
  - 3> sp_addtype type_of_price, 'tinyint', 'null'

Create Rule 限制規則

- 1> create rule rule_of_age as @age between 18 and 65
- 2> create rule rule_of_color as @color in ('Red', 'Yellow', 'Blue', ...)
- 3> create rule rule_of_price as @price between 80 and 250

Create Default 設定內定值為何？

- 1> create default default_of_age as 25
- 2> create default default_of_color as 'Red'
- 3> create default default_of_price as 100

將屬性與 Rule/Default Bind

- 1> sp_bindrule 'rule_of_price' 'Books.price'
- 2> sp_bindefault 'default_of_price' 'Books.price'

Unbind Rule/Default

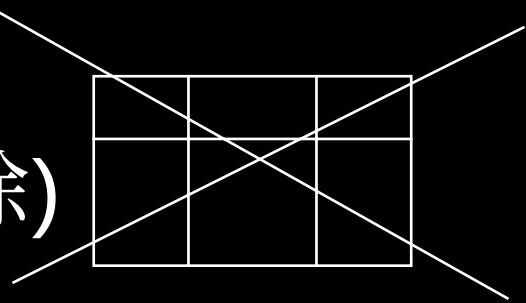
- 1> sp_unbindrule 'Books.price'
- 2> sp_unbindefault 'Books.price'

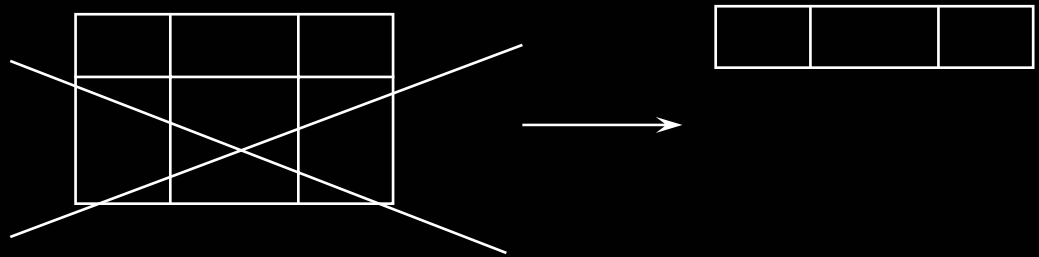
刪除自定型態

- 1> sp_droptype type_of_age

基底關聯表上的 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* (
  colums-name data-type [null | not null], ...
  [, primary key (*column-commalist*)]
  [, foreign key (*column-commalist*)
  references *base-table* [(*column-commalist*)]
  [on delete *option*] (SQL Server 2000 有提供)
  [on update *option*] ,...] (SQL Server 2000 有提供)
  [, check (*condition*)]
)

常用的資料型態

- bigint (整數資料型態) 佔 8 個位元組 (新增)
- int (整數資料型態) 佔 4 個位元組
- smallint (整數資料型態) 佔 2 個位元組
- tinyint (整數資料型態) 佔 1 個位元組
- char(n) (字元資料型態) 佔 *n* 個位元組
- varchar(n) (字元資料型態) 最多佔 *n* 位元組
- float (實數資料型態) 佔 8 個位元組
- real (實數資料型態) 佔 4 個位元組
- bit (整數資料型態) 佔 1 個位元

常用的資料型態 (續)

- datetime (日期資料型態) 佔 8 個位元組
- smalldatetime (日期資料型態) 佔 4 個位元組
- money (代表金錢的數值) 佔 8 個位元組
- smallmoney (代表金錢的數值) 佔 4 個位元組
- Table (SQL Server 2000 新增的資料型態)
- image (存放影像資料) 最多可以到 2G
- text (文章) (存放本文資料) 最多可以到 2G

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

Collation 的設定

- 想知道系統中有多少 Collation，還可以使用下列指令來查詢：
  - 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；
  - CS 是 Case Sensitive 的縮寫，表示將大、小寫字母視為不同；
  - AI 是 Accent Insensitive 的縮寫，表示忽略抑、重音符號，

Alter Table—更改綱要

- 加入一個新的欄位。
- 定義/刪除一個已存在欄位之內定值。
- 刪除一個已存在的欄位。(SQL Server 2000 可以)
- 加入一條新的整合限制條件到該關聯表上。(可以用觸發程序來達成)。
- 刪除關聯表上已存在的整合限制條件。
- 將現有的欄位名稱改名。(在 SQL Server 中可以用 sp_rename 為之)

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)

- 刪除一個已存在的欄位—其通式為：
  alter table *base-table-name* **drop column** *column-name*
- 新定義一個已存在欄位之內定值—其通式為：
  alter table *base-table-name* **add constraint** *constraint-name* **default** *constant-expression* **for** *column-name*

Alter Table—更改綱要 (續)

- 刪除一個已存在的欄位之內定值—其通式為：

  alter table *base-table-name* drop constraint *constraint-name*

  其中的 *constant-name* 必須要與先前所給的名稱一致。

- 加入一條新的檢查條件到該關聯表上—其通式為：

  alter table *base-table-name* add constraint *constraint-name* check (*expression*)

  其中的 *expression* 所算出來的結果必須是一個可以產生布林代數的值 (True/False)

Alter Table—更改綱要 (續)

- 刪除該關聯表上一條已存在的整合限制條件—其通式與上頁投影片(刪除內定值)一樣。

  不論是加入「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。

- 改變現有欄位的資料型態—其通式為：

  alter table *table-name* alter column *column-name new_data_type*

Alter Table—更改綱要 (續)

- 若要將現有欄位的名稱改名，在SQL Server中可以用 sp_rename 來做。舉例來說，以下的指令會將 Books 中的 *bookname* 欄位改成 *name*：

```
sp_rename 'Books.bookname', 'name', 'column'
```

- 在 MS SQL Server 上做綱要修正，只要使用SQL Server Enterprise Manager，在進入資料庫後於表格上按滑鼠右鍵，點選 [Design Table] 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*

視界上的 DDL

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

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

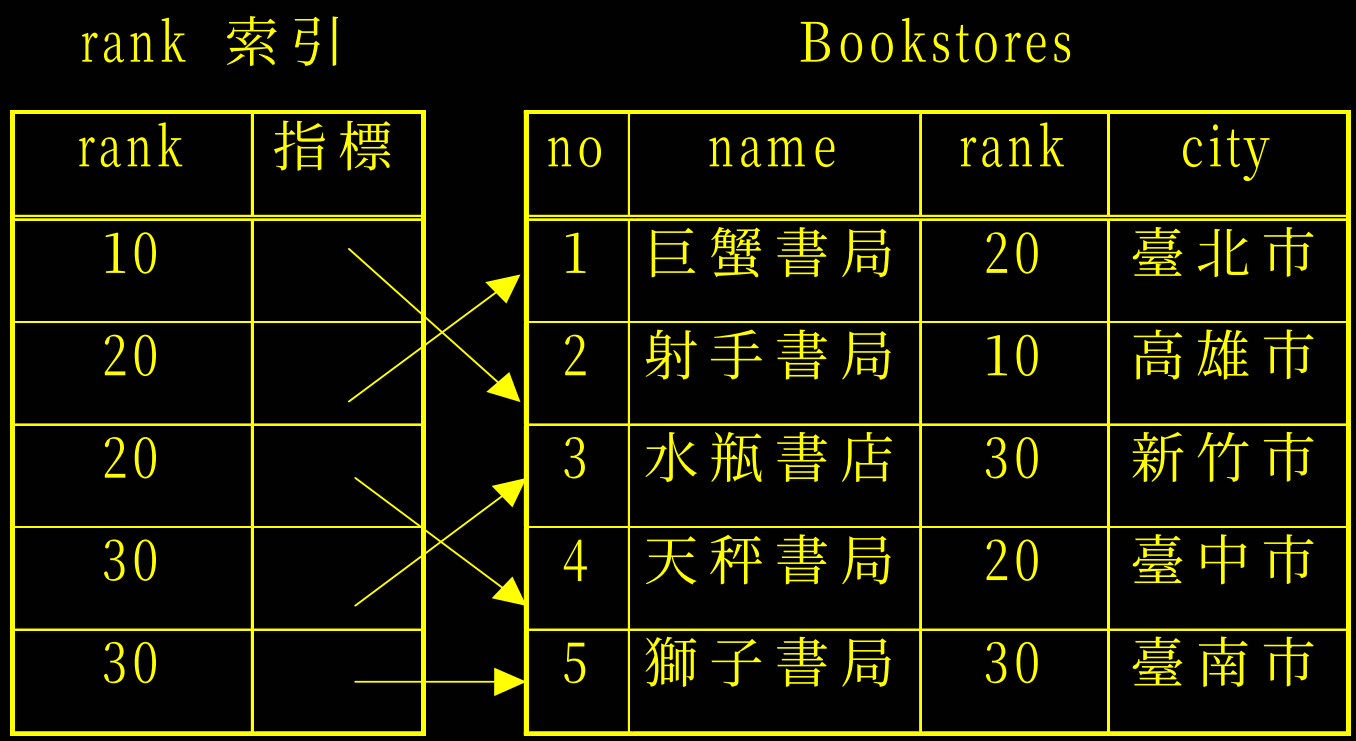
產生視界→

Odd_No_Bookstores

no	*name*	*rank*	*city*
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

索引上的 DDL

- Primary index vs. Secondary index (如下例)



非密集索引 (Nondense index)

- 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* (
  (*colums-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*

資料處理語言 (DML)

- **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** *a-list-of-attribute-name*
  [**Having** *a-true/false group-condition*]] (跟著 Group by)
  [**Order by** *a-list-of-attribute-name*]
  (查詢結果可建立另一關聯表)

Select 子句

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

Books

id	*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

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

price
120
170
140
190

bookname		*new_price*
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)

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
```

no	*name*	*rank*	*city*	*no*	*id*	*quantity*
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)

First.*name*	Second.*name*
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 First.no < 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*
巨蟹書局	巨蟹書局
射手書局	射手書局
水瓶書店	水瓶書店
天秤書局	天秤書局
獅子書局	獅子書局

First.*name*	Second.*name*
巨蟹書局	天秤書局
天秤書局	巨蟹書局
水瓶書店	獅子書局
獅子書局	水瓶書店

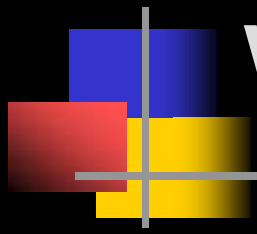
刪除多餘部份

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
```

no	*name*	*rank*	*city*
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 2000 現在允許使用 = 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 */
```

等位合併時對 Null 的處理

- 做「等位合併」時，會發現有些合併欄位中含有 **null**，這時合併欄位值為 **null** 的那些值組通常都會被捨棄，而使得結果比真正的資料要來得少。

Students

no	*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	—	民生路x號
5	大衛	B	—	仁愛路三段x號

Zipdata

zip	*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

做合併查詢時的差別

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

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

InnerJoin vs. Left OuterJoin

結果比較

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

漏了兩筆

no	name	class	zip	address	district
1	瑪莉	A	100	中華路一段x號	台北市中正區
2	志明	B	104	中正路二段x號	台北市中山區
3	阿倫	A	105	中山北路一段x號	台北市松山區
4	春嬌	B	—	民生路x號	—
5	大衛	B	—	仁愛路三段x號	—

不全的兩筆可用人工補足

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'
```

no	*name*	*rank*	*city*
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')
```

no	*name*	*rank*	*city*
2	射手書局	10	高雄市
3	水瓶書局	30	新竹市

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')
```

no	*name*	*rank*	*city*
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

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)
```

no	*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

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

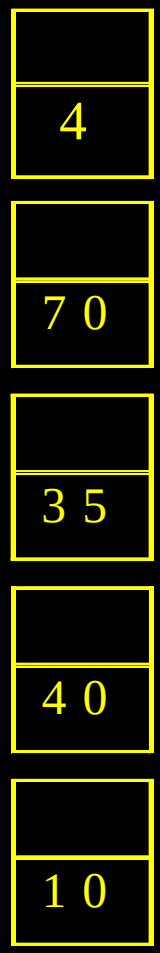
5

Bookstores

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

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



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

Group by 子句

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

no	
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

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

Group by 子句 (續)

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

no	
1	130
2	70
3	20
4	90

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

Group by 子句 (續)

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

no	
1	21.66
2	35
3	20
4	30

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

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

no	
1	130
4	90

no	
1	130
2	70
3	20
4	90

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

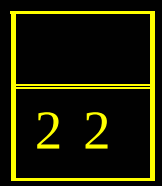
巢狀式查詢 (Nested Query)

- 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
```



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

- 再算出

```
select name, rank
from Bookstores
where rank < 22
```

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)
```

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

- 再算出

```
select name
from Bookstores
where rank = 20
```

name
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

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

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」
- 先將第一筆取出後，以Bookstores.no = 1 去執行 select *

1	巨蟹書局	20	臺北市

```
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

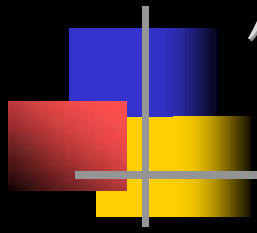
傳回集合的巢狀子查詢 (續)

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

≡

- select 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

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

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

Books

id	*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

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

數層含有 in 的巢狀查詢

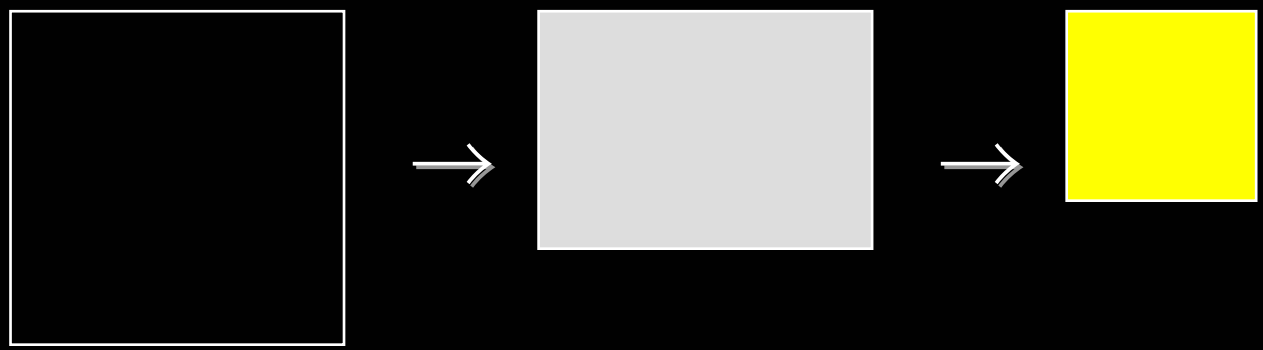
- select distinct name from Bookstores where no in (select no from Orders where id in (
  select id from Books))

  ||

- 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)

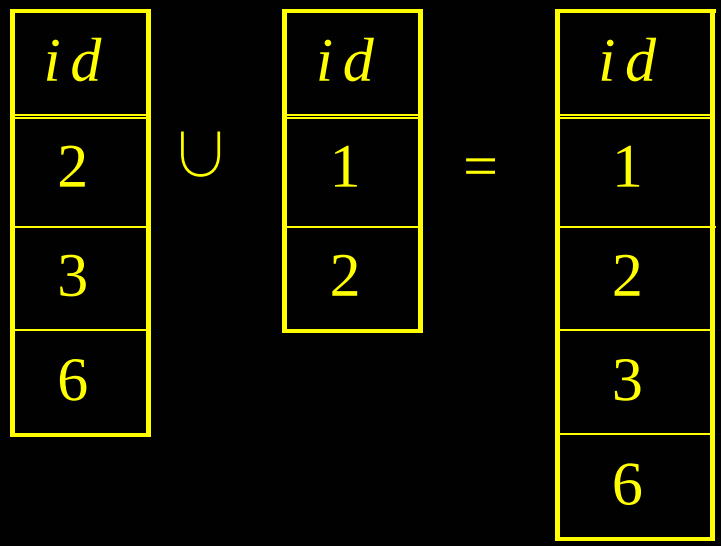
```
select no
from Bookstores
where not exists
        (select *
        from temp
        where not exists
                (select *
                from Orders
                where no = Bookstores.no
                and id = temp.id))
```

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



- 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 Orders.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*	*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
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)

bookname				
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

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

DML 的異動部份

- insert into *tablename* [(*field1, field2, …, fieldi*)]
  values (*exp1, exp2, …, expi*) | *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_book_sold

id	*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

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

id	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

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

刪除條件的另一種寫法

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

較易理解

=

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

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*, ... where *predicate*
- update Bookstores set name = '元智書坊', rank = rank + 10, city = '中壢市' where no = 2

Bookstores

no	*name*	*rank*	*city*
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

id	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

no	id	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

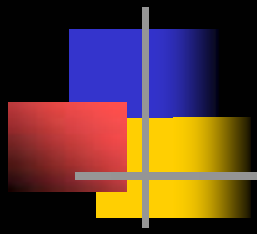
- update Orders set quantity = 0 where 120 = (select price from Books where Books.id = Orders.id)

較易理解

=

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

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

no	…	*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

no	*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

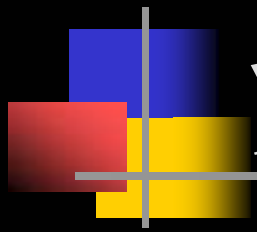
ToBeUpdated

no	*Address*	*Cell_phone*
A0038	花蓮市	093x123355
…	…	…
A1008	新店市	092x787866
B0035	中和市	093x566788
…	…	…
B0096	高雄市	091x002820

Members

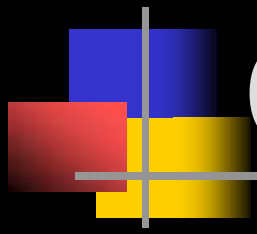
no	…	*Address*	*Cell_phone*
A0001	…	中壢市	093x112233
A0002	…	鳳山市	093x223344
…	…	…	…
A0038	…	花蓮市	093x123355
…	…	…	…
A1008	…	新店市	092x787866
…	…	…	…
A1788	…	豐原市	093x777766
B0001	…	蘆洲市	093x556688
B0035	…	中和市	093x566788
…	…	…	…
B0096	…	高雄市	091x002820
…	…	…	…
B8899	…	彰化市	095x008120

```
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)

- 主要的指令有兩個：Grant 與 Revoke 指令。
  - Grant 指令用來授予使用權限,
  - 而 Revoke 指令則是用來取回使用權限。
- 也可以直接利用 MS SQL Server 所提供的 Graphical User Interface 來建立：
  - 可由 Server 的項目下建立 Login 帳號
  - 在資料庫的 Properties 中則可以建立 User 帳號



Grant—授予使用權限

Grant {All [privilege] | *privilege_list* }

on {*table_name* [(*column_list*)] |

view_name [(*column_list*)] |

stored_proc_name}

to {Public | *user_list*}

[**with Grant option**]

Grant—授予使用權限

- *privilege_list* 中可以是 select、insert、update、delete 的列表。不同的系統可能會有不同的權限，可能還可以增加 execute、references、index 等。
- 小心使用 with Grant option，它代表被授權的使用者有權力再將此一使用權限授予其他使用者
- 用了with Grant option 後續可配合 Revoke 以 cascade 將擴散出去的使用權限回收。

使用範例

- 希望讓 Books 中的 (bookname, price, author, publisher) 讓 jesse, chris, annie, frank, mike 可以做更新以及刪除的動作時：

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

- 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 {All | *privilege_list* }
on {*table_name* [(*column_list*)] |
    view_name [(*column_list*)] |
    stored_proc_name}
from {Public | *user_list*}
[cascade]

使用範例

- 希望取回 jesse, chris, annie 對Books中的(bookname, price, author, publisher) 做更新以及刪除的使用權時，則可以下達如下的指令來取回授權：

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

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，則不需要列舉一長串的使用者名稱在 *user_list* 上，只要：
- Grant all on Books to public
- Revoke all on Books from tom, john

SQL Server 上的存取控制

- 在 SQL Server 上具有登入帳號 (Login Account) 並不表示具有存取資料庫的權限，必須還要取得該資料庫的使用權才能使用該資料庫
- 新增一個登入帳號

```
1> sp_addlogin new_loginame, …
2> go
```

- 想轉移資料庫擁有權的話，可以利用 sp_changedbowner 指令來達成

新增資料庫的使用者

- 不是擁有者但是想使用某一個資料庫的話，則必須經由該資料庫的擁有者或 sa 使用 sp_adduser 指令來將你的帳號新增為該資料庫的使用者
- login 帳號與 database user 帳號可以不一樣，但是除了sa 與 owner 的 database username 都叫 dbo 以外，一般都設成一樣

轉移資料庫的擁有權

- 資料庫的現有使用者是不能被轉移擁有權的, 如果真要轉移給這樣的對象時, 請先將此使用者由該資料庫的使用者中移除

```
1> use database /* 要先 use目標資料庫 database */
2> go
3> sp_changedbowner newowner
4> go
```

新增資料庫的使用者

```
1> use database /* 要先 use 目標資料庫 */
2> go
3> sp_adduser loginame1, loginame2, ...
     /* 可以多個帳號 */
4> go
```

- 若要移除則使用 sp_dropuser *loginname*

各種使用權的幾個觀念

- 登入 SQL Server 的 login name：只准許登入 SQL Server 而已，除了系統內建的資料庫以及自己擁有的資料庫之外，並沒有其它資料庫的使用權。
  - 任何時候我們都可以使用 select suser_sname() 來查詢自己目前的 login name

各種使用權的幾個觀念

- 資料庫的 user name：要使用任何一個資料庫當然先具備登入 SQL Server 的 login name，然後再由該資料庫的擁有者賦予資料庫的 user name。
  - 資料庫擁有者在該資料庫中的 user name一律稱為 dbo，而一般使用者的 user name 通常與其 SQL Server 的 login name 一樣，但經由 dbo 的指定，也可以不一樣。
  - 使用 select user_name() 可以取得自己在該資料庫的 user name。

各種使用權的幾個觀念

- 每一個資料庫的物件 (如：關聯表、索引等) 在建立後也將建立該物件的使用者視為其擁有者，所以可能 dbo 並不擁有其資料庫中的某個物件。
  - 由於 dbo 無法刪除擁有某個物件的使用者，所以我們建議不要賦予資料庫使用者有建立物件的權限，比較容易管理。

各種使用權的幾個觀念

- 資料庫的擁有者 dbo：也就是建立該資料庫的 SQL Server login name。如果你想知道那各個資料庫的 dbo 為何，可以用sp_helpdb 指令來查詢。
  - 注意：sa 在存取任何資料庫的 user name 都叫做 dbo。

各種使用權的幾個觀念

- 資料庫的管理者 sa：可以執行任何命令，以及使用任何資料庫。
  - 由於 sa 的權利太大，所以通常會讓不同的使用者扮演各種不同的角色 (roles) 去執行 sa 的工作，然後將 sa 這個使用者鎖住 (使用sp_lockuser *loginame*, lock 來達成)，以免一個人的權利太大，造成困擾。(但 SQL Server 2000 已經沒有此一指令了)
  - 同時不同的角色也有彼此監督的功能。

SQL vs. 關聯式代數

- $\sigma_p(R)$

  select *
  from R
  where *p*

- R ⨯ S

  select *
  from R, S

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

  select distinct *a, b, c,…, z*
  from R

SQL vs. 關聯式代數 (續)

- R – S

```
select * from R
difference                          (UniSQL/X 有提供)
select * from S
```

```
select * from R                     (標準 SQL 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)

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

  select *
  from R, S
  where *p*　(合併條件)

SQL vs. 關聯式代數 (續)

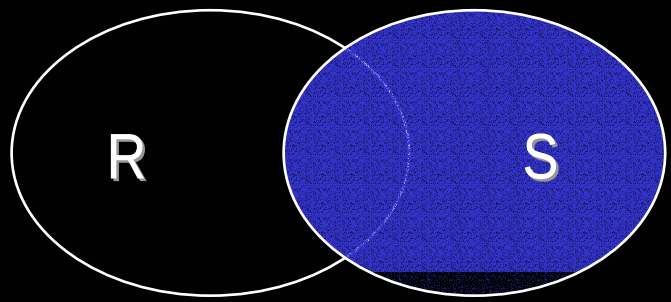
- R∩S

  select * from R
  intersect　　(UniSQL/X 有提供)
  select * from S

標準 SQL 可以用 R ⋈ S 的退化情況來模擬

或使用 R∩S = R – (R – S) = S – (S – R)

見 6.2.2.4 節



用 R ⋈ S 的退化情況來模擬

- R(*r1, r2, …, rn*) 交集 S (*s1, s2, …, 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∩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)
```

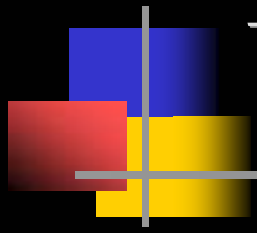
SQL *vs.* 關聯式代數 (續)

- R [X, Y] ÷ S [Y]

```
select distinct X
from R R1
where not exists
          (select  *
           from S
           where not exists
                (select *
                 from R R2
                 where R2.X = R1.X
                 and R2.Y = S.Y))
```

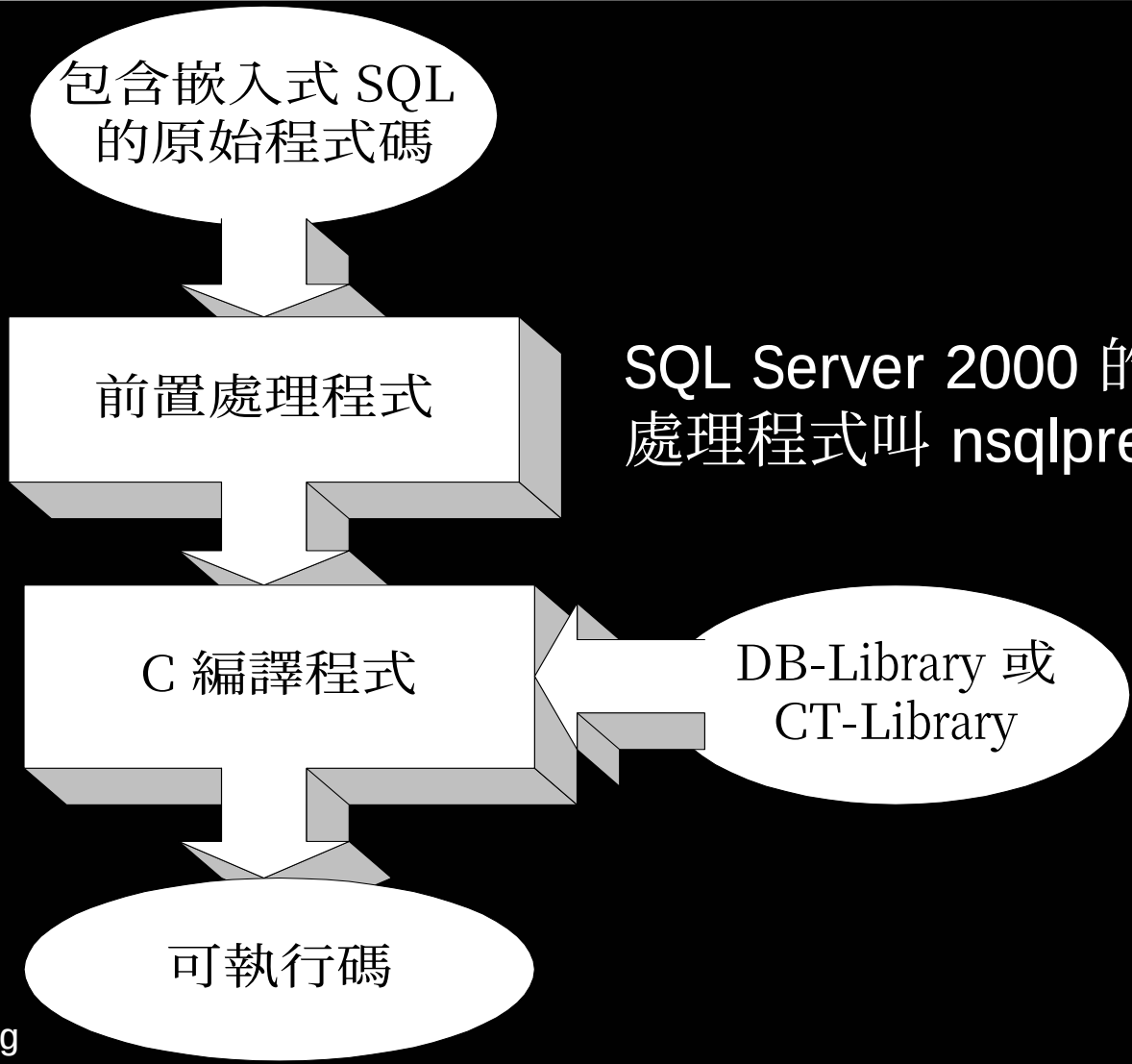
“所有” = “沒有一個不”

嵌入式 SQL (Embedded SQL)



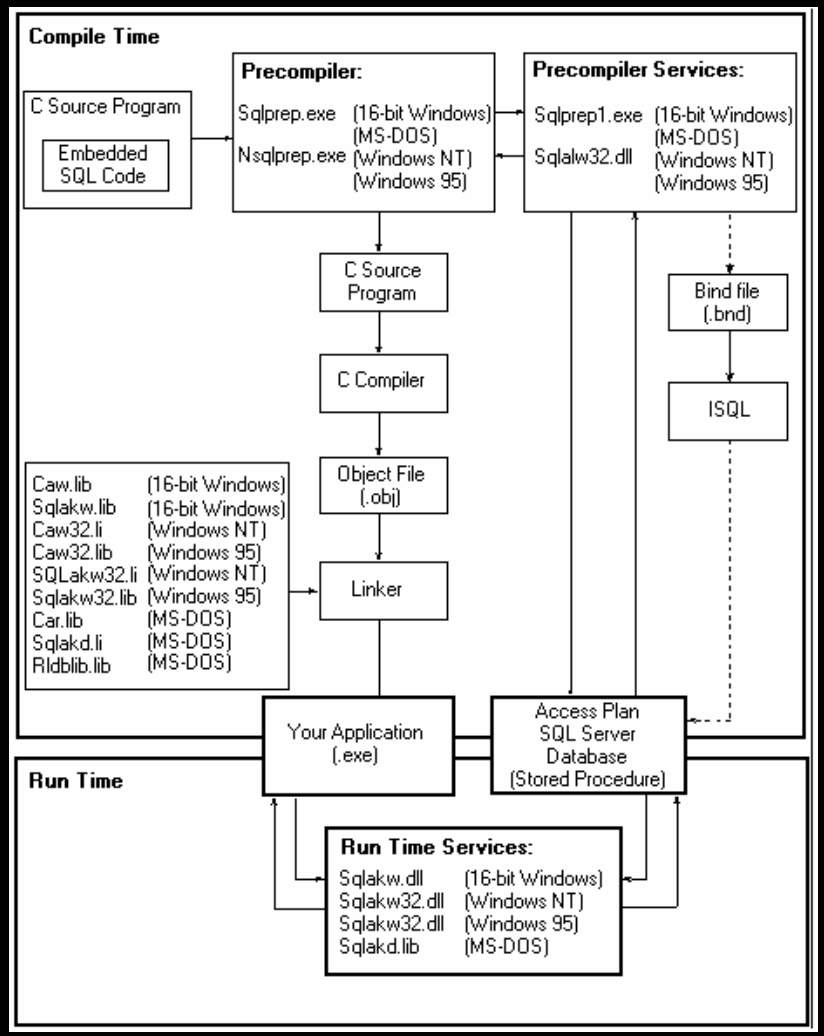
- 交談式 SQL 查詢僅能夠在線上做查詢與修改而已，所能發揮的影響有限
- 具備程式開發之能力，才能讓其影響發揮最大效力
- SQL 也可以嵌入高階的程式語言中
- 高階語言專注於輸出、入與流程之控制
- SQL 則專注於資料庫的存取、加入、刪除等 (Stored Procedures 也是)

SQL 嵌入高階語言的編譯程序



SQL Server 2000 的前置處理程式叫 nsqlprep.exe

MS SQL Server 的處理程序



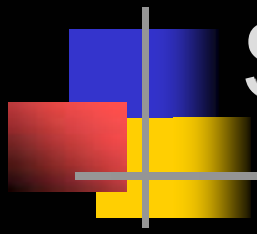
SQL 嵌入高階語言中的方式

- 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8.3 所示。
- 緊密耦合 (Tightly-Coupled)，如： Microsoft Visual Basic

Transact-SQL 嵌入 C 語言範例

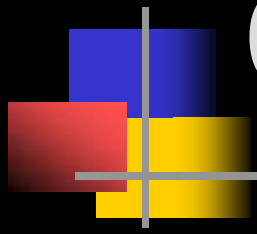
```
EXEC SQL include sqlca;
EXEC SQL begin declare section;
        char    bookname[20];
        char    author[12];
        int      price;
EXEC SQL end declare section;
            ..................
if (...) {
    EXEC SQL select bookname, author, price
                      into :bookname, :author, :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

  EOF

- 關閉資料指標並釋放空間

  EXEC SQL close X

Cursor 應用範例 (當作習題)

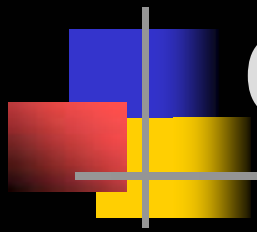
- 請見本章習題問答題第 22 題之習題解答

```
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 2000 on-Line help
  搜尋鍵入：DB-Library for C

  (See also DB-Library for C functions)

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sqlsource)

```
EXEC SQL begin declare section;
        char sqlsource[256];
EXEC SQL end declare section;
```

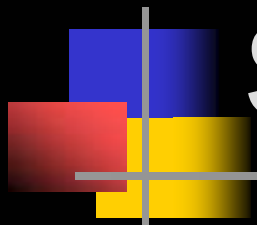
2.將字串變數的內容設定成欲執行的SQL指令

```
strcpy(sqlsource, "delete from Books where id = 3");
```

3. 以 prepare 指令將其定義成一個準備執行的SQL命令變數 (在此為sqlobj)

```
EXEC SQL prepare sqlobj from :sqlsource;
```

4. 以execute指令執行之　EXEC SQL execute sqlobj;

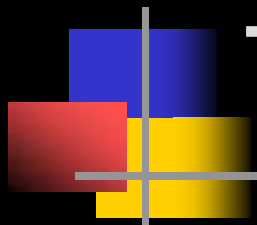


SQL Server 的程式化功能

- 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

Transac-SQL 流程控制指令

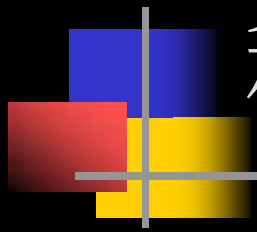
- if … else：為一條件式指令，可以用 exists 或not exists 來檢查某個查詢指令是否有傳回任何資料。
- begin … end：將一群 SQL 指令群組起來。
- goto …：跳到某個使用者所定義的標籤上執行。
- while …：為一迴圈指令 。
- break：中斷迴圈執行。
- continue：回到迴圈的第一句繼續執行。
- waitfor {delay *time* | time *time*}：延遲 *time* 後再執行或等待某個時間到達後繼續執行。
- return：結束副程式回到原呼叫處。



Transac-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。



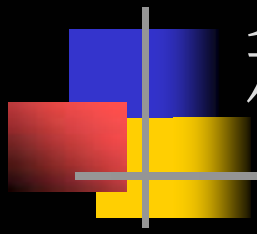
程式範例

- 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。

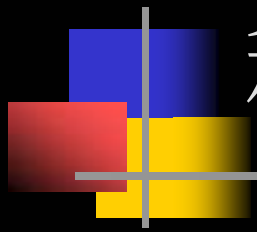
```
if exists (select * from sysobjects
                where name = 'Books' and xtype = 'U')
drop table Books
```



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話，不論是否價格已經超過 200 元都必須要停止。

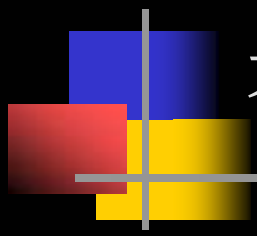
```
declare @count int
select @count = 0
while (exists (select * from books where price < 200)) and
        (@count < 3)
begin
    update books set price = price * 1.1 where price < 200
    select @count = @count + 1
end
```



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

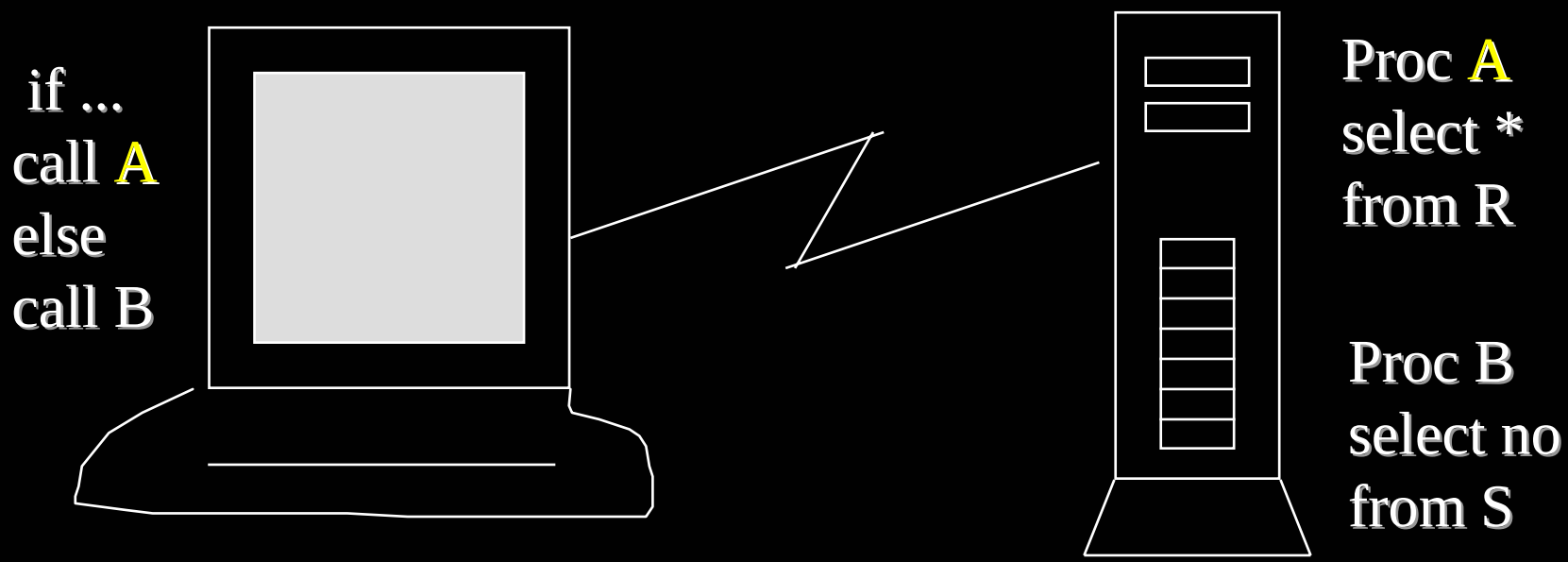
- 可以宣告表格變數當作暫存表格來使用

```
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
```

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 (續)



SQL Server 的預儲程序 (續)

- create proc *procedure_name*
  [*(@parameter1 datatype* [*= value*][, *@parameter2 datatype* [*= 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 (@name varchar(40) = '臺北市')

```
as
    if (@name = null)
    begin
        select name, city
        from Bookstores
        where city is null
    end
else
begin
    select name, city
    from Bookstores
    where city = @name
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 master 資料庫中取得
- 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。SQL Server 最多容許 16 層。
@@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 @name = '水瓶書局'

預儲程序的撰寫原則與刪除

- 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*

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

1	巨蟹書局	20	臺北市

⋈

刪除

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

Orders

no	*id*	*quantity*
1	1	30
1	2	20
1	3	40
1	4	20
1	5	10
1	6	10
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_name(), 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)

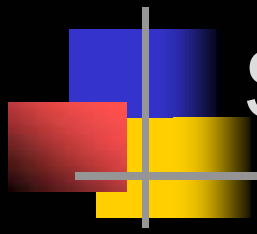
觸發規則的缺點

- 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 2000 的使用者自訂函數

- 使用者自訂函數 (User-Defined Functions) 讓使用者可以以函數的傳回值來取代其名稱。
- 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。
- Server 2000的使用者自訂函數依傳回值分為數值型和表格型兩種，而且使用者自訂函數的完整名稱 (*database_name.owner_name.function_name*) 都必須是唯一的。

SQL Server 2000 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 2000 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 2000 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

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
  - 有那些關聯表？
  - 各有那些欄位？
  - 資料型別 (值域) 為何?
  - 使用那個欄位當主鍵？
  - 是否有建索引？

查看系統目錄的例子

- select * /* 在 BOB 資料庫中 */
  from sysobjects
  where name = 'Books'
- select * /* 在 master 資料庫中 */
  from sysdatabases
  where name = 'master'

透過系統預儲程序來查較清楚

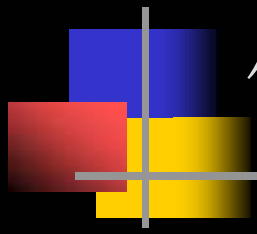
- 上述的查詢結果，可能會讓人有點失望，
- 它可能是一些內定的識別碼讓人看不懂，使得查詢的結果不具可讀性，
- 因為整個系統目錄的綱要對使用者來說並不是很熟悉，所以上面的查詢對於使用者來說意義不大
- SQL Server 額外提供了一些內建的預儲程序，如：sp_help、sp_helpkey、sp_columns、…等，讓使用者可以取得簡單明瞭的資訊。

使用內建預儲程序 (1)

- 如果要查看資料庫中所有關聯表、視界的名稱、建立者、維度等資訊

- 1> use BOB /* 要先 use 目標資料庫 BOB */
  2> exec sp_help
  3> go

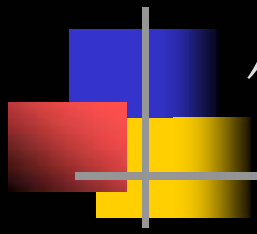
- 如果只執行一行預儲程序, 可以不加上 exec
- 如果執行很多列預儲程序的呼叫, 則必須加上 exec



使用內建預儲程序 (2)

- 如果要查看關聯表的所有屬性名稱、資料型態等資訊

- 1> use BOB /* 要先use目標資料庫BOB */
  2> exec sp_columns @*table_name* = 'Books'
  3> go



使用內建預儲程序 (3)

- 如果要查看關聯表的所有索引名稱、索引欄位數等資訊
- 1> use BOB
  2> exec sp_statistics @*table_name* = 'Books'
  3> go

系統目錄的異動

- 不可以用 SQL 對系統目錄下達新增、刪除或更改的動作
- 系統目錄的新增：Create Table, Create Index, Create View, sp_adduser、sp_addlogin 、…
- 系統目錄的刪除：Drop Table, Drop Index, Drop View, sp_dropuser、sp_droplogin、…
- 系統目錄的更新：Alter Table, sp_rename、sp_renamedb、sp_password、…

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
- “Optimizing SQL” by P. Gulutzan and T. Pelzer (1995)
- 沒有絕對的鐵律可以告訴我們：如何查詢最好？
- 僅提供一些原則性的做法供參考

加速 SQL查詢的技巧 (續)

- 有建索引的欄位可省去多餘的 Order by
- 可加入虛擬條件強迫系統採用 index

select * from Books where price >= 0 (較快) and discount > 0.2 order by price (可去除此句)	select * from Books (較慢) where discount > 0.2 order by price

加速 SQL查詢的技巧 (續)

- 儘量少用巢狀子查詢 (Subquery)

select * from Books where author || publisher in (select aname, pname from author, publisher) (較快)	select * from Books (較慢) where author in (select aname from author) and publisher in (select pname from publisher)
select * from Books (較快) where author || publisher = '孟子元智書坊'	select * from Books (較慢) where author = '孟子' and publisher = '元智書坊'

加速 SQL查詢的技巧 (續)

- 小心使用 LIKE 運算
- 儘量自行將條件限制在最小範圍內

select * from Books (較快) where author between 'A y' and 'AzY'	select * from Books where author like 'A_y'

加速 SQL查詢的技巧 (續)

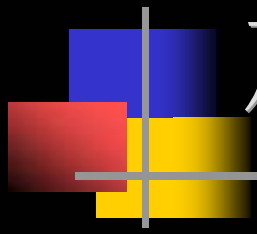
- $\sigma_{C1 \text{ or } C2}(\mathbf{R}) \equiv \sigma_{C1}(\mathbf{R}) \cup \sigma_{C2}(\mathbf{R})$
- 能用 OR 的條件儘量不用 Union

select * from Books (較快) where price > 160 or author = ‘孟子’	select * from Books (較慢) where price > 160 union select * from Books where author = ‘孟子’

加速 SQL查詢的技巧 (續)

- Skew Data 的條件儘量拆開列舉
- 例如：大部份的 price 都落在 300 ~ 350 間

select * from Books (較快) where price > 160 or price <160	select * from Books where price <> 160



加速 SQL查詢的技巧 (續)

- 善用狄摩根定律 not (A = 5 and B = 6) 可以換成 A <> 5 or B <> 6 可能可以提昇效率
- Column IS NOT NULL 可以換成 Column >= "　(空字串)

加速 SQL查詢的技巧 (續)

- 對於大量關聯表的乘積 (Cartesian Product), 可以先建造兩個乘積後的暫存關聯表, 並對它建索引, 再做合併。

create table Tmp (*C1* int) insert into Tmp select (*num+amount*)*2 from Country, Invoices create index I on Tmp (*C1*) select * from Tmp, Items (較快) where *C1 = price*	select * from Country, Invoice, Items (較慢) where (*num+amount*) * 2 = *price*

加速 SQL 查詢的技巧 (續)

- 儘量以 "*column1* >= MAX(*column2*)" 替代 "*column1* >= all *column2*"

select *amount* from Invoices (較快) where *amount* >= (select MAX (*price*) from Items)	select *amount* from Invoices where *amount* >= all (select *price* from Items)

加速 SQL查詢的技巧 (續)

- 擅用遞移律

select amount from Invoices, Items (較快) where Invoices.no = ‘1420’ and Items.no = ‘1420’	select amount from Invoices, Items (較慢) where Invoices.no = Item.no and Items.no = ‘1420’

加速 SQL查詢的技巧 (續)

- IN 換成 OR 來做

select * from Invoices (較快) where *no* between 20 and 25 and *no* <> 23	select * from Invoices (較慢) where *no* in (20, 21, 22, 24, 25) 會被轉換成 select * from Invoices where *no* = 20 or *no* = 21 or *no* = 22 or *no* = 24 or *no* = 25

加速 SQL查詢的技巧 (續)

- 以合併來取代 IN 的 Subquery

select name (較快) from Bookstores, Orders where Bookstores.no = Orders.no	select name (較慢) from Bookstores where no in (select no from Orders)

加速 SQL 查詢的技巧 (續)

- 以 IN 來取代 EXISTS 的 Subquery

select name (較快) from Bookstores, Orders where no in (select no from Orders);	select name (較慢) from Bookstores where exists (select * from Orders where Bookstores.no = Orders.no);

- 先行計算運算式 (Col1 = 2000/1000* 3) 寫成 (Col1 = 6)
- 關於其他加速 SQL Server 的效能調整技巧, 請參考網站 http://www.sql-server-performance.com

KID Curriculum Series: Kid 01