



Stockholm, Sweden, by Frank S.C. Tseng  
(<http://www2.nkfust.edu.tw/~imfrank>)

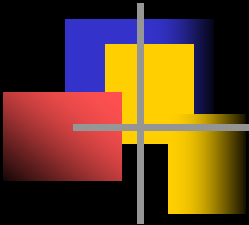
## 第八章 視 界



# 本章內容

---

- 8.1 前言
- 8.2 視界 (View) 的定義
- 8.3 視界的資料處理
- \*8.4 定義遞迴 (Recursive) 式查詢視界
- 8.5 以視界達成邏輯上的資料獨立
- 8.6 視界的優、缺點
- 8.7 後記—理論與實務之間的落差

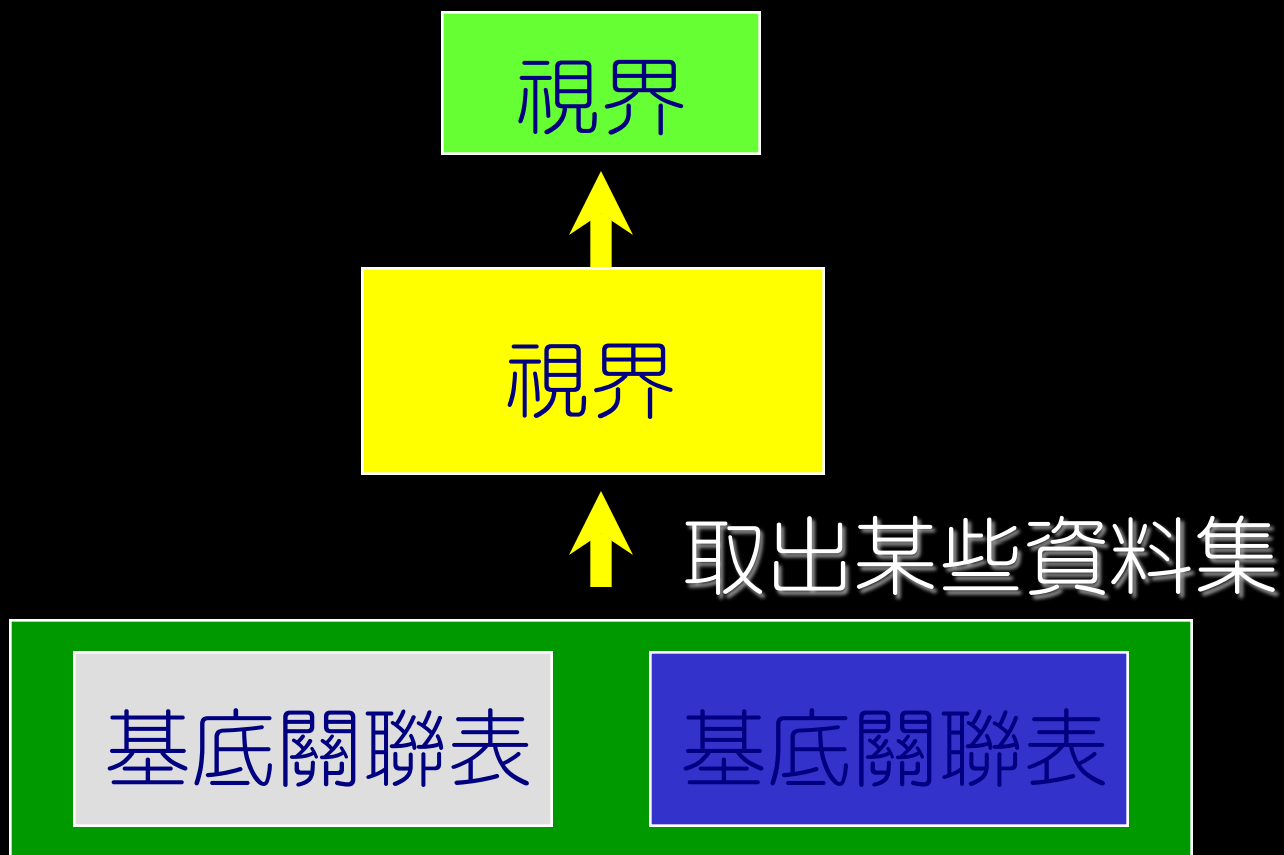


# 前言

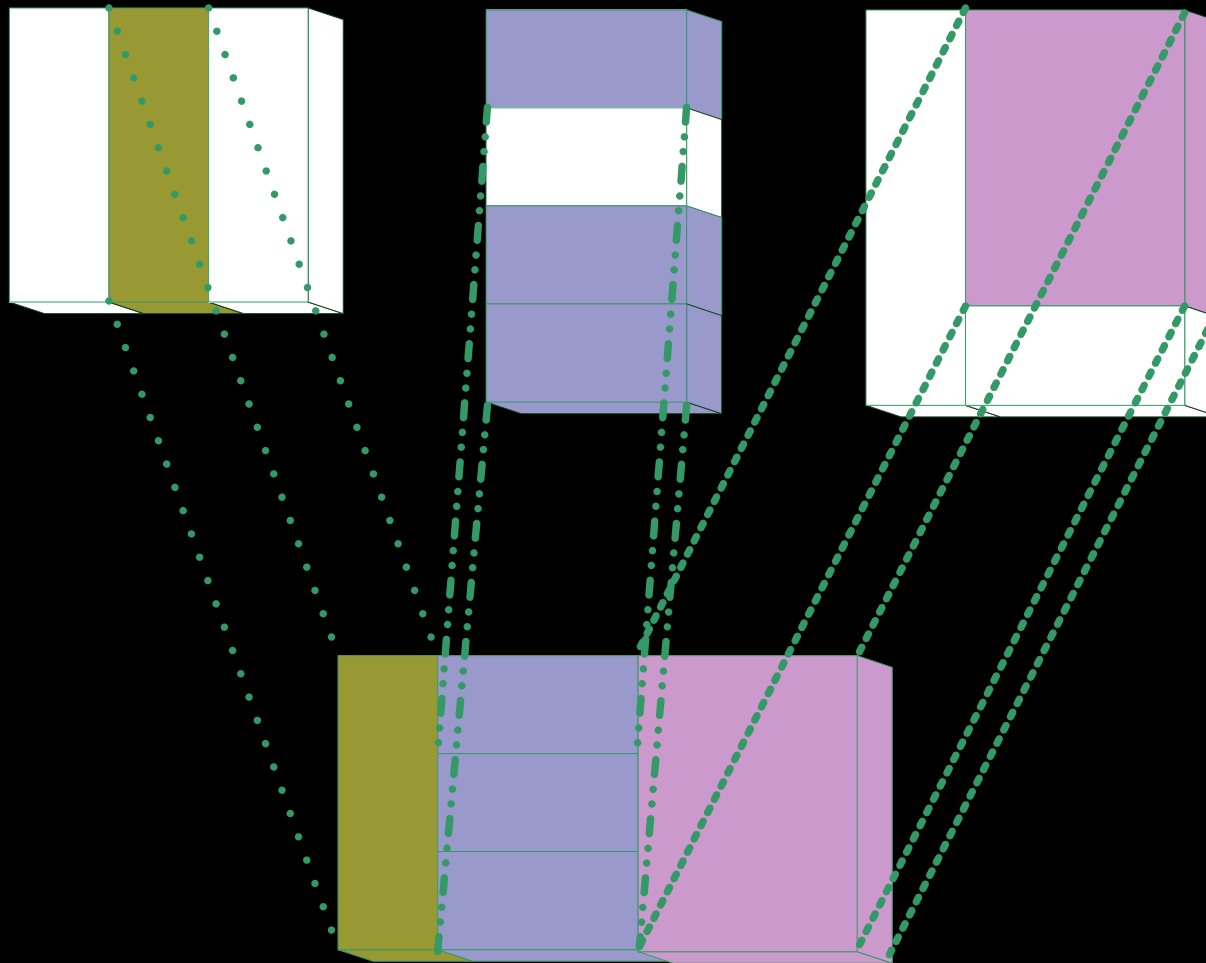
---

- 視界 (View) 在關聯式系統中的地位相當於 ANSI/SPARC 架構上的外部層 (External Level)
- 可以將它看成是一種關聯表
- 只是一個**虛構關聯表** (Virtual Relation) ，實際上並沒有存放真正的資料
- 在資料庫中只存放其定義—外部層與概念層之間的映對 (External/Conceptual Mapping)

# 視界的資料來源



# 視界的概念







# 視界的用途

---

- 讓不同使用者對於資料有不同的觀點、
- 讓不同使用者對於資料有不同的使用範圍。
- 定義不同的視界，將不該讓使用者看到的資料過濾
- 有保密的作用。
- 絕大部份的視界僅能做查詢，不能做更新。  
但是仍有某類的視界可以做更新動作。



# 常用的視界種類

---

- 可以分成以下三大類：
  - 行列子集視界 (Row-and-Column Subset Views)
  - 合併多個關聯表的視界 (Join Views)
  - 統計總覽視界 (Statistical Summary Views)  
(透過 Group By)

# 常用的視界種類 (續)

## ■ 行列子集視界 (Row-and-Column Subset Views)

<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

基底關聯表



<i>no</i>	<i>name</i>	<i>city</i>
1	巨蟹書局	臺北市
3	水瓶書店	新竹市
4	天秤書局	臺中市
5	獅子書局	臺南市

視界



# 常用的視界種類 (續)

## ■ 合併多個關聯表的視界 (Join Views)

**Employee**

<i>id</i>	<i>name</i>	<i>dept</i>
1	Frank	A
2	Mike	B
3	Jesse	C
4	John	B

**Department**

<i>dept</i>	<i>name</i>	<i>manager</i>
A	Marketing	Paul
B	Engineering	Mary
C	Accounting	Annie

合併後產生視界

**Emp\_Manager**

<i>id</i>	<i>name</i>	<i>manager</i>
1	Frank	Paul
2	Mike	Mary
3	Jesse	Annie
4	John	Mary

# 常用的視界種類 (續)

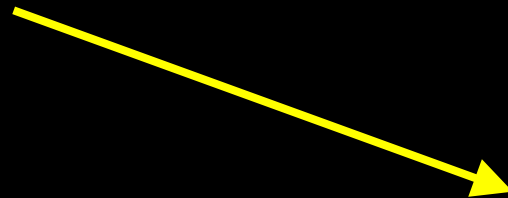
## Orders

<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

- 統計總覽視界 (透過 Group By)  
(Statistical Summary Views)

## Total\_Order

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





# 視界的定義

---

- 透過 SQL 來定義

```
create view view_name [(column_name [,  
    column_name], ... )]  
as SQL_subquery
```

- SQL\_subquery 中有些系統不能含有 Union 子句，如：  
SQL Server 7.0 (但 SQL Server 2000 已經可以了)
- SQL\_subquery 中不能含有 Order By 子句
- 含有 Group By 子句的視界，不被允許更改資料

# 視界範例

- create view **Good\_Bookstores**  
as select no, name, city  
from Bookstores where rank > 10;

<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>no</i>	<i>name</i>	<i>city</i>
1	巨蟹書局	臺北市
3	水瓶書店	新竹市
4	天秤書局	臺中市
5	獅子書局	臺南市

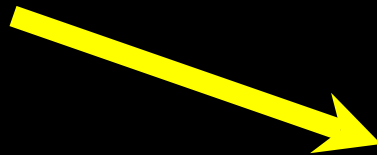
**Good\_Bookstores**

# 需指定視界屬性欄位之情況

## ■ 該欄位為內建的聚合函數

create view No\_of\_Books\_Ordered (id, total\_quantity) as  
select id, SUM(quantity) from Orders group by id;

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



No_of_Books_Ordered	
id	total_quantity
1	60
2	100
3	40
4	50
5	50
6	10



# 需指定視界屬性欄位之情況(續)

- 該欄位為算術運算式

```
create view Books_Discount(id, bookname,  
author, newprice, publisher)  
as select id, bookname, author, price * 0.8, publisher from Books
```

- 該欄位為一字串常數 (e.g. “打八折後的售價” )
- 兩個欄位雖來自不同的關聯表卻有一樣的名稱





# 視界的刪除與定義查詢

---

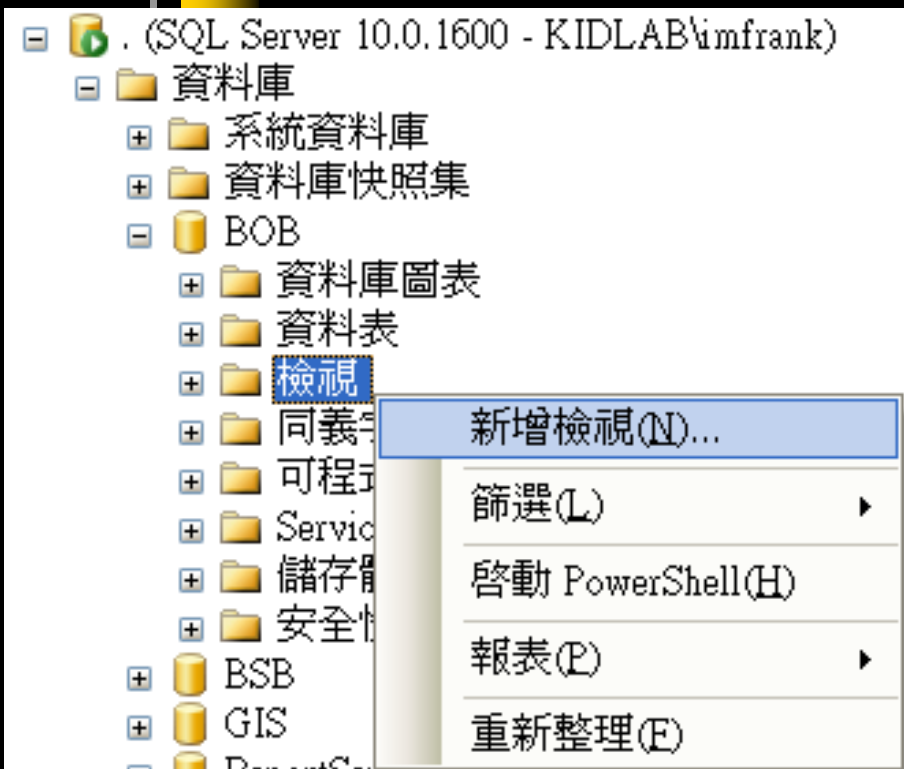
- 將視界刪除的語法為：

drop view *view\_name*

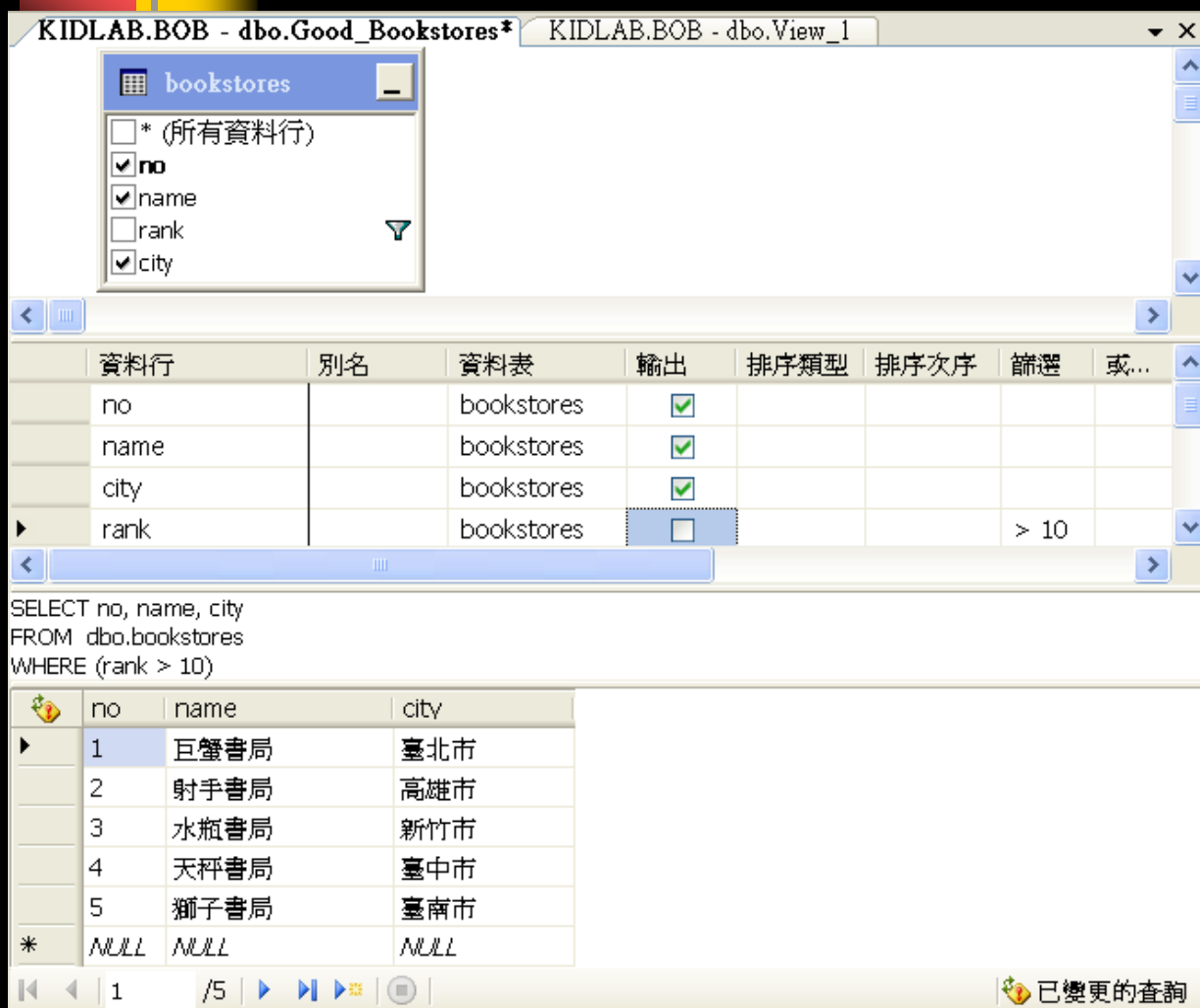
- 在SQL Server上，若要查詢現有視界的定義，可以呼叫預儲程序sp\_helptext 來做，其語法為：

sp\_helptext *view-name*

# 在 SQL Server 2008 上建立視界



# 在 SQL Server 2008 上建立視界



KIDLAB.BOB - dbo.Good\_Bookstores\*

bookstores

- ☐ \* (所有資料行)
- ☒ no
- ☒ name
- ☐ rank
- ☒ city

資料行	別名	資料表	輸出	排序類型	排序次序	篩選	或...
no		bookstores	<input checked="" type="checkbox"/>				
name		bookstores	<input checked="" type="checkbox"/>				
city		bookstores	<input checked="" type="checkbox"/>				
rank		bookstores	<input type="checkbox"/>			> 10	

```
SELECT no, name, city
FROM dbo.bookstores
WHERE (rank > 10)
```

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

已變更的查詢

整個介面完全與  
Query-by-Example  
一模一樣

選擇名稱

輸入檢視表名稱(E):

Good\_Bookstores

確定 取消

# 在 SQL Server 2008 上建立視界





# 視界上的資料處理

---

- 在視界上做查詢，百分之一百沒有問題
- 但是要在視界上做更新，會有許多困難
- 行列子集視界若包含了原基底關聯表的主鍵，則在理論上做更新沒有問題。
- 早期在 80 年代時，一般的資料庫管理系統通常不管理論上是否可以做更新，通常是不提供視界上的更新功能的。
- 目前的產品則多已提供部分更新功能了

# 視界上的資料處理 (續)

- create view Good\_Bookstores  
as select no, name, city  
from Bookstores where rank > 10

- 使用者下達

```
select *  
from Good_Bookstores  
where city <> '臺北市'
```

會被轉換成



```
select no, name, city  
from Bookstores  
where city <> '臺北市'  
and rank > 10;
```

- 請驗證一下兩個查詢的結果是否一致？



# 驗證一下上述兩查詢是否一致？

- `select no,name, city  
from Bookstores  
where city <> '臺北市'  
and rank > 10`

<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

- `select *  
from Good_Bookstores  
where city <> '臺北市'`

<i>no</i>	<i>name</i>	<i>city</i>
1	巨蟹書局	臺北市
3	水瓶書店	新竹市
4	天秤書局	臺中市
5	獅子書局	臺南市



# 視界上的資料處理 (續)

- 以第 13 頁投影片的 `No_of_Books_Ordered` 為例

```
select id, total_quantity  
from No_of_Books_Ordered  
where total_quantity > 50 and id <> 2
```

會被轉成

```
select id, SUM(quantity) as total_quantity  
from Orders where id <> 2  
group by id          (有having 就有 group by)  
having SUM(quantity) > 50
```

- `where` 的條件若有聚合函數，則要改成 `having`

# 驗證一下上述兩查詢是否一致？

```
select id, total_quantity  
from No_of_Books_Ordered  
where total_quantity > 50 and id <> 2
```

No\_of\_Books\_Ordered

<i>id</i>	<i>total_quantity</i>
1	60
2	100
3	40
4	50
5	50
6	10



<i>id</i>	<i>total_quantity</i>
1	60

# 驗證一下是否一致？(續)

select id, SUM(quantity) as total\_quantity  
from Orders where id <> 2 group by id  
having SUM(quantity) > 50

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



id	Total_quantity
1	60
3	40
4	50
5	50
6	10



id	total_quantity
1	60

與上頁結果一致



# 視界可以更新的例子

create view **No\_City**  
as select no, city  
from Bookstores

有保留主鍵的  
行列子集視界在  
理論上可以更新

<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	臺北市
5	獅子書局	30	臺南市



# 視界無法更新的例子

create view **Rank\_City**  
as select distinct rank, city  
from Bookstores

看不到這一筆  
(如何更新其內容?)



<i>rank</i>	<i>city</i>
20	臺北市
10	高雄市
30	新竹市
20	臺北市
30	臺南市



# No\_City 可以新增之分析

<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	臺北市
5	獅子書局	30	臺南市

↑ 新增到 Table

7	(Null)	(Null)	中壢市
---	--------	--------	-----

<i>no</i>	<i>city</i>
1	臺北市
2	高雄市
3	新竹市
4	臺北市
5	臺南市

↑ 新增 View

7	中壢市
---	-----

# No\_City 可以刪除之分析

<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	臺北市
5	獅子書局	30	臺南市

刪除 Table 中  
的第 4 筆

<i>no</i>	<i>city</i>
1	臺北市
2	高雄市
3	新竹市
4	臺北市
5	臺南市

刪除 View 中  
的第 4 筆

# No\_City 可以更新之分析

<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	嘉義市
5	獅子書局	30	臺南市

更新 Table 中的 '臺北市'  
成為 '嘉義市'

<i>no</i>	<i>city</i>
1	臺北市
2	高雄市
3	新竹市
4	嘉義市
5	臺南市

更新 View 中的 '  
臺北市'  
成為 '嘉義市'

# Rank\_City 無法新增之原因

<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	臺北市
5	獅子書局	30	臺南市

↑ 新增到 Table

<Null>	<Null>	25	中壢市
--------	--------	----	-----

<i>rank</i>	<i>city</i>
20	臺北市
10	高雄市
30	新竹市
30	臺南市

↑ 新增 View

25	中壢市
----	-----

# Rank\_City 無法刪除之原因

<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	臺北市
5	獅子書局	30	臺南市

不知道要刪除 Table 中的那一筆  
(20, '臺北市') ?

<i>rank</i>	<i>city</i>
20	臺北市
10	高雄市
30	新竹市
30	臺南市

刪除 View 中的  
(20, '臺北市') 那一筆

# Rank\_City 無法更改之原因

<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	臺北市
5	獅子書局	30	臺南市

不知道要更改 Table 中的那一筆  
(20, 臺北市) 成為 (20, 嘉義市) ?

<i>rank</i>	<i>city</i>
20	臺北市
10	高雄市
30	新竹市
30	臺南市

更改 View 中  
(20, '臺北市') 成為  
(20, '嘉義市')



# 容易產生疑惑的視界

- 假設 Good\_Bookstores1 的定義如下:

```
create view Good_Bookstores1  
as select no, name, rank, city  
from Bookstores  
where rank > 10
```

在 Good\_Bookstores1  
中看不見這一筆

<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

# 容易產生疑惑的視界(續)

- 新增一筆 (2, "?????", ??, "????????") 到 View 中會被拒絕？！
- 將 (3, ...) 改成 (2, ...) 也會被拒絕？！
- 新增 (8, ..., 5, ...) 會在 Good\_Bookstores1 中看不到
- 可以用 check option 來避免

<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

8	元智書坊	5	中壢市
---	------	---	-----

(但在 SQL Server Management Studio 中無效)



# 行列子集視界

```
create view Simple_Cheap_Books
as select id, bookname, price, author
from Books
where price < 150
```

Simple\_Cheap\_Books

<i>id</i>	<i>bookname</i>	<i>price</i>	<i>author</i>
1	三國演義	120	羅貫中
4	西遊記	140	吳承恩
5	水經注	120	酈道元

- 可以不允許使用者修正 price 或新增  $\geq 150$  的 price
- 也可以允許新增或修改，但是使用者會看不到更新後的全貌



# 合併的視界

## Employee

<i>id</i>	<i>name</i>	<i>dept</i>
1	Frank	A
2	Mike	B
3	Jesse	C
4	John	B

## Department

<i>dept</i>	<i>name</i>	<i>manager</i>
A	Marketing	Paul
B	Engineering	Mary
C	Accounting	Annie

```
create view Emp_Manager  
as select id, E.name, manager  
   from Employee E, Department D  
  where E.dept = D.dept
```

## Emp\_Manager

<i>id</i>	<i>name</i>	<i>manager</i>
1	Frank	Paul
2	Mike	Mary
3	Jesse	Annie
4	John	Mary

# 合併的視界通常無法更新

Emp\_Manager

<i>id</i>	<i>name</i>	<i>manager</i>
1	Frank	Paul
2	Mike	Mary
3	Jesse	Annie
4	John	Mary

- 將 (4, John, Mary) 改成 (4, John, Annie) 有困難:
  - 到底是要將 John 換到部門 C ?
  - 還是要將部門 B 的主管換成 Annie ?
- 奇怪的是 SQL Server 居然允許上例更新!!!??? 而且採用 '將部門 B 的主管換成 Annie' 的做法

# 合併視界通常無法更新 (續)

Employee

<i>id</i>	<i>name</i>	<i>dept</i>
1	Frank	A
2	Mike	B
3	Jesse	C
4	John	C

Department

<i>dept</i>	<i>name</i>	<i>manager</i>
A	Marketing	Paul
B	Engineering	Mary
C	Accounting	Annie

Employee

<i>id</i>	<i>name</i>	<i>dept</i>
1	Frank	A
2	Mike	B
3	Jesse	C
4	John	B

Department

<i>dept</i>	<i>name</i>	<i>manager</i>
A	Marketing	Paul
B	Engineering	Annie
C	Accounting	Annie

SQL Server 2000 預設採用這種方式, 會有 '副作用' :  
(Mike 的主管也自動換成了 'Annie', 會造成問題 !!!)



# 統計總覽視界

```
create ciew Total_Order (no, quantity)
as select no, SUM(quantity)
from Orders
Group by no
```

Total\_Order

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

- 此類視界在理論或實際上是絕對無法新增或更改的



# 視界在理論上應遵循的原則

- 對視界的新增、刪除、修改在基底關聯表上不可以有兩種以上的可能方式。
- 對視界的新增、刪除、與修改，不可以違反基底關聯表上的限制條件。
- 對視界的新增、刪除、與修改，不可以反過來對該視界造成「副作用」。
- 對視界的新增、刪除、與修改，不可以對基底關聯表造成「副作用」。





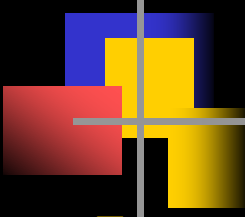
# ANSI 在視界更新上的規範

- 以下指令如果包含在 Create View 中的話，則該視界只能供讀取，無法更改：
  - Select 子句中包含了 Distinct 關鍵字。
  - 在 Select 子句中含算術運算式、聚合函數
  - 引用了兩個以上的關聯表，如：在 From 子句或巢狀子查詢中參用一個以上的關聯表，或 Union 兩個以上關聯表。
  - 在 From 或巢狀子查詢中參用了無法更改的視界。
  - 查詢中包含了有 Group By 或 Having 子句。



## \*定義遞迴 (Recursive) 式查詢視界

- SQL的八大子句中的 `with common_table_exp` 子句簡稱為 **CTE**，也稱為**共同表格運算式**，
- 它利用 Select 子句產生暫存結果，可以當作暫存的視界 (View) 或衍生性關聯表 (Derived Relation)，用來進行**遞迴查詢 (Recursive Query)**
- 如：公司組織結構，或物料需求規劃 (Material Resource Planning, MRP) 上常用的「用料表」(Bill-Of-Material, **BOM**) 等之計算
- 我們建議將這類遞迴查詢寫成**視界**



# 利用 CTE 形成遞迴的效果

- 一般是遵循以下三個步驟：
  - 透過 with 宣告該 CTE 關聯表名稱 (可以多個)，以及所含的屬性清單
  - 透過 as 建立 CTE 的完整 SQL 查詢，當中可以使用到該 CTE 關聯表本身，以形成遞迴效果。
  - 運用一般的 DML 查詢，擷取由 CTE 所定義的表格內容。
- 指令如下：

```
with table_name(column_name [...n]) [...n]  
as (  
    CTE_query_expression  
)
```

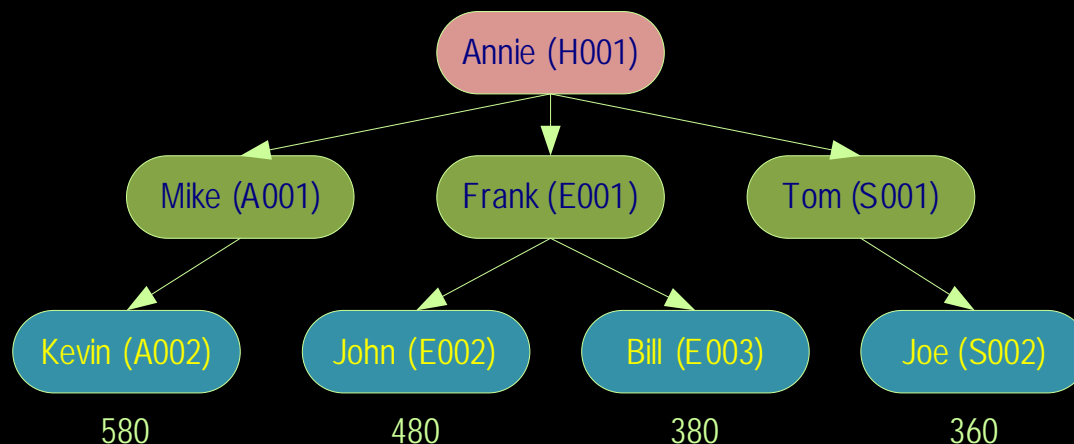


# 使用 CTE 進行樹狀遞迴式查詢

- 一般的遞迴查詢處理可以拆解成兩個部份：
  - 「錨定成員」(Anchor Member) 的產生：撰寫讓遞迴終止的條件 (一般稱為 Trivial Condition)，並利用 UNION ALL 將下述步驟所找出的結果聯集起來。
  - 「遞迴成員」(Recursive Member) 的產生：利用另一個 SQL 指令引用 CTE 關聯表本身，形成遞迴呼叫。

# 先看一個範例

- 業務員的銷售績效會累積到直屬經理的績效上，經理的累積績效又會累積到其直屬上司的績效上，依此類推。



往上累加

Sales

<u>eno</u>	name	manager	achievement
A002	Kevin	A001	580
E002	John	E001	480
E003	Bill	E001	380
S002	Joe	S001	360

SalesManagers

<u>eno</u>	name	manager
H001	Annie	—
A001	Mike	H001
E001	Frank	H001
S001	Tom	H001



# 以 CTE 求解

■ SQL 寫法如下 (請將 (1), (2), (3) 去除才能執行) :

- (1) with Rollup(*eno, name, manager, achievement*)
- (2) as ( -- 以下SQL求出錨定成員  
select *eno, name, manager, achievement* from Sales  
union all -- 以下SQL找出所有遞迴成員  
select *S.eno, S.name, S.manager, R.achievement*  
from Rollup R, SalesManagers S  
where *R.manager = S.eno*  
) -- 以上定義 Rollup, 下方則是直接引用 Rollup
- (3) select *eno, name, sum(achievement) as achievement*  
from Rollup  
group by *eno, name*  
order by *achievement desc, eno asc*



## (2) 執行過程說明

- (2) 的第一個 SQL 用來求取錨定成員：也就是直接  
求出各業務員的業績：

*select eno, name, manager, achievement from Sales*

<i>eno</i>	<i>name</i>	<i>manager</i>	<i>achievement</i>
A002	Kevin	A001	580
E002	John	E001	480
E003	Bill	E001	380
S002	Joe	S001	360

查出的結果形成 Rollup 的初步內容

## (2) 執行過程說明

- (2) 的第二個 SQL 則由 Rollup 與 SalesManagers 找出上司與部屬間的關係，然後將業績 (Achievement) 由部屬傳給上司：

```
Select S.eno, S.name, S.manager, R.achievement  
from Rollup R, SalesManagers S  
where R.manager = S.eno
```

<i>Eno</i>	<i>name</i>	<i>Manager</i>	<i>achievement</i>
A001	Mike	H001	580
E001	Frank	H001	480
E001	Frank	H001	380
S001	Tom	H001	360

遞迴第一層產生的結果

<i>Eno</i>	<i>name</i>	<i>Manager</i>	<i>achievement</i>
H001	Annie	NULL	580
H001	Annie	NULL	480
H001	Annie	NULL	380
H001	Annie	NULL	360

遞迴第二層產生的結果



## (2) 執行過程說明

整個  
Rollup  
最後  
產生的  
內容

Rollup			
<i>eno</i>	<i>name</i>	<i>manager</i>	<i>achievement</i>
S002	Joe	S001	360
A002	Kevin	A001	580
E002	John	E001	480
E003	Bill	E001	380
A001	Mike	H001	580
E001	Frank	H001	480
E001	Frank	H001	380
S001	Tom	H001	360
H001	Annie	NULL	580
H001	Annie	NULL	480
H001	Annie	NULL	380
H001	Annie	NULL	360

} 後續要加總

} 後續要加總



## (3) 執行過程說明

(3) 指令執行後的結果得到每個人的總業績

```
select eno, name, sum(achievement) as achievement  
from Rollup  
group by eno, name  
order by achievement desc, eno asc
```

<i>eno</i>	<i>name</i>	<i>achievement</i>
H001	Annie	1800
E001	Frank	860
A001	Mike	580
A002	Kevin	580
E002	John	480
E003	Bill	380
S001	Tom	360
S002	Joe	360



# 定義成 View

create view Rollup

As

with Rollup(*eno*, *name*, *manager*, *achievement*)

as ( -- 以下SQL求出錨定成員

select *eno*, *name*, *manager*, *achievement* from Sales

union all -- 以下SQL找出所有遞迴成員

select S.*eno*, S.*name*, S.*manager*, R.*achievement*

from Rollup R, SalesManagers S

where R.*manager* = S.*eno*)

select *eno*, *name*, sum(*achievement*) as *achievement*

from Rollup group by *eno*, *name*

- 後續直接對 Rollup 查詢 (可加上 order by 子句)

select *eno*, *name*, *achievement*

from Rollup

order by *achievement* desc, *eno* asc



# 控制遞迴的深度

- SQL Server 2008 預設的執行深度是 100 層，超過此數目會跳出執行。
- 在針對 CTE 關聯表查詢時，可在 SQL 的最後加上 **OPTION (MAXRECURSION  $n$ )** 的設定，並將  $n$  設成 0 即可解除 100 層的限制
- 或是設定更小的深度 (如： $n = 5$ ) 來限定之：

```
select eno, name, achievement
from Rollup
order by achievement desc, eno asc
option (MAXRECURSION 5)
```

# 使用 CTE 進行圖形遞迴式查詢

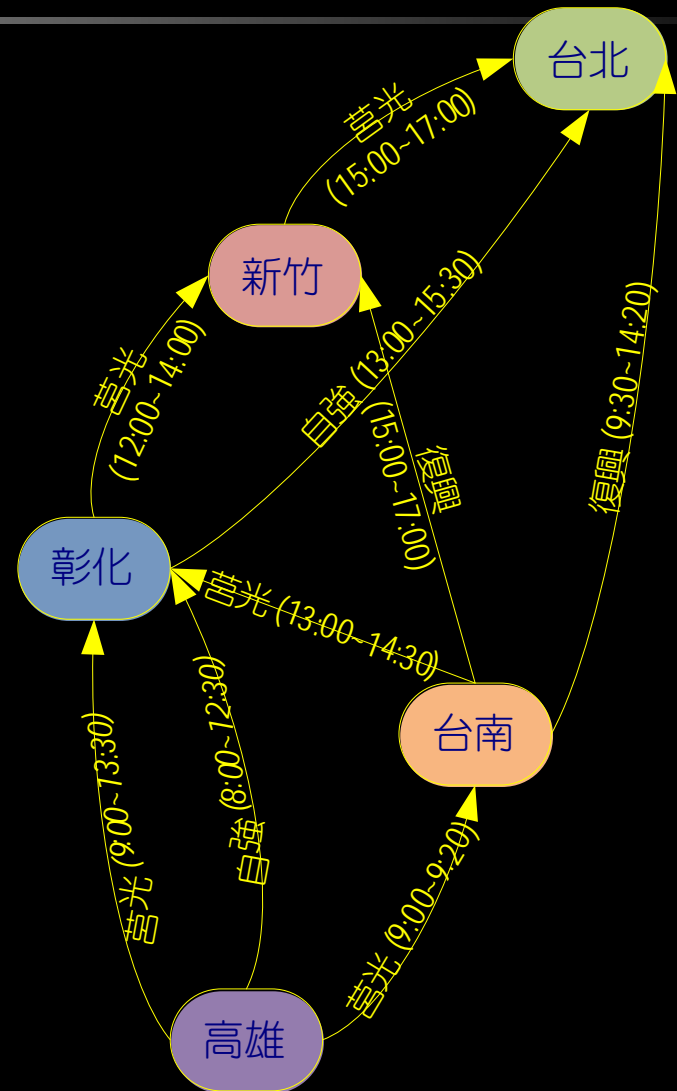
- CTE 遞迴除了適合樹狀結構的計算之外，也適合解決圖形上的問題。
- 範例：假設台鐵的北上列車班次如下圖所示

Schedule

車次	車種	起站	終站	開車時間	到站時間	經過站名
1001	莒光	高雄	彰化	900	1330	高雄
1010	莒光	彰化	新竹	1200	1400	彰化
1013	莒光	高雄	台南	900	920	高雄
1021	莒光	台南	彰化	1300	1430	台南
1038	自強	高雄	彰化	800	1230	高雄
1041	復興	台南	新竹	1500	1700	台南
1051	復興	台南	台北	930	1420	台南
1101	莒光	新竹	台北	1500	1700	新竹
1203	自強	彰化	台北	1300	1530	彰化

# 台鐵的北上列車班次圖示

- 目標：建立 CTE 表格，推導任兩城市之間所有可能的搭車組合。
- 原則是：只要原搭乘列車的 [到站時間] 小於欲搭乘列車的 [開車時間] 即可成功接駁。





# SQL 程式撰寫

- (1) with Travel(車次, 車種, 起站, 終站, 開車時間, 到站時間, 經過站名)
- (2) as ( -- 以下SQL求出錨定成員  
select 車次, 車種, 起站, 終站, 開車時間, 到站時間,  
cast(rtrim(經過站名)+'('+cast(開車時間 as char(4))+')' as varchar(100)) as 經過站名  
from Schedule  
union all -- 以下SQL找出所有遞迴成員  
select i.車次, i.車種, i.起站, o.終站, i.開車時間, o.到站時間,  
cast(rtrim(i.經過站名)+'->('+cast(i.到站時間 as char(4))+')'+  
rtrim(i.終站)+'('+cast(o.開車時間 as char(4))+')' as varchar(100))  
from Travel i, Schedule o  
where i.終站 = o.起站 AND i.到站時間 < o.開車時間)
- (3) select 車次, 車種, 起站, 終站, 開車時間, 到站時間,  
cast(rtrim(經過站名)+'->('+cast(到站時間 as char(4))+')'+rtrim(終站)  
as varchar(100)) as 經過站名  
from Travel
- cast() 及 rtrim() 函數只是用來讓輸出美觀的程式碼。

## (2) 執行過程說明

- (2) 的第一個 SQL 用來求取錨定成員，結果就是 Schedule 關聯表本身先構成 Travel 的內容。

Schedule

車次	車種	起站	終站	開車時間	到站時間	經過站名
1001	莒光	高雄	彰化	900	1330	高雄
1010	莒光	彰化	新竹	1200	1400	彰化
1013	莒光	高雄	台南	900	920	高雄
1021	莒光	台南	彰化	1300	1430	台南
1038	自強	高雄	彰化	800	1230	高雄
1041	復興	台南	新竹	1500	1700	台南
1051	復興	台南	台北	930	1420	台南
1101	莒光	新竹	台北	1500	1700	新竹
1203	自強	彰化	台北	1300	1530	彰化





## (2) 執行過程說明

---

- (2) 的第二個 SQL 由 Travel 與 Schedule 找出各站間的接駁關係是否成立：
  - 搭乘列車之 [終站] 站名要與所欲搭乘列車之 [起站] 站名一樣，以及
  - 搭乘列車 [到站時間] 要小於所欲搭乘列車之 [開車時間]：  
where i.終站 = o.起站 AND i.到站時間 < o.開車時間

# 產生的 Travel 結果

車次	車種	起站	終站	開車時間	到站時間	經過站名
1001	莒光	高雄	彰化	900	1330	高雄(900 )->(1330)彰化
1010	莒光	彰化	新竹	1200	1400	彰化(1200)->(1400)新竹
1013	莒光	高雄	台南	900	920	高雄(900 )->(920 )台南
1021	莒光	台南	彰化	1300	1430	台南(1300)->(1430)彰化
1038	自強	高雄	彰化	800	1230	高雄(800 )->(1230)彰化
1041	復興	台南	新竹	1500	1700	台南(1500)->(1700)新竹
1051	復興	台南	台北	930	1420	台南(930 )->(1420)台北
1101	莒光	新竹	台北	1500	1700	新竹(1500)->(1700)台北
1203	自強	彰化	台北	1300	1530	彰化(1300)->(1530)台北
1038	自強	高雄	台北	800	1530	高雄(800 )->(1230)彰化(1300)->(1530)台北
1013	莒光	高雄	彰化	900	1430	高雄(900 )->(920 )台南(1300)->(1430)彰化
1013	莒光	高雄	新竹	900	1700	高雄(900 )->(920 )台南(1500)->(1700)新竹
1013	莒光	高雄	台北	900	1420	高雄(900 )->(920 )台南(930 )->(1420)台北
1010	莒光	彰化	台北	1200	1700	彰化(1200)->(1400)新竹(1500)->(1700)台北



# 撰寫 Travel 成為 View

- create view Travel  
as  
with Travel(車次, 車種, 起站, 終站, 開車時間, 到站時間, 經過站名)  
as ( -- 以下SQL求出錨定成員  
select 車次, 車種, 起站, 終站, 開車時間, 到站時間,  
cast(rtrim(經過站名)+'('+cast(開車時間 as char(4))+')' as varchar(100)  
as 經過站名  
from Schedule  
union all -- 以下SQL找出所有遞迴成員  
select i.車次, i.車種, i.起站, o.終站, i.開車時間, o.到站時間,  
cast(rtrim(i.經過站名)+ '->('+cast(i.到站時間 as char(4))+')'+  
rtrim(i.終站)+ '('+ cast(o.開車時間 as char(4))+ ')' as varchar(100))  
from Travel i, Schedule o  
where i.終站 = o.起站 AND i.到站時間 < o.開車時間)  
select 車次, 車種, 起站, 終站, 開車時間, 到站時間,  
cast(rtrim(經過站名)+'->('+cast(到站時間 as char(4))+')'+rtrim(終站)  
as varchar(100)) as 經過站名  
from Travel
- 後續想從高雄到台北，就可以 SQL 查詢 Travel 視界如下：

```
select * from Travel where 起站 = '高雄' and 終站 = '台北'
```

# 查詢 Travel (起站 '高雄' , 終站 = '台北')

- select \* from Travel where 起站 = '高雄' and 終站 = '台北'

車次	車種	起站	終站	開車時間	到站時間	經過站名
1038	自強	高雄	台北	800	1530	高雄(800 )->(1230)彰化(1300)->(1530)台北
1013	莒光	高雄	台北	900	1420	高雄(900 )->(920 )台南(930 )->(1420)台北

- 第一筆資料的意義：1038 的自強號，可在 12:30 抵達彰化，然後 13:00 可以轉搭另一班列車，在 15:30 抵達台北。
- 第二筆資料的意義類似



# Travel 的變形 (請同學修改程式碼)

- 如果有票價資料的話，請算出以下幾個問題：
  - 給定 [起站] 與 [終站]，找出票價最便宜的接駁方式。
  - 給定 [起站] 與 [終站]，找出乘車時間最短的接駁方式。
  - 給定 [起站] 與 [終站]，找出等車時間總和最少的接駁方式。
  - 給定 [起站] 與 [終站]，找出轉車次數最少的接駁方式
  - 給定 [起站] 與 [終站] 與某 [中間站]  $m$ ，找出不經過  $m$  的接駁方式
  - 給定 [起站] 與 [終站]，加上固定的 [乘車費用]  $f$ ，找出可以讓我們從起站到達終站的接駁方式 (或不給定起站、終站，找出所有配對結果)
  - 給定某個 [時間]  $t$ ，以及 [起站] 與 [終站]，找出在  $t$  之前一定要抵達 [終站] 的接駁方式
  - 再加上人數的因素，例如：區分全票、半票與老人優待票的不同情況，或多人分別持不同固定乘車費用從不同起站坐到某個固定終站的乘車與接駁方式。
  - ... (see Page 8-28)



# 以視界達成邏輯上的資料獨立

- 外部層與概念層之間有一個「外部層/概念層映對」(External/Conceptual Mapping)
- 若概念層的結構被修改過，則可以透過更改「外部層/概念層映對」達成邏輯資料獨立
- 不會影響應用程式或使用者做查詢的感覺



# 基底關聯表會改變結構的原因

---

- 基底關聯表欄位不敷使用，需要成長 (Base Table Growth)
- 重組基底關聯表結構 (Base Table Restructuring)
- 可能因正規化 (Normalization) 或效率上的考量，將關聯表使用很頻繁的部份獨立出來，成為另一個關聯表放在速度較快的磁碟機上。

# 重組基底關聯表的結構

切割後將它變成 View

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

切成兩個

<i>no</i>	<i>rank</i>
1	20
2	10
3	30
4	20
5	30

使用較頻繁  
可放在快速的  
磁碟機上





# 視界的缺點

---

- 要經過系統的一個轉換動作，效率比較差，
- 對視界做新增、刪除或更新動作會有某些限制。無法用相同的對待方式看視界與關聯表
- 在視界上可定義層層的視界，視界的建立記錄與相互間的依存關係要記錄起來，以免搞混，以免在刪除視界、關聯表或新增、刪除或更改基底關聯表中的欄位名稱及資料型態時，產生不一致的情形。



# 視界的優點

---

- 可以達成邏輯資料獨立的目的。
- 簡化使用者的觀點，使用者只需專注在他所感興趣的部份
- 將基底關聯表上的資料處理工作移到視界上的定義來做
- 自動對隱藏的資料提供保密措施



# 建立一個具保密措施的個人視界

```
create table Employees (  
  id int not null,  
  name char(20) not null,  
  dept char(12) not null,  
  salary money not null)
```

Employees

<i>id</i>	<i>name</i>	<i>dept</i>	<i>salary</i>
1	Jim	Accounting	\$25,000
2	Michael	Engineering	\$35,000
3	Grace	Marketing	\$30,000
4	Annie	Accounting	\$28,750
5	Jesse	Marketing	\$40,000
6	Andy	Engineering	\$39,500

- create view myrecord  
as select \* from Employees  
where name = *suser\_sname*();



# 視界在理論與實務之間的落差

- with check option 在 SQL Server Enterprise Manager 的管理介面上似乎並沒有作用
- 定義 Good\_Bookstores1 時限定  $rank > 10$ ，所以 Good\_Bookstores1 沒有  $rank \leq 10$  的資料。
- 將某一筆值組的  $rank$  改成 5 (假設是「水瓶書局」)，那麼你將會發現系統是允許的，可是改完後的 Good\_Bookstores1 中卻看不到它的存在 (見下頁的圖)



# 視界在理論與實務之間的落差

- 將 Employee、Department 兩個表格合併後產生 Emp\_Manager 這個視界的內容如下所示：

	id	name	manager
▶	1	Frank	Paul
	2	Mike	Mary
	3	Jesse	Annie
	4	John	Mary
*	NULL	NULL	NULL



# 視界在理論與實務之間的落差

- 針對 Emp\_Manager 做更新動作，假定將 John 的 *manager* 由 Mary 改成 Annie。這個動作反應到基底關聯表上有兩種可能的作法，違反了 8.3.4 節的第 1 條規則，**理論上應該是不允許的**。
- 但是，MS SQL Server 2008 不但允許，甚至於還造成了 Mike 的 *manager* 也被改成 Annie 的「副作用」，見下圖。換句話說，它並沒有遵循 8.3.4 節的第 3 條規則：對視界的新增、刪除、或修改動作，在反應到其基底關聯表之後，不可以反過來對該視界造成「副作用」(Side-Effect)(見下頁)

# 視界在理論與實務之間的落差

	id	name	manager
▶	1	Frank	Paul
	2	Mike	Annie
	3	Jesse	Annie
	4	John	Annie
*	NULL	NULL	NULL

副作用

# 視界在理論與實務之間的落差

- 原因是：SQL Server 2008 採用 8.3.2 節的第二種作法來執行 (把部門 B 的主管換成 Annie)。
- 將 Department 的內容打開就可以得到證實。

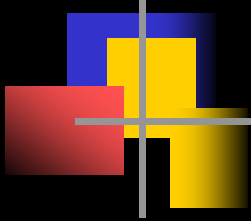
	dept	name	manager
	A	Marketing	Paul
▶	B	Engineering	Annie
	C	Accounting	Annie
*	NULL	NULL	NULL





# 視界在理論與實務之間的落差

- 對於更改視界的做法是否合宜？！我們覺得有討論的必要。
- 除非萬不得已，否則我們不鼓勵使用這些視界的更新功能，因為它很容易造成錯誤的情況。
- 我們更希望開發資料庫管理系統的廠商，能加以補強，將這些會造成副作用的新增、刪除、或修改動作一一擋掉，以避免不必要的困擾。
- 經過測試: IBM DB2, Oracle 預設都是不可以更改上述這些會產生副作用的 Views



本章結束  
The End.