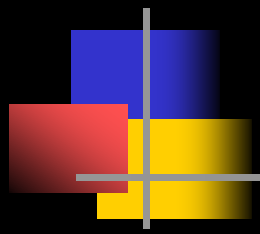
第六章 關聯式模式的資料運算

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



本章內容

- 6.1 前言
- 6.2 關聯式代數 (Relational Algebra)
- 6.3 關聯式計算 (Relational Calculus)

前　言

- 關聯式模式的兩套運算模式
  - 關聯式代數 (Relational Algebra)
  - 關聯式計算 (Relational Calculus)

  → 綜合成SQL
- 兩套運算模式能力相當，但本質有所差異
- 任何一套查詢語言只要具備以上兩者之一的能力便稱為「關聯完整性」(Relationally-Complete)

兩套運算模式的比較

	關聯式代數 (Relational Algebra)	關聯式計算 (Relational Calculus)
1.	為一程序式的查詢語言 (procedural language)，比較低階。	非程序式的查詢語言 (non-procedural language)，接近人類所使用的自然語言，比較高階。
2.	必須明白地指出運算的順序，表達“如何” (how) 一步步完成查詢。	不須明白地指出運算的順序，只須說明查詢所要的是 “什麼” (what)。
3.	必須提供基本的運算 (operations)，如：選擇 (Select)、投影 (Project)、聯集 (Union)等。	不須提供基本的運算 (operations)，而完全是以「完構語式」(well-formed formula) 來說明查詢所要的是 “什麼”。

兩套運算模式的比較 (續)

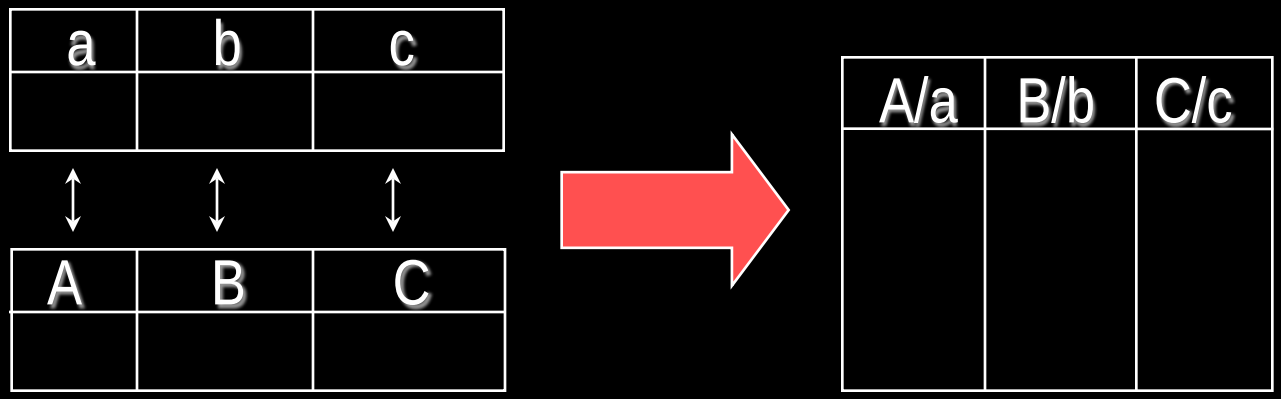
	關聯式代數 (Relational Algebra)	關聯式計算 (Relational Calculus)
4.	以集合的基本運算 (如: union、intersection) 與關聯式運算子(如：select 、join) 為基礎。	以集合表示法與 predicate calculus 為基礎。例如： $\{ t \mid t \in R \wedge t.A = \text{“data”} \}$。
5.	表示能力與關聯式計算相等，而且每一個關聯式代數的查詢皆可轉成相對的關聯式計算查詢。	表示能力與關聯式代數相等，而且每一個關聯式計算的查詢皆可轉成相對的關聯式代數查詢。
6.	可以直接實現 (implement)，並且可以作為查詢最佳化 (Query Optimization) 時的樹狀結構中間型式 (可以轉成 Query Tree)。	通常是以轉成關聯式代數來實現 (implement)。

關聯式代數的簡介

- 傳統的集合運算子：
  - 聯集運算 (Union)，符號為 $\cup$
  - 交集運算 (Intersection)，符號為 $\cap$
  - 差集運算 (Difference)，符號為 \ (或 –)
  - 乘積運算 (Cartesian Product)，符號為 $\times$
- 特殊的關聯運算子：
  - 選擇運算 (Selection)，符號為 σ (Sigma)
  - 投影運算 (Projection)，符號為 π (Pi)
  - 合併運算 (Join)，符號為 $\bowtie$
  - 除法運算 (Divide)，符號為 $\div$

聯集相容 (Union-Compatible)

- 聯集、交集、差集做運算時, 運算元必須符合聯集相容的條件
  - 兩關聯表必須有相同的屬性集
  - 相對的屬性都要定義於相同的值域 (Domain) 上



封閉性 (Closure)

- 「封閉性」：運算所得到的結果仍是關聯表
- 這個關聯表的標題部份可以由原運算元的標題部份繼承而來
- 實體部份則是依不同運算元有不同的定義 (下一節說明)
- 封閉性讓關聯式代數得以巢狀方式出現：
  $\pi_Y(\sigma_p(R1) - \pi_x(R2))$

各運算子的詳細說明

- 關聯式代數的八個運算子中，可再分成
  - 原始運算子 (Primitive Operators)：
    σ π $-$ $\cup$ $\times$
  - 非原始運算子 (Non-primitive Operators)：
    $\bowtie$ $\div$ $\cap$
- 非原始運算子可以用原始運算子來模擬
  正如：算術中的乘法可以用加法來模擬一般
  5 * 3 = 5 + 5 + 5 (三次)

關聯式代數的八個運算子

- 以上述兩個不同的維度來呈現關聯式代數的八個運算子

	傳統的集合運算子	特殊的關聯運算子
原始運算子	×、∪、–	σ、π
非原始運算子	∩	⋈、÷

原始運算子 (Primitive)

- 卡迪生乘積 (Cartesian Product)，符號 R1 × R2
- 聯集運算 (Union)，符號 R1 ∪ R2
- 差集運算 (Diffenence)，符號 R1 – R2
- 選擇運算 (Selection)，符號 $\sigma_P(R)$
- 投影運算 (Projection)，符號 $\pi_A(R)$

※最後兩個為一元運算子

乘積運算 (Cartesian Product)

R_1

U	*V*
u1	v1
u2	v2
u3	v3

×

R_2

X	*Y*
x1	y1
x2	y2

=

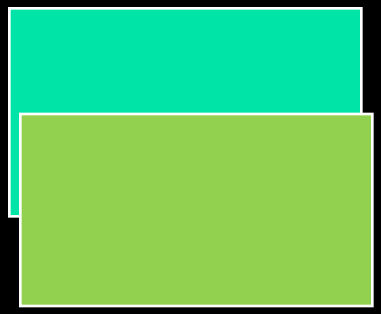
$R_1 \times R_2$

U	*V*	*X*	*Y*
u1	v1	x1	y1
u2	v2	x1	y1
u3	v3	x1	y1
u1	v1	x2	y2
u2	v2	x2	y2
u3	v3	x2	y2

- 若 R1 有 *a* 個屬性，R2 有 *b* 個屬性的話
  則 R1╳R2 會包含 $a + b$ 筆記錄
- 若 R1 有 *m* 筆記錄，R2 有 *n* 筆記錄的話
  則 R1╳R2 會包含 $m * n$ 筆記錄

聯集運算 (Union)

- R_1 與 R_2 必須要「聯集相容」才能做聯集



R_1

X	Y
x1	y1
x3	y3

∪

R_2

X	Y
x1	y1
x2	y2

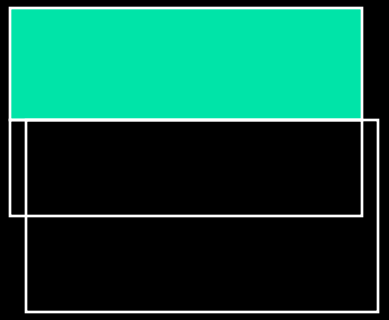
=

$R_1 \cup R_2$

X	Y
x1	y1
x2	y2
x3	y3

差集運算 (Set Difference)

- R_1 與 R_2 必須要「聯集相容」才能做差集



R_1

X	*Y*
x1	y1
x3	y3

—

R_2

X	*Y*
x1	y1
x2	y2

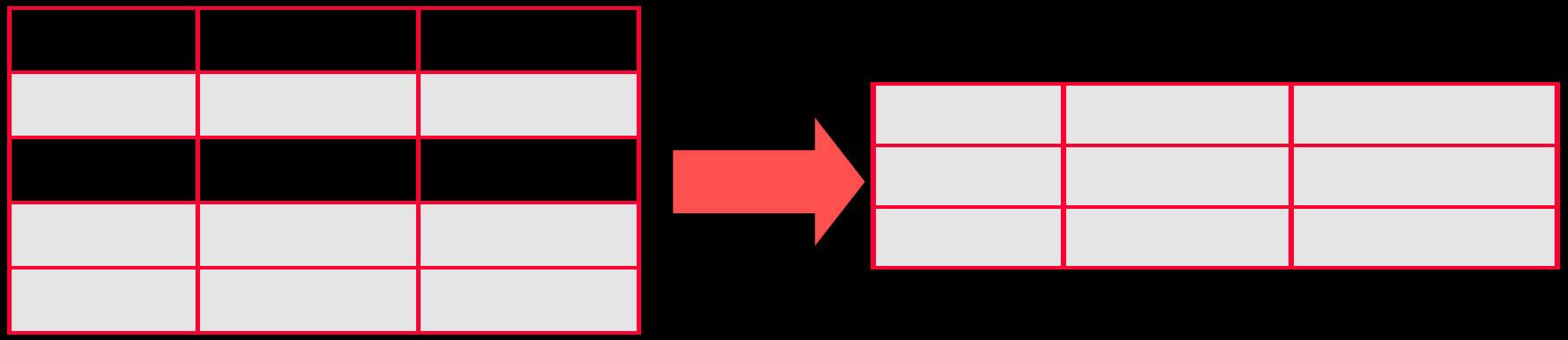
=

$R_1 - R_2$

X	*Y*
x3	y3

選擇運算 (Selection)

- 選擇運算是一元運算子，含有兩個參數
  - 選取條件 (Predicate) P
  - 關聯表名稱 R
- 從 R 中選出符合條件 P 的值組



選擇運算的範例

$\sigma_{price \geq 140}$ (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

選擇運算的範例 (續)

$\sigma_{id > 3}$ (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

選擇運算的範例 (續)

$\sigma_{price \geqq 140 \text{ and } id > 3}$ (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

查詢轉換與定性最佳化

- $\sigma_{C1 \text{ and } C2}(R) \equiv \sigma_{C1}(R) \cap \sigma_{C2}(R)$
- $\sigma_{C1 \text{ or } C2}(R) \equiv \sigma_{C1}(R) \cup \sigma_{C2}(R)$
- $\sigma_{not(C)}(R) \equiv R - \sigma_{C}(R)$
- 可以將一個選擇查詢變成兩個以上的查詢，適合平行處理
- 是定性查詢最佳化 (Qualitative Query Optimization) 的基礎

投影運算 (Projection)

- $\pi_{X1, X2,..., Xn}$ (R) 將 R 中的屬性 X1, X2,..., Xn 投影出來

$\pi_{bookname, price}$ (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~~

非原始運算子

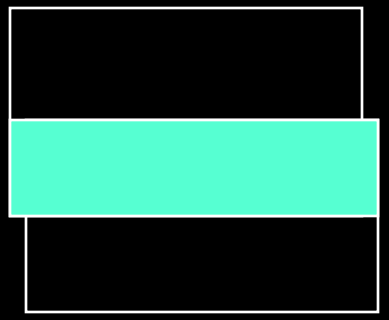
- 交集運算 (Intersection)，符號 $R_1 \cap R_2$
- 合併運算 (Join)，符號 $R_1 \bowtie R_2$
- 除法運算 (Divide)，符號 $R_1 \div R_2$

※全部都是二元運算子

※全部都可以由原始運算子來模擬

交集運算 (Intersection)

- R_1 與 R_2 必須要「聯集相容」才能做交集



R_1

X	Y
x1	y1
x3	y3

$\cap$

R_2

X	Y
x1	y1
x2	y2

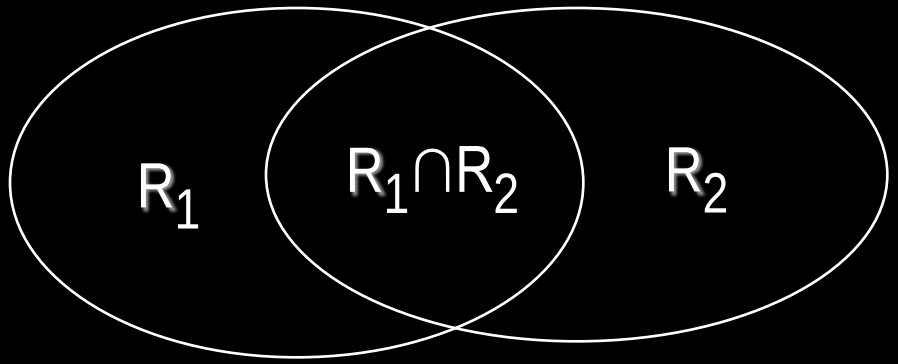
$=$

$R_1 \cap R_2$

X	Y
x1	y1

為什麼交集不是原始運算子？

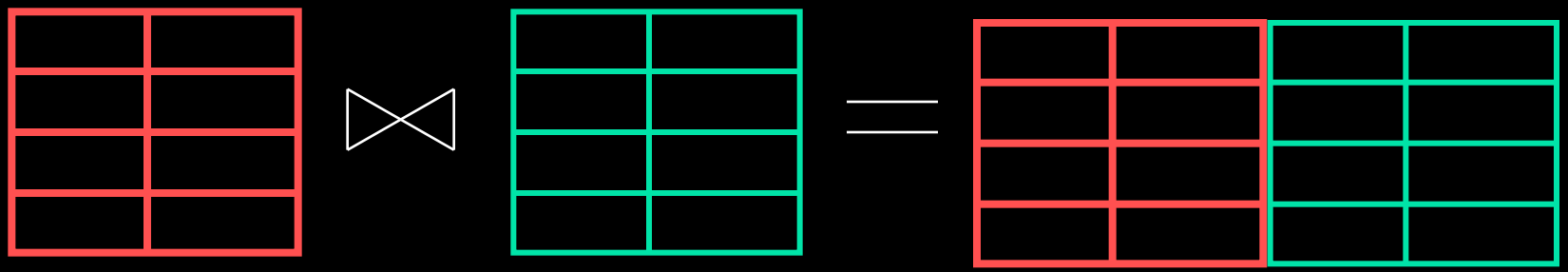
- 因為 $R_1 \cap R_2$
  $= R_1 - (R_1 - R_2)$
  $= R_2 - (R_2 - R_1)$



Venn Diagram

合併運算 (Join)

- 合併運算含有三個參數
  - 合併條件 (Join Predicate) P
  - 關聯表 R_1
  - 關聯表 R_2
- 將 R_1 與 R_2 依合併條件 P 做合併 $R_1 \bowtie_P R_2$



合併運算的範例

Emp

no	*name*	*dept_no*
1	張三	1
2	李四	2
3	王五	1
4	毛六	2

等位合併
(Equi-join)

Dept

no	*name*	*mgr_no*
1	會計部	3
2	工程部	4

通常是將外來鍵與所參考的主鍵做合併
(請練習白色部分的合併結果以及圖4.2)

Emp ⋈ Emp.*dept_no* = Dept.*no* Dept

Emp.*no*	Emp.*name*	*dept_no*	Dept.*no*	Dept.*name*	*mgr_no*
1	張三	1	1	會計部	3
2	李四	2	2	工程部	4
3	王五	1	1	會計部	3
4	毛六	2	2	工程部	4

自然合併運算 (Natural Join)

- 在執行等位合併運算後，結果中一定會有兩欄的資料完全相同
- 去除其中一欄仍不影響整體意義，我們稱為「自然合併」

Emp ⋈ Emp.*dept*_no = Dept.*no* Dept

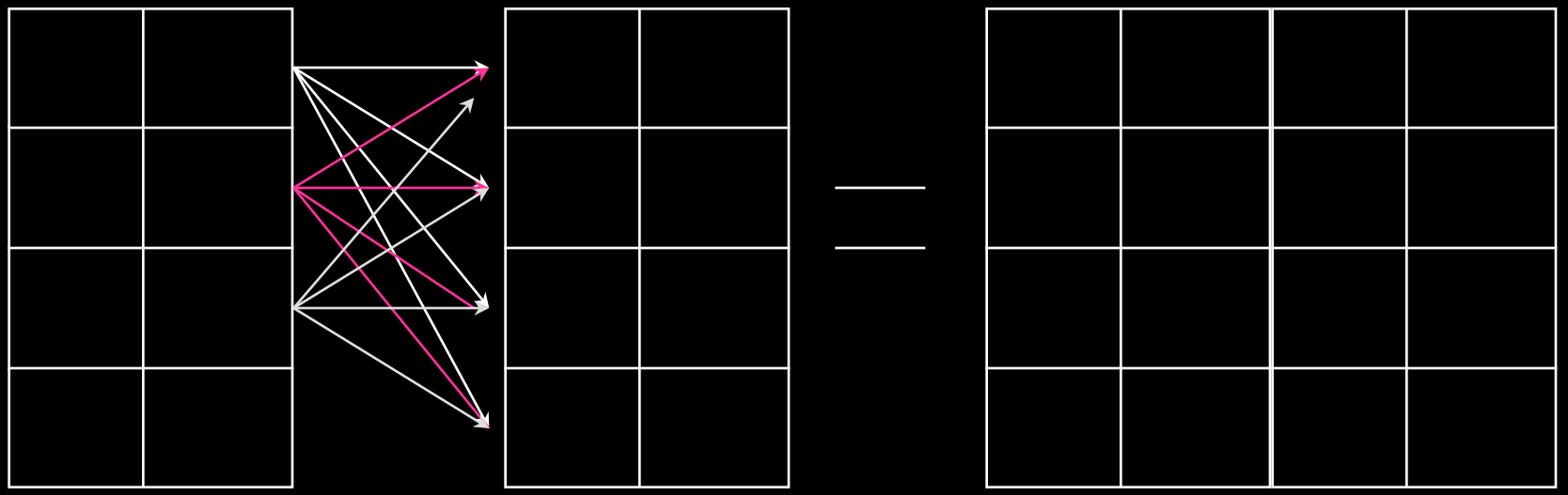
Emp.*no*	Emp.*name*	*dept_no*	Dept.*no*	Dept.*name*	*mgr_no*
1	張三	1	1	會計部	3
2	李四	2	2	工程部	4
3	王五	1	1	會計部	3
4	毛六	2	2	工程部	4

為什麼合併運算不是原始運算子？

- 因為 $R_1 \bowtie_P R_2 = \sigma_P (R_1 \times R_2)$
- 所以合併運算的結果最多只可能會產生 $m * n$ 筆記錄 (如果 R_1 (R_2) 有 m (n) 筆)
- 如果 P 是 $\varnothing$ 的話，$R_1 \bowtie R_2 = R_1 \times R_2$
- 合併運算是一個非常耗費時間的運算

合併運算的實現方式(一)

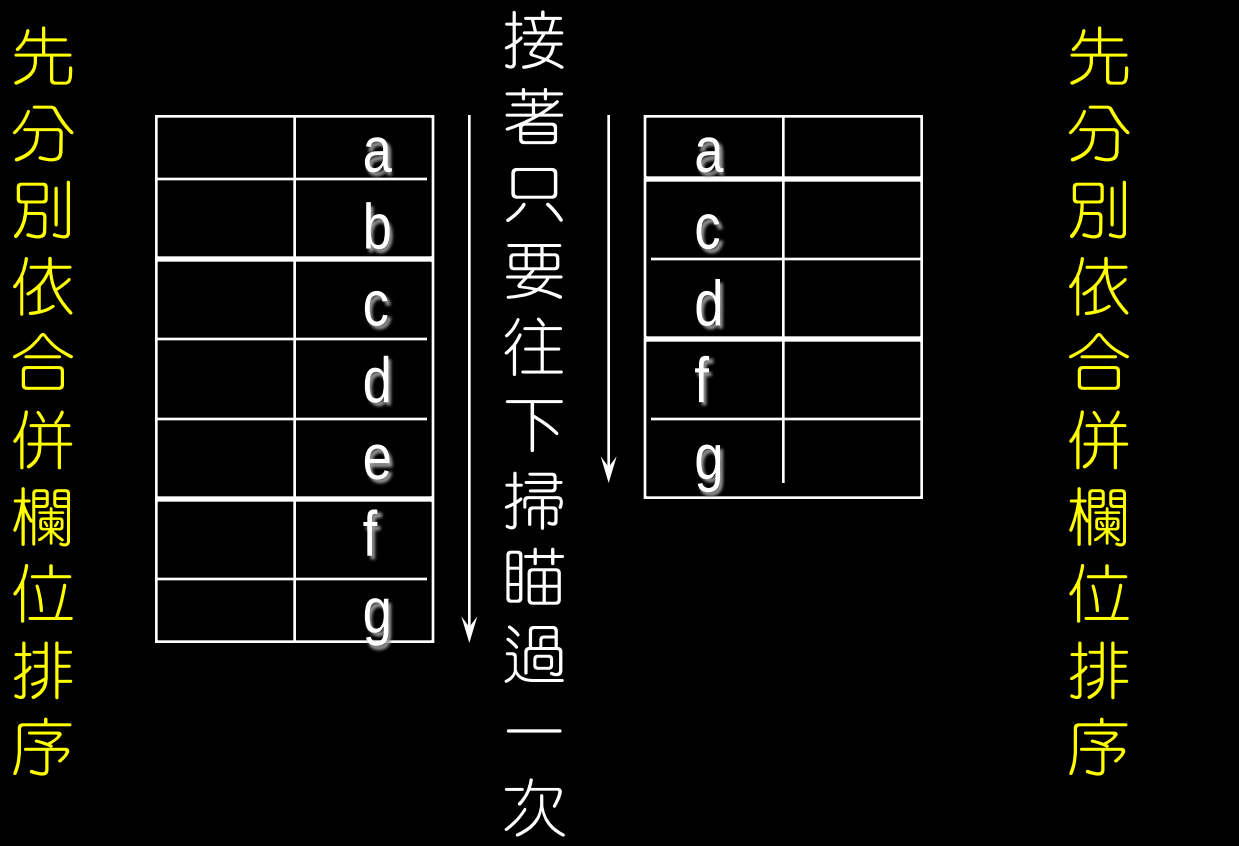
- 巢狀迴路法 (Nested Loop)



m 筆　　*n* 筆　　需要 O(*m* * *n*) 個步驟

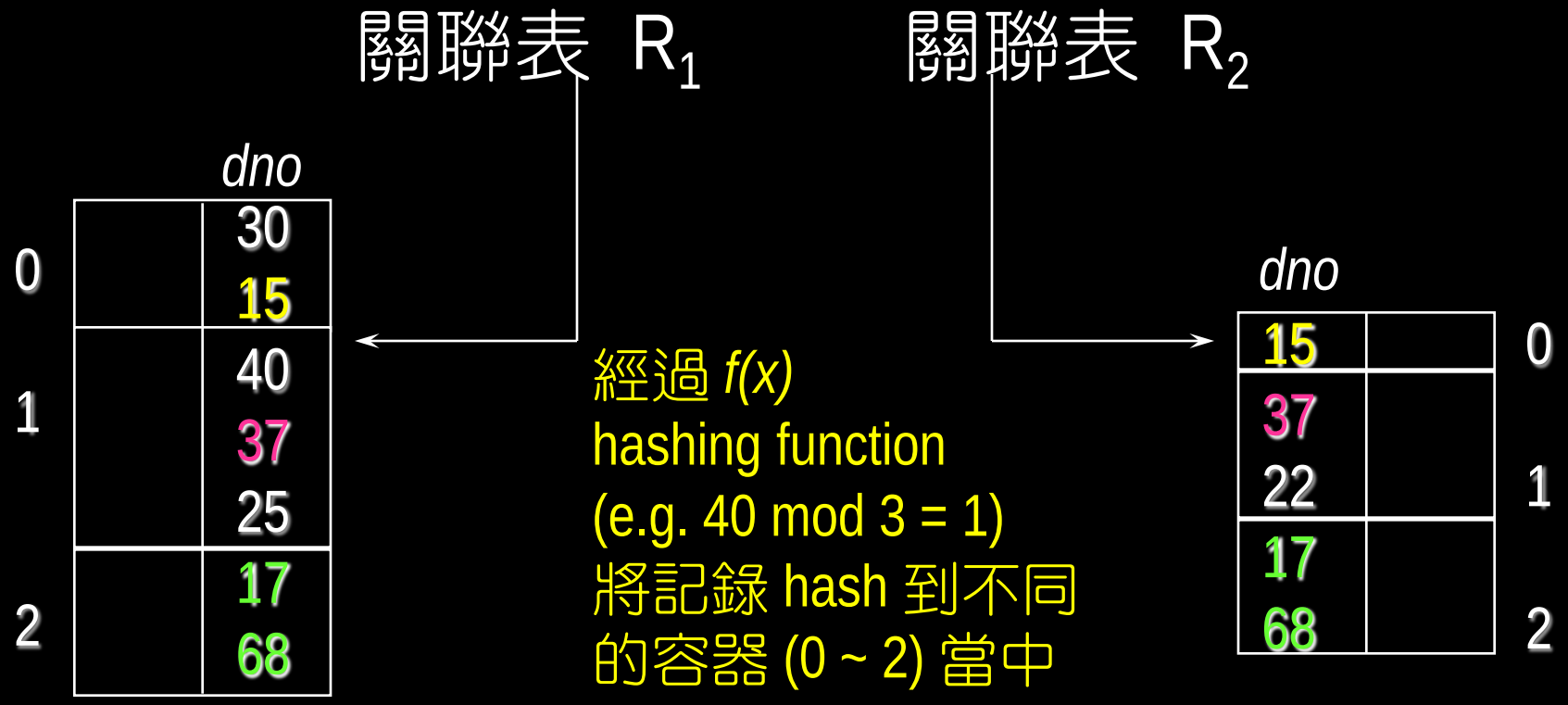
合併運算的實現方式(二)

- 排序合併法 (Sort/Merge)



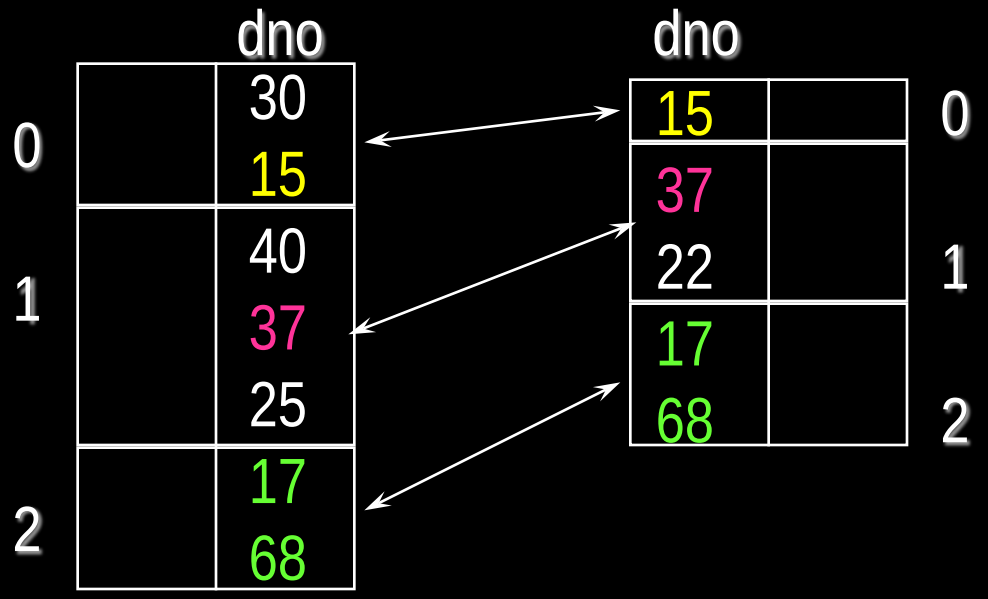
合併運算的實現方式(三)

- 雜湊法 (Hashing)：適用於等位合併



雜湊法 (Hashing) 續上頁

- 經過上述的作法將各值組對應到不同的容器中後，我們只要分別將 R_1 中屬於第 *i* 個 容器的資料與 R_2 中屬於第 *i* 個 容器的資料做比對即可
- 其觀念很像我們在玩拼圖一般：先依顏色相同的分類後再開始拼



合併運算的變形

- 「半合併」(Semi-join)：用於分散式資料庫上的查詢最佳化處理上 (See "資料庫系統理論與實務" Chap. 13 或 "資料庫系統進階實務" Chap. 5)
- 「外部合併」(Outer-join)：用於異質性分散式資料庫上的整合運算處理上 (See "資料庫系統理論與實務" Chap. 14 或 "資料庫系統進階實務" Chap. 7)
- 相對於「外部合併」而言，前面所提到的合併運算又稱為「內部合併」(Inner-join)

除法運算 (Division)

- 除法運算含有兩個運算元
  - 關聯表 R1 (當做被除表)
  - 關聯表 R2 (當做除表)
- $R1(X, Y) \div R2(Y) = \{ t \mid \{t\} \times R2(Y) \subseteq R1(X, Y)\}$
- 就像 38 ÷ 5 = 7 ... 3 一樣 ($7 * 5 \leq 38$)
- 通常用在查詢「找出符合所有 XX 的 XXX」
- 見下頁的例子

找出供應所有菜色的餐館！

Restaurants

hotel_name	*cuisine_name*
凱悅	台菜
凱悅	湘菜
凱悅	川菜
凱悅	西餐
希爾頓	湘菜
希爾頓	川菜
希爾頓	西餐

÷

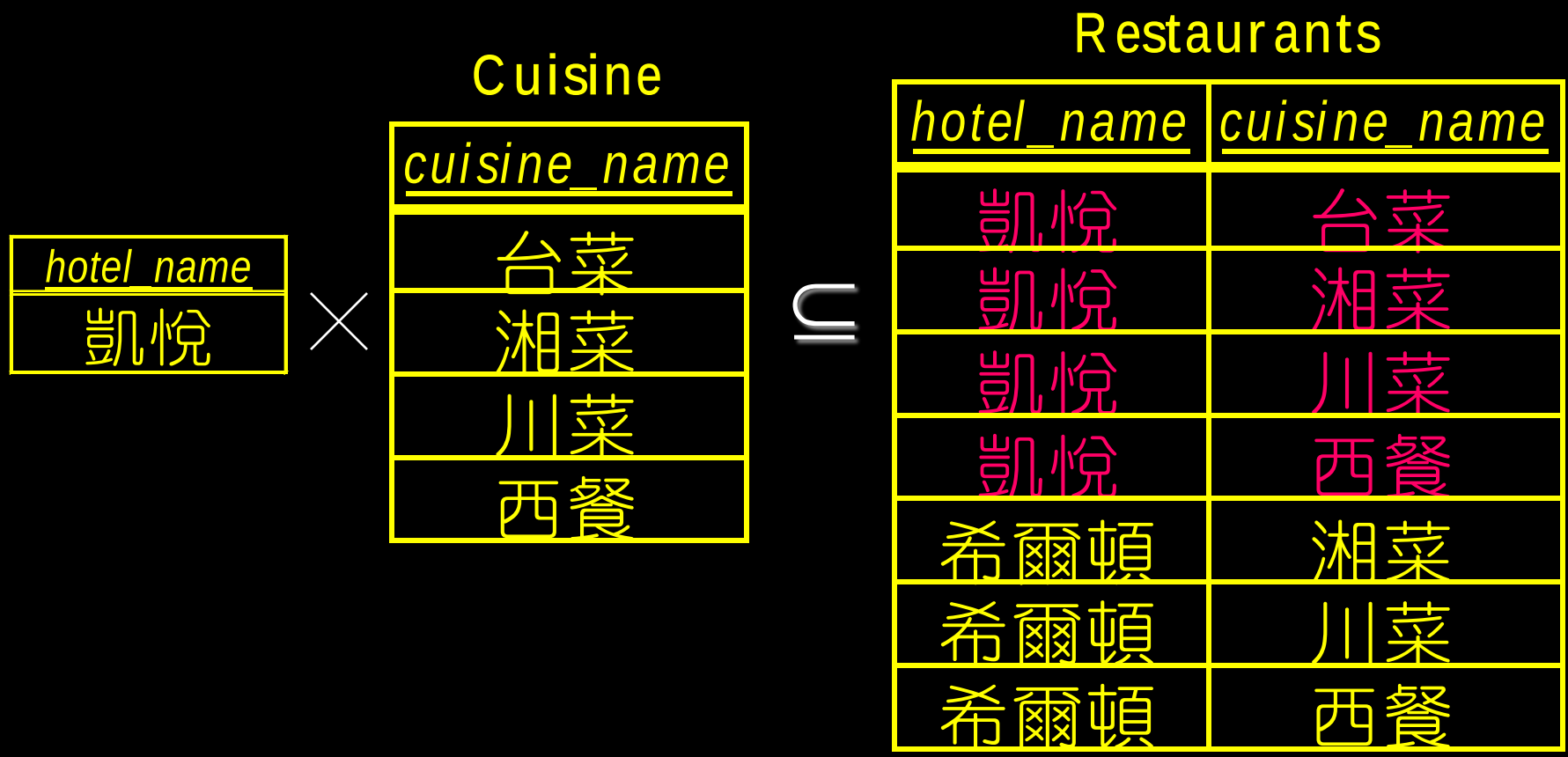
Cuisine

cuisine_name
台菜
湘菜
川菜
西餐

=

hotel_name
凱悅

驗證一下


hotel_name
凱悅
Cuisine
cuisine_name
台菜
湘菜
川菜
西餐
Restaurants
hotel_name
cuisine_name
凱悅 台菜
凱悅 湘菜
凱悅 川菜
凱悅 西餐
希爾頓 湘菜
希爾頓 川菜
希爾頓 西餐

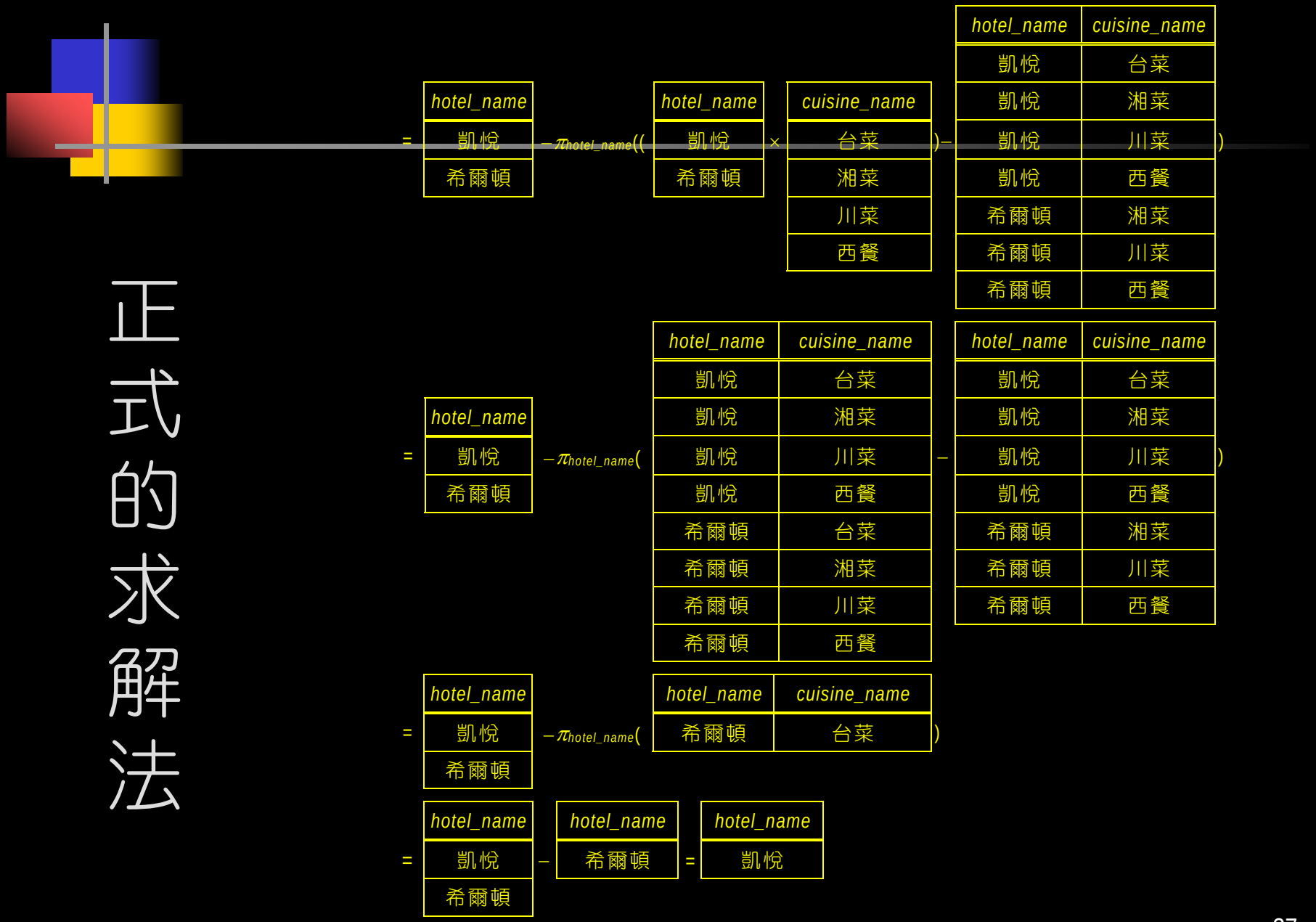
為什麼除法運算不是原始運算子？

- $R1(X, Y) \div R2(Y) =$
  $\pi_X(R1) - \pi_X((\pi_X(R1) \times R2) - R1)$
- 另外，$R1(X, Y) \div R2(Y) = \bigcap_{t \in R2} (\pi_X(\sigma_{Y=t}(R1)))$
- 除法運算的結果最多只可能會產生 m/n 筆記錄 (如果 R1 (R2) 有 m (n) 筆)
- 除法運算是一個非常耗費時間的運算

正式的求解法

Restaurant(*hotel_name*, *cuisine_name*) ÷ Cuisine(*cuisine_name*)

= π_{hotel_name}(Resuaurant) − π_{hotel_name}((π_{hotel_name}(Restaurant) × Cuisine) − Restaurant)



除法運算是對稱的

- 試看看你知道下面的答案嗎？意義為何？

Restaurant

hotel_name	*cuisine_name*
凱悅	台菜
凱悅	湘菜
凱悅	川菜
凱悅	西餐
希爾頓	湘菜
希爾頓	川菜
希爾頓	西餐

÷

Hotel

hotel_name
凱悅
希爾頓

= ?

關聯式代數的特性

- 結合律 (Associativity)
- 交換律 (Commutativity)
- 分配律 (Distributivity)
- 這些特性可以用來做查詢的轉換，是「定性查詢最佳化」(Qualitative Query Optimization) 的基礎
- 也可以輔助「定量查詢最佳化」

結合律 (Associativity)

- 1. $(A \cup B) \cup C = A \cup (B \cup C)$
- 2. $(A \times B) \times C = A \times (B \times C)$
- 3. $(A \cap B) \cap C = A \cap (B \cap C)$
- 4. $(A \bowtie_p B) \bowtie_q C = A \bowtie_p (B \bowtie_q C)$
- 但是 $(A - B) - C \neq A - (B - C)$
- 而且 $(A \div B) \div C \neq A \div (B \div C)$

以反例證明之 (差集)

A

X	Y
x1	y1
x2	y2
x3	y3

B

X	Y
x2	y2
x3	y3
x4	y7

C

X	Y
x3	y3
x4	y7
x5	y8

A − B

X	Y
x1	y1

−

C

X	Y
x3	y3
x4	y7
x5	y8

=

(A − B) − C

X	Y
x1	y1

以反例證明之(續)

A

X	Y
x1	y1
x2	y2
x3	y3

B

X	Y
x2	y2
x3	y3
x4	y7

C

X	Y
x3	y3
x4	y7
x5	y8

A

X	Y
x 1	y 1
x 2	y 2
x 3	y 3

–

B − C

X	Y
x 2	y 2

=

A − (B − C)

X	Y
x 1	y 1
x 3	y 3

以反例證明(除法)

A

W	*X*	*Y*	*X'*
w1	x1	y2	x1
w1	x1	y2	x2
w2	x2	y2	x1
w2	x2	y2	x2
w3	x3	y1	x1
w4	x4	y4	x2

B

Y	*X'*
y2	x1
y2	x2

C

X
x1

A ÷ B

W	*X*
w1	x1
w2	x2

(A ÷ B) ÷ C

W
w1

以反例證明(除法)

A

W	X	Y	X'
w1	x1	y2	x1
w1	x1	y2	x2
w2	x2	y2	x1
w2	x2	y2	x2
w3	x3	y1	x1
w4	x4	y4	x2

B

Y	X'
y2	x1
y2	x2

C

X
x1

B ÷ C

Y
y2

A ÷ (B ÷ C)

W	X	X'
w1	x1	x1
w1	x1	x2
w2	x2	x1
w2	x2	x2

交換律 (Commutativity)

- $\pi_{\Sigma}(\sigma_{Y=k}(A))$?= $\sigma_{Y=k}(\pi_{\Sigma}(A))$

  如果 $Y \in \Sigma$的話，則等號成立

  否則等號不成立

- $A \cap B = B \cap A$
- $A \cup B = B \cup A$
- $A \times B = B \times A$
- $A \bowtie_p B = B \bowtie_p A$
- $A - B \neq B - A$
- $A \div B \neq B \div A$

以反例證明 (差集)

A

X	Y
x1	y1
x2	y2
x3	y3

B

X	Y
x2	y2
x3	y3
x4	y7

A - B

X	Y
x1	y1

≠

B - A

X	Y
x4	y7

分配律 (Distributivity)

- 假定 A(X, Y) 而且 B(Y, Z)
- $\sigma_{A.X\ \theta\ C1\ \text{and}\ B.Z\ \theta\ C2}(A \bowtie_q B)$

  $= (\sigma_{A.X\ \theta\ C1}(A)) \bowtie_q (\sigma_{B.Z\ \theta\ C2}(B))$
- $(A \cup B) \bowtie_p (C \cup D) = (A \bowtie_p C) \cup$

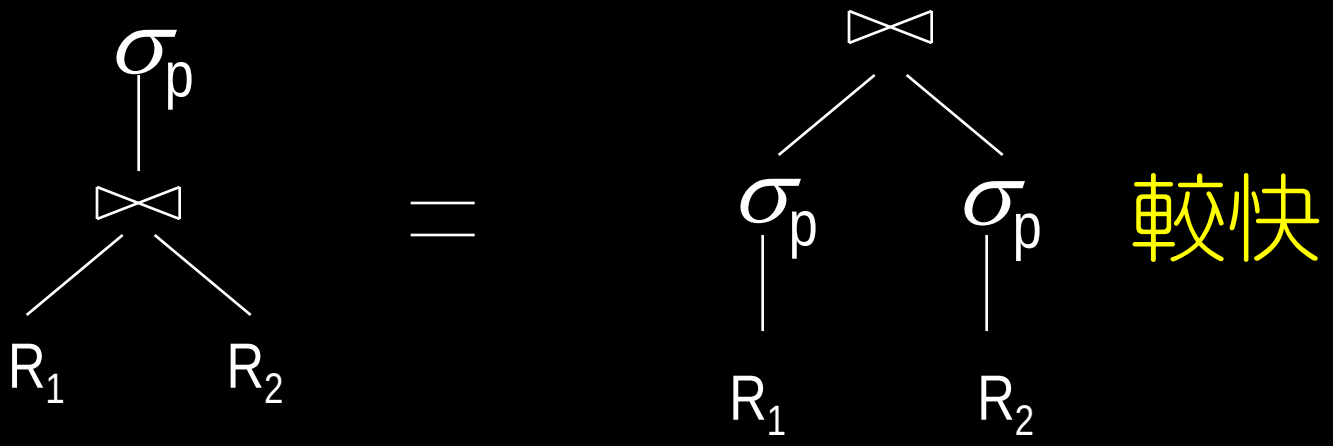
  $(A \bowtie_p D) \cup (B \bowtie_p C) \cup (B \bowtie_p D)$

查詢轉換 (Query Transformation)

- $\sigma_{C1 \text{ and } C2}(R) \equiv \sigma_{C1}(R) \cap \sigma_{C2}(R)$
- $\sigma_{C1 \text{ or } C2}(R) \equiv \sigma_{C1}(R) \cup \sigma_{C2}(R)$
- $\sigma_{not(C)}(R) \equiv R - \sigma_{C}(R)$
- 如果 S 包含於 T 中，則 $\pi_S(\pi_T(A)) = \pi_S(A)$
- See [S. Ceri and G. Pelagatti (1984)] for more details.

定性查詢最佳化的原則

- 將二元運算 (尤其是合併、除法) 延遲到越後面做越好
- 將一元運算 (選擇、投影) 提到前面做
- 通常是以 Query Tree 來表示一串查詢動作



範例

- $\sigma_{X = 'x1' \text{ and } Z = 'z1'} (A \bowtie_{A.Y = B.Y} B)$

A

X	Y
x1	y1
x2	y2
x3	y3
x4	y4

B

Y	Z
y1	z1
y2	z2
y3	z3
y1	z2

範例 (續)

- 先算 $A \bowtie_{A.Y = B.Y} B$ 可得

$$A \bowtie_{A.Y = B.Y} B$$

X	*A.Y*	*B.Y*	*Z*
x1	y1	y1	z1
x1	y1	y1	z2
x2	y2	y2	z2
x3	y3	y3	z3

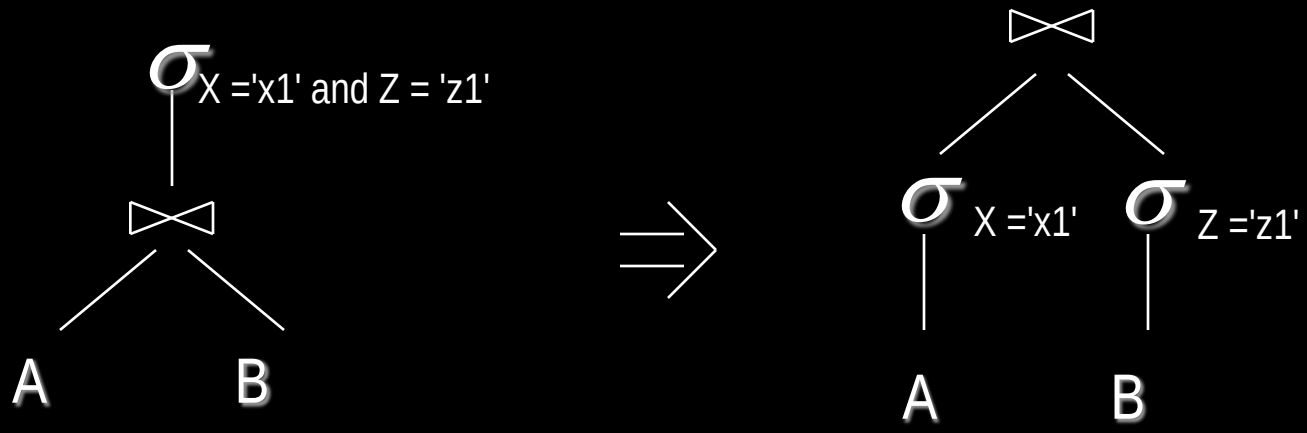
範例 (續)

- 再算 $\sigma_{X='x1' \text{ and } Z='z1'}(A \bowtie_{A.Y = B.Y} B)$ 可得

X	*A.Y*	*B.Y*	*Z*
x1	y1	y1	z1

經過查詢轉換後的做法

- 將 $\sigma_{X='x1' \text{ and } Z='z1'}(A \bowtie_{A.Y = B.Y} B)$ 轉換成 $(\sigma_{X='x1'}(A)) \bowtie_{A.Y = B.Y} (\sigma_{Z='z1'}(B))$



較快

查詢轉換後的做法 (續)

$\sigma_{X='x1'}(A)$

X	Y
x1	y1

$\sigma_{Z='z1'}(B)$

Y	Z
y1	z1

$(\sigma_{X='x1'}(A)) \bowtie_{A.Y = B.Y} (\sigma_{Z='z1'}(B))$

X	A.Y	B.Y	Z
x1	y1	y1	z1

退化的情況 (Degenerate Case)

- 如果 P 是 ∅ 的話，A ⋈ B = A × B
- 若 A 與 B 為「聯集相容」，而且合併的條件是：所有 A 與 B 的相對屬性都相等，而結果取「自然合併」 (Natural Join) 時，則 $A \bowtie_p B = A \cap B$

A

X	Y
x1	y1
x3	y3

∩

B

X	Y
x1	y1
x2	y2

=

A ∩ B

X	Y
x1	y1

See 7.6.2.2 節

A

X	Y
x1	y1
x3	y3

$\bowtie_p$

B

X	Y
x1	y1
x2	y2

=

$A \bowtie_p B$

A.X	A.Y	B.X	B.Y
x1	y1	x1	y1

取 自然合併 可得

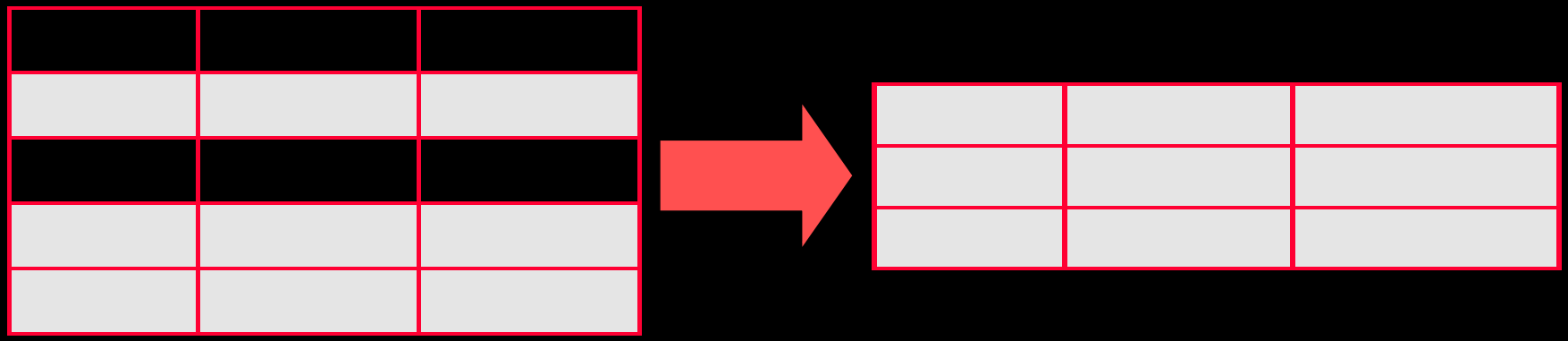
$A \bowtie_p B$

A.X	A.Y
x1	y1

P : A.X = B.X and A.Y = B.Y

關聯式代數之結果的主鍵

- 關聯式代數所求得的結果仍是一個關聯表
- 此結果關聯表的主鍵依如下規則取得：
  - $\sigma_p(A)$ ：結果的主鍵同關聯表 A 的主鍵，PK_A
    因為原主鍵仍然存在，所有主鍵值仍不會重覆



關聯式代數之結果的主鍵 (續)

- $\pi_{x_1, x_2, \ldots, x_n}(A)$：若投影出來的屬性中包含原主鍵 PK 的話，則投影後的主鍵仍為 PK，否則便由所有屬性組成整個投影後的主鍵。

$\pi_{bookname,\ price}$ (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~~

關聯式代數之結果的主鍵 (續)

- A×B ：假定 A 的主鍵為 PK_A、B 的主鍵為 PK_B，則 A×B 的主鍵為 (PK_A, PK_B)

A

PK_A	X
a1	b1
a2	b2
a3	b3

×

B

PK_B	Y
x1	y1
x2	y2

=

A × B

PK_A	X	PK_B	Y
a1	b1	x1	y1
a2	b2	x1	y1
a3	b3	x1	y1
a1	b1	x2	y2
a2	b2	x2	y2
a3	b3	x2	y2

關聯式代數之結果的主鍵 (續)

- A ∪ B ：A與B的共同主鍵所構成 (因為聯集相容)，或由所有屬性構成

A

PK_A	X
a1	b1
a3	b3

∪

B

PK_B	X
a1	b1
a2	b2

=

A ∪ B

PK_A /PK_B	X
a1	b1
a2	b2
a3	b3

A

PK_A	X
a1	b1
a3	b3

∪

B

X	PK_B
a1	b1
a2	b2

=

A ∪ B

PK_A	PK_B
a1	b1
a2	b2
a3	b3

關聯式代數之結果的主鍵 (續)

- A－B ： A的主鍵 PK_A
- A ∩ B ： A的主鍵 PK_A，或 B 的主鍵 PK_B

A

PK_A	X
a1	b1
a3	b3

—

B

PK_B	X
a1	b1
a2	b2

=

A — B

PK_A	X
a3	b3

A

PK_A	X
x1	y1
x3	y3

∩

B

PK_B	X
x1	y1
x2	y2

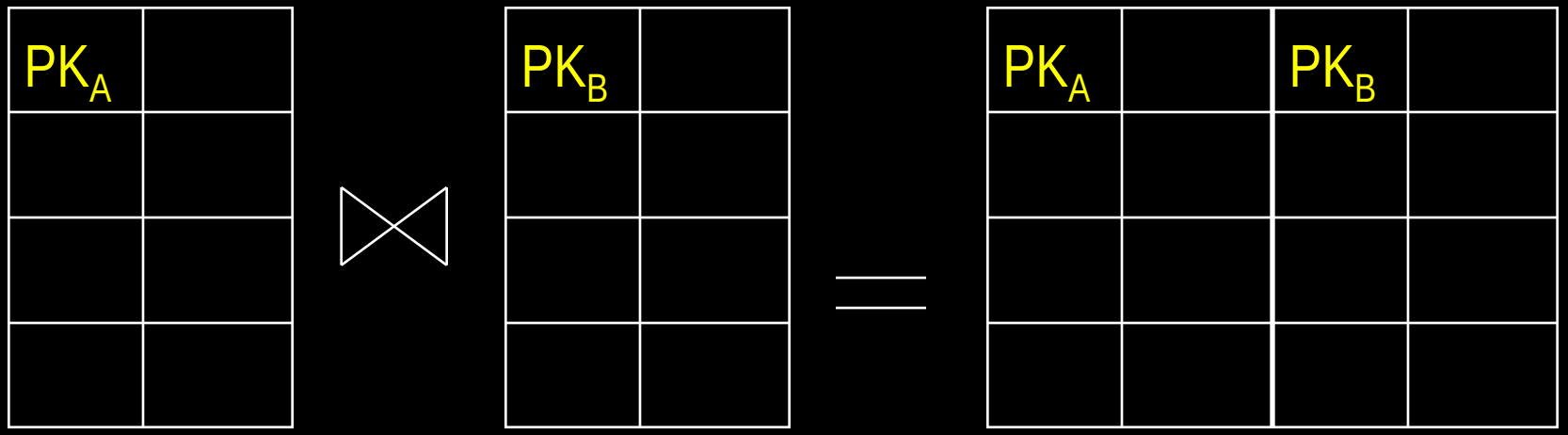
=

A ∩ B

PK_A /PK_B	X
x1	y1

關聯式代數之結果的主鍵 (續)

- $A \bowtie B$ ： (PK_A, PK_B)
- $A \div B$ ： 一般是由除得結果的所有屬性構成



關聯式代數的用途

- 定義更新資料所需的範圍，See 第七章
- 模擬「新增」、「刪除」、「修改」
  - Bookstores ∪ {(6, '元智書坊', 10, '中壢市')}
  - Bookstores – {(5, '獅子書局', 30, '台南市')}
  - Bookstores – {(5, '獅子書局', 30, '台南市')} ∪ {(5, '獅子書局', 30, '台中市')}

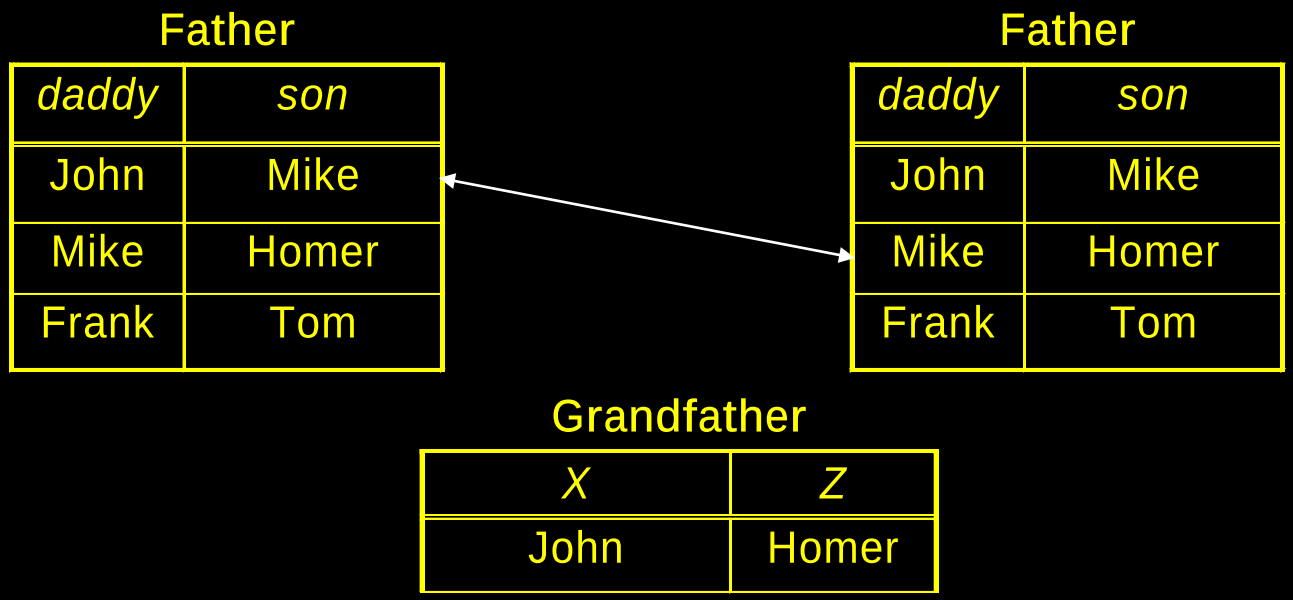
關聯式代數的用途 (續)

- 定義「視界」的範圍，See 第八章
- 定義查詢所需的範圍，並將它存成另一個關聯表 (瞬間關聯表)
- 用來定義欲保護資料之範圍 (資料安全)
- 定義整合限制規則的範圍
- 定性查詢最佳化

關聯式代數的用途 (續)

- 將知識規則轉換成關聯式代數來執行
- Grandfather (*X*, *Z*) :- Father (*X*, *Y*), Father (*Y*, *Z*).

  = $\pi_{F1.X,\ F2.Z}(F1(X, Y) \bowtie_{F1.Y = F2.Y} F2(Y, Z))$



關聯式計算

[範例] “請找出訂購「紅樓夢」的書局名稱 name 與所在城市 city”

- 以關聯式代數來完成此查詢：
  - 先將 Bookstores, Orders, Books 做合併運算
  - 將上述結果以 *bookname* = “紅樓夢” 條件來選擇
  - 最後將上述結果的 *name* 與 *city* 投影出來

$$\pi_{name,\ city}\left(\sigma_{bookname\ =\ '紅樓夢'}(\text{Bookstores} \bowtie_{Bookstores.no\ =\ Orders.no} \text{Orders} \bowtie_{Orders.id\ =\ Books.id} \text{Books})\right)$$

必須一步一步寫出步驟 (描述 How?)

關聯式計算(續)

- 以關聯式計算來完成此查詢 (只要描述 What 即可)：

"取得書局的名稱 *name* 與其所在城市 *city*，條件是 Orders 關聯表中要存在一筆該書局編號 *no* 的訂購資料，同時該筆資料的書籍編號 *id* 與 Books 關聯表中書名 *bookname* 為「紅樓夢」的書籍編號 *id* 相同。"

- 也可以 Predicate Calculus 寫成

$$\{ (r.name, r.city) \mid (\exists r)(\exists s)(\exists t)(r \in \text{Bookstores} \wedge s \in \text{Orders} \wedge t \in \text{Books} \wedge r.no = s.no \wedge s.id = t.id \wedge t.bookname = \text{'紅樓夢'})\}$$

Relational Calculus *vs.* 1st Order Predicate Calculus

- 「關聯式計算」是從「第一階賓辭邏輯」(the First Order Predicate Calculus) 發展出來的，
- 「第一階賓辭邏輯」是「數學邏輯」(Mathematical Logic) 的分支，
- 第一階的意義為：「限量詞」(Quantifier) 所限制的變數只能是簡單的個體，不可以是一個集合。

$$\{(r.name, r.city) \mid (\exists r)(\exists s)(\exists t)(r \in \text{Bookstores} \wedge s \in \text{Orders} \wedge t \in \text{Books} \wedge r.no = s.no \wedge s.id = t.id \wedge t.bookname = \text{‘紅樓夢’})\}$$

賓辭邏輯 vs.資料庫查詢語言

- 將「賓辭邏輯」的觀念應用在資料庫的查詢語言上，最早是由 J.L. Kuhns [J.L. Kuhns (1967)] 在1967年提出的 ；
- 以它來定義「關聯式計算」的查詢語言，則是 1972 年的 E.F. Codd [E.F. Codd (1972)]。
- 當時 E.F. Codd 還提出了「關聯完整性」(Relational Completeness) 的觀念。

關聯式計算的種類

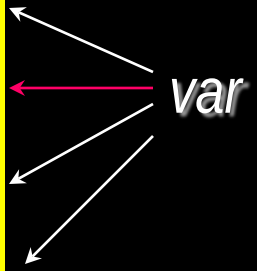
- 「值組導向關聯式計算」(Tuple-Oriented Relational Calculus)—Ingres 系統 (由 Michael Stonebraker發展) 原來所使用的 QUEL 便屬於此類
- 「值域導向關聯式計算」(Domain-Oriented Relational Calculus)—Query-By-Example (QBE) (由 M.M. Zloof 所提出) 便屬於此類

值組導向關聯式計算

- 值組變數 (Tuple Variables)：
  range of *var* is *relation* 表示變數 *var* 的值域範圍是 *relation*，也就是說 *var* 可以代表 *relation* 中的任何一筆記錄

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



值組導向關聯式計算語法

- 使用值組變數描述所要查詢的條件，語法如下：
  var.*attribute* [where <wff>]
  where 的條件是一個「完構語式」(Well-Formed Formula, 簡稱 wff)
- Well-Formed Formula 可以用遞迴方式定義各種條件的組合 (請見下頁)

完構語式的語法

- 以 BNF (Backus-Normal Form, 或 Backus-Naur Form) 來撰寫：

<wff>:- <condition>　(例如： *s.rank* > 20)
　　| NOT <wff>　(或寫成 “¬<wff>”)
　　| <wff> AND <wff>　　(或寫成 “<wff> ∧ <wff>”)
　　| <wff> OR <wff>　　(或寫成 “<wff> ∨ <wff>”)
　　| IF <condition> THEN <wff>
　　　　(≡ NOT <condition> OR <wff>)
　　| EXISTS *var* (<wff>)　(或寫成 “∃ *var* (<wff>)”)
　　| FORALL *var* (<wff>)　(或寫成 “∀ *var* (<wff>)”)
　　| (<wff>);

- 其中 *var* 代表值組變數，<condition> 代表純數值 (Scalar) 間的比較 (=, <, >, >=, <=, ≠)，純數值可以是屬性值或常數值。

限量詞的種類

- 限量詞有兩種：
  - EXISTS：稱為「存在限量詞」(Existential Quantifier)，其數學代號為 ∃，用來表示必須有某一個成員符合括弧中的 <wff> 條件，才能使 EXISTS *var* (<wff>) 為「真」(True)。
  - FORALL：稱為「總體限量詞」(Universal Quantifier)，其數學代號為 ∀，用來表示所有成員都必須符合括弧中的 <wff> 條件，才能使 FORALL *var* (<wff>) 為「真」(True)。

值組導向關聯式計算—範例

- “請找出位於臺北市或 *rank* 大於 20 的書局名稱”

其值組導向關聯式計算可以寫成：

range of *s* is Bookstores　(宣告 tuple variable *s*)

s.name where *s.city* = ‘臺北市’ OR *s.rank* > 20

總體限量詞與存在限量詞的巢狀結構

- 總體限量詞與存在限量詞可以互相層層包覆，以形成巢狀結構
- 透過某些轉換法則，可以將它們都提到最外層。有興趣的讀者請參考數學邏輯或人工智慧的書籍：
  - N.J. Nilson, *Principles of Artificial Intelligence*, [N.J. Nilson (1983)]；
  - M.R. Genesereth and N.J. Nilson, *Logical Foundations of Artificial Intelligence* [M.R. Genesereth & N.J. Nilson (1988)]。

一些完構語式 <wff> 的例子

- NOT (*t.city* = '臺南市')
- *t.rank* = *s.rank* AND *t.no* ≠ 2
- EXISTS *t* (*t.no* = *s.no* AND *s.no* = 2)
- FORALL *s* (*s.rank* > 10)

完構語式的等式關係

- IF <condition> THEN <wff> ≡ NOT <condition> OR <wff> (可由真值表推得)

<cond>	<wff>	IF <cond> THEN <wff>
0	0	1
0	1	1
1	0	0
1	1	1

<cond>	<wff>	NOT <cond> OR <wff>
0 (1)	0	1
0 (1)	1	1
1 (0)	0	0
1 (0)	1	1

NOT <cond>

完構語式的等式關係 (續)

- FORALL *var* (<wff>) ≡ NOT EXISTS *var* (NOT <wff>)

  「所有 *var* 都符合 <wff>」的意義就等於

  「不存在一個 *var* 不符合 <wff>」

- EXISTS *var* (<wff>) ≡ NOT FORALL *var* (NOT <wff>)

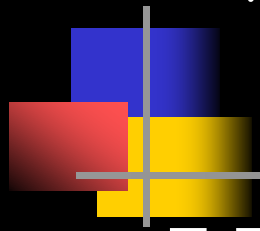
  「存在 *var* 符合 <wff>」的意義就等於

  「並非所有的 *var* 都不符合 <wff>」

牽涉兩個值組變數的例子

- “請列出訂購某本書籍，且數量為 40 本的書局名稱。”
- 值組導向關聯式計算的查詢為：
  range of *s* is Bookstores
  range of *o* is Orders
  s.name where *s.no* = *o.no* AND *o.quantity* = 40
- 請同學們想想看，能否理解上述查詢的含意?

將值組導向關聯式計算轉成關聯式代數



- E.F. Codd (1972) 年提出演算法 (Algorithm)。
- 只適用於值組導向關聯式計算的條件中沒有 OR 的情況，
- 原理簡單容易理解如何將值組導向關聯式計算轉換成關聯式代數。

將值組導向關聯式計算轉成關聯式代數 (Algorithm)

- 假設所有 Forall 與 Exists 都已經提到最外層：
  - 針對 where 條件的每個值組變數 *ti*，
  - 從 <wff> 中找出所有 *ti.attr* θ *cj* 形式的比較條件 (如：*t1.city* = “臺北市”)，拿來篩選 *ti* 所對應的關聯表 R_i：$\sigma_{t1.attr\ \theta c1 \text{ and } t1.attr\ \theta c2}(R_1)$ = 產生 R’
  - 若沒有上述形式的比較條件，則以 ∅ (空集合) 做為篩選條件 (將所有關聯表都做 Cartesian Product) 產生 R’
  - 再以所有合併條件 (如： *t.city* = *s.city*) 去選擇上述的關聯表 R’ = 得到 $\sigma_{t1.attr\ \theta t2.attr}(R')$

將值組導向關聯式計算轉成關聯式代數 (續)

- 從右至左對於所有的限量詞 (Quantifiers) (包含 EXISTS 和 FORALL) 採取下面的做法：
  - 若是EXISTS *s*，則去除 *s* 對應關聯表上的所有屬性，其餘都投影出來，
  - 若是 FORALL *t*，則將結果除以 *t* 所相對應的關聯表
- 最後將所需的答案投影出來

一個轉換的範例

- “請列出 *rank* 大於等於 20 且訂購所有書籍的書店名稱”。
  其值組導向關聯式計算可以寫成：

range of *s* is Bookstores

range of *o* is Orders

range of *t* is Books

s.name where FORALL *t* (EXISTS *o* (*s.rank* >= 20 AND *o.no* = *s.no* AND *t.id* = *o.id*)))

一個轉換的範例 (續)

- 第一步驟完後，得到三個結果
  - 完整的 Books 關聯表：有 6 筆值組 (此乃因為 Books 上沒有選擇條件)
  - 完整的 Orders 關聯表：有 12 筆值組 (此乃因為 Orders 上沒有選擇條件)
  - $\sigma_{rank >= 20}$(Bookstores)：有 4 筆值組 (因為 Bookstores 上有一個選擇條件 rank >= 20)

一個轉換的範例 (續)

- 第二步驟將上面三個結果做乘積運算
  - R' = Books $\times$ Orders $\times$ $\sigma_{rank >= 20}$(Bookstores)：
    共有 6 * 12 * 4 筆值組 = 288 筆值組
- 第三步驟以合併條件來選取上述關聯表 R'

$$R'' = \sigma_{Orders.no = R.no \text{ and } Books.id = Orders.id}(R')$$
$$= \sigma_{Orders.no = R.no \text{ and } Books.id = Orders.id}(Books \times Orders \times R)$$
$$= (Books \bowtie_{Books.id = Orders.id} Orders \bowtie_{Orders.no = R.no} R)$$

得到下頁的結果...

第三步驟得到的結果

Books ⋈Books.*id* = Orders.*id* Orders ⋈Orders.*no* = R.*no* R

no	*name*	*rank*	*city*	*id*	*bookname*	*author*	*price*	*publisher*	*no*	*id*	*quantity*
1	巨蟹書局	20	臺北市	1	三國演義	羅貫中	120	古文出版社	1	1	30
1	巨蟹書局	20	臺北市	2	水滸傳	施耐庵	170	中庸出版社	1	2	20
1	巨蟹書局	20	臺北市	3	紅樓夢	曹雪芹	170	春秋出版社	1	3	40
1	巨蟹書局	20	臺北市	4	西遊記	吳承恩	140	聊齋出版社	1	4	20
1	巨蟹書局	20	臺北市	5	水經注	酈道元	120	易經出版社	1	5	10
1	巨蟹書局	20	臺北市	6	道德經	老子	190	大唐出版社	1	6	10
3	水瓶書店	30	新竹市	2	水滸傳	施耐庵	170	中庸出版社	3	2	20
4	天秤書局	20	臺中市	2	水滸傳	施耐庵	170	中庸出版社	4	2	20
4	天秤書局	20	臺中市	4	西遊記	吳承恩	140	聊齋出版社	4	4	30
4	天秤書局	20	臺中市	5	水經注	酈道元	120	易經出版社	4	5	40

一個轉換的範例 (續)

- 第四步驟由右至左處理 FORALL 與 EXISTS
  - 首先遇見 EXISTS *o*，則由上面的結果中去除所有 Orders 上的屬性，得到
    R''' = $\pi_{no, name, rank, city, id, bookname, author, price, publisher}$(R'')
  - 接著遇見 FORALL *t*，則將結果除以 Books，得到下頁的結果

第四步驟得到的結果

$\pi_{no,\ name,\ rank,\ city,\ id,\ bookname,\ author,\ price,\ publisher}(R'')$

no	name	rank	city	id	bookname	author	price	publisher
1	巨蟹書局	20	臺北市	1	三國演義	羅貫中	120	古文出版社
1	巨蟹書局	20	臺北市	2	水滸傳	施耐庵	170	中庸出版社
1	巨蟹書局	20	臺北市	3	紅樓夢	曹雪芹	170	春秋出版社
1	巨蟹書局	20	臺北市	4	西遊記	吳承恩	140	聊齋出版社
1	巨蟹書局	20	臺北市	5	水經注	酈道元	120	易經出版社
1	巨蟹書局	20	臺北市	6	道德經	老子	190	大唐出版社
3	水瓶書店	30	新竹市	2	水滸傳	施耐庵	170	中庸出版社
4	天秤書局	20	臺中市	2	水滸傳	施耐庵	170	中庸出版社
4	天秤書局	20	臺中市	4	西遊記	吳承恩	140	聊齋出版社
4	天秤書局	20	臺中市	5	水經注	酈道元	120	易經出版社

第四步驟 (續) 與第五步驟

- 接著將結果除以 *t* 所對應的關聯表 Books，得到：

$$R''' \div Books$$

no	*name*	*rank*	*city*
1	巨蟹書局	20	臺北市

- 第五步驟將所需的欄位 *name* 投影出來：

$$\pi_{name}(R''' \div Books)$$

name
巨蟹書局

值域導向關聯式計算

- 值域導向關聯式計算以值域變數定義查詢語法
- 值域變數 (vs. 值組變數)

成員條件 (Membership Condition)

- 「值域導向關聯式計算」比「值組導向關聯式計算」多提供了「成員條件」(Membership Condition) 的條件形式，
- 成員條件可以用來宣告值域變數和設定屬性等於某值的查詢條件，其語法如下：

$$relation\ (X_1{:}v_1, X_2{:}v_2, \ldots)$$

其中 *relation* 是關聯表名稱，X_i 是屬性名稱，v_i 是 X_i 的值域變數或是屬性值。

值域變數不用在查詢前宣告

- 值域變數不用在查詢前宣告，只要在查詢時的 where 條件中宣告。
- 「值域導向關聯式計算」的語法如下：

  var [where <wff>]

  var 為值域變數，<wff> 的定義如下頁所示
- ...

值域導向關聯式計算的完構語式

- <wff> 定義如下:

<wff>:- <condition>　　(例如：*r* > 20)

| <membership-condition>　(例如: **Bookstores**(name: s, city:'臺北市'))

| NOT <wff>　　(或寫成 "¬ <wff>")

| <wff> AND <wff> (或寫成 "<wff> ∧ <wff>")

| <wff> OR <wff>　(或寫成 "<wff> ∨ <wff>")

| IF <condition> THEN <wff>　　(或寫成 "<condition> ⇒ <wff>")

| EXISTS var (<wff>)　　(或寫成 "∃ var (<wff>)")

| FORALL var (<wff>)　　(或寫成 "∀ var (<wff>)")

| (<wff>);

值域導向關聯式計算—範例 1

- “請找出位於臺北市或 *rank* 大於 20 的書局名稱。”
- 其值域導向關聯式計算可以寫成：

  其值域導向關聯式計算的寫法是：

  s where Bookstores(*name*: *s*, *city*: ‘臺北市’) OR
  (EXISTS *r* (*r* > 20 AND Bookstores(*name*: *s*, *rank*: *r*)))

- 注意：當條件是「值域變數 θ 常數值」，且 θ 是非等號 (=) 的比較符號時，必須使用 EXISTS 來加以描述。

值域導向關聯式計算—範例 2

- “請列出訂購某本書籍，且數量為 40 本的書局名稱。”

  值域導向關聯式計算的寫法為：

  s where EXISTS *n* (Bookstores(*no*: *n*, *name*: *s*) AND
  Orders(*no*: *n*, *quantity*: 40))

值組導向 vs. 值域導向

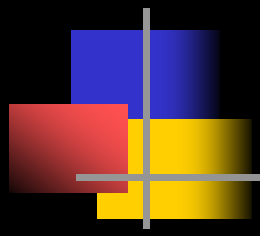
- 請列出訂購紅樓夢的書局名稱與所在城市。
  - 以值組導向關聯式計算表示較複雜

```
range of s is Bookstores
 range of o is Orders
range of t is Books
s.name, s.city where EXISTS o (s.no = o.no AND
          EXISTS t (o.id = t.id AND t.bookname = '紅樓夢')
```

  - 以值域導向關聯式計算表示較簡單

s, *c* where EXISTS *i* EXISTS *n* (Bookstores(*no:n*, *name:s*, *city:c*) AND Orders(*no:n*, *id:i*) AND Books(*id:i*, *bookname:* '紅樓夢'))

其意義為 Join



Query-By-Example (QBE)

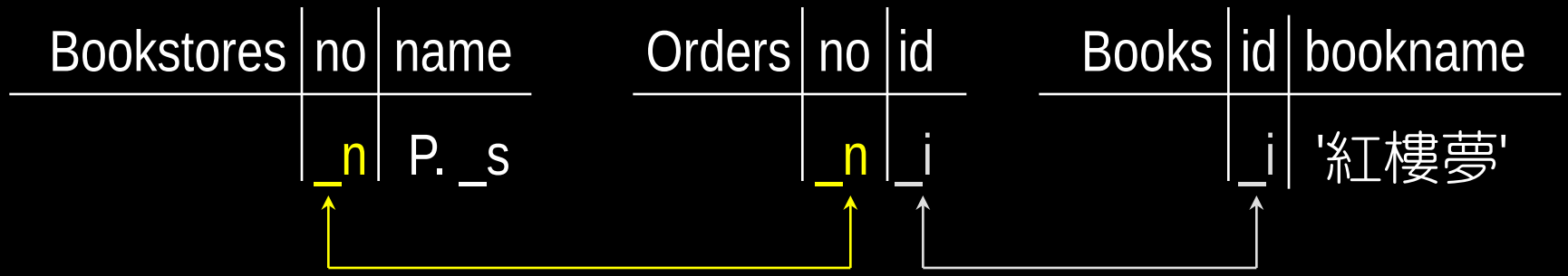
- 以值域導向關聯式計算為基礎所發展出來
- 由 M.M. Zloof 於 1975 年所提出
- 讓使用者直接在表格中填入所要的資料與條件，不必撰寫複雜的指令
- 早期人性化人機介面的濫觴

牽涉多個關聯表的QBE範例

- 請列出訂購 '紅樓夢' 的書局名稱。
  - 其值域導向關聯式計算的寫法為：

s where EXISTS *i* EXISTS *n* (Bookstores(*no:n*, *name:s*) AND Orders(*no:n*, *id:i*) AND Books(*id:i*, *bookname:* '紅樓夢'))

  - QBE寫法為：



QBE 範例 (1)

- 列出所有書局的編號 *no*、名稱 *name*、等級 *rank* 與所在城市 *city*，列出時先將等級由大至小排列，若等級相同則以書局編號由小到大排列

Bookstores	*no*	*name*	*rank*	*city*
P.	AO(2)		DO(1)	

- DO(1) ： DO 表 Descending Order，(1) 表示主要排序欄位
- AO(2) ： AO 表 Ascending Order，(2) 表示次要排序欄位

QBE範例 (2)

- 列出等級大於 20 而且位於 '臺南市' 的書局名稱

Bookstores	*name*	*rank*	*city*
	P.	>20	‘臺南市’

- 寫在同一列的條件代表 AND

QBE範例 (3)

- 列出等級大於 20 或是位於 '臺南市' 的書局名稱

Bookstores	*name*	*rank*	*city*
	P.	>20	
	P.		'臺南市'

- 寫在不同列的條件代表 OR
  另一種寫法

Bookstores	*name*	*rank*	*city*		*CONDITIONS*
	P.	_r	_c		_r > 20 OR _c = '臺南市'

QBE範例 (4)(5)

- 列出等級介於 10 與 25 之間的書局名稱

Bookstores	*name*	*rank*	*rank*
	P.	>=10	<=25

- 請將所有位於 '高雄市' 的書局等級都改成 25

Bookstores	*name*	*rank*	*city*
		U. 25	'高雄市'

QBE範例 (6)(7)

- 請將所有位於 '高雄市' 的書局都刪除

Bookstores	*name*	*rank*	*city*
D.			‘高雄市’

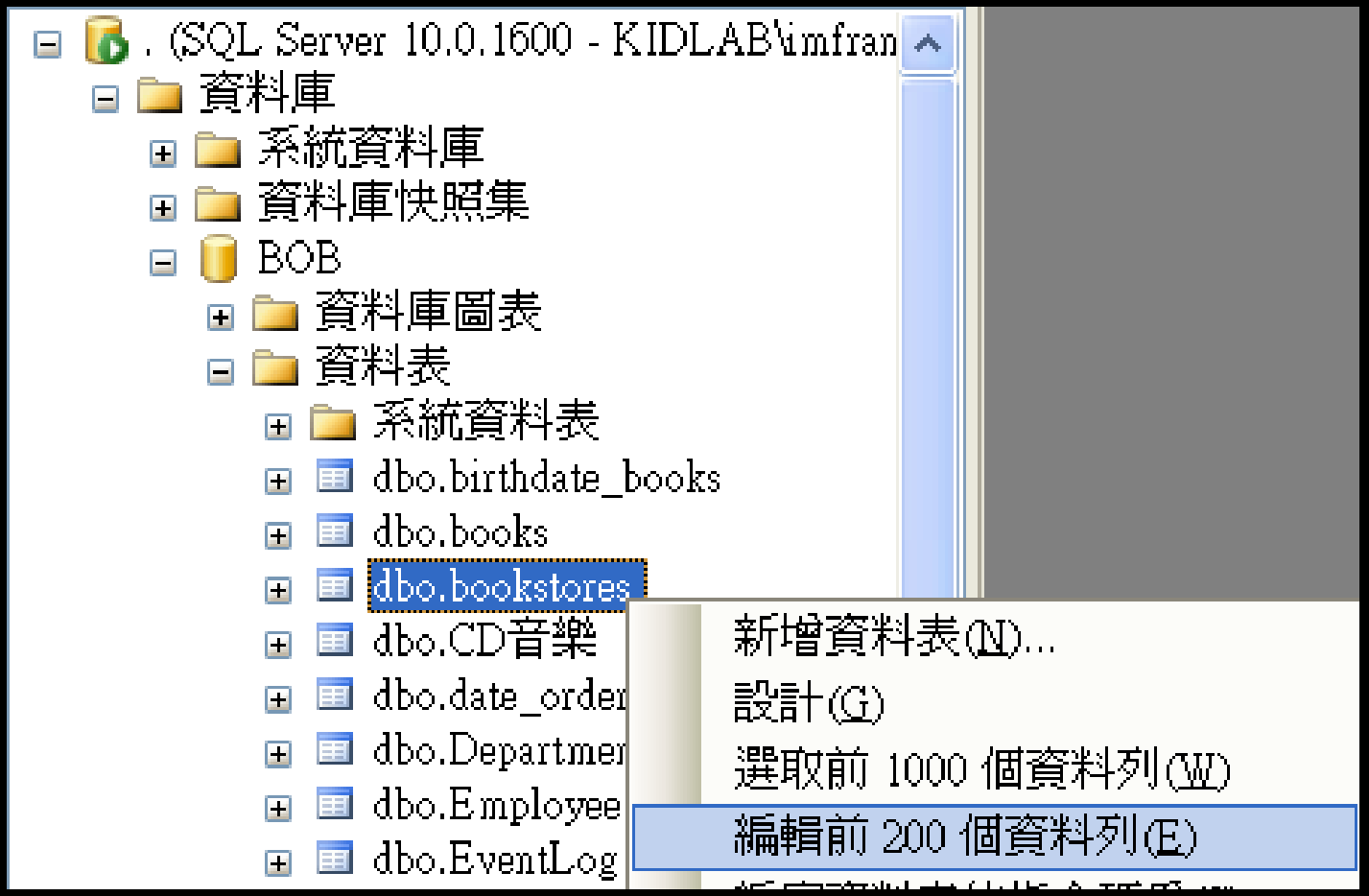
- 新增一筆書局資料 (7, '元智書坊', 25, '中壢市')

Bookstores	*no*	*name*	*rank*	*city*
I.	7	‘元智書坊’	25	‘中壢市’

QBE 在使用上的限制

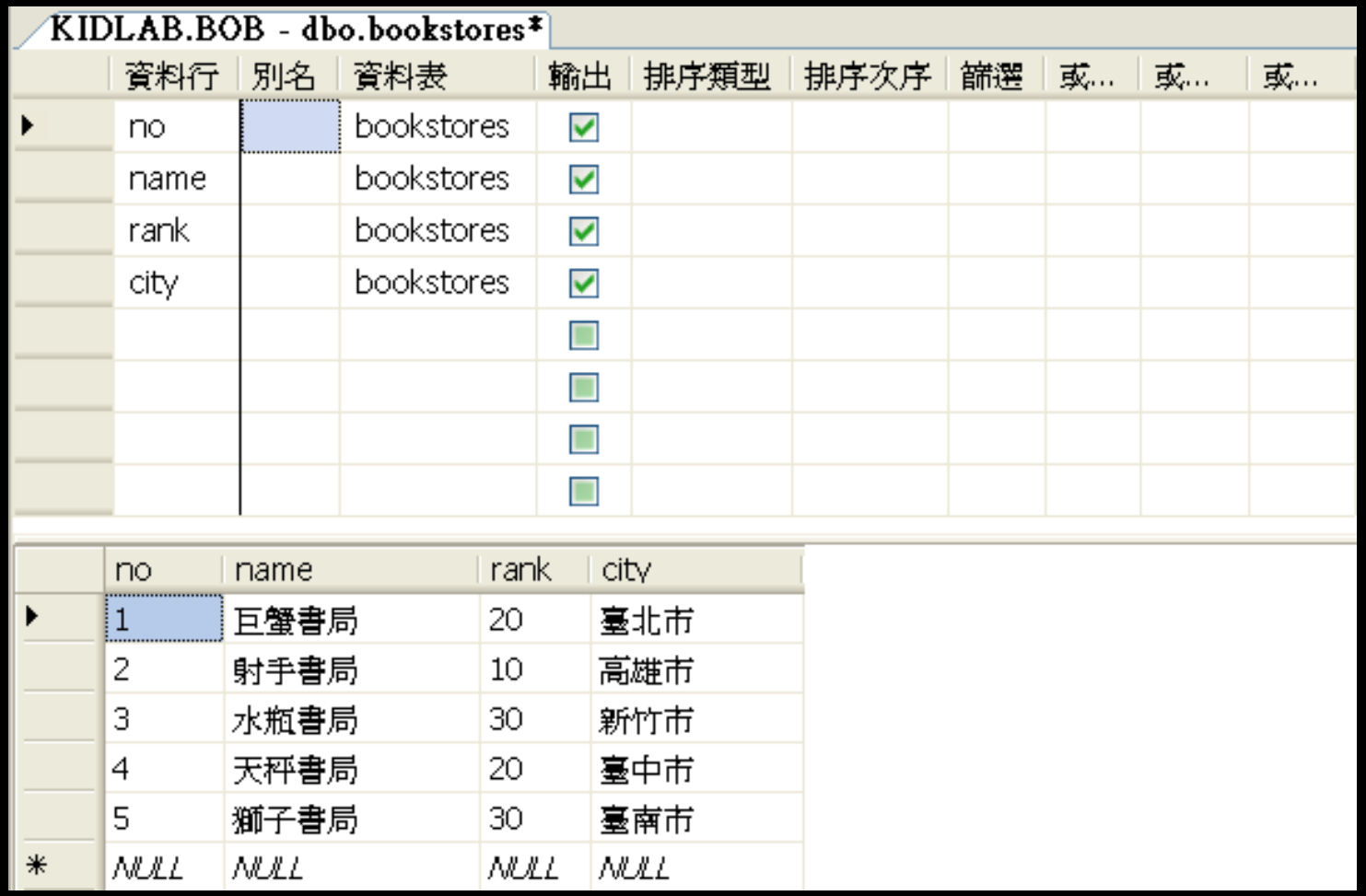
- 無法表示「總體限量詞」(FORALL, ∀) 與「存在限量詞」(EXISTS, ∃)
- 總體限量詞 (Universal Quantifier)
- 存在限量詞 (Existential Quantifier)
- 也沒有辦法表示兩者的前、後順序關係
- 但仍不失為一人性化人機介面的始祖

SQL Server 2008 的 QBE 介面

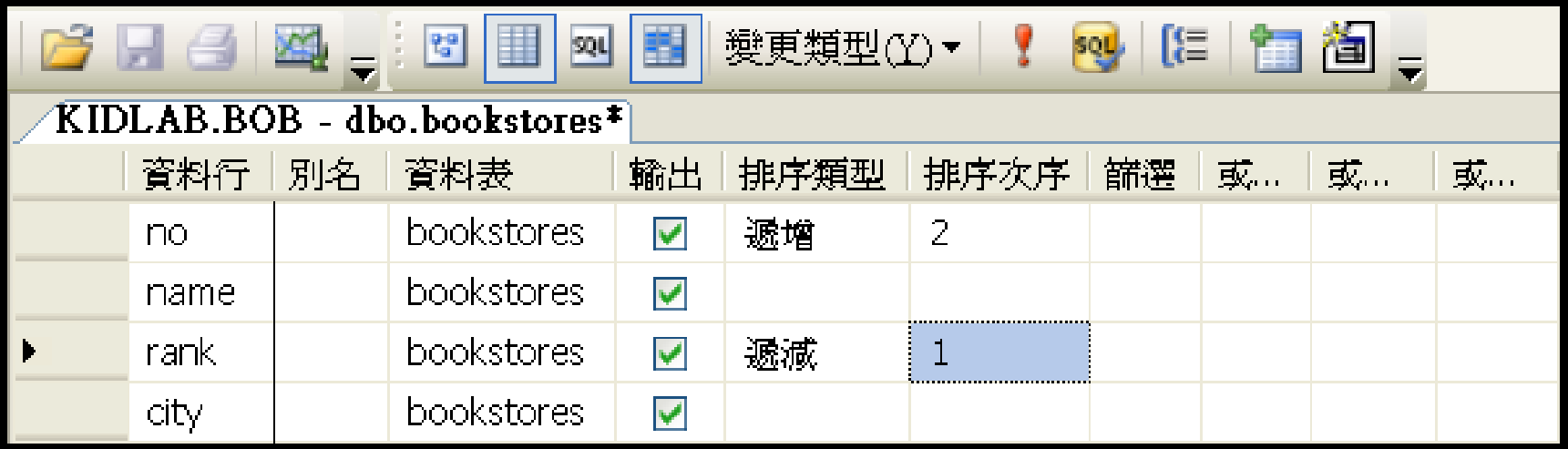


SQL Server 2008 的 QBE 介面

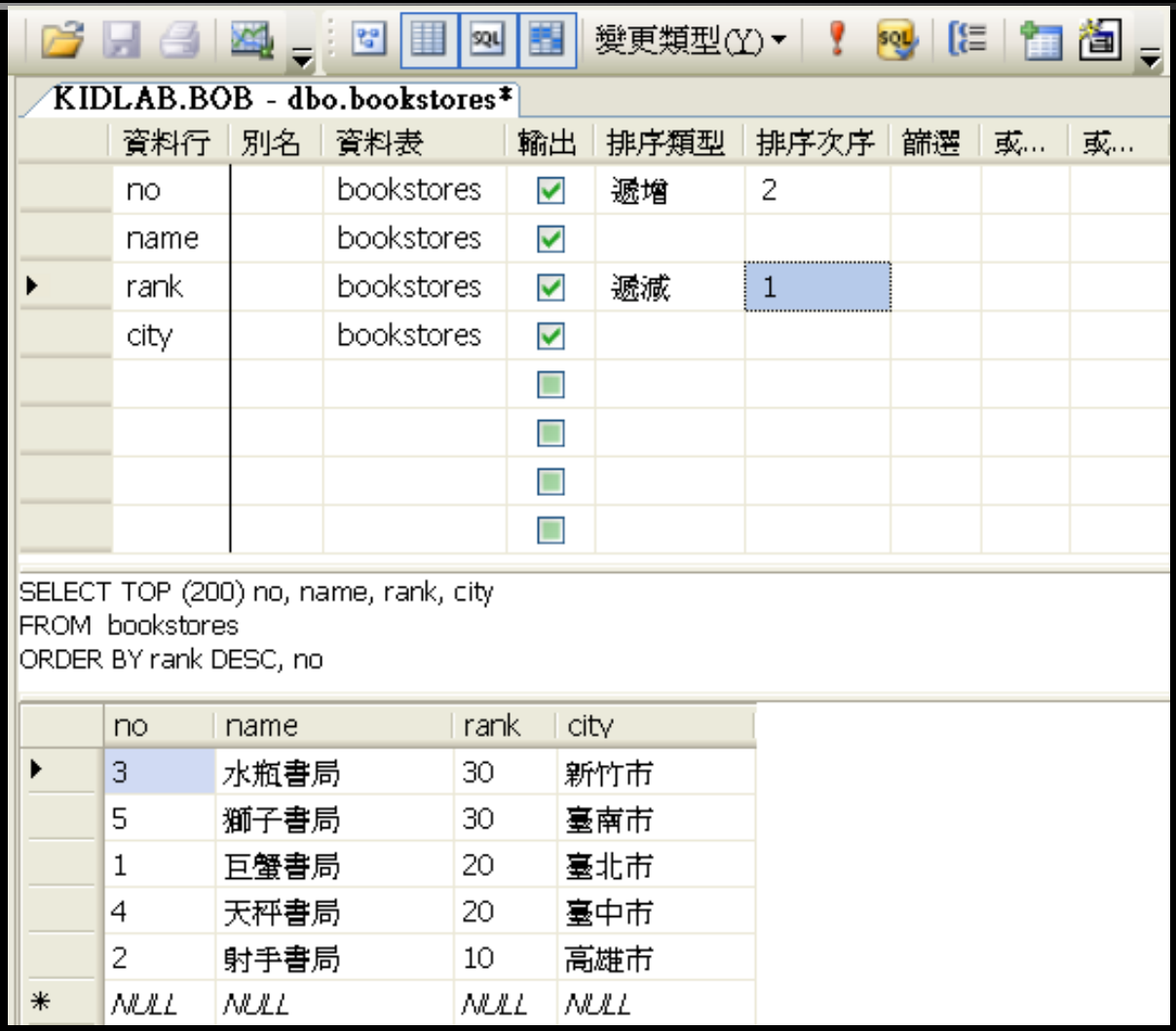
SQL Server 2008 的 QBE 介面



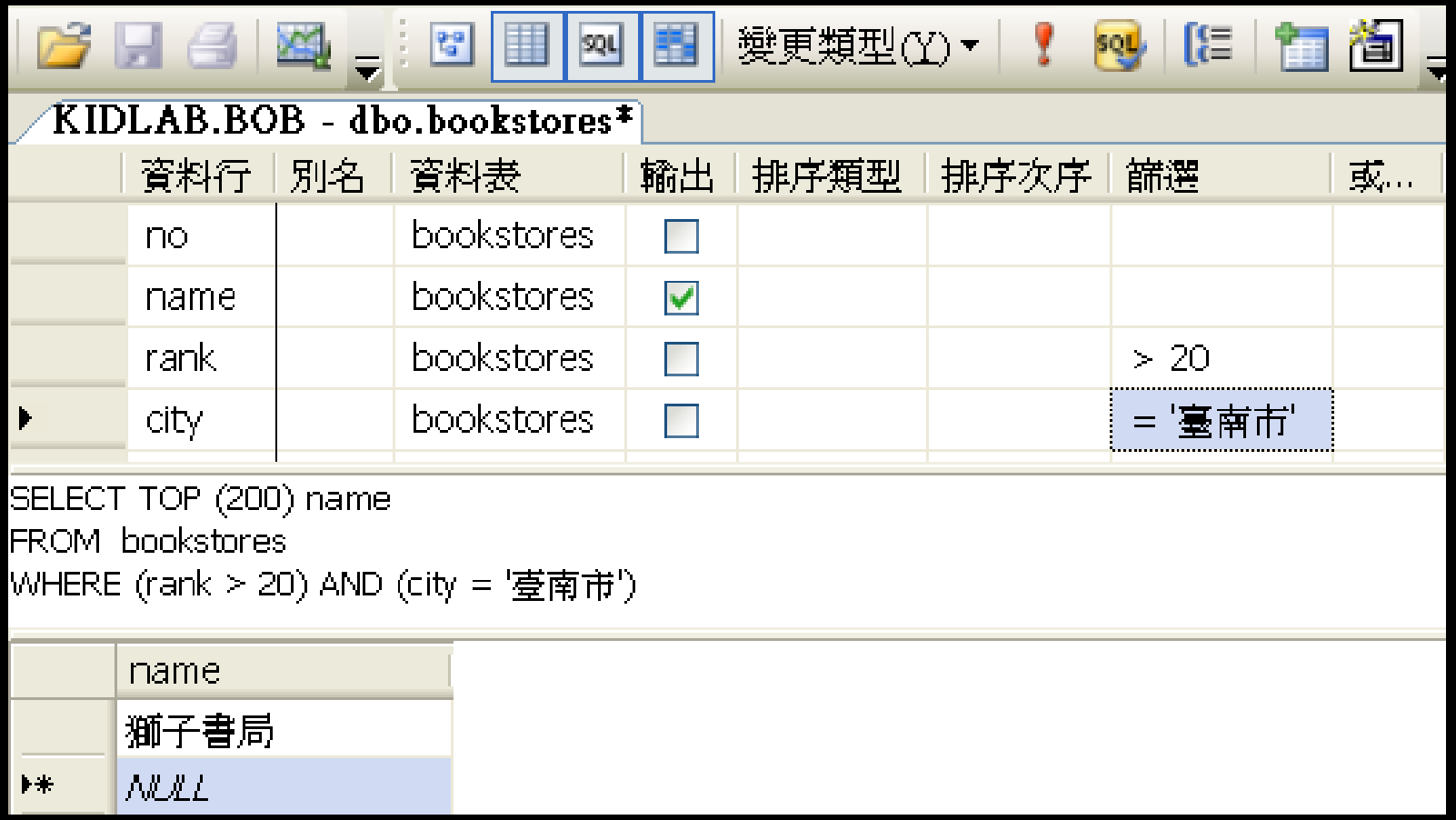
SQL Server 2008 的 QBE 介面



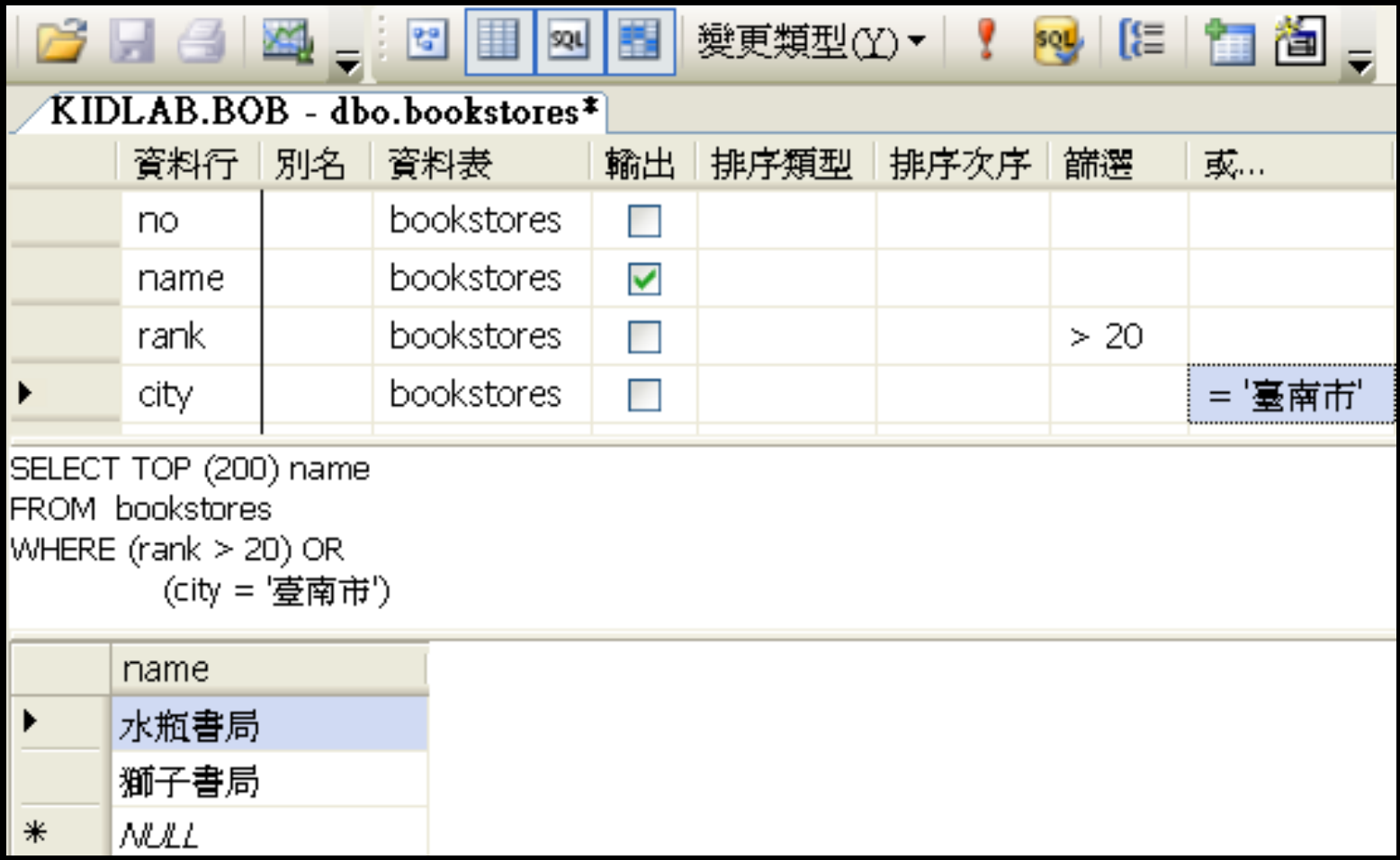
SQL Server 2008 的 QBE 介面



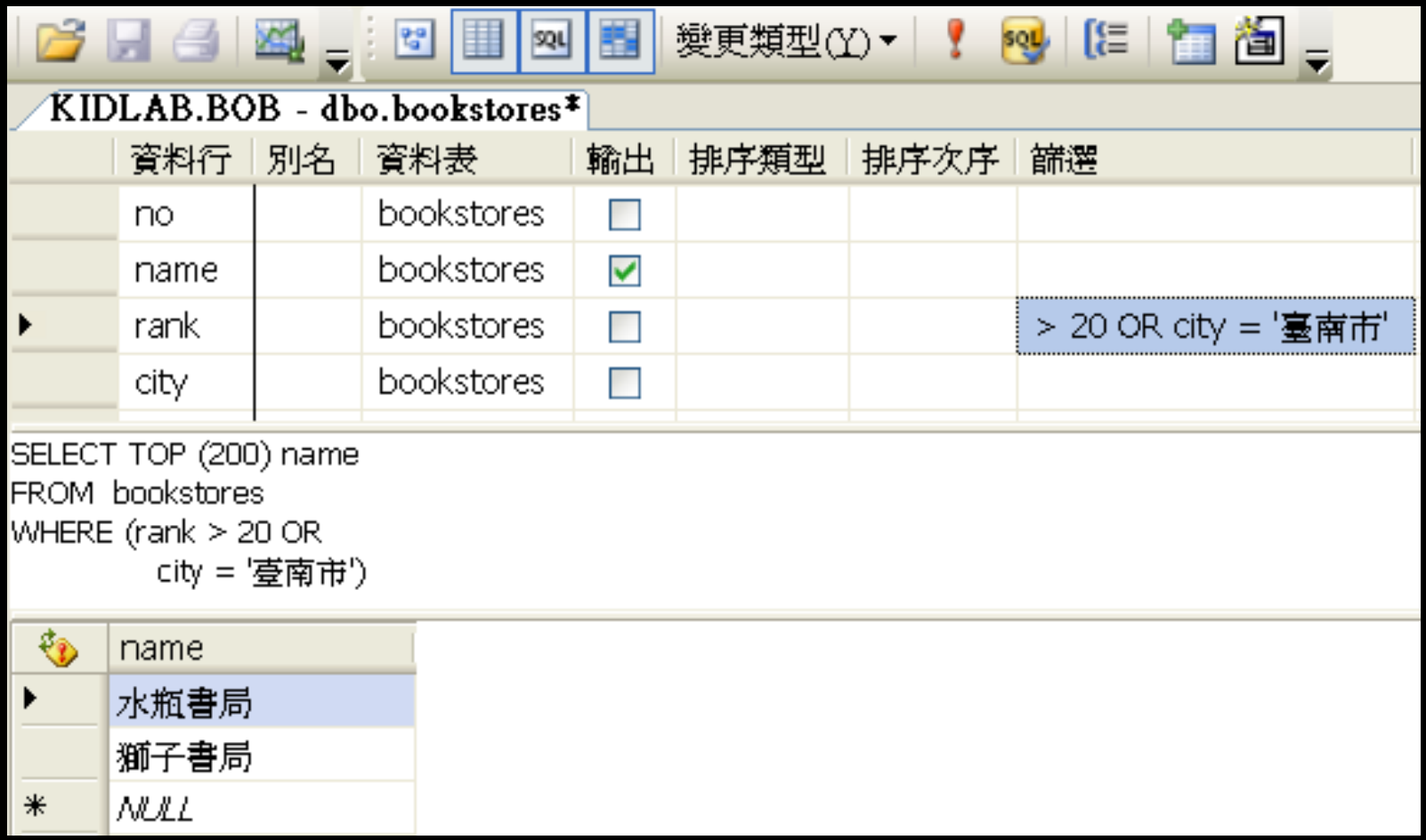
SQL Server 2008 的 QBE 介面



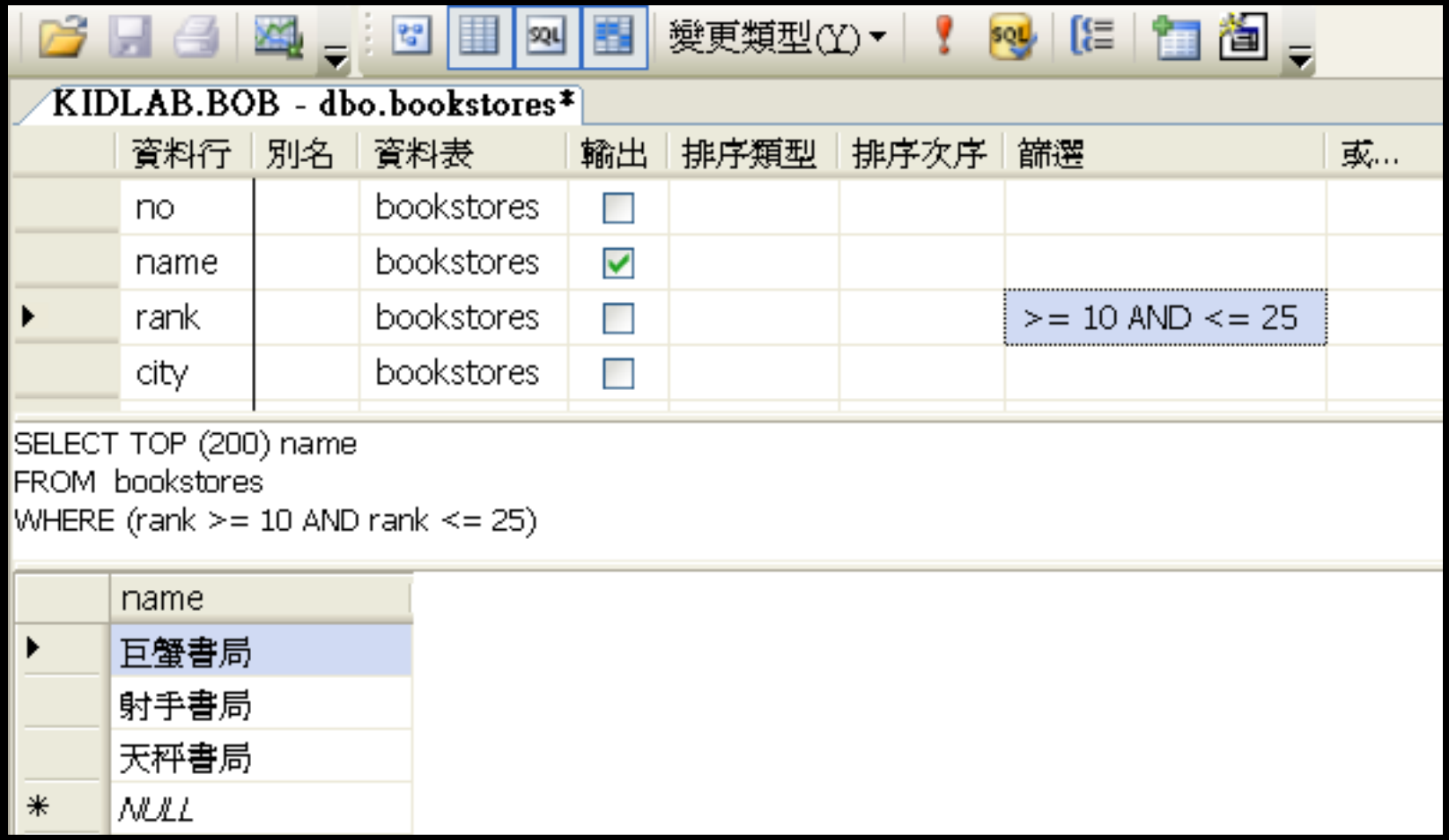
SQL Server 2008 的 QBE 介面



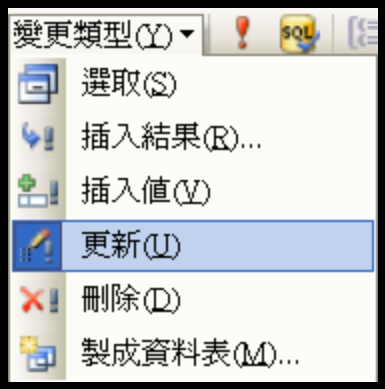
SQL Server 2008 的 QBE 介面

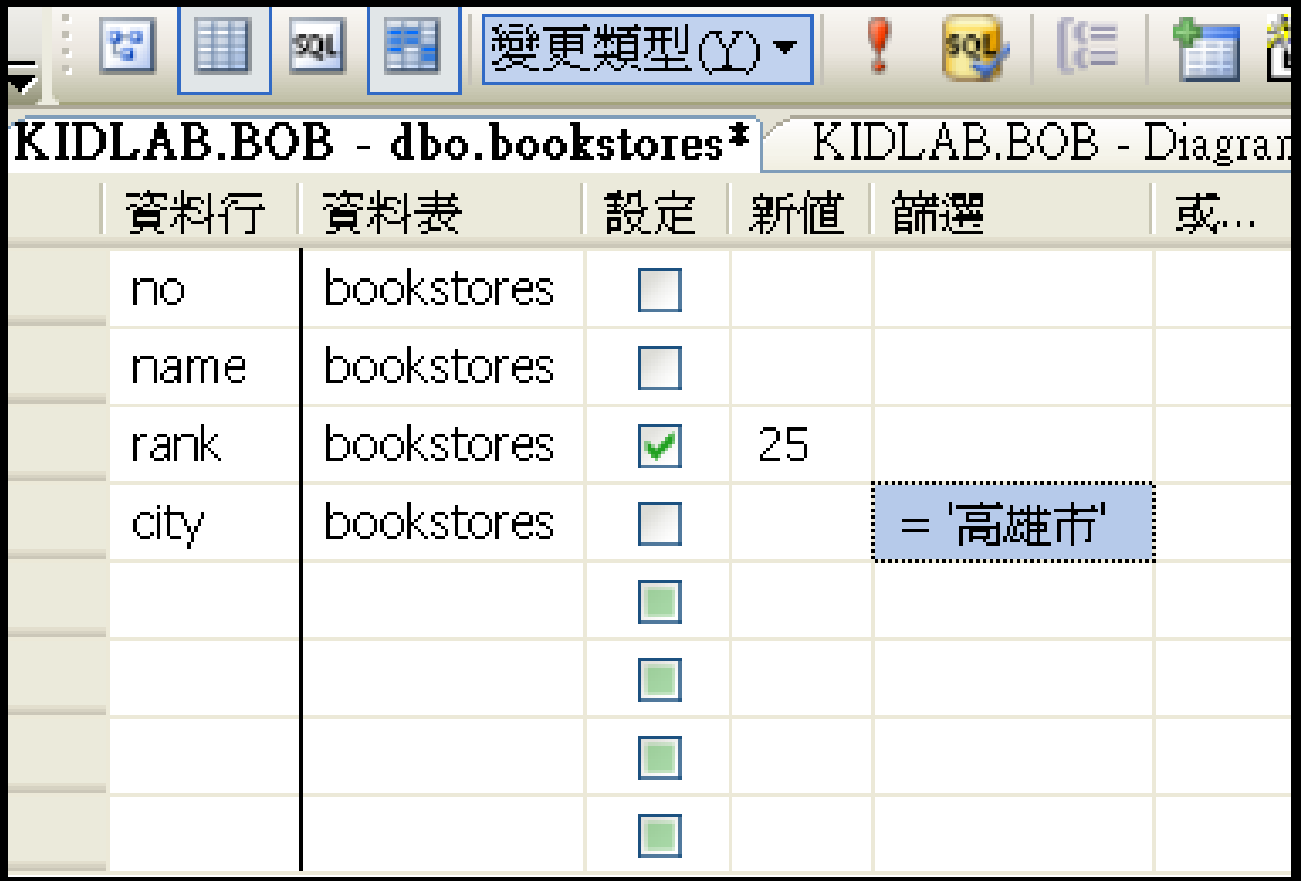


SQL Server 2008 的 QBE 介面

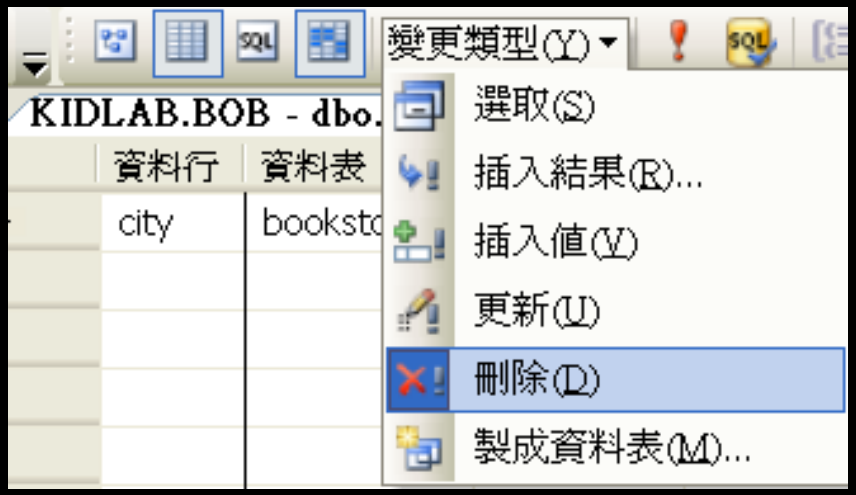


QBE 介面 (更新的例子)


變更類型(Y)
選取(S)
插入結果(R)...
插入值(V)
更新(U)
刪除(D)
製成資料表(M)...

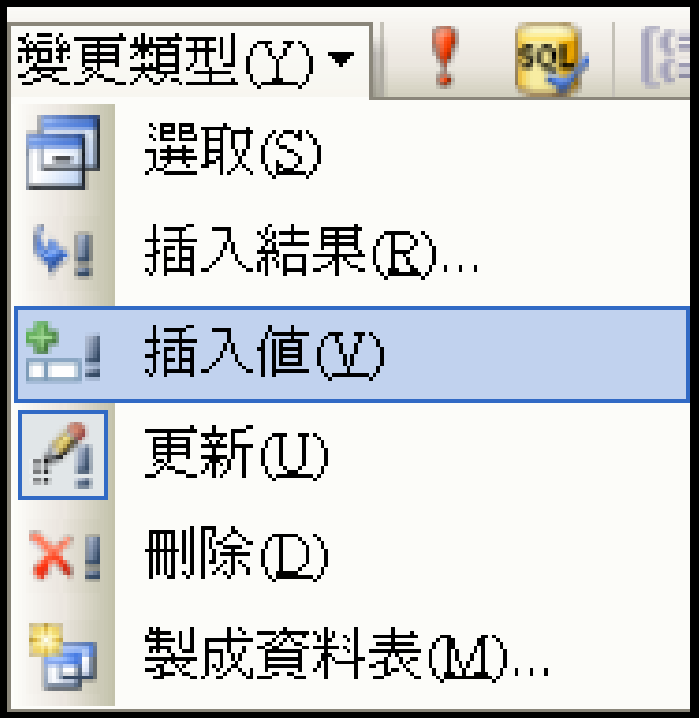

變更類型(Y)
KIDLAB.BOB - dbo.bookstores*
KIDLAB.BOB - Diagram
資料行
資料表
設定
新值
篩選
或...
no
bookstores
name
bookstores
rank
bookstores
25
city
bookstores
= '高雄市'

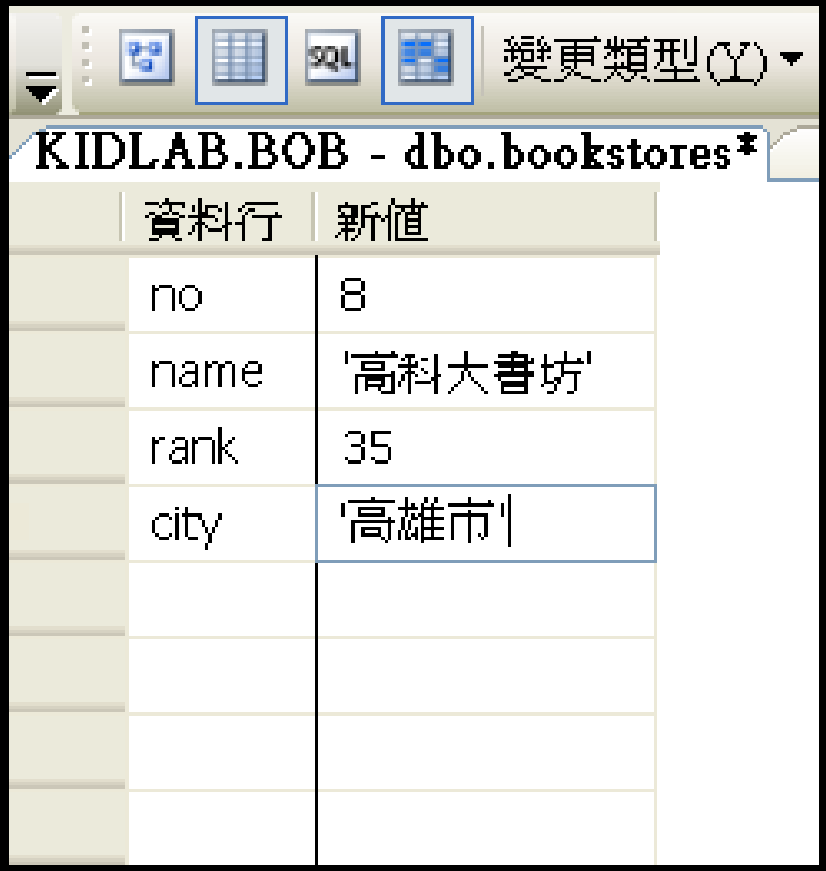
QBE 介面 (刪除的例子)


變更類型(Y)
KIDLAB.BOB - dbo.
資料行
資料表
city
booksto
選取(S)
插入結果(R)...
插入值(V)
更新(U)
刪除(D)
製成資料表(M)...

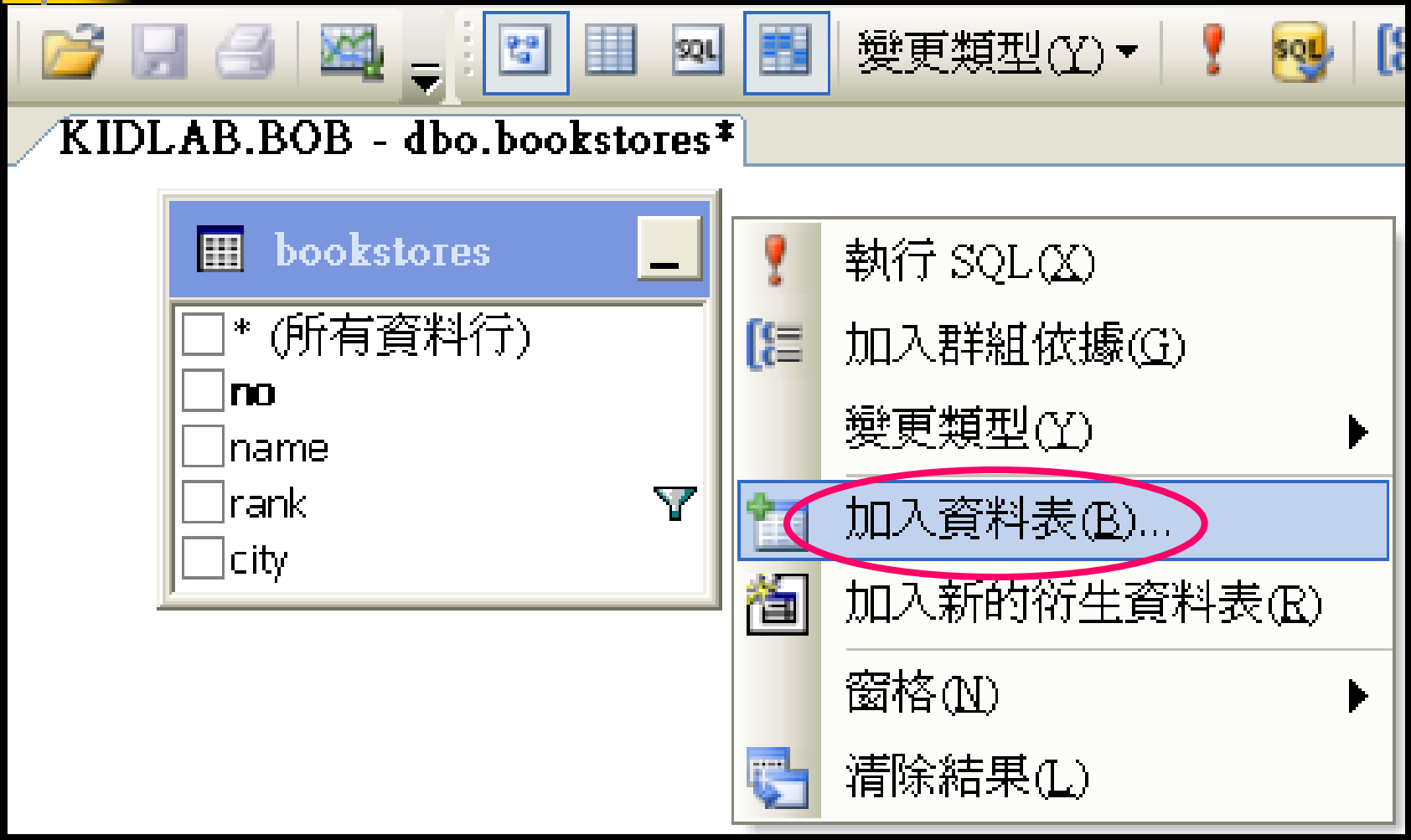

變更類型(Y)
KIDLAB.BOB - dbo.bookstores*
KIDLAB.E
資料行
資料表
篩選
或...
city
bookstores
= '高雄市'

QBE 介面 (新增的例子)

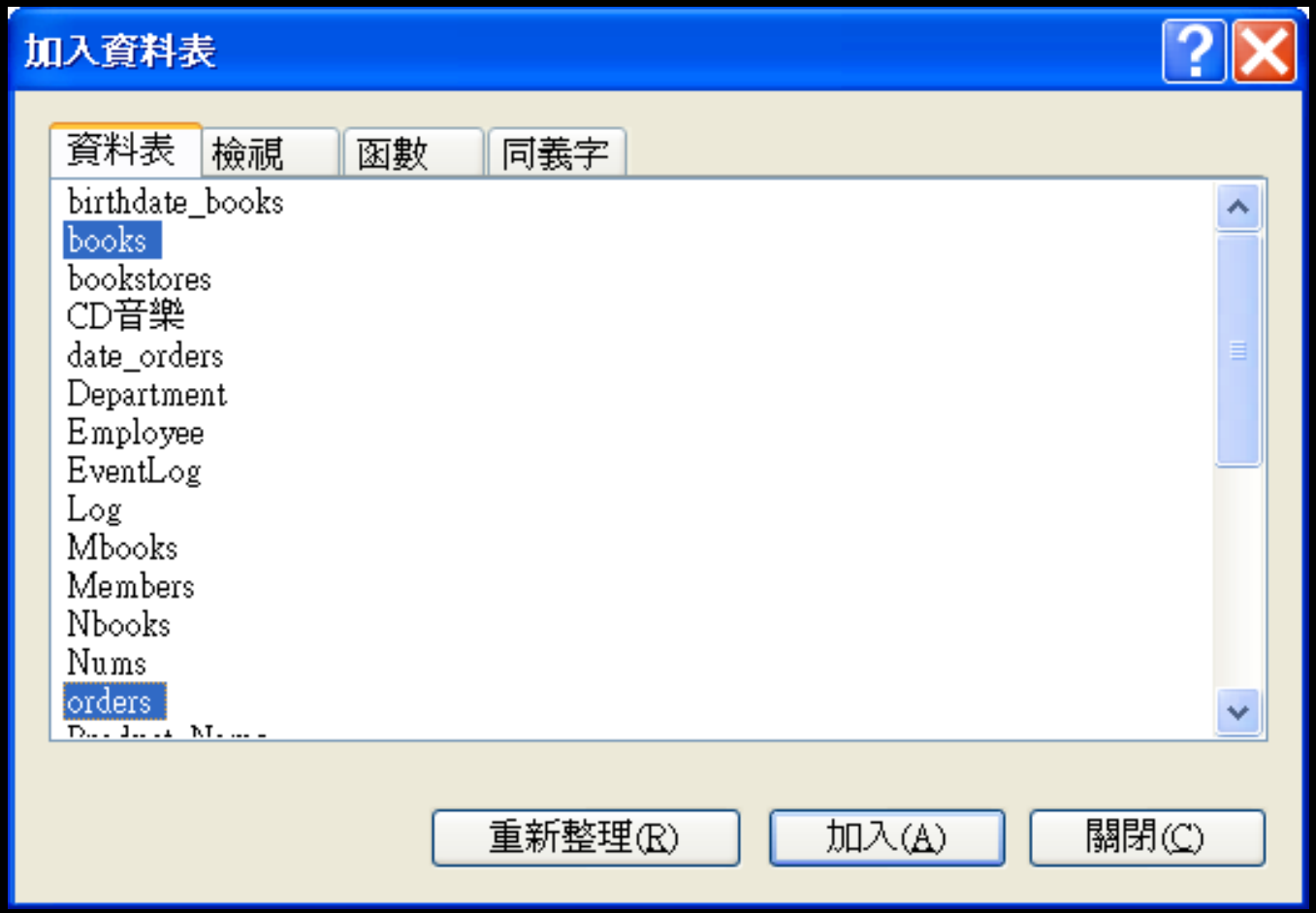

變更類型(Y)
選取(S)
插入結果(R)...
插入值(V)
更新(U)
刪除(D)
製成資料表(M)...


變更類型(Y)
KIDLAB.BOB - dbo.bookstores*
資料行
新值
no
8
name
'高科大書坊'
rank
35
city
'高雄市'

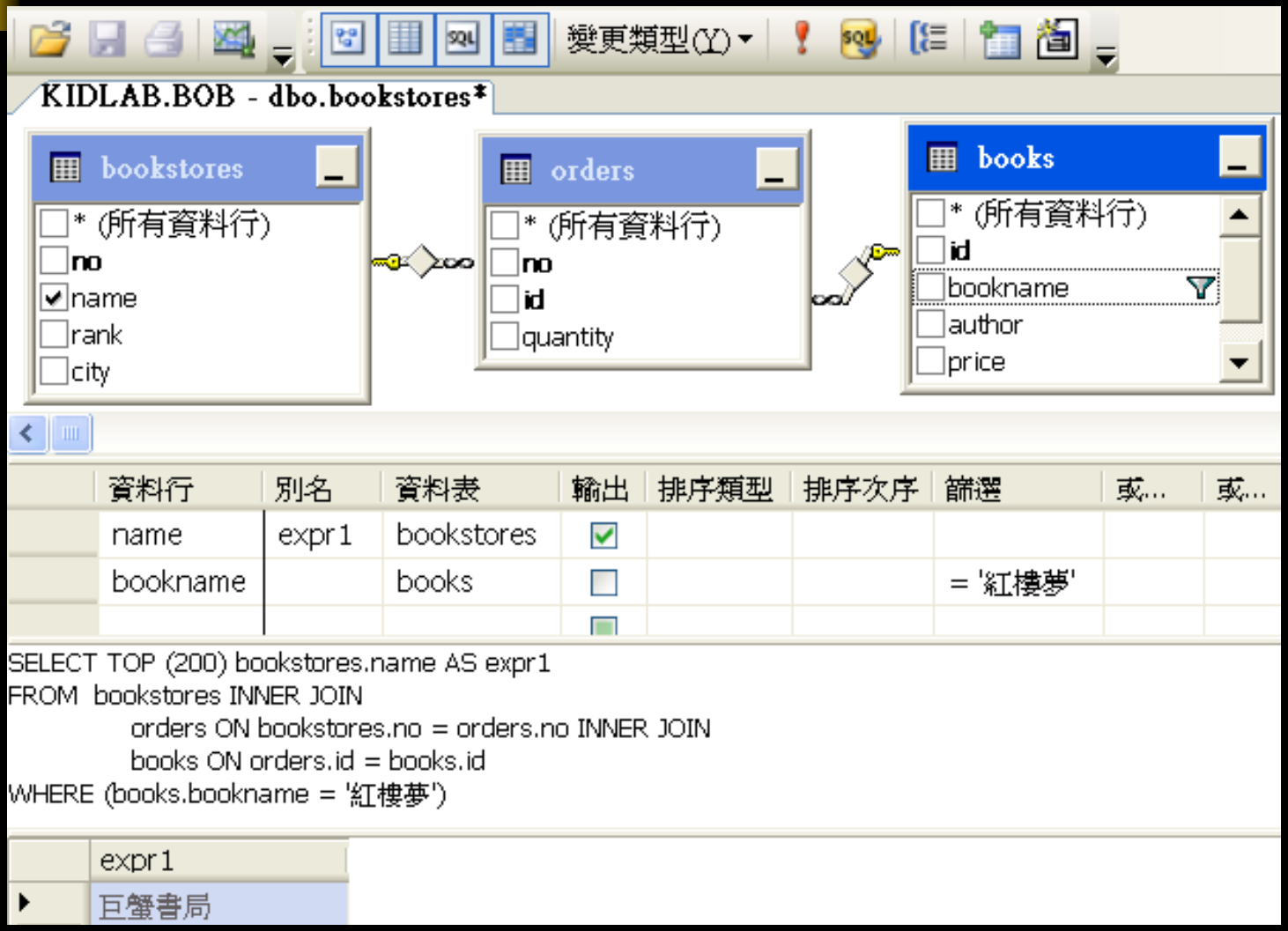
查詢多個表格的 QBE 介面



SQL Server 2008 的 QBE 介面



SQL Server 2008 的 QBE 介面



本章結束
The End.