資料庫系統
Class 7: Query Processing and Optimization

逢甲資工 許懷中

QUERY PROCESSING (查詢處理)

關聯式代數 (Relational Algebra)

- Selection (σ)
- Projection (Π)
- Cartesian Product (×)
- Join
  - Natural Join (⋈)
  - Semijoin (⋉)(⋊)
  - Antijoin (▷)

Selection (σ)

相當於 SQL Query 中使用的 WHERE closure

Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$$\sigma_{A=B \text{ and } D>5}(r)$$

A	B	C	D
α	α	1	7
β	β	23	10

Projection (Π)

相當於 SQL 中的
SELECT statement

Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\Pi_{A,D}(r)$

A	D
α	7
α	7
β	3
β	10

A	D
α	7
β	3
β	10

Cartesian Product (×)

Relation r

A	B
α	1
β	2

Relation s

C	D	E
α	10	a
β	20	b
γ	30	c

r × s

A	B	C	D	E
α	1	α	10	a
α	1	β	20	b
α	1	γ	30	c
β	2	α	10	a
β	2	β	20	b
β	2	γ	30	c

Natural Join (⋈)

superheroes

name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

publishers

publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

superheroes ⋈ publishers

name	alignment	gender	publisher	yr_founded
Magneto	bad	male	Marvel	1939
Storm	good	female	Marvel	1939
Mystique	bad	female	Marvel	1939
Batman	good	male	DC	1934
Joker	bad	male	DC	1934
Catwoman	bad	female	DC	1934

*from https://stat545-ubc.github.io/bit001_dplyr-cheatsheet.html

Semijoin (⋉, left-join)

superheroes

name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

publishers

publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

superheroes ⋉ publishers

name	alignment	gender	publisher	yr_founded
Magneto	bad	male	Marvel	1939
Storm	good	female	Marvel	1939
Mystique	bad	female	Marvel	1939
Batman	good	male	DC	1934
Joker	bad	male	DC	1934
Catwoman	bad	female	DC	1934
Hellboy	good	male	Dark Horse Comics	NA

*from https://stat545-ubc.github.io/bit001_dplyr-cheatsheet.html

Semijoin (⋉, left-join)

publishers

publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

superheroes

name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

publishers ⋉ superheroes

publisher	yr_founded	name	alignment	gender
DC	1934	Batman	good	male
DC	1934	Joker	bad	male
DC	1934	Catwoman	bad	female
Marvel	1939	Magneto	bad	male
Marvel	1939	Storm	good	female
Marvel	1939	Mystique	bad	female
Image	1992	NA	NA	NA

*from https://stat545-ubc.github.io/bit001_dplyr-cheatsheet.html

Antijoin (▷)

superheroes

name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

publishers

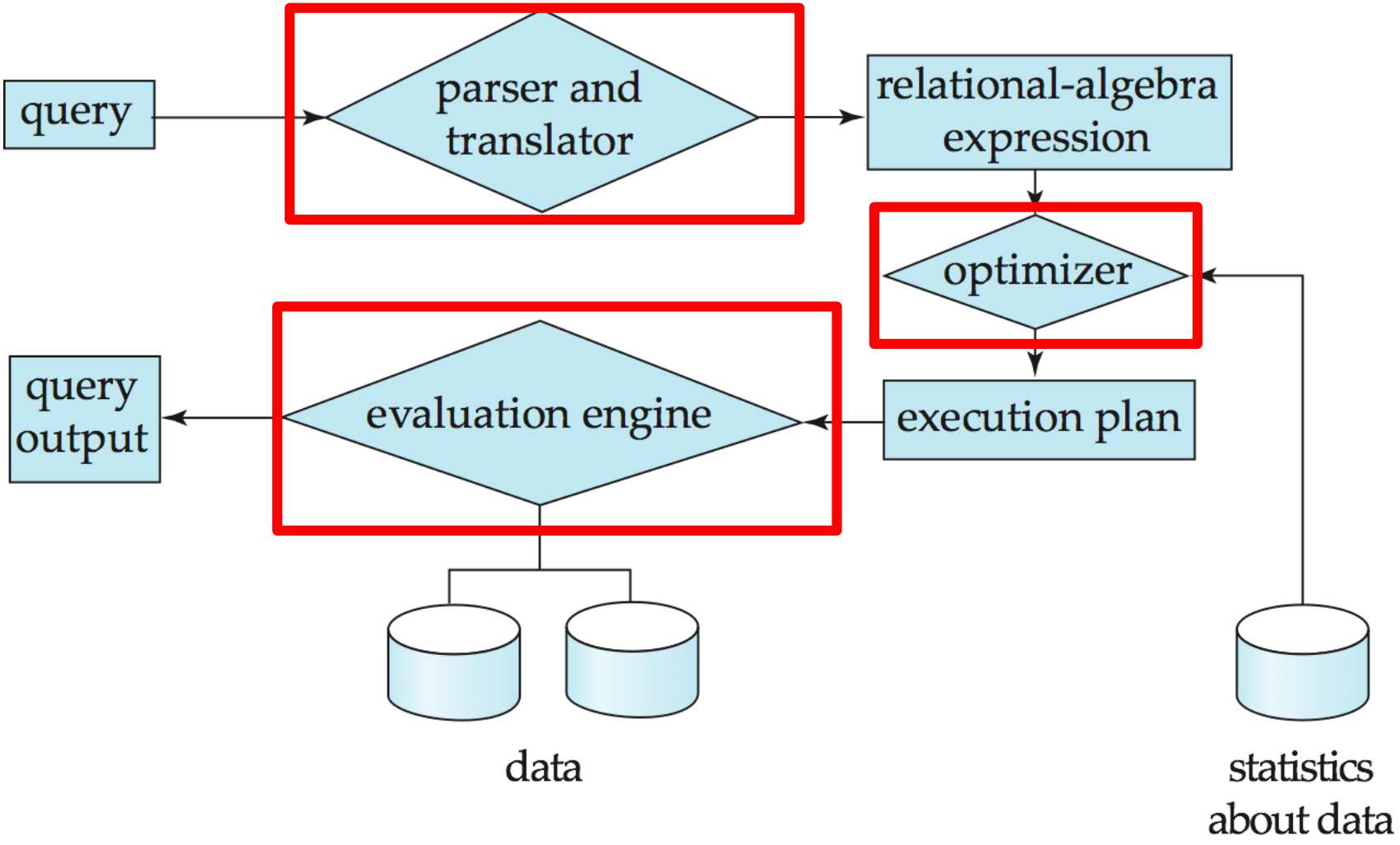
publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

superheroes ▷ publishers

name	alignment	gender	publisher
Hellboy	good	male	Dark Horse Comics

*from https://stat545-ubc.github.io/bit001_dplyr-cheatsheet.html

Query Processing



Query Optimization

- 每個 SQL 查詢都可以被轉化為幾個關聯代數的表示方式。
  - E.g., $\sigma_{salary<75000}(\prod_{salary}(instructor))$
    $\prod_{salary}(\sigma_{salary<75000}(instructor))$
- 可從多種演算法選擇一種來執行每個關聯代數。
- 同理，每個關聯代數可以進行多種方式的評估。
- 用來評估查詢的序列基本操作，稱為查詢執行計劃
  - 對 salary 做索引，找出 salary < 75000 的教師
  - 對整個 relation 做搜索，拋棄 salary ≥ 75000 的紀錄

Query Optimization (cont.)

- 從所有等效的評估計畫終挑出 cost 最低的
- Cost 是根據資料庫過去存取紀錄的統計數據所得
- Cost => 評估計畫的回應時間
  - 磁碟存取次數、CPU 與存儲裝置間的通訊次數
- 磁碟存取成本
  - 讀寫頭搜尋次數
  - 讀取的 block 數
  - 寫入的 block 數
- 寫入成本一般比較高，一方面寫入速度較慢，另一方面寫入後系統需要回讀以確認寫入成功

測量 Query Cost

- 為了簡化討論
  - 只考慮 block 的傳輸量 (忽略讀取/寫入的差異)
  - 不考慮 CPU costs (真實狀況需要考慮)
- 一個需要傳輸 b 個 block 與進行 S 次讀寫頭搜尋的 Query 其 Cost 為
  - $b * t_T + S * t_S$
  - t_T – time to transfer one block
  - t_S – time for one seek
- 利用 buffer 或 cache 可以減少 query cost

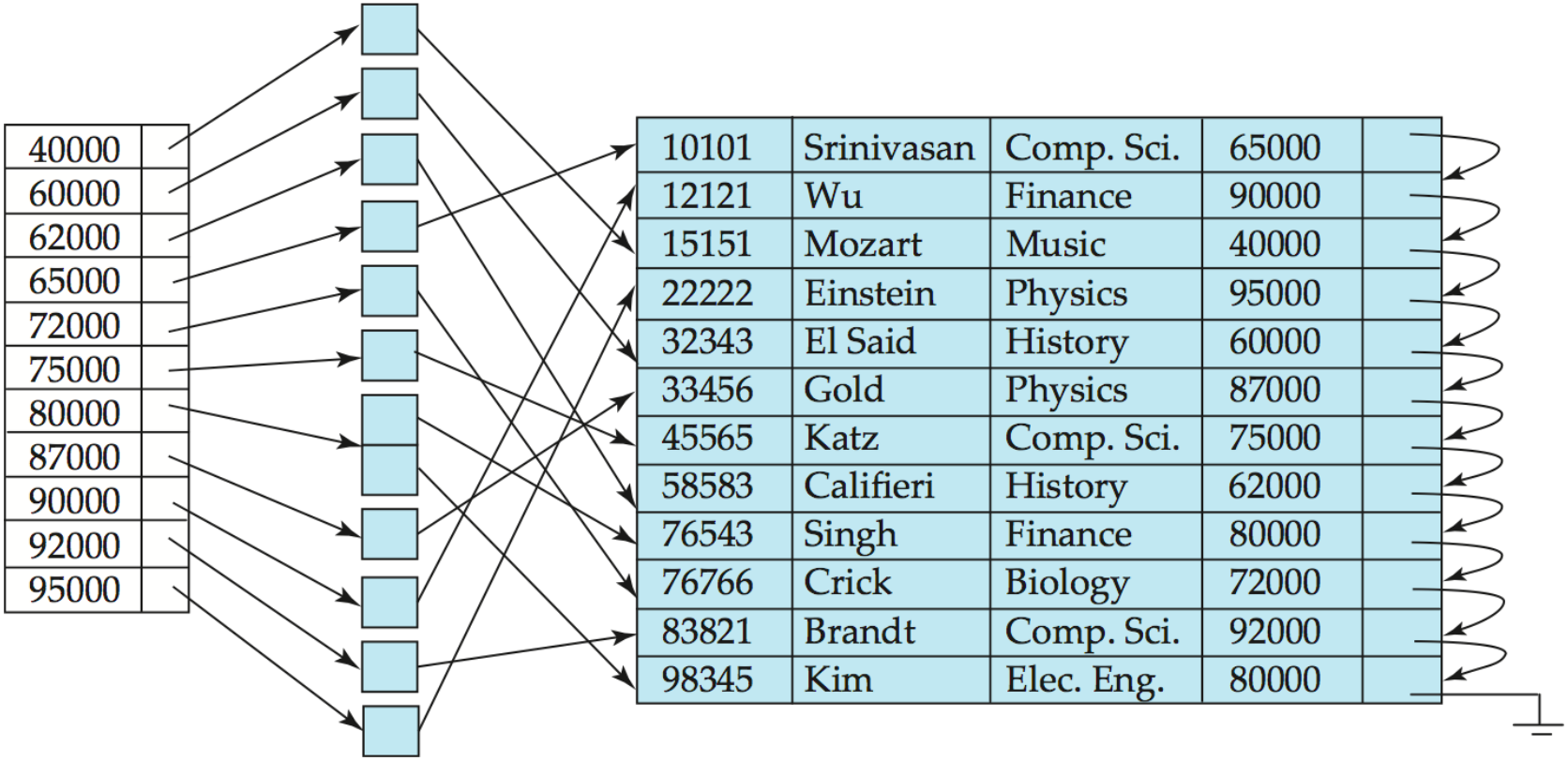
Selection Operation

- b_r 是一個 relation 的 block 數
- Cost for **linear search** (file scan)
  - $B_r * t_T + t_S$ (全面搜尋)
  - $(b_r/2) * t_T + t_S$ (找到所需要的 record 就停止)

Selection Operation (cont.)

- Cost for **index scan**
  - primary index
    - Cost = $(h_i + 1) * (t_T + t_S)$ (找到特定的鍵值)
    - Cost = $h_i * (t_T + t_S) + t_S + t_T * b$ (找到符合條件的紀錄)
  - secondary index
    - Cost = $(h_i + 1) * (t_T + t_S)$ (找到特定的候選鍵值)
    - Cost = $(h_i + n) * (t_T + t_S)$ (找到符合條件的紀錄, n 個)

Secondary Indices



Indices to non-candidate keys

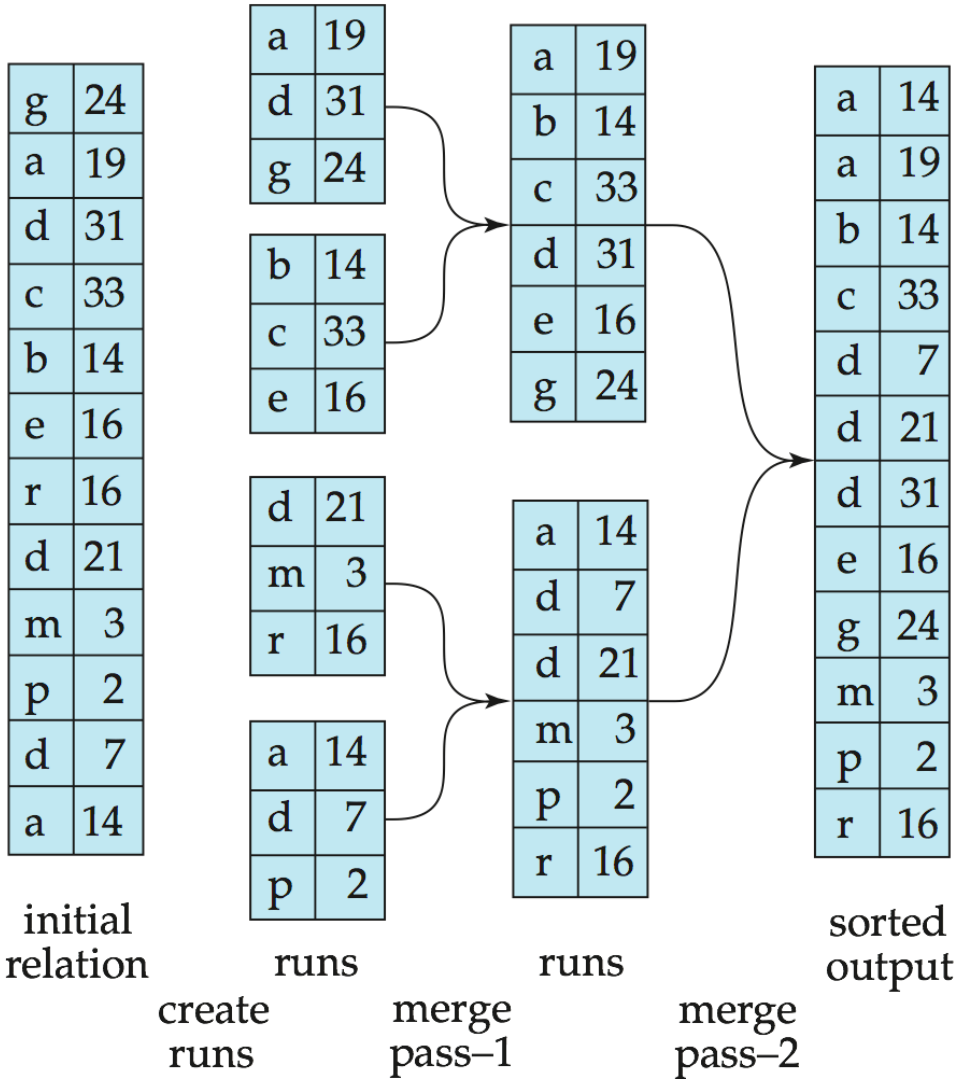
Selection with Comparisons

- $\sigma_{A \leq V}(r)$ or $\sigma_{A \geq V}(r)$
- Primary index 可以直接根據對到的 index 進行 sequential search
- Secondary index 需要進一步經過指標找到目標 block

Complex Selection

- $\sigma_{\theta_1 \wedge \theta_2 \wedge \ldots \theta_n}(r)$
  - 如果 θ_i 有 index，以 有 index 的條件為主，其他後續再篩選
- $\sigma_{\theta_1 \vee \theta_2 \vee \ldots \theta_n}(r)$
  - 如果所有條件都有索引，使用索引搜尋，否則進行 linear search
- $\sigma_{\neg\theta}(r)$
  - Linear Search
  - 假如符合$\neg\theta$ 的紀錄很少，而且有 index 可用，正向搜尋後捨去

排序


g 24
a 19
d 31
c 33
b 14
e 16
r 16
d 21
m 3
p 2
d 7
a 14
a 19
d 31
g 24
b 14
c 33
e 16
d 21
m 3
r 16
a 14
d 7
p 2
a 19
b 14
c 33
d 31
e 16
g 24
a 14
d 7
d 21
m 3
p 2
r 16
a 14
a 19
b 14
c 33
d 7
d 21
d 31
e 16
g 24
m 3
p 2
r 16
initial relation
runs
runs
sorted output
create runs
merge pass–1
merge pass–2

Cost of Merge Sort

- 存取成本
  - b_r 表示包含關聯 r 紀錄的區塊數量
  - M 代表每次可以 load 進主記憶體 sort 的 block 數
  - 合併路徑所需的總數量: $\lceil \log_{M-1}(b_r/M) \rceil$
  - 讀取每個關聯區塊，並將其重新寫入，造成總共 2 br 個區塊傳輸。
  - 傳輸區塊總數: $b_r (2\lceil \log_{M-1}(b_r / M) \rceil + 1)$
- Disk Seeking 成本
  - 每個合併路徑增加 $2\lceil b_r / M \rceil$ 個 disk seeking
  - 假設合併階段，每個 run 每次讀取 b_b個資料區塊
  - 總 disk seeking數目:

$$2\lceil b_r / M \rceil + \lceil b_r / b_b \rceil (2\lceil \log_{M-1}(b_r / M) \rceil - 1)$$

計算 Join 所需的 Cost

- $r \bowtie_\theta s$

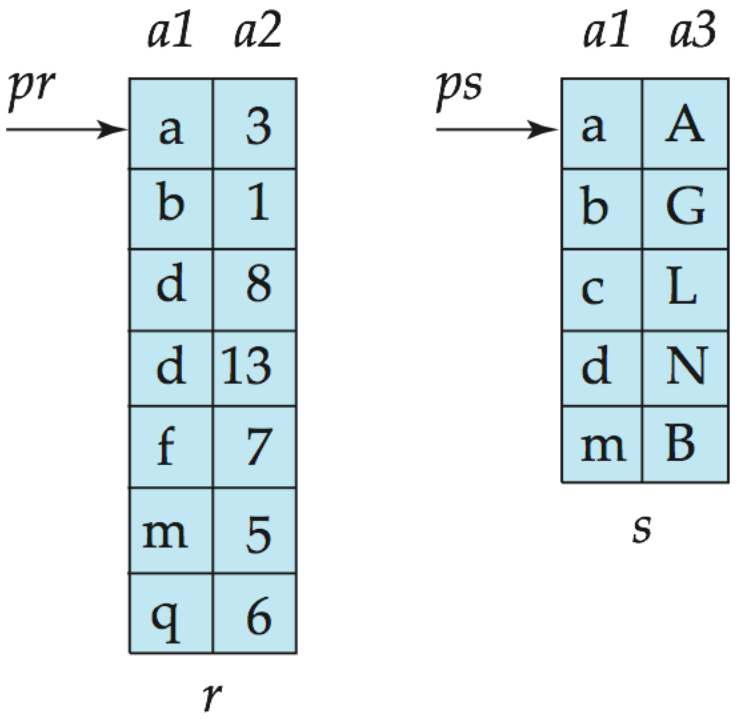
```
for each tuple t_r in r do begin
    for each tuple t_s in s do begin
        test pair (t_r , t_s) to see if they satisfy the join condition θ
        if they do, add t_r · t_s to the result;
    end
end
```

Join with Indexing

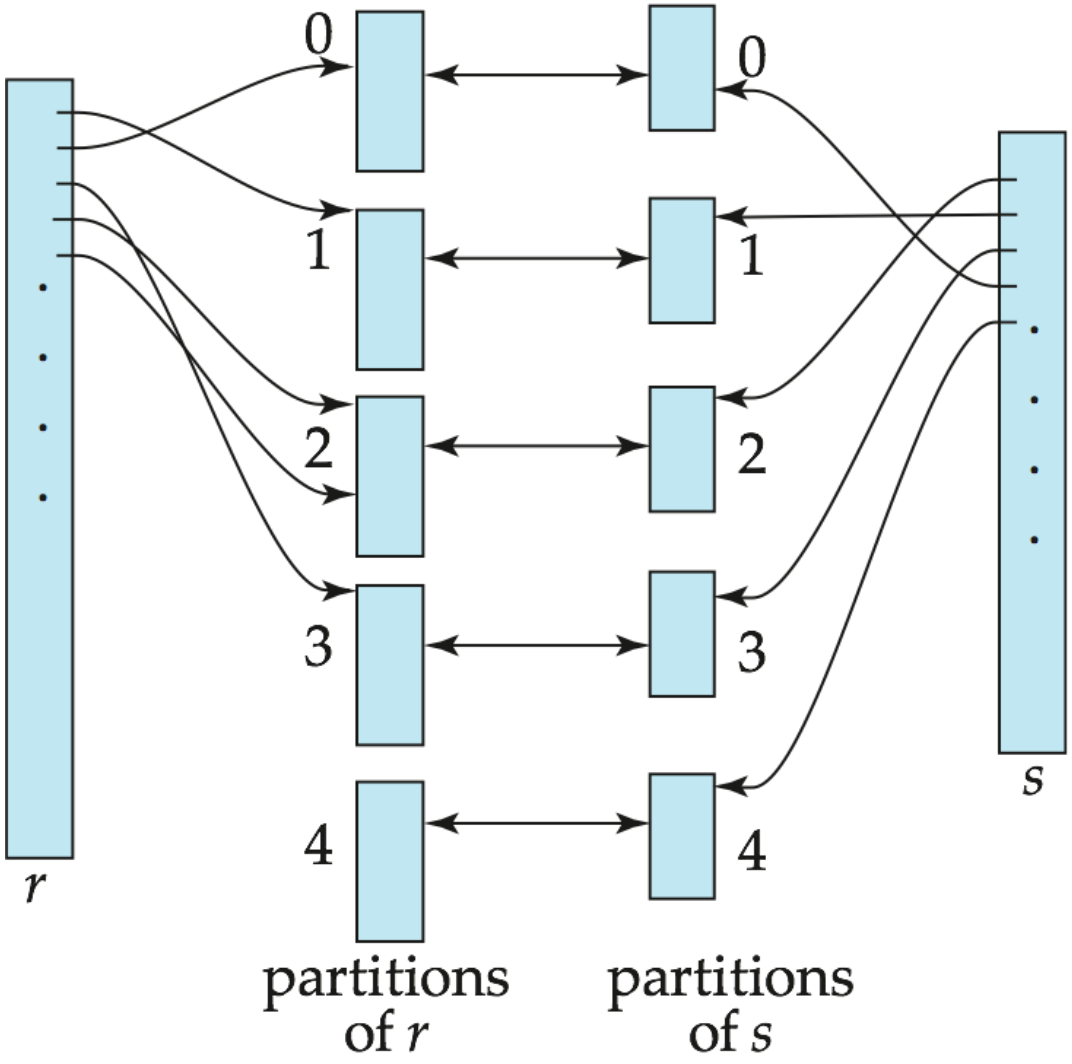
- 內部迴圈有 index => 取代 linear search
- 外部迴圈有 index ，加快條件 match 速度
- 結合的時間成本: $b_r(t_t + t_s) + n_r * c$

Merge Join

- Sort then Join



Join with hashing



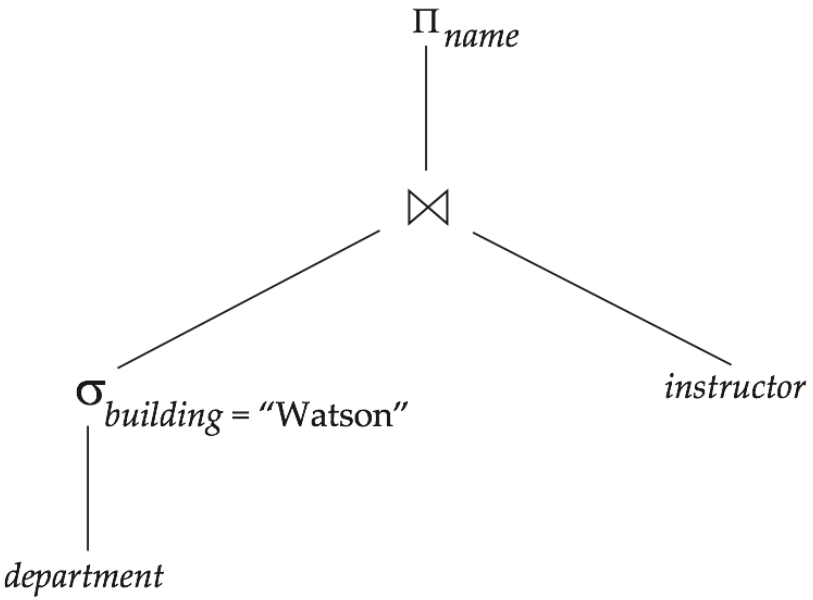
Aggregation

- 排序或雜湊不是消除元組具有相同值的分組屬性，而是將它們收集成群組，在每組應用聚集運算獲得結果
  - 可以在建構群體時，實現聚合運算，如總和、最小值、最大值、計數和平均
    - 對於總和、最小值、最大值、計數: 保持目前在群組中找到的元組聚集值
    - 透過總和除以計數來得到平均值

管線 Pipelining

- 結合幾個關聯運算到一個管線，使一個運算結果可在管線中傳遞到下一個運算

$$\sigma_{building="\text{Watson}"}(department)$$

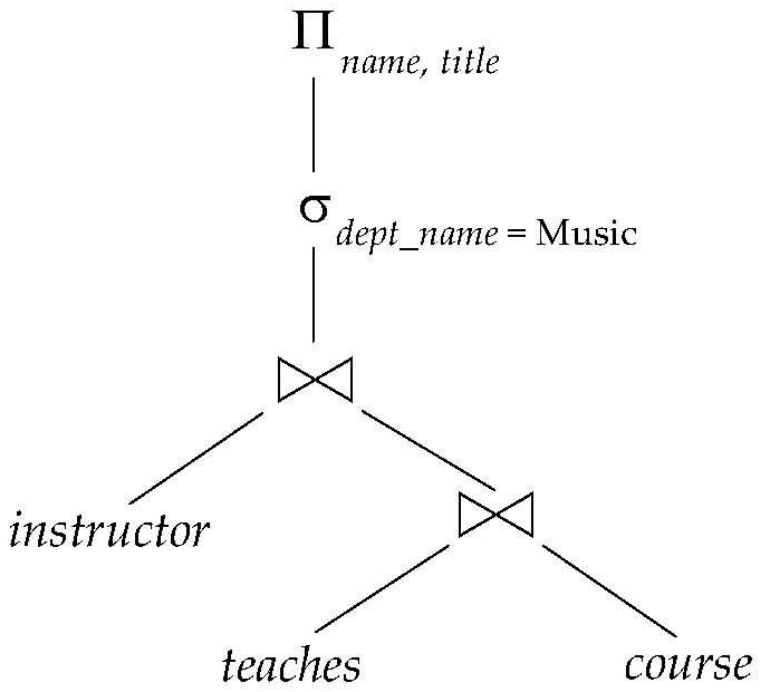


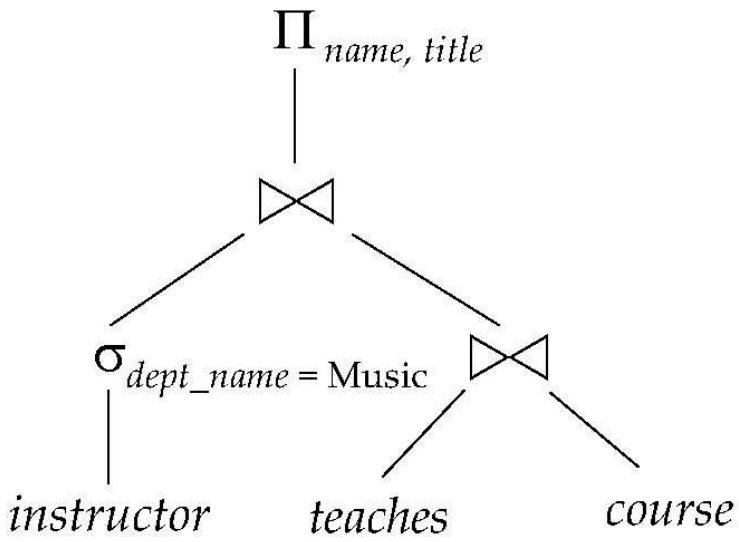
管線的優劣

- 不儲存中間結果，成本較低
- Demand-driven pipeline (pulling model)
  - 有需求才計算
- Producer-drive pipeline (pushing model)
  - 將前端產生的資料先存入 buffer 備用
- 並非總是可行，例如排序與 hash join
  - 中間結果需要全部被檢查，否則無法進行下一階段

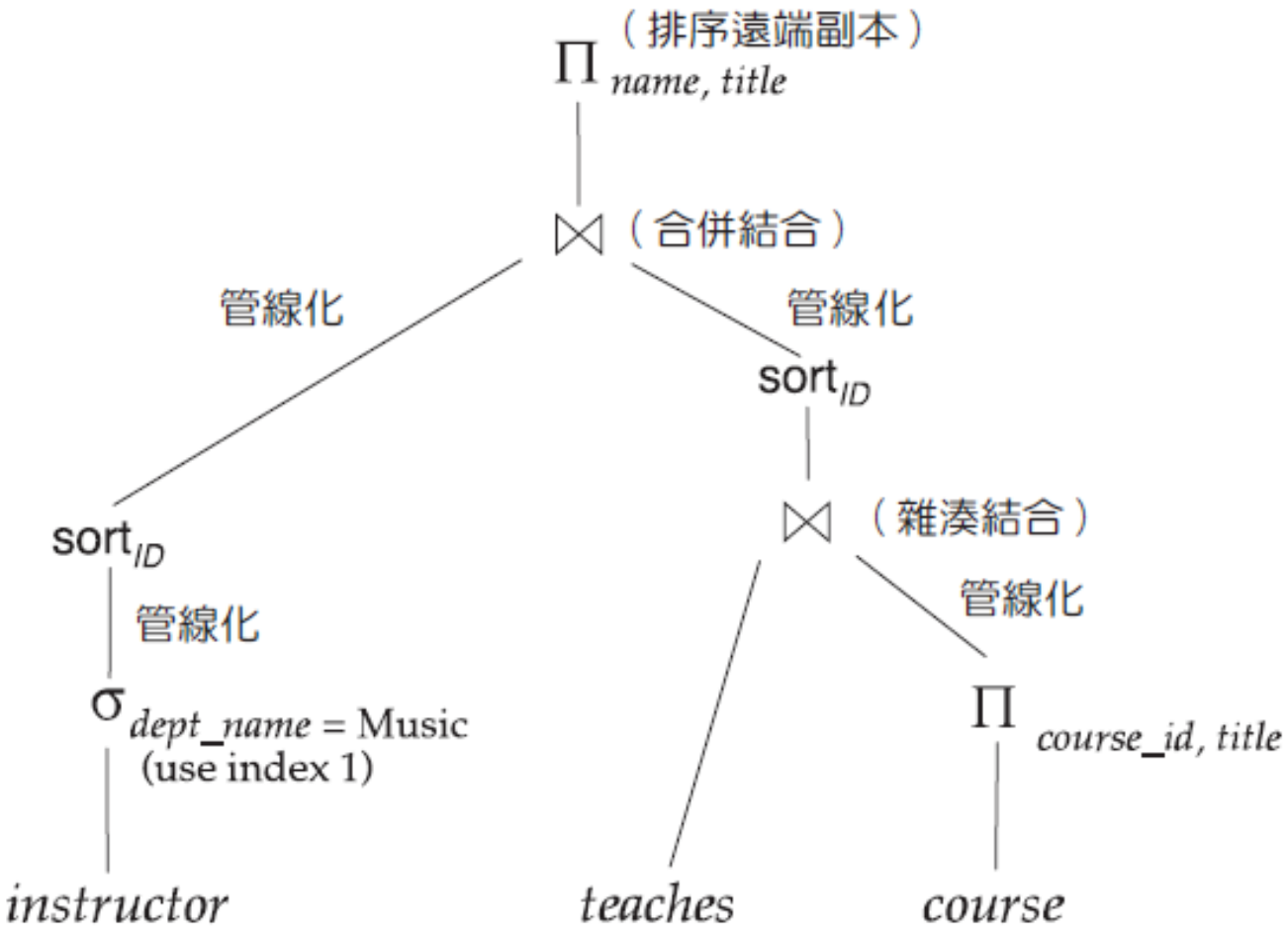
QUERY OPTIMIZATION

Query Evaluation





Query Evaluation (cont.)



如何進行 Query Evaluation？

- Query Evaluation 的步驟
  - 對給定的表達式產生邏輯上等價的表達式 (equivalence rules)
  - 對每個產生的表達式註解並產生替代查詢評估計劃
  - 估計每個評估計劃的成本，然後選擇一個成本最低的計劃
- 評估基礎
  - 關聯大小、索引深度
  - 對於中間結果的估算
  - 對於演算法的估算

等價規則

1. 兩個條件 AND 的 selection 可以被分解成為一連串獨立的 selection

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. Selection 是**可交換的(commutative)**

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3. 一連串的 projection 只需要留下最後一個，其他可以省略

$$\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{L_n}(E))\ldots)) = \Pi_{L_1}(E)$$

4. 選擇可以和 cartesian product 與 natural join 結合
   a. . $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$
   b. . $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

等價規則 (cont.)

5. 條件 Join 是可交換的

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

6. (a) Natural Join 符合結合律:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

(b)下列的 條件 Join 符合結合律:

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

其中 θ_2只涉及 E_2 和 E_3 的屬性

等價規則 (cont.)

7. 在下列狀況中，selection 可分散在條件 Join 中進行:

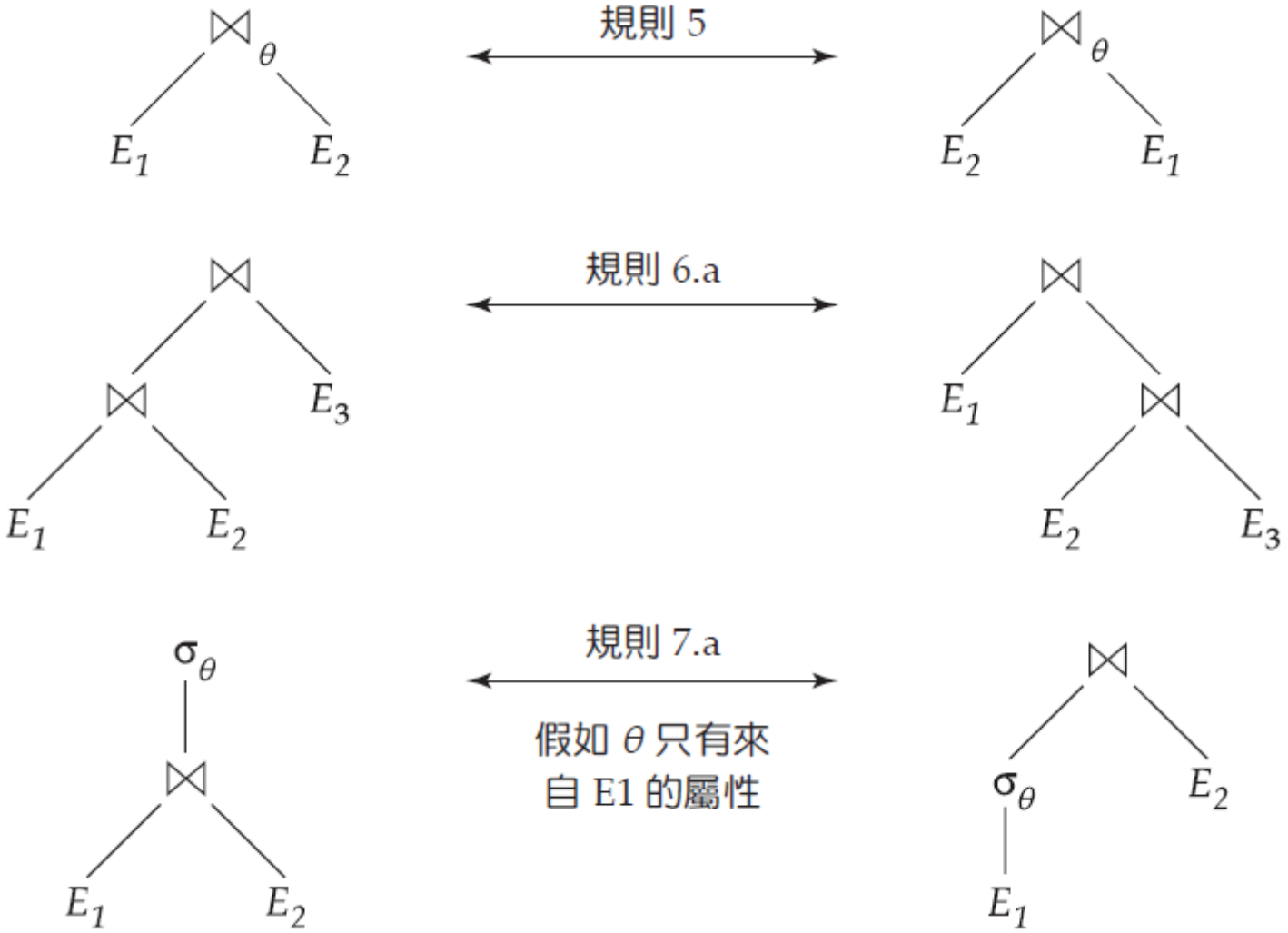
(a) 當**selection** 條件 θ_0 的所有屬性在只涉及被 **Join** 的其中一個表達式 （例如，E_1）的屬性時

$$\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2$$

(b) 當選擇條件 θ_1 只涉及E_1 屬性且 θ_2 只涉及E_2 屬性時

$$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$$

等價規則 (cont.)


規則 5
θ
E1
E2
E2
E1
規則 6.a
E3
E1
E2
E1
E2
E3
規則 7.a
假如 θ 只有來
自 E1 的屬性
σθ
E1
E2
σθ
E2
E1

等價規則 (cont.)

8. 假如符合以下條件，投影操作可分佈在條件 Join 操作:

(a)假設結合條件 θ只涉及$L_1 \cup L_2$ 的屬性，那麼:

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

(b)考慮結合 $E_1 \bowtie_\theta E_2$

- 讓 L_1 和 L_2 分別是 E_1 和 E_2 的屬性
- 讓 L_3是涉及結合條件 θ 的 E_1 屬性，但並不在$L_1 \cup L_2$ 內
- 讓 L_4 是涉及結合條件 θ 的 E_2屬性，但並不在$L_1 \cup L_2$ 內

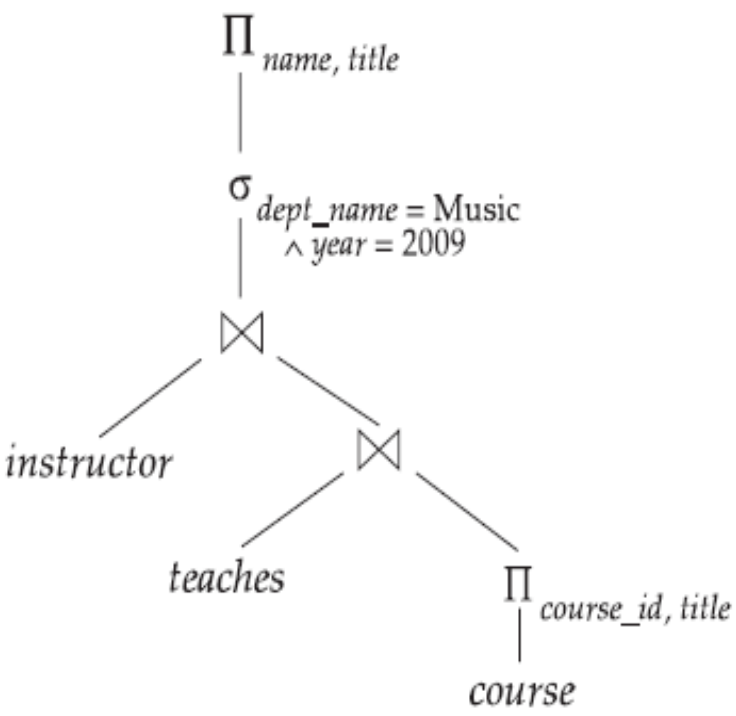
那麼，

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$$

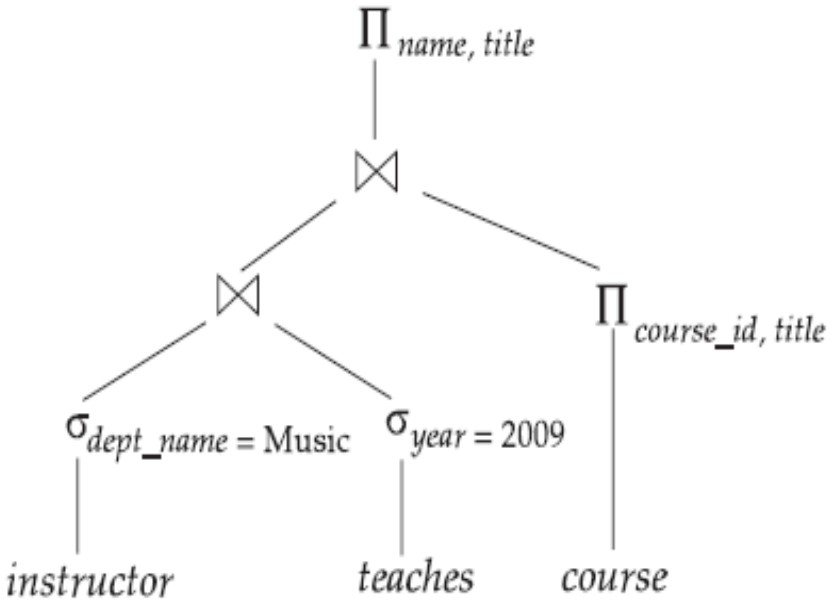
實例轉換

- 查詢: 找出所有音樂系在2009年開過課程的教師名稱，以及他們所開設的課程
  - $\Pi_{name,\ title}(\sigma_{dept\_name=\text{“Music”} \wedge gear\ =\ 2009}$
    $(instructor \bowtie (teaches \bowtie \Pi_{course\_id,\ title}\ (course))))$
- 我們可以先使用規則 (Rule 6a) 來轉換結合:
  - $\Pi_{name,\ title}(\sigma_{dept\_name=\text{“Music”} \wedge gear\ =\ 2009}$
    $((instructor \bowtie teaches) \bowtie \Pi_{course\_id,\ title}\ (course)))$
- 後一種形式的表達使得規則7.a(「提早執行選項」)可以派上用場，造成子表達式: $\sigma_{dept\_name\ =\ \text{“Music”}}\ (instructor)$ $\sigma_{\ year\ =\ 2009}\ (teaches)$

實例轉換



(a) 最初的表達式



(b) 多個轉換後的表達式

實例轉換

- 查詢範例: $\Pi_{name,\ title}(\sigma_{dept\_name=\text{“Music”}}(instructor) \bowtie teaches) \bowtie \Pi_{course\_id,\ title}(course))))$
- 我們計算子表達式

$$(\sigma_{dept\_name = \text{“Music”}}(instructor \bowtie teaches)$$

我們得到一個關聯，其架構是:
(ID, name, dept_name, salary, course_id, sec_id, semester, year)

- 我們可以透過推動基於等價規則8.a 和8.b 的投影，從架構裡去除一些屬性:

$$\Pi_{name,\ title}(\Pi_{name,\ course\_id}(\sigma_{dept\_name=\text{“Music”}}(instructor) \bowtie teaches)) \bowtie \Pi_{course\_id,\ title}(course))))$$

- 透過去除不必要的屬性，我們減少中間結果的行數，因此我們減少了中間結果的大小

結合順序的實例

- 對於所有關聯 $r_1, r_2,$ 和 r_3,

$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

- 若 $r_2 \bowtie r_3$ 相當大且 $r_1 \bowtie r_2$ 的值很小, 那我們會選擇以下公式

$$(r_1 \bowtie r_2) \bowtie r_3$$

如此一來我們會計算並儲存到較小的暫時關聯

結合順序的實例

- 考慮以下的表達式

$$\Pi_{name,\ title}(\sigma_{dept\_name=\text{“Music”}}(instructor) \bowtie teaches) \bowtie \Pi_{course\_id,\ title}(course))))$$

- 可以先選擇計算 $teaches \bowtie \Pi_{course\_id,\ title}(course)$,然後再結合其結果為 $\sigma_{dept\_name=\text{“Music”}}(instructor)$

  然而結果很可能是一個大的關聯.

- 我們知道一所大學有很多科系, 而有可能只有一小部分的大學教師與音樂系相關
  - 所以最好先計算以下式子

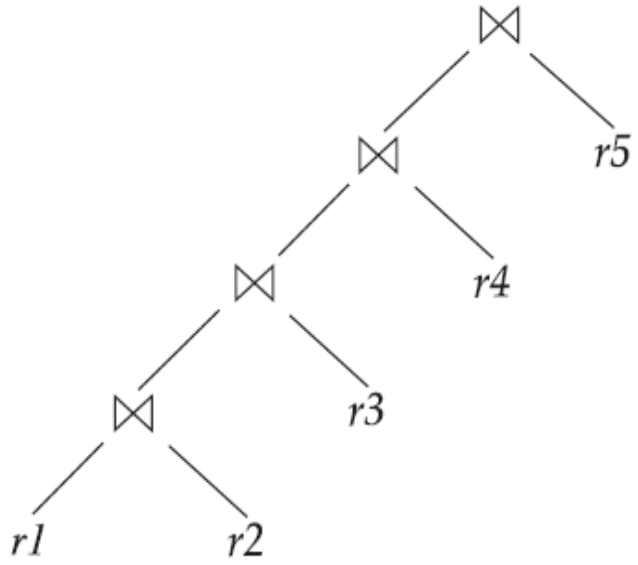
$$\sigma_{dept\_name=\text{“Music”}}(instructor) \bowtie teaches$$

尋找最佳結合順序 (動態規劃)

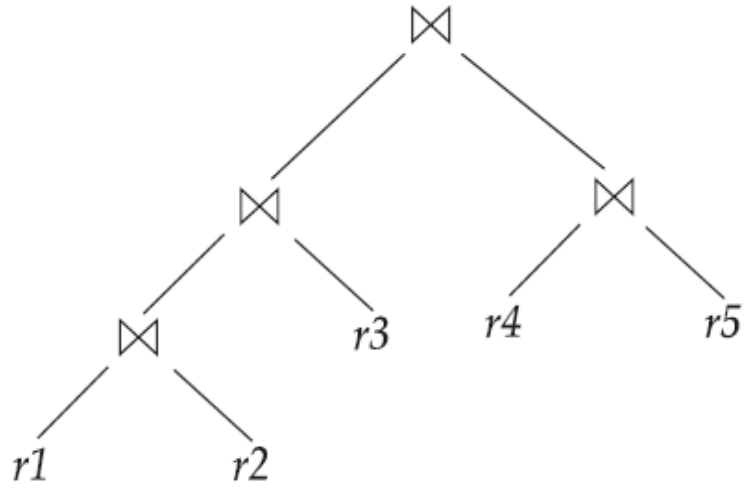
```
procedure findbestplan(S)
    if (bestplan[S].cost ≠ ∞)
            return bestplan[S]
    // else bestplan[S] has not been computed earlier, compute it now
    if (S contains only 1 relation)
            set bestplan[S].plan and bestplan[S].cost based on the best way
            of accessing S  /* Using selections on S and indices on S */

    else for each non-empty subset S1 of S such that S1 ≠ S
            P1= findbestplan(S1)
            P2= findbestplan(S - S1)
            A = best algorithm for joining results of P1 and P2
            cost = P1.cost + P2.cost + cost of A
            if cost < bestplan[S].cost
                    bestplan[S].cost = cost
                    bestplan[S].plan = "execute P1.plan; execute P2.plan;
                                        join results of P1 and P2 using A"
    return bestplan[S]
```

左深結合樹 (left-deep join tree)

- 右邊輸入用來做結合的都是關聯，
  並非結合的中間結果



(a) 左深結合樹



(b) 非左深結合樹

成本的最佳化

- 使用動態規劃來做密集樹最佳化
  - 時間複雜度為 $O(3^n)$.
  - 空間複雜度為 $O(2^n)$
- 找出 n 個關連的最佳左深結合樹:
  - 用: **for each** relation r in S

    let S1 = S – r
  - 取代‘ “**for each** non-empty subset $S1$ of S such that $S1 \neq S$”
- 若只有考慮左深樹, 則最佳結合順序的時間複雜度降為 $O(n\ 2^n)$