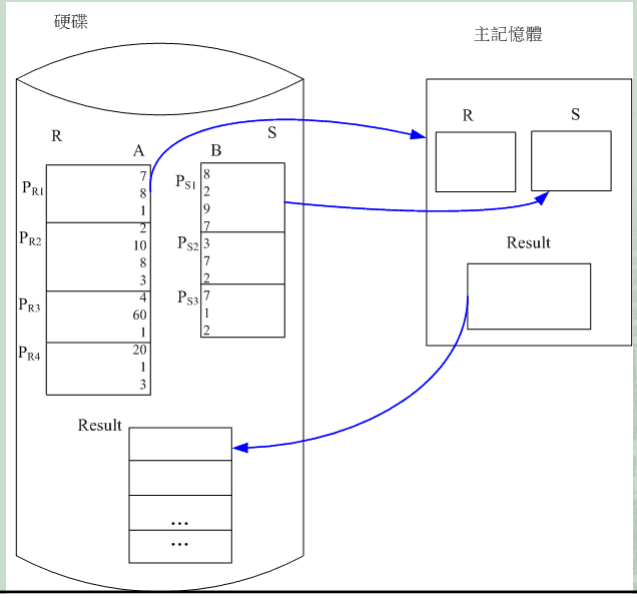
第十一章進階的查詢處理與最佳化

- JOIN的處理方式和成本推估
- PROJECT的處理方式和成本推估
- 集合運算子的處理方式和成本推估
- 彙總函數的處理方式和成本推估
- 多個關聯代數運算子的整合處理方式
- 實體資料庫設計與微調

JOIN的處理方式

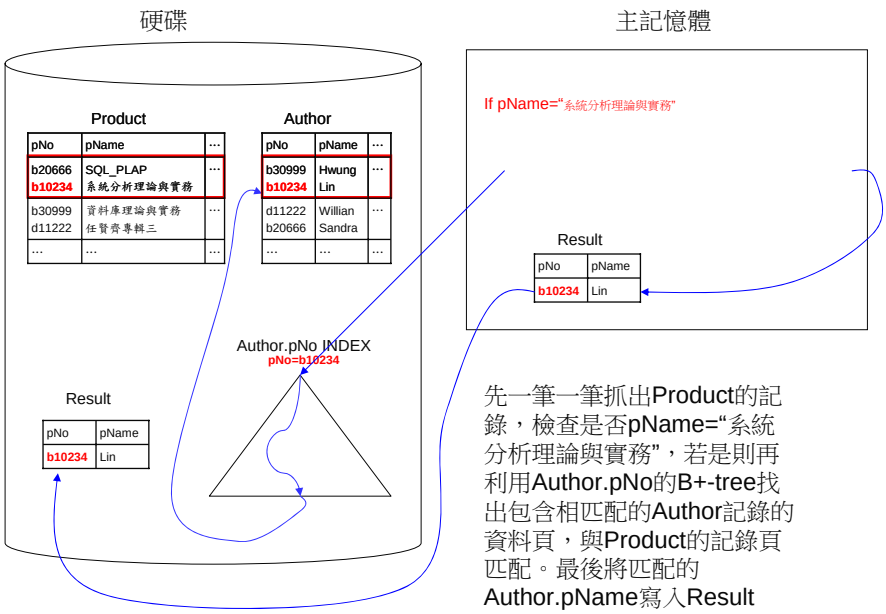
- **JOIN**運算子包括兩個資料表，其運算往往是最耗時的
- **R** $\bowtie_{R.A=S.B}$ **S**有三種處理方式
  - （JNL）巢狀迴圈法
    - 假設主記憶體裡有三個資料頁，分別用來容納資料表R，資料表S和資料表Result
    - 每擷取一個R資料頁，就要擷取所有S的資料頁，以便兩兩配對。如下頁圖所示

JOIN的處理方式（Cont.)



JOIN的處理方式（Cont.)

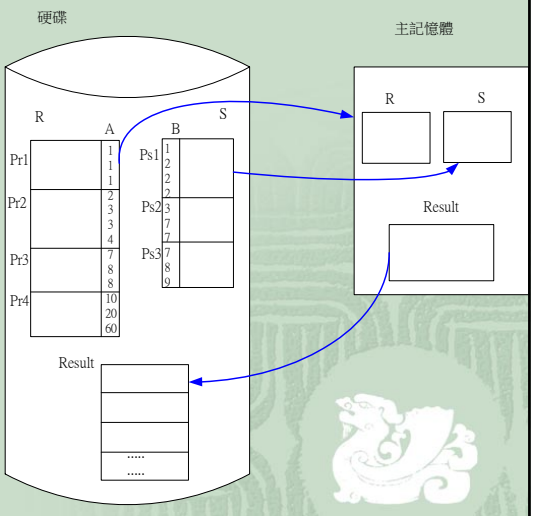
- （JIL）索引迴圈法
  - 利用其中一個匹配屬性的索引結構



先一筆一筆抓出Product的記錄，檢查是否pName="系統分析理論與實務"，若是則再利用Author.pNo的B+-tree找出包含相匹配的Author記錄的資料頁，與Product的記錄頁匹配。最後將匹配的Author.pName寫入Result

JOIN的處理方式（Cont.）

- （JSM）排序合併法
  - 假設在硬碟裡，兩個資料表的記錄恰好是依照R.A和S.B排序
  - 按次序輪流將R和S分別載入主記憶體一次即可完成將所有R的記錄和相匹配的S記錄結合



JOIN處理方式的成本推估

- $R \bowtie_{R.A=S.B} S$
  - （JNL）巢狀迴圈法
    - 資料表R的載入成本：b_R
    - 資料表S的載入成本：$b_R \times b_S$
    - 因此總成本為$b_R + b_R \times b_S$
  - （JIL）索引迴圈法
    - 資料表R的載入成本：b_R
    - 資料表S和S.B索引的載入成本
      - case 1: S.B為非群聚索引：$r_R \times (x_B + s_B)$
      - case 2: S.B為群聚索引：$r_R \times (x_B + \left\lceil \frac{s_B}{bfr_S} \right\rceil)$
    - 因此總成本為
      - case 1: S.B為非群聚索引：$b_R + r_R \times (x_B + s_B)$
      - case 2: S.B為群聚索引：$b_R + r_R \times (x_B + \left\lceil \frac{s_B}{bfr_S} \right\rceil)$

JOIN處理方式的成本推估(Cont.)

- （JSM）排序合併法
  - 資料表R的載入成本：b_R
  - 資料表S的載入成本：b_S
  - 因此總成本為$b_R + b_S$
  - 如果R和S的記錄並沒有依照R.A和S.B排序，要採用（JSM），則要加上排序的成本

    $K \times (b_R \log_2 b_R + b_s \log_2 b_s)$

範例一

- 假設$r_{Product}$ = 100,000，$b_{Product}$ = 5000（因此$bfr_{Product}$ = 20），r_{Author} = 200,000，b_{Author} = 500（因此bfr_{Author} = 400），有二個索引Product.pNo和Author.pNo，且此二索引皆為群聚索引。$x_{Product.pNo}$=3，$x_{Author.pNo}$=3，$s_{Author.pNo}$=5。
- 考慮以下查詢式的成本

  Product $\bowtie_{\mathbf{Product.pNo=Author.pNo}}$ Author

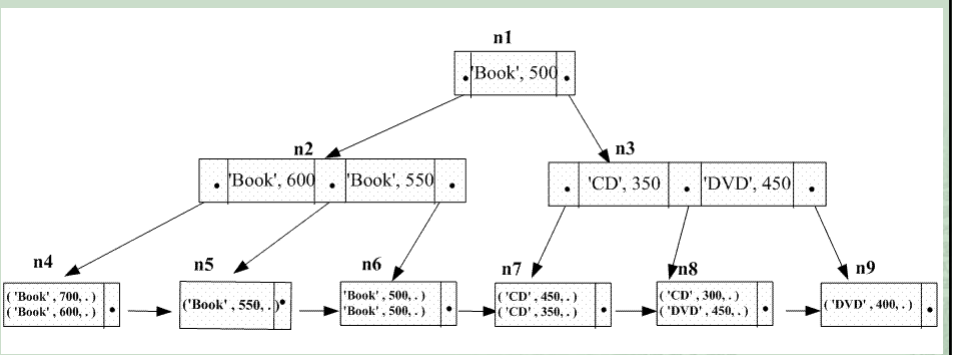
範例一

- （JNL）：
- case 1: 以Product為外部迴圈：$b_{Product}+b_{Product}\times b_{Author}=5000+5000\times 500=2,505,000$
- case 2: 以Author為外部迴圈：$b_{Author}+b_{Product}\times b_{Author}=500+5000\times 500=2,500,500$
- （JIL）：
- case 1: 利用Product.pNo的索引。由於Product.pNo為主鍵，所以sProduct.pNo=1，因此，成本為$b_{Author}+r_{Author}\times(x_{Product.pNo}+1)=500+200000\times 4=800,500$
- case 2: 利用Author.pNo的索引。成本為 $b_{Product}+r_{Product}\times(x_{Author.pNo}+1)=5000+100000\times(3+1)=405,000$。
- （JSM）：因為Product.pNo和Author.pNo為群聚索引，可知Product和Author裡的記錄已分別按照Product.pNo和Author.pNo排序。因此成本為 $b_R+b_S=5000+500=5,500$。

PROJECT的處理方式

- 允許重複的PROJECT運算子實作方式有兩種
  - (PL) 資料頁循序搜尋
  - (PI) 利用索引結構
    - 假設有(catalog, unitPrice)的索引結構
    - 處理以下查詢句只要搜尋該索引結構的葉節點即可

      Q4: SELECT catalog, unitPrice
      FROM Product;
    - 參考下頁圖9-7



練習11-2

- 考慮上頁圖9-7的索引，若要利用該索引結構來處理以下查詢句

```
SELECT catalog, unitPrice
FROM Product;
```

  請問要造訪哪些硬碟頁？

- Ans:

  n1, n2, n4, n5, n6, n7, n8, n9

PROJECT的處理方式(Cont.)

- 不允許重複的PROJECT運算子
  - RESULT = π <屬性串列>(R)

```
SELECT DISTINCT catalog, unitPrice
FROM Product;
```

  - 實作方式其實也類似，但要作刪除重複
  - 三種方式
    - (PLS) 資料頁循序搜尋和排序
    - (PLH) 資料頁循序搜尋和雜湊
    - (PIND) 利用索引結構

PROJECT的處理方式(Cont.)

- (PLS) 資料頁循序搜尋和排序

catalog	unitPrice	
'Book'	500	
~~'Book'~~	~~500~~	刪除重複
'Book'	550	
'Book'	600	
'Book'	700	
'CD'	300	
'CD'	350	
'CD'	450	
'DVD'	400	
'DVD'	450	

PROJECT的處理方式(Cont.)

- (PLH) 資料頁循序搜尋和雜湊
  - 將資料表R的每一資料頁裡的每一筆記錄<屬性串列>的屬性值取出，再將由這些屬性值所形成的記錄利用雜湊法分配到雜湊桶去，最後造訪雜湊桶一次刪除重複屬性值
  - 範例：雜湊函數為F(r) = r % 5

F(<'Book', 500>) = 4
F(<'Book', 550>) = 2
F(<'Book', 600>) = 0
F(<'Book', 700>) = 3
F(<'CD', 300>) = 0
F(<'CD', 350>) = 1
F(<'CD', 450>) = 4
F(<'DVD', 400>) = 3
F(<'DVD', 450>) =0

B0	<'CD', 300>	<'Book', 600>	<'DVD', 450>
B1	<'CD', 350>		
B2	<'Book', 550>		
B3	<'DVD', 400>	<'Book', 700>	
B4	<'Book', 500>	<'CD', 450>	~~<'Book', 500>~~

刪除重複

PROJECT的處理方式(Cont.)

- (PIND) 利用索引結構
  - 假設所要Project出來的屬性均存在一個索引結構裡，同於(PI)
  - 由於其葉節點已按照所要的屬性值排序，屬性值相同的記錄會被排在一起，造訪一次即可取出不重複的屬性值
  - 參考圖9-7

範例二

- 假設$r_{Product}$ = 100,000，$b_{Product}$ = 5000，$d_{catalog}$= 100，$x_{catalog}$=2，$bl1_{catalog}$=100。
- 考慮以下查詢句：

SELECT catalog

FROM Product

(PL) 成本為$b_{Product}$ = 5000

(PI) 成本為$x_{catalog}$－1+$bl1_{catalog}$ = 2－1+100 = 101

範例二

- 考慮以下查詢句：

SELECT DISTINCT catalog
FROM Product

假設$b_{Product_catalog}$ = 50

（PLS）成本為$b_{Product}+b_{Product_catalog}+K\times b_{Product_catalog}\log_2 b_{Product_catalog}+b_{Product_catalog}$
$\approx 5000+50+K\times 50\times \log_2 50+50=5100+282K$。

（PLH）成本為
$b_{Product}+2r_{Product}+b_{Product\_catalog}=5000+200000+50=205050$

（PIND）成本為$x_{Product}-1+bl1_{Product} = 2-1+100 = 101$

交集（R∩S）的處理方式

- 兩種方式
  - （IS）排序法
    - 將R的記錄由小排到大，再將S的記錄由小排到大，最後兩兩比對
- 練習11-4：參考圖11-3，若我們要用排序法做R.A ∩ S.B，請列出資料頁載入主記憶體的次序
- Ans: Pr1, Ps1, Pr2, Ps2, Pr3, Ps3, Pr4

交集（R∩S）的處理方式

- （IH）雜湊法
  - 將R的記錄按雜湊法分配到雜湊桶去，再將S的每一筆記錄藉由相同的雜湊函數到某一個雜湊桶，檢查是否存在相同的R記錄
  - 參考圖11-4

聯集和差集的處理方式

- **聯集（R∪S）的處理方式**
  - 只要將R和S的記錄合起來 即可
  - 重複的記錄去除用排序法或雜湊法
- **差集（R－S）的處理方式**
  - 可以用排序法和雜湊法
  - 但比對時與交集的處理方式相反

練習 11-5

- 參考圖11-3 ，若我們要用排序法做S.B–R.A，請列出資料頁載入主記憶體的次序
- Ans:

  Ps1, Pr1, Pr2, Ps2, Pr3, Ps3, Pr4

集合處理方式的成本推估

- 考慮以下交集運算：Result=**R ∩ S**
  - （IS）排序法：假設資料表R和S已經排序好，總成本為$b_R + b_S$
  - **（IH）雜湊法：**
    - 將R放入雜湊桶，成本：$b_R + 2r_R$
    - 載入每一個S的資料頁，成本：b_S
    - 對於S的每一筆記錄，需讀入並檢查雜湊桶的記錄，假設每一個雜湊桶僅有一頁，成本：r_S
    - **總成本為$b_R + 2r_R + b_S + r_S$**

彙總函數處理方式

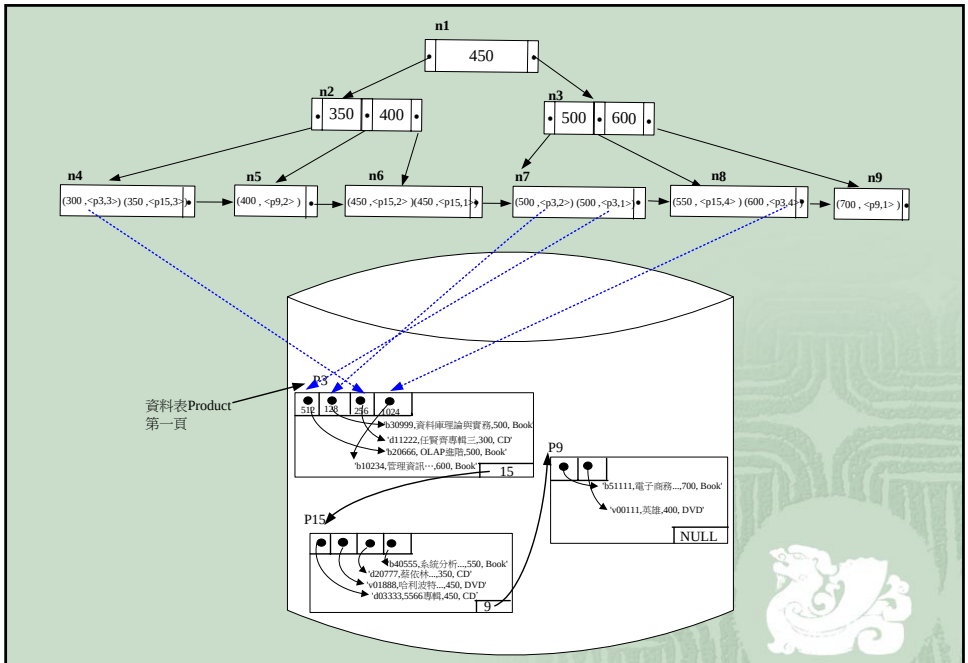
- 沒有分群的簡單彙總

```
SELECT AVERAGE(unitPrice)
FROM Product;
```

  - （AL）資料頁循序搜尋
  - （AI）利用索引結構
    - 假設彙總函數的屬性有建置索引
    - 將存在葉節點裡的該屬性值做彙總運算

練習 11-7

- 參考下頁圖9-6的索引結構，分別用（AL）和（AI）的方式處理SUM（unitPrice），請問其分別造訪了哪些硬碟頁？
- Ans:
  - (AL): P3, P15, P9
  - (AI): n1, n2, n4, n5, n6, n7, n8, n9



彙總函數處理方式(Cont.)

- 練習11-8：參考上頁圖9-6的索引結構，若用（AI）處理MAX（unitPrice），請問需造訪哪些硬碟頁？
- Ans:

  n1, n3, n9

彙總函數處理方式(Cont.)

- 有分群的簡單彙總
  - 範例

```
SELECT AVERAGE(unitPrice)
FROM Product
GROUP BY catalog;
```

  - 三種處理方式
    - （AGS）排序分群
    - （AGH）雜湊分群
    - （AGI）索引分群

彙總函數處理方式(Cont.)

- （AGS）排序分群

‘Book’	500	
‘Book’	500	
‘Book’	600	→ （’Book’, 570）
‘Book’	550	（’CD’, 366）
‘Book’	700	（’DVD’, 425）
‘CD’	300	
‘CD’	350	
‘CD’	450	
‘DVD’	450	
‘DVD’	400	

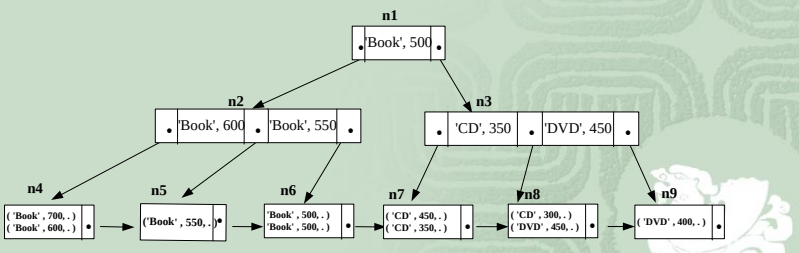
彙總函數處理方式(Cont.)

- （AGH）雜湊分群
  - 將所有的記錄按照分群屬性的雜湊函數分配到雜湊桶
  - 再將每一雜湊桶的記錄按照分群屬性排序
  - 最後將每一雜湊桶造訪一遍
  - 假設H(‘Book’) = 1, H(‘CD’) = 0, H(‘DVD’) = 0

B0	<‘CD’, 300>	<‘CD’, 350>	<‘CD’, 450>	<‘DVD’, 450>	<‘DVD’, 400>
B1	<‘Book’, 500>	<‘Book’, 500>	<‘Book’, 600>	<‘Book’, 550>	<‘Book’, 700>

彙總函數處理方式(Cont.)

- （AGI）索引分群
  - 剛好存在一個包括分群屬性和彙總函數屬性的索引
  - 造訪葉節點的屬性值，即可算出每一群的彙總函數值
  - 參考下圖



彙總函數處理方式的成本推估

- 沒有分群的簡單彙總

  SELECT AVERAGE(A)
  FROM R;

  - （AL）資料頁循序搜尋

    b_R

  - （AI）利用索引結構

    $x_A - 1 + bI1_A$

彙總函數處理方式的成本推估(Cont.)

- 有分群的簡單彙總

  SELECT AVERAGE(A2)
  FROM R
  GROUP BY A1;

- （AGS）排序分群，假設這些屬性佔了b_{A1_A2}個資料頁

  $b_R + K \times b_{A1\_A2} \times \log_2 b_{A1\_A2} + b_{A1\_A2}$

- （AGH）雜湊分群

  $b_R + 2r_R + b_{A1\_A2}$

- （AIG）索引分群

  $x_{A1,A2} - 1 + bI1_{A1,A2}$

範例四

- 假設$r_{Product}$ = 100,000， $b_{Product}$ = 5,000，假設unitPrice有建置一索引，$x_{unitPrice}$ = 3，$bI1_{unitPrice}$ = 400。
- 考慮以下查詢：

```
SELECT SUM(unitPrice)
FROM Product
```

  - （AL）：$b_{Product}$ = 5000
  - （AI）：$x_{unitPrice}-1+bI1_{unitPrice}$ =3－1+400=402

範例四(Cont.)

- 再考慮以下查詢：

```
SELECT SUM(unitPrice)
FROM Product
GROUP BY catalog
```

- 假設(catalog, unitPrice)有建置一索引，$b_{catalog, unitPrice}$ = 1000，$x_{catalog, unitPrice}$ = 4，$bI1_{catalog, unitPrice}$ = 1000。且Product並沒有按（catalog, unitPrice）排序好。
  - （AGS）：$b_{Product} + K \times b_{catalog\_unitPrice} lg b_{catalog\_unitPrice} + b_{catalog\_unitPrice}$ =5000 + K×1000lg1000+1000=10,000K+6,000
  - （AGH）：$b_{Product} + 2r_{Product} + b_{catalog\_unitPrice}$ =5000 + 200,000+1000=206000
  - （AIG）：$x_{catalog\_unitPrice}-1+bI1_{catalog\_unitPrice}$=4－1+1000=1003

練習11-9

- 在範例七中，若想節省（AGS）成本，則群聚索引該建置在catalog或unitPrice上？
- Ans:

  catalog上，因分群屬性是catalog。此時只要將Product的資料頁造訪一遍，即可計算出各類商品的平均定價，因此成本為$b_{Product}$=5,000。

多個關聯代數運算子的整合處理方式

Temp1=$\sigma_{pName='系統分析理論與實務'}$（Product）;

Temp2= Temp1 $\bowtie_{Temp1.pNo=Author.pNo}$ Author;

Result= π_{name}（Temp2）;

- 如**PROJECT, SELECT**和**JOIN**常相鄰出現
- 我們可以Product當外部迴圈的資料表，但Product的記錄必須先符合pName='系統分析理論與實務'才能與Author的記錄匹配。此外，只有匹配的Author.name才寫入Result。

範例五

- 假設$r_{Product}$ = 100,000， $b_{Product}$ = 5,000，（因此$bfr_{Product}$ = 20），r_{Author} = 200,000，b_{Author} = 500（因此bfr_{Author} = 400），有三個索引Product.catalog，Product.pNo和Author.pNo，其中Product.catalog為群聚索引，其他兩個索引為非群聚索引。$x_{catalog}$=2，$d_{catalog}$=100（因此$s_{catalog}$ = 1000），$bl1_{catalog}$=100，$x_{Product.pNo}$=3，$x_{Author.pNo}$=3，$s_{Author.pNo}$=2。
- 考慮以下查詢句：

  SELECT pName, name
  FROM Product **AS** P, Author **AS** A
  WHERE P.pNo=A.pNo **AND** catalog = 'CD';

範例五(Cont.)

- 三種可能的處理方式：
- 僅利用Product.catalog的索引：透過Product.catalog的索引結構來處理$\sigma_{catalog='CD'}$和巢狀迴圈法（JNL）來處理$\bowtie_{P.pNo=A.pNo}$Author。
  - 透過Product.catalog的索引找出第一個catalog='CD'的資料頁指標：$x_{catalog}$=2
  - 巢狀迴圈法：$b_{Product.catalog='CD'} + b_{Product.catalog='CD'} \times b_{Author}$=50+50×500=25050
  - 故總成本為 25,052

範例五(Cont.)

- 利用Product.catalog的索引和Author.pNo的索引：透過Product.catalog的索引結構來處理$\sigma_{catalog = 'CD'}$和索引迴圈法（JIL）來處理$\bowtie_{P.pNo=A.pNo}$。
  - 透過Product.catalog的索引找出第一個catalog=’CD’的資料頁指標：$x_{catalog}=2$
  - catalog = ‘CD’的產品記錄資料頁數：$b_{Product.catalog='CD'}=50$
  - 利用Author.pNo的索引結構抓取出相對應的Author記錄：$s_{catalog} \times (x_{Author.pNo}+s_{Author.pNo})=1000 \times 5=5000$
  - 故總成本為 5,052

範例五(Cont.)

- 僅利用Product.pNo的索引：將每一筆Author記錄利用Product.pNo的索引結構找出匹配的Product，之後再檢查是否其catalog=’CD’
  - 即是索引迴圈法（JIL），故其總成本為$b_{Author}+r_{Author} \times (x_{Product.pNo}+s_{Product.pNo})=500+200,000 \times (3+1)=800,500$

實體資料庫設計與微調

- 實體資料庫設計（Physical database design）的目的是透過設計適當的資料表與索引來提高資料庫應用系統在運作時的執行效率。
- 資料庫微調（Database tuning）則是在實體資料庫設計後，資料庫應用系統且實際上線以後，根據上線結果來調整資料表與索引的設計，或是修改資料庫應用系統裡的SQL查詢句來進一步提升資料庫應用系統的效率。
- 三種方式：
  - 索引的選定與建立
  - 調整資料庫綱目
  - 修改SQL查詢句

索引的選定與建立

- 列出一些需要被最佳化的查詢句。這些查詢句通常是重要的查詢句且執行速度不如預期。
- 觀察那些查詢句WHERE子句裡的查詢條件。依據之前成本推估的經驗，找出一些適合建立索引的屬性。
- 依據以下三點找出最適合的屬性來建立索引：
  - 每一個建立的索引都會被用於某些查詢句的最佳化。
  - 一個索引最好能被多個查詢句使用到。
  - 如果使用到一索引的查詢次數遠少於對該索引的修改次數，慎重考慮是否要建立該索引。

範例六

- 考慮以下的SQL查詢句：

  SELECT pName, name
  FROM Product **AS** P, Author **AS** A
  WHERE P.pNo=A.pNo **AND** catalog = ‘CD’；

- 檢視WHERE子句後，發現有三個屬性可以建立索引：
  - Product.pNo
  - Author.pNo
  - Product.catalog
- 在做過如範例五的推估後，很明顯的最有效率的執行方案是第二種，也就是說我們該在Product.catalog上建立群聚索引，最好也在Author.pNo上建立一般索引。

索引的選定與建立

- 有些DBMS提供一個資料庫微調系統（Database tuner），用來提供該建立哪些索引的建議
- 一個比較實際的作法是依據使用者對查詢最佳化的瞭解，概略性的選定並建立一些索引，經過一段時間後若發現其對資料增刪造成太大負擔，再替換一些索引，靠經驗慢慢調出一套適合你們環境的索引設定。

調整資料庫綱目

- 有以下三種方式：
  - 反正規化（Denormalization）
    - 考慮圖8-12(a)的關聯綱目：
      - Transaction（pNo, tNo, amount, salePrice, invNo）
    - 和8-12(b)的關聯綱目
      - Transaction1(pNo, invNo, amount, salePrice)
      - Transaction2(invNo, tNo)
    - 考慮“列出售價超過300之交易商品的pNo, tNo, salePrice屬性值”，第一種方式的效率較高。

調整資料庫綱目（Cont)

- 資料表的垂直切割
  - 考慮4-3的Transaction關聯綱目：
    - Transaction(tNo, transMid, method, transTime, tNo, bankId, bankName, cardType, cardId, dueDate)
  - 若付款相關屬性與其他屬性很少一起被用到，則可分解如下：
    - TransInfo(tNo, transMid, method, transTime)
    - TransPayment(tNo, bankId, bankName, cardType, cardId, dueDate)

調整資料庫綱目（Cont)

- 資料表的水平切割
  - 考慮圖4-3的Product資料表，假設有很多的查詢只針對書籍，則可將Product水平切割成Book和OtherProduct兩個資料表
  - 將Product設成VIEW，如下：

```
CREATE VIEW Product AS
((SELECT * FROM Book)
UNION
(SELECT * FROM OtherProduct));
```


查詢句的改寫

- 有些DBMS對於WHERE子句裡稍複雜的數學運算和字串比對都不作最佳化
  - WHERE unitPrice/1000 > 5 ➔ WHERE unitPrice > 5000
  - WHERE name LIKE ‘黃%’ ➔ WHERE lastName = ‘黃’
- 有些DBMS不會試圖將巢狀查詢句改寫成一般查詢句再行處理，因此能用一般查詢句完成的，就不要用巢狀查詢句
- DISTINCT處理較費時，除非必要，否則應當避免。
- 能夠用一個（非巢狀）查詢句來處理的，盡量不要用多個，以避免暫存檔的產生。
- SQL 敘述盡量不要置於迴圈內，以免因需下達大量的SQL敘述，導致效率變差

查詢句的改寫(Cont.)

- 有時巢狀查詢句可以改寫成一或多個SQL查詢句，更有效率，如下：
  - 在每一類商品裡，找出定價最高的商品編號、商品類別和定價

```
SELECT pNo, catalog, unitPrice
FROM Product AS P
WHERE unitPrice =
      SELECT MAX(unitPrice)
      FROM Product AS M
      WHERE P.catalog=M.catalog;
```

→

```
INSERT INTO HighestPrice (catalog,
    unitPrice)
SELECT catalog, MAX(unitPrice)
 FROM Product
 GROUP BY catalog;

SELECT pNo, catalog, unitPrice
FROM Product NATURAL JOIN
    HighestPrice;
```


查詢句的改寫(Cont.)

- 分群的處理較費時，有些查詢的分群其實並不是必要的，如下：
  - 找出書的最高定價

```
SELECT catalog, MAX(unitPrice)
FROM Product
GROUP BY catalog
HAVING catalog = 'Book';
```

→

```
SELECT catalog, MAX(unitPrice)
FROM Product
WHERE catalog = 'Book';
```