效能校調

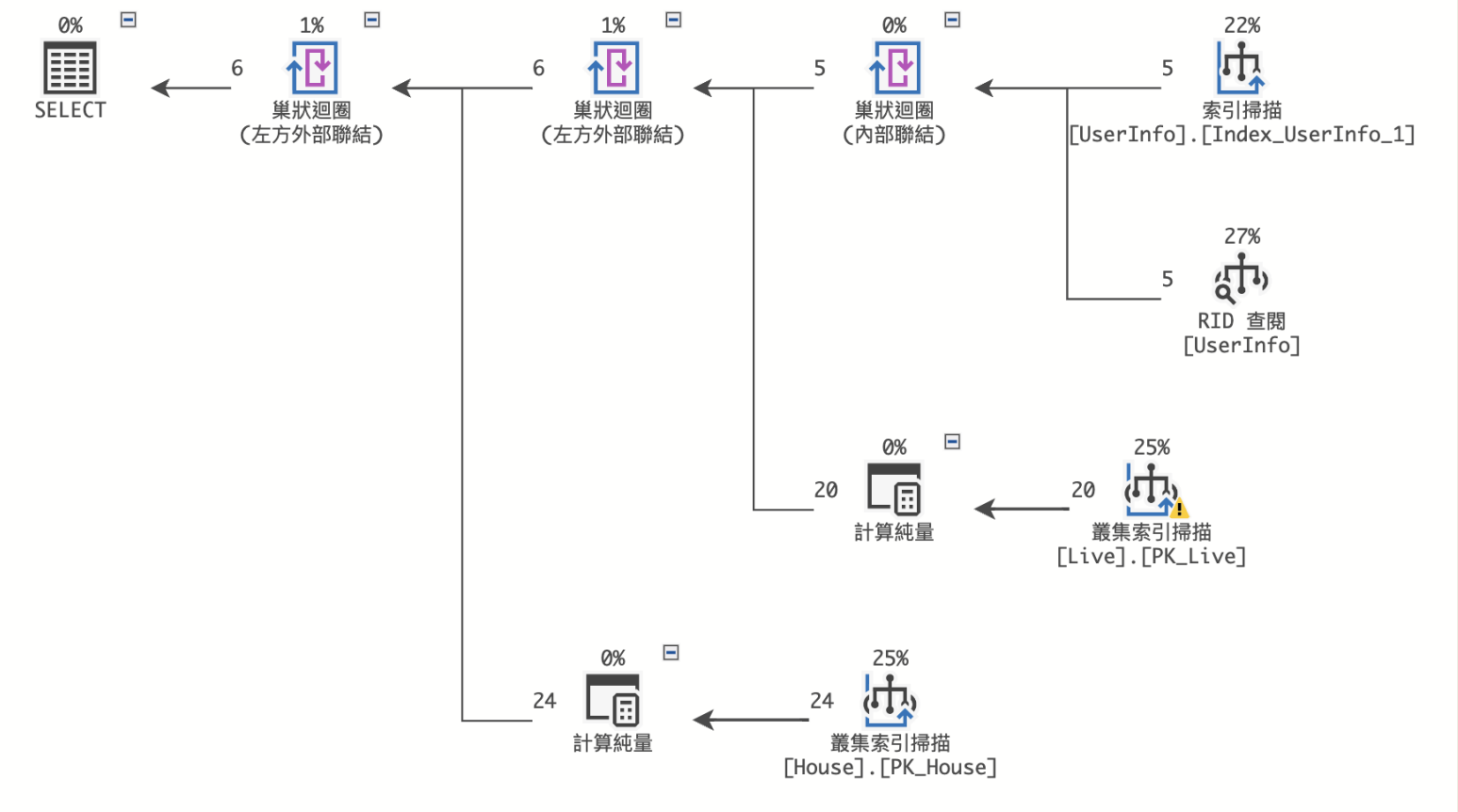
朱克剛

叢集與非叢集索引

- 資料表中的資料順序會與叢集索引順序一致
- 一個資料表只能有一個叢集索引，主索引預設就是叢集索引
- 有叢集索引的資料表稱為叢集資料表，沒有叢集索引的資料表稱為堆積或堆疊（heap）
- 非叢集索引有獨立的索引資料儲存區，透過資料列定位器指向實際資料
- 沒有索引的資料搜尋稱為「資料表掃描」

工具

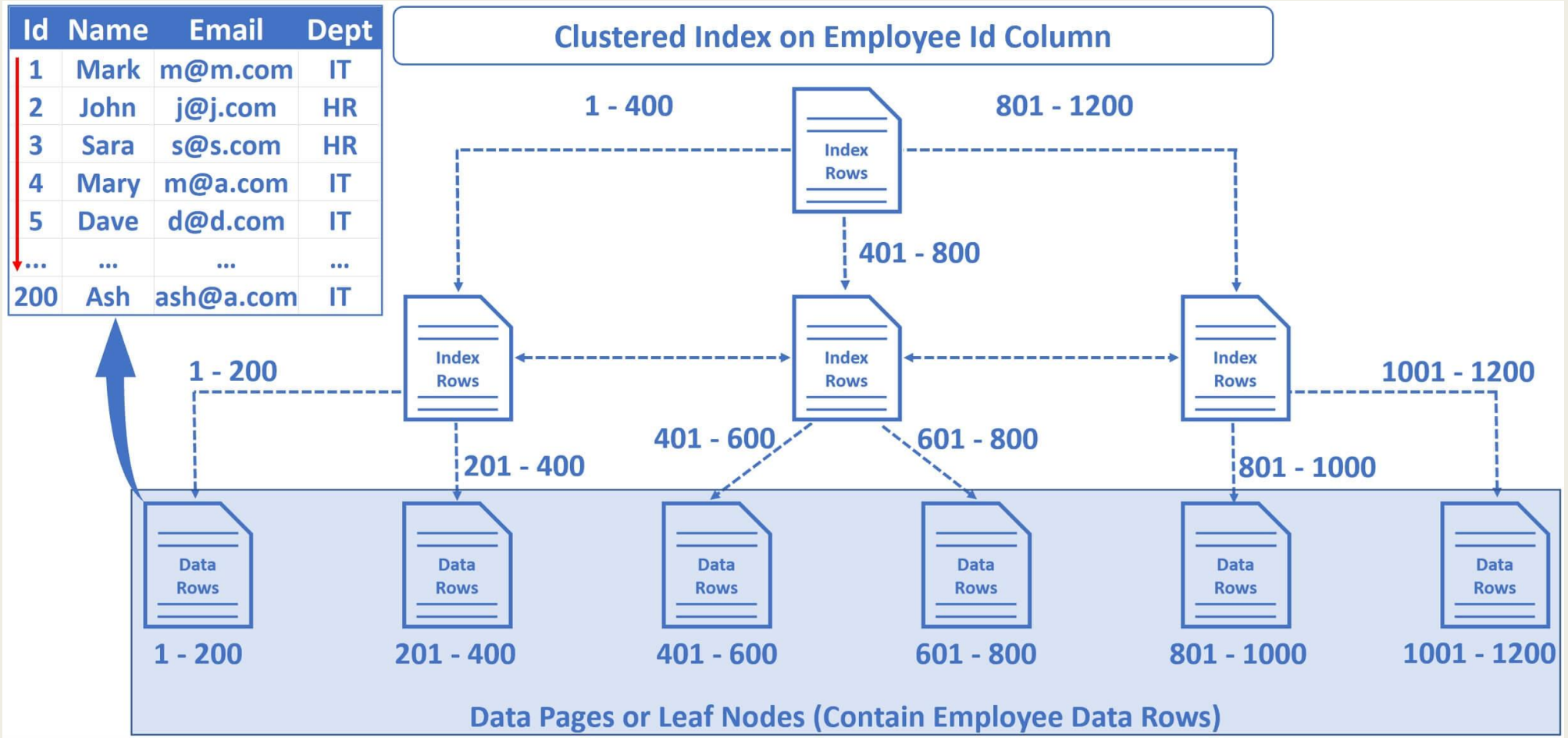
- 透過查詢計畫觀察查詢最佳化工具如何運作
- SQL Server 判斷不使用索引反而比較快時就不會使用到索引



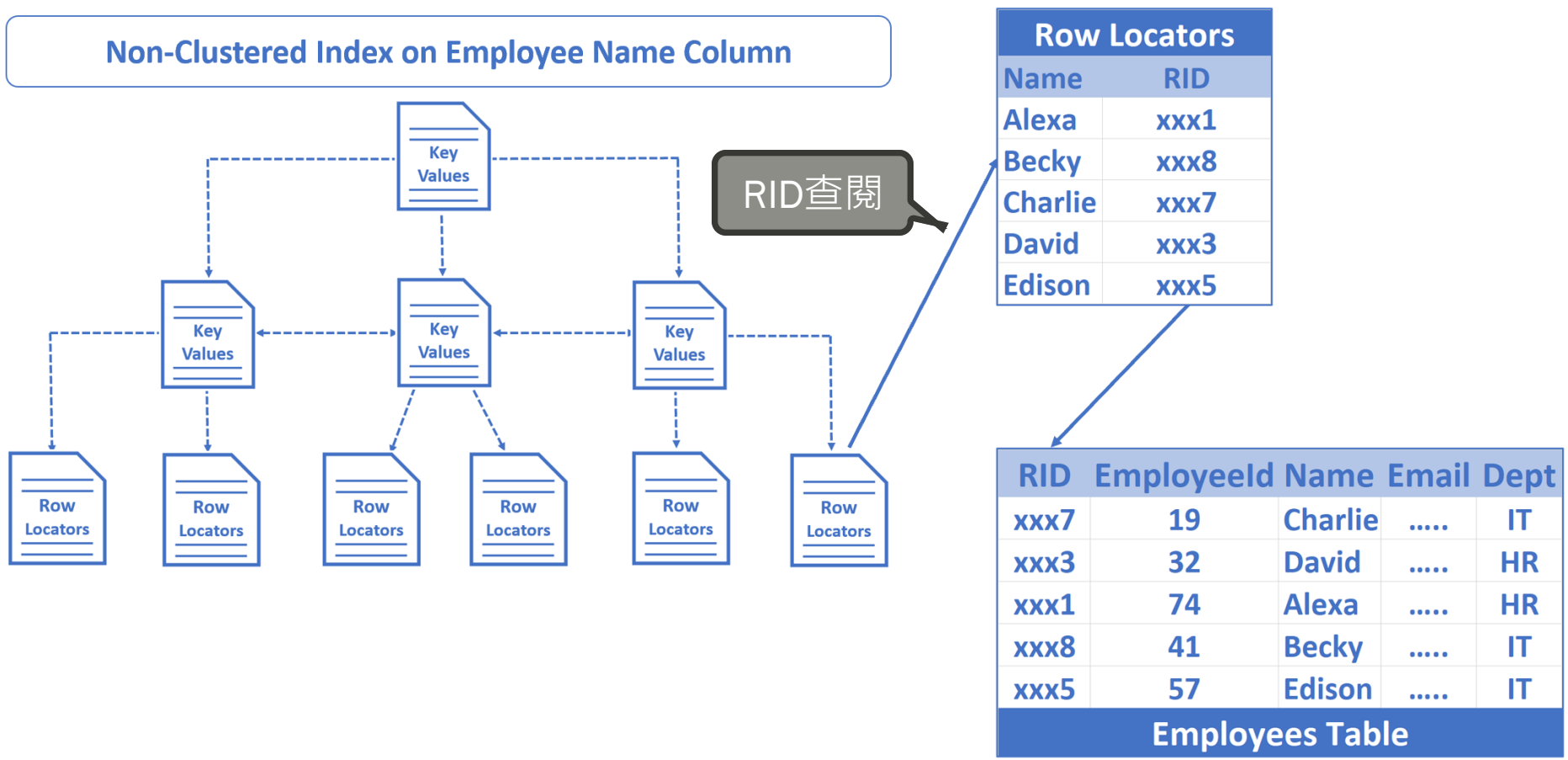
搜尋 優於 掃描

search 優於 scan

叢集索引結構


Id Name Email Dept
1 Mark m@m.com IT
2 John j@j.com HR
3 Sara s@s.com HR
4 Mary m@a.com IT
5 Dave d@d.com IT
... ...
200 Ash ash@a.com IT
Clustered Index on Employee Id Column
1 - 400
801 - 1200
Index Rows
401 - 800
Index Rows
Index Rows
Index Rows
1 - 200
1001 - 1200
401 - 600
601 - 800
201 - 400
801 - 1000
Data Rows
Data Rows
Data Rows
Data Rows
Data Rows
Data Rows
1 - 200
201 - 400
401 - 600
601 - 800
801 - 1000
1001 - 1200
Data Pages or Leaf Nodes (Contain Employee Data Rows)

非叢集索引結構


Non-Clustered Index on Employee Name Column
Key Values
Key Values
Key Values
Key Values
Row Locators
Row Locators
Row Locators
Row Locators
Row Locators
Row Locators
RID查閱
Row Locators
Name RID
Alexa xxx1
Becky xxx8
Charlie xxx7
David xxx3
Edison xxx5
RID EmployeeId Name Email Dept
xxx7 19 Charlie ..... IT
xxx3 32 David ..... HR
xxx1 74 Alexa ..... HR
xxx8 41 Becky ..... IT
xxx5 57 Edison ..... IT
Employees Table

何時使用叢集索引

- JOIN 的欄位（被參考的欄位）
- 當下列的查詢經常是 select * 的時候，建議使用叢集索引
- 用作 >、<、>=、<= 與 between ... and ...
- Order By 或 Group By 欄位，因為他們都跟排序有關

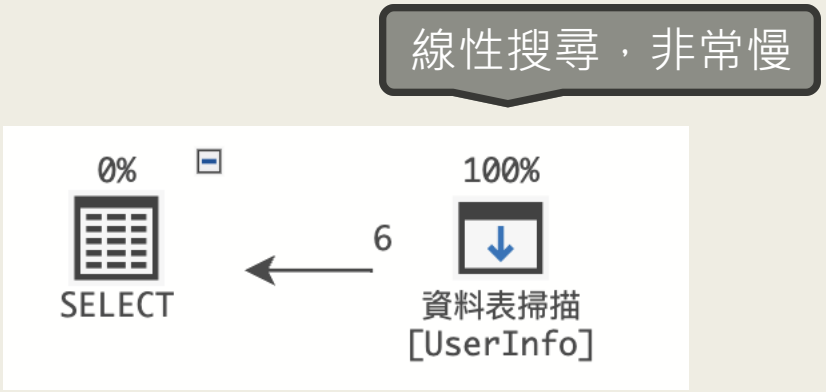
何時使用非叢集索引

- 使用 JOIN 或 GROUP BY 的欄位
- 用來做搜尋條件的欄位
- 要注意過多的非叢集索引會影響資料異動效能
- 非叢集索引只適合資料筆數少的查詢，若資料筆數多，查詢計畫有可能會改為掃描造成查詢效率變低

測試一

- 將 UserInfo 資料表中的所有索引與主鍵都刪除
- 執行下列指令看查詢計畫

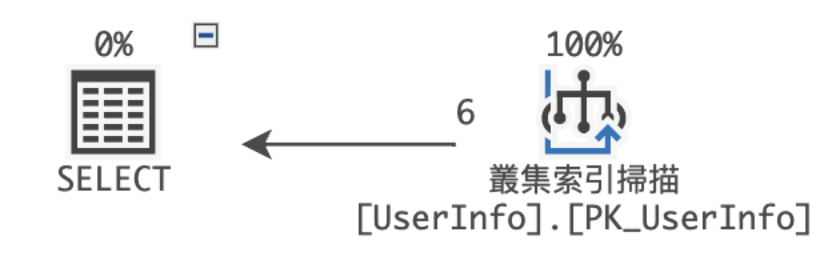
```
select * from UserInfo
```



測試二

- 將 UserInfo 的 uid 設定為主索引，並且設定為叢集索引
- 執行下列指令

```
select * from UserInfo
```



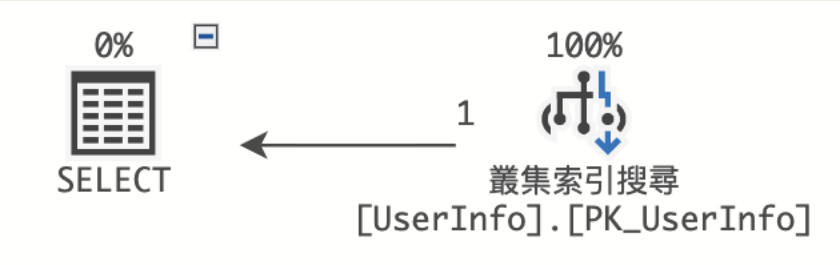
- 雖然使用了索引，但還是「掃描」，一樣慢

使用了索引

測試三

- 執行下列指令，讓搜尋條件包含了叢集索引

```
select * from UserInfo where uid = 'A03'
```

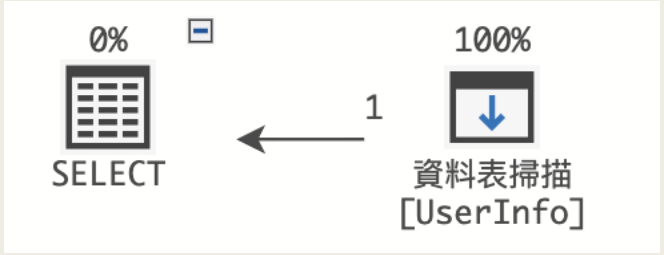


- 現在使用了索引搜尋，此為二位元搜尋，效率極高

索引包含 - 測試一

- 在非叢集索引中 include 其他非索引欄位，提高資料查詢速度
- 移除 UserInfo 所有索引與主鍵，設定 uid 為非叢集索引。當查詢結果需要顯示 cname 欄位資料時

```
select * from UserInfo where uid = 'A01'
```



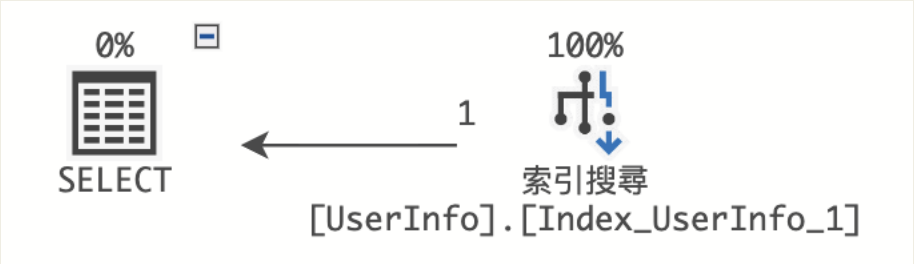
- 以為用到了索引，事實上則沒有

索引包含 - 測試二

- 應將 cname 欄位包含至 uid 索引中，注意不是建立 uid 與 cname 的複合欄位索引
- 執行下列指令

```
select * from UserInfo where uid = 'A01'
```

- 此時就會變成索引搜尋，速度極快

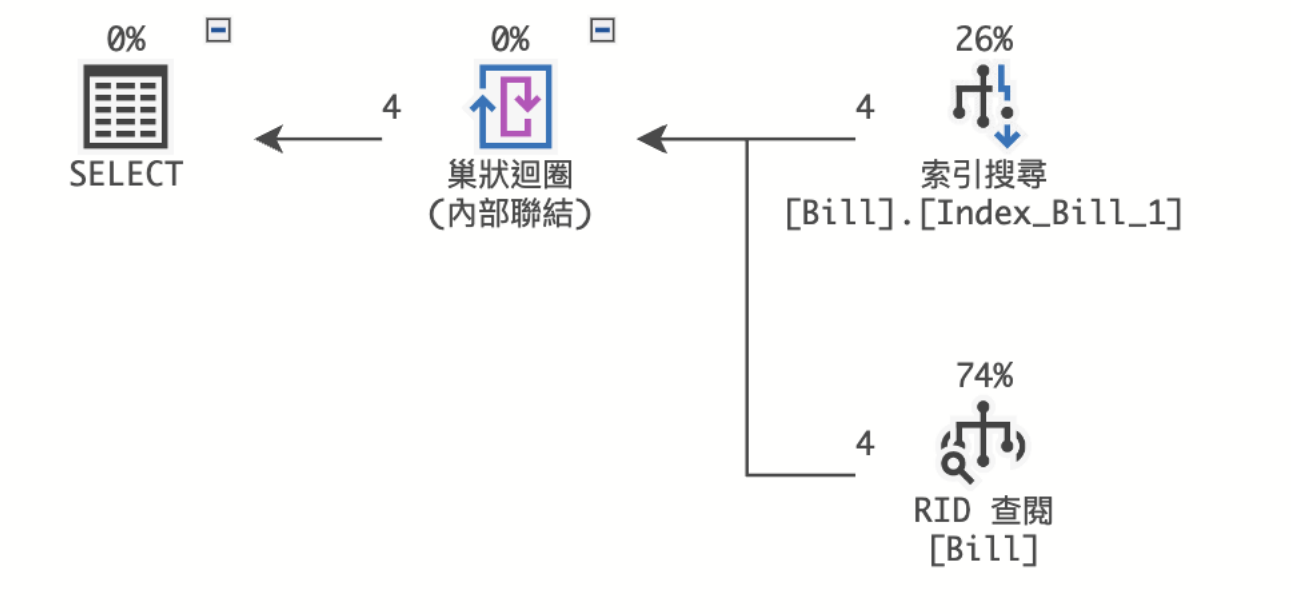


RID查閱

```
insert into Bill (tel, fee, dd)
values ('1111', 400, getdate() + rand())
go 3000
```

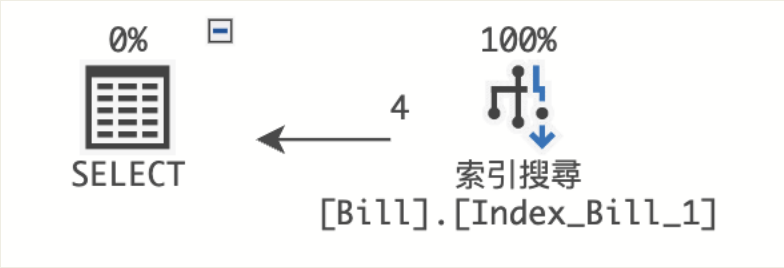
- 移除 Bill 資料表 PK 與所有索引並新增 fee 為索引
- 任意新增三千筆資料到 Bill 資料表
- 執行下列查詢

```
select tel, fee, dd from Bill where fee = 700
```



校調 RID 查閱

- 將 tel、dd 欄位包含在 fee 欄位索引，注意，並非複合欄位索引
- 執行同樣查詢指令，RID 查閱不見了



比較有沒有 RID 查閱的效能

- 若 Bill_tmp 資料表的內容與 Bill 一樣，且 fee 索引沒有包含其他欄位

```
select * into Bill_tmp from Bill

CREATE NONCLUSTERED INDEX [Index_Bill_tmp_1]
ON [dbo].[Bill_tmp]([fee] ASC);
```

- 同時對下列指令進行查詢計畫

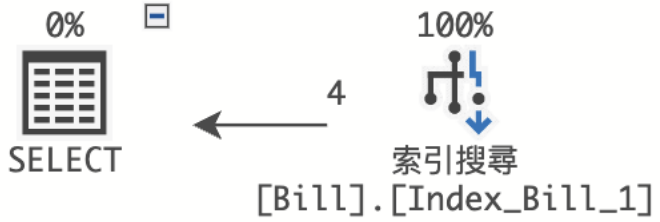
```
select tel, fee, dd from Bill where fee = 700
select tel, fee, dd from Bill_tmp where fee = 700
```

- 結果在下張投影片

結果　訊息　**查詢計劃**　方案樹狀　頂層作業

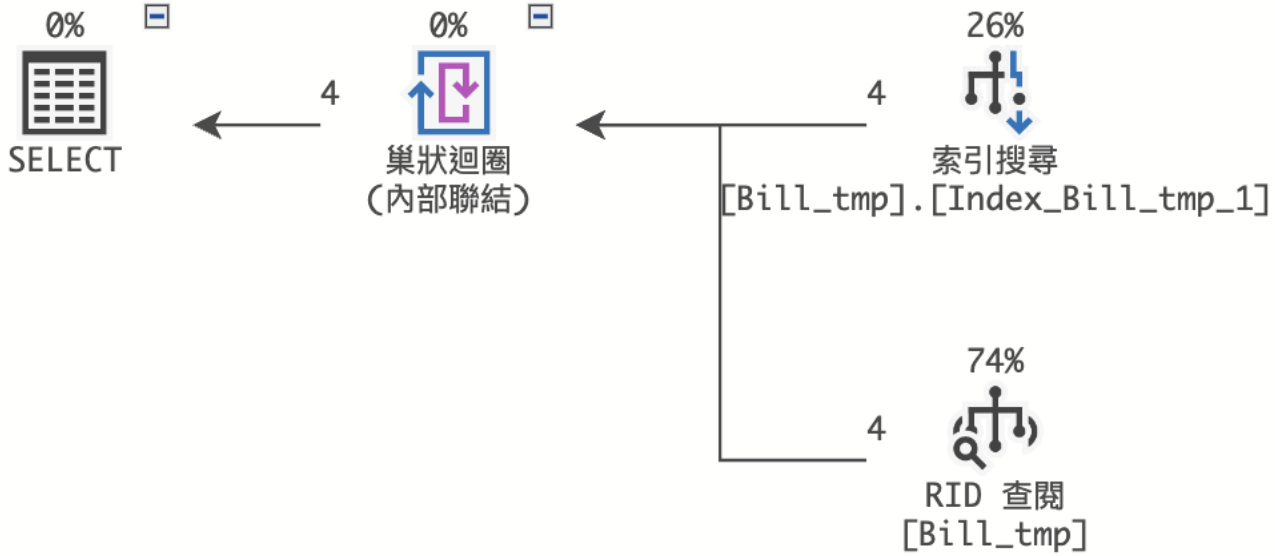
查詢 1: 查詢成本 (相對於指令碼): 20.68%

```
SELECT [tel],[fee],[dd] FROM [Bill] WHERE [fee]=@1
```



查詢 2: 查詢成本 (相對於指令碼): 79.32%

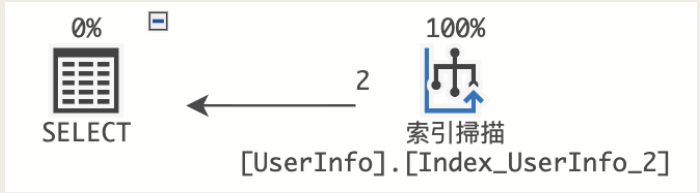
```
SELECT [tel],[fee],[dd] FROM [Bill_tmp] WHERE [fee]=@1
```



查詢條件使用了函數

- 只要是「掃描」就是慢，雖然用到了索引
- 對欄位使用函數或運算時，就會採用掃描，即便該欄位設了索引也一樣

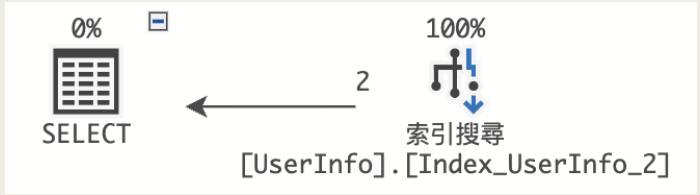
```
select cname from UserInfo where left(cname, 1) = '王'
```



看看實際讀了幾筆資料

- 應改成

```
select cname from UserInfo where cname like '王%'
```



小心使用 LIKE

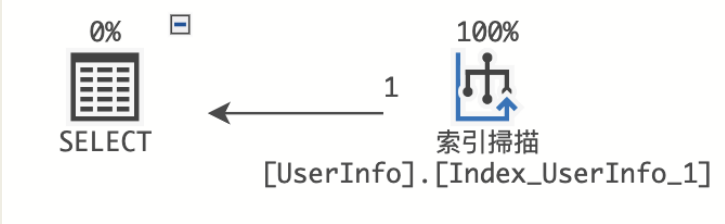
- 使用 LIKE 時，% 放在前面索引就發揮不了作用

```
select cname from UserInfo where cname like '%毛'
```

還有那些字串處理函數

- 子字串處理：LEFT()、RIGHT()、SUBSTRING()
- 頭尾去空白：LTRIM()、RTRIM()、TRIM()
- 大小寫：UPPER()、LOWER()
- 字串相加：CONCAT()
- 小心使用這些函數，使用在查詢條件中的欄位時，就不會使用索引
- 例如 UserInfo 無 PK，且 cname 設定索引

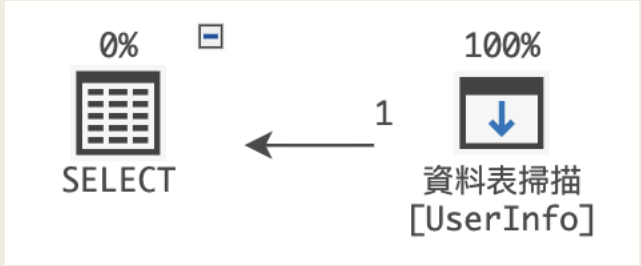
```
select cname from UserInfo where trim(cname) = '王大明'
```



索引不一定都會使用

- 有時建立了索引，但查詢計畫會依據實際狀況，並不一定會使用到索引
- 將 UserInfo 的索引與主鍵刪除，建立 cname 欄位的非叢集索引
- 執行下列指令，感覺上會使用到索引，事實則沒有

```
select * from UserInfo where cname = '王小毛'
```



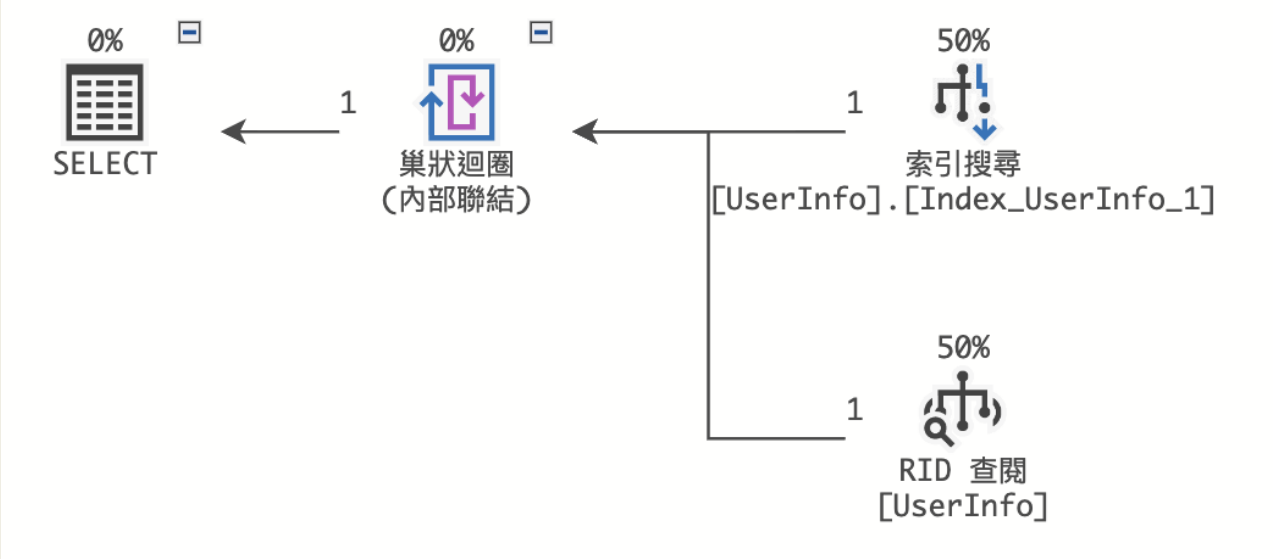
- 原因：因資料太少，查詢計畫覺得此時不用索引比用索引快
- 當資料多的時候（例如六百萬筆資料），就會改為索引搜尋

強制使用索引

- 有時需要改變最佳化工具挑選到的索引，可以使用 with 語句來更換索引，或強制使用索引

```
select *
from UserInfo with(index(Index_UserInfo_1))
where cname = '王小毛'
```

- 若使用索引要做更多的事情
- with (index(0)) 表示不使用索引



複合欄位索引

稱為主要欄位

- 當索引包含兩個以上欄位時，查詢條件須含索引中「第一個欄位」才會使用到索引
- 例如一個資料表有三個欄位：a、b、c，設定這三個欄位的複合欄位索引。每個欄位的排序分別是：a(+)、b(+)、c(-)
- 當查詢條件有 a 欄位時，就會用到索引，例如

```
select * from NewTable where a = 20                  ✅
select * from NewTable where a = 20 and c = 10       ✅
select * from NewTable where b = 20 or c = 10        ❌
select * from NewTable where b = 10                  ❌
```

複合欄位索引與排序

- 索引中的欄位排序會跟查詢結果排序有關，例如：每個欄位的排序分別是：a(+)、b(+)、c(-)
- 根據排序列出下表

  a(+)、b(+)、c(-)
  a(+)、b(+)
  a(+)
  a(-)
  a(-)、b(-)
  a(-)、b(-)、c(+)

- 只要 Order By 的欄位符合這些順序中的一項，就會使用索引中的排序，例如

```
select * from NewTable order by a desc, b desc
```
✅

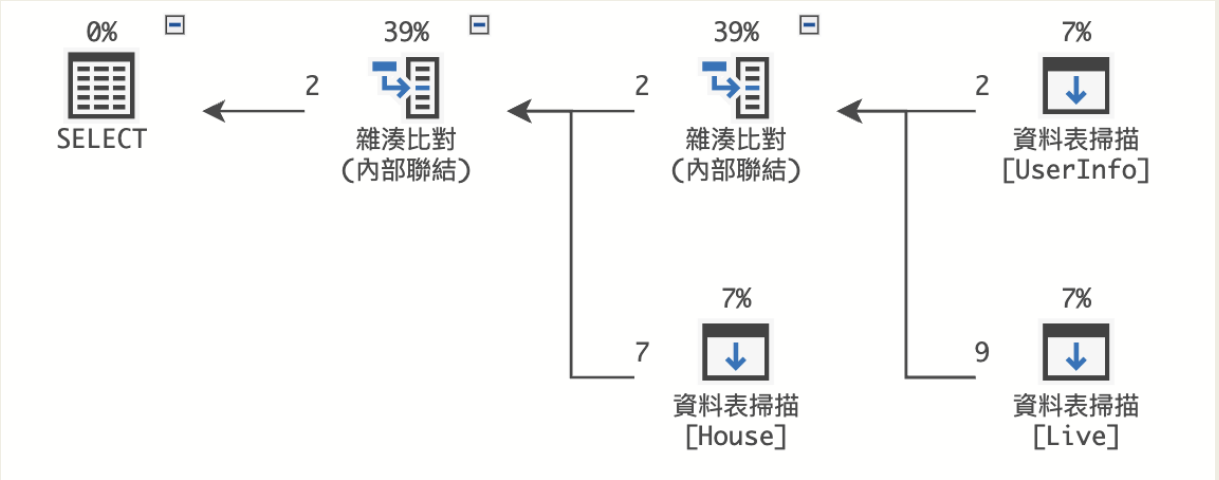
```
select * from NewTable order by a desc, b
```
❌

JOIN – 1/3

- 移除 UserInfo、Live、House 主鍵，並且不設定任何索引

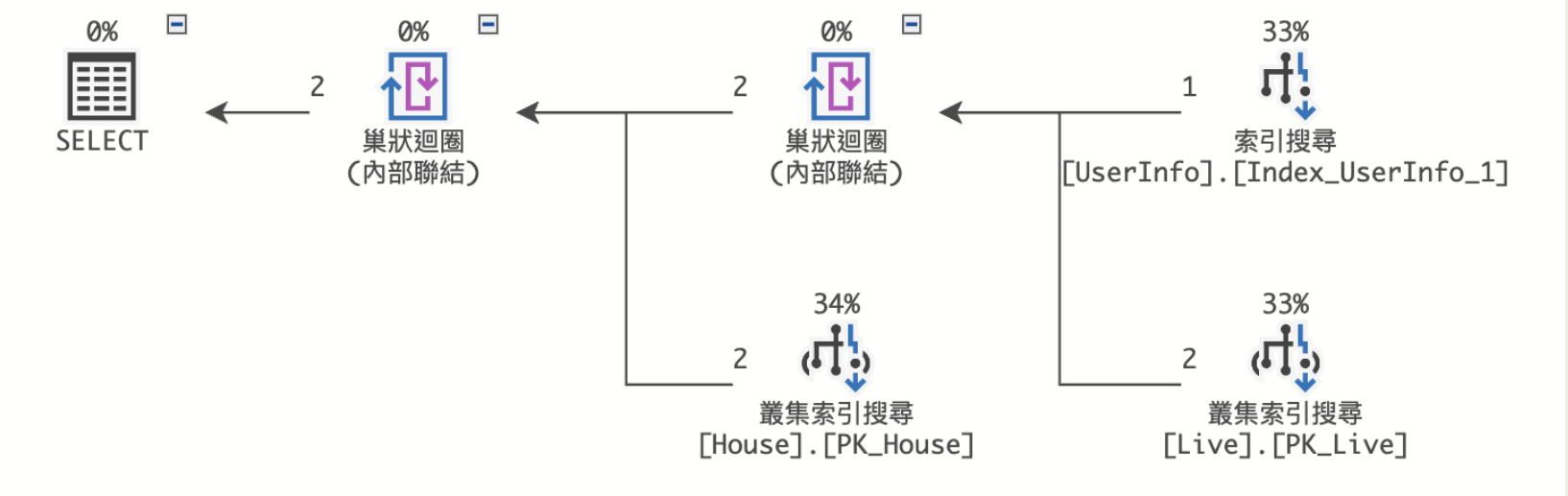
```
select UserInfo.uid, cname, address
from UserInfo, Live, House
where
   UserInfo.uid = Live.uid and Live.hid = House.hid
   and cname = '王大明'
```

- 三個資料表均使用最慢的資料表掃描



JOIN – 2/3

- 設定 PK：UserInfo(uid)、Live(uid + hid)、House(hid)
- 設定 Index：UserInfo(cname)
- 三個資料表都變成快速的搜尋

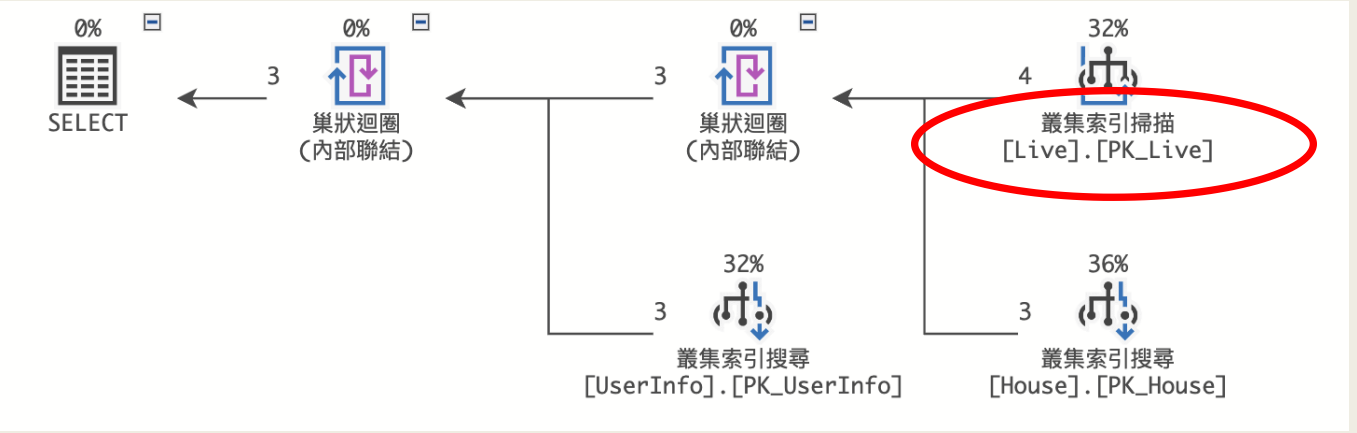


JOIN – 3/3

- House 資料表中的 address 欄位設定 unique index
- 查詢條件改為地址

```
select UserInfo.uid, cname, address
from UserInfo, Live, House
where
   UserInfo.uid = Live.uid and Live.hid = House.hid
   and address = '台北市南京東路1號'
```

- 為什麼 Live 資料表變成掃描？如何校調？

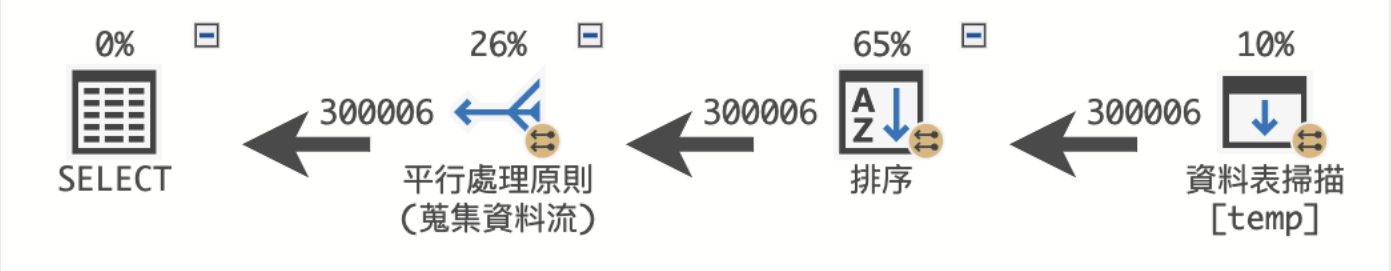


平行處理

- 建立資料

```
select * into temp
go
insert into temp select * from UserInfo
go 50000
```

- 使用多 CPU 與單一 CPU




```
select * from temp order by uid

select * from temp order by uid
option (maxdop 1)
```

限制只使用1顆CPU

索引檢視表（indexed view）

- 能夠建立索引的 View 稱為索引檢視表
- 索引檢視表實際儲存資料，並且與來源資料表資料同步
- 索引檢視表可以更新與刪除資料
- 建立方式如下

```
create view dbo.vw_users with schemabinding
as
select uid, cname
from dbo.UserInfo
where cname <> '' and cname is not null
```

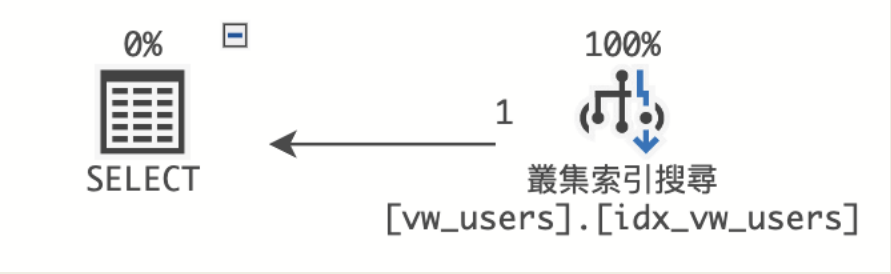
建立索引

- 必須先建立 unique cluster index 才能建立非叢集索引

```
create unique clustered index idx_vw_users
on dbo.vw_users (uid)
```

- 執行下列指令就會用到索引了

```
select * from vw_users where uid = 'A01'
```



新增與索引

- 沒有索引與只有叢集索引的兩資料表，在新增資料時兩者效能完全一樣
- 當資料表的非叢集索引越多，新增資料時效能就越低

更新與索引

- 更新指令具有 where 條件，這裡會受到索引影響
- 更新指令所更新的欄位會影響與之有關的索引，並不會影響所有索引

刪除與索引

- 刪除指令會影響所有索引，並且 where 部分會受到索引而影響效能
- 刪除所有資料時，使用 truncate 指令，效能遠高於 delete
- truncate 刪除資料後，不會維護相關索引，但 delete 會

查詢索引破碎狀況

```
SELECT
   OBJECT_NAME(dt.object_id),
   si.name,
   dt.avg_fragmentation_in_percent,
   dt.avg_page_space_used_in_percent
FROM (
   SELECT
      object_id,
      index_id,
      avg_fragmentation_in_percent,
      avg_page_space_used_in_percent
   FROM sys.dm_db_index_physical_stats (DB_ID(), NULL, NULL, NULL, 'DETAILED')
   WHERE   index_id <> 0

) AS dt  INNER JOIN sys.indexes si
   ON si.object_id = dt.object_id AND si.index_id  = dt.index_id
```

重建與重組時機

官方並未對此值
有任何建議

- 重組
  - avg_fragmentation_in_percent 的值 between 10 and 15
  - avg_page_space_used_in_percent 的值 between 60 and 75
  - *指令：alter index index_name on TableName REORGANIZE*

改 all 表示所有索引

- 重建
  - avg_fragmentation_in_percent 的值 > 15
  - avg_page_space_used_in_percent 的值 < 60
  - *指令： alter index index_name on TableName REBUILD*