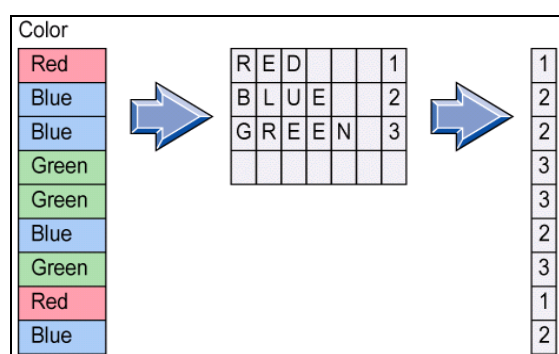


## IQ Index 的建立技巧

雖然標題提到『技巧』二字，但請讀者不要誤會這是篇介紹 IQ 有什麼隱藏『密技』的秘笈。其實關於如何在 IQ 內建立 Index、指令怎麼打、有哪些重要參數，在 Sybase 官方的 Administration Guide 裡已經介紹的很詳細，不是這篇文章的重點；本文所要討論的，主要是在 IQ 所提供的好幾種索引之間，到底差別何在，使用者該如何應用。

### IQ 的特殊設計架構

如果對 IQ 特殊的設計架構有所了解的話，我們會發現其實在很多狀況下，不用建立什麼特殊的索引就已經可以得到比傳統 RDBMS 還好的效能。



上圖即為 IQ 特殊的『以欄位為主』的資料儲存架構。當其他傳統型 RDBMS 必須替 Color 這個欄位定義出像 varchar(20)之類的長度時，IQ 卻只需要透過一張 Lookup Table，外帶一個 Byte 的實際資料儲存區就夠了。以這麼一道簡單的指令為例：

```
select color from products where color='Blue'
```

products 這個資料表如果有五十個欄位，在傳統式的 RDBMS 裡，不會因為這道指令中只查詢了 color 一個欄位而減少任何 I/O，資料庫依然會先把整筆資料都讀取出來，再萃取出使用者想要的部份。因此隨著欄位愈多、資料筆數愈多，這麼簡簡單單的一句語法，就可以讓舊式 RDBMS 感受到效能衝擊。

但 IQ 就比較沒有這個困擾，他會先透過 Lookup Table 找出其對應的數值(2)，然後就只針對此欄去搜尋所有數值為 2 的資料即可。因此而得到的效益有：最節省的儲存空間(1 Byte)、最節省的 I/O 次數(因無須讀取整筆資料，只針對單一欄位)、以及較少的查詢時間(雖然是文字資料，但以數值方式被儲存及查詢)。使得在沒有建立額外索引的情況下，IQ 的效能就已經超出傳統資料庫系統甚多。

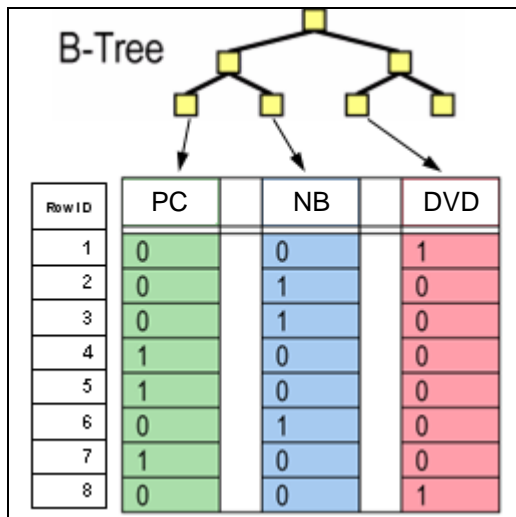
### 三大基本索引

即使 IQ 這種以欄位為主的儲存架構存在著上述的優點，當資料量龐大的時候，依然會出現速度上的一些瓶頸；若再考慮到同時還要進行諸如 `group by` 或 `join` 等動作，則適度的選用最恰當的索引類型就成了 DBA 最重要的工作。這裡 IQ 準備了三種不同類型的基本索引供使用者選擇，分別是：High Group(HG)、Low Fast(LF)、及 Non High Group(NHG)。

考慮這一道 SQL Statement：

```
select product_name from orders where product_name='PC'
```

雖然 IQ 已經可以只針對 `product_name` 這一欄做資料存取，而不必讀入完整的整筆資料，速度有所提升，但當資料量大時，IQ 還是要做『Full Column Scan』(不是 Full Table Scan 喔)，在沒有索引的情況下，仍是件很耗時的工作。但此時應該選用三大基本索引中的哪種較適合呢？假設該公司的產品一共有 500 多種，那我們會建議使用者針對 `product_name` 這一欄建立 LF Index。



LF Index 是一個以 B-Tree + Bit Map 為主要架構的索引，通常適用於欄位 Unique 值(不是資料筆數喔)小於 1500 的情況。比較起傳統 RDBMS 僅以 B-Tree 方式來建構索引，IQ 額外結合了 Bit Map 的機制，因而使得此索引的效能更佳。使用者可在有 `join`、`group by` 需求的欄位、或經常用於 `ad-hoc` 查詢中的欄位上建立此索引，當可大大增強指令的執行速率。

再考慮另一個不同的 Statement：

```
select order_number from orders where order_number >= 16000
```

此例中所使用的訂單資料表(`orders`)約有 20000 多筆資料，而 `order_number`(訂單編號)為其主鍵，此時針對此欄我們該使用哪一種 Index 呢？很明顯的不能使用 LF，因為 `order_number` 的 Unique 值高達 20000，已超過 LF 的適用範圍。其實這時使用者反而不必傷腦筋去評估或設

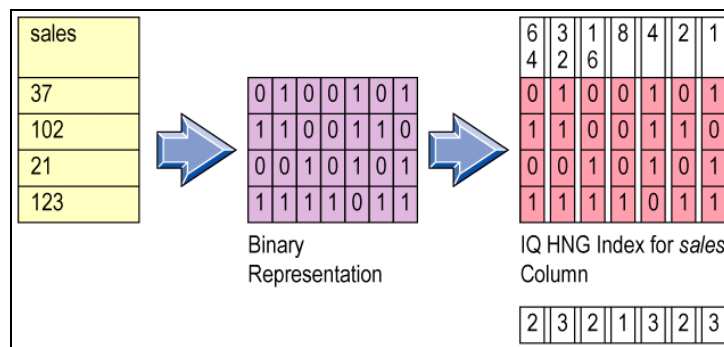
計該用哪種索引，因為針對像是 Primary Key、Foreign Key 等欄位，IQ 會自動幫他建立起 HG 的索引。

HG Index 與 LF 一樣，是一種 B-Tree + Bit Map 架構的索引，但不同於 LF 的是，它適用於更高的 Unique 值(1500 以上)，而且可以結合多個欄位共組 Index，但也因此使得它在實作的細節上比 LF 複雜，會占用更多的儲存空間，且日後因資料異動所導致的索引變動也會耗費更多資源。這些都可算是 HG 的相對缺點。

如果不想耗費那麼多的資源去維護一個 HG Index，使用者也許可以考慮使用 NHG。以下列指令為例：

```
select sum(sales) from orders where sales>=100
```

這道查詢指令想要針對訂單資料表內所有銷售數量(sales)做加總。由於目標欄位 sales 擁有超過 1500 個 Unique 值，因此首先排除了 LF 的適用性；接著由於此欄幾乎不會與其他資料表產生 join 的關係、而且也很少拿來當作 group by 的基準，因此可以不考慮非常耗資源的 HG，而改用 NHG。



上圖顯示出了 NHG 索引的實體架構。讀者們可以看到這是一種不儲存數值，而改存數值之二進位各個冪值的一種索引。這種索引除了跟 HG 一樣可適用於較高的 Unique 值(1500 以上)以外，還特別適合用在 Range Search、或與 Aggregate Function 結合，但不適用於需要做 group by 或 join 的場景。以範例中的指令來說，為了求得 sum(sales)，IQ 僅需做如下運算即可：

$$(2 * 64) + (3 * 32) + (2 * 16) + (1 * 8) + (3 * 4) + (2 * 2) + (3 * 1) = 283$$

總結下來，使用者將來在判斷某個欄位究竟適用 LF、HG、及 NHG 中的哪種索引時，可依據下列經驗法則做選擇的依據：

- 若該欄位是 Primary Key 或 Foreign Key，則不必選擇，IQ 會自動建立 HG。
- 若 Unique 值小於 1500，則優先考慮 LF。
- 當 Unique 值大於 1500，且此欄位經常用於 group by 與 join 的運算，則優先考慮 HG。

- 當 Unique 值大於 1500，但此欄位不會用在 group by 或 join 中的話，則可選擇 NHG。

### 其他特殊類型的索引

考慮下列 SQL 指令：

```
select purchase_price-list_cost from products where  
purchase_price>list_cost
```

這道指令嘗試著在實際進貨價格(purchase\_price)大於表列成本(list\_cost)時，列出其價差。如果依照上面提過的索引建立技巧，使用者可能會選擇在 list\_cost 與 purchase\_price 欄位上分別建立 NHG Index(高於 1500 的 Unique 值、無 group by 或 join 的需求)。但考慮到 IQ 有一種特別為『比對』這個目的而設計的特殊索引：Compare(CMP)，使用者當優先使用此索引為佳。

purchase_price	<	=	>	list_cost
15.55	0	1	0	15.55
39.95	1	0	0	49.99
5.00	0	0	1	4.50
63.50	1	0	0	79.95

上圖是 CMP Index 的架構圖。讀者可以明顯看出，這也不是一種儲存數值的索引，而是一種儲存兩個欄位間的『比對結果』的索引，因此特別適用於判斷條件中使用了『>』、『=』或『<』的情況。

再考慮這麼一道指令：

```
select customer_name from customers where customer_address  
contains 'Taipei'
```

很明顯的，這道指令嘗試著將住址中帶有'Taipei'字眼的所有客戶名稱都列出來。由於像住址這一類的欄位，通常包含一個很長的字串，不論用 LF、HG 或 NHG 都不容易得到最佳搜尋或比對的效果，因此 IQ 另外設計了 Word(WD)這一類型的索引，專門用來處理字串搜尋。通常 WD 最適合與 contains 合用，不過當然使用在 like 之中也可以。

最後我們再假設一個狀況：

```
select * from orders where order_date>='2009/8/1'
```

雖然通常日期的顯示方式看起來像是一個字串，但實際儲存的時候並非文字，因此日期欄位上通常還是適用之前介紹過的 LF、HG 與 NHG，且選擇的準則也相同。

但若是像這樣的一道指令，IQ 就會有不同的做法：

```
select * from orders where datediff(day,order_date,getdate())<=30
```

這道指令使用了 datediff 函數，嘗試著將所有最近 30 天內的新訂單資料讀取出來，此時 IQ 會建議使用者盡量在 order\_date 這個欄位上建立 DATE、TIME 或 DTTM 的索引。

這其中 DATE Index 適用於 date 資料型別的欄位，TIME 則是用於 time 型別，至於 DTTM 則是給 datetime 型別或 timestamp 使用的。通常這三種索引會搭配日期或時間函數一起使用，若查詢中用到了日期欄位，但並未使用日期時間函數，則此類索引的效益就不顯著了。

### 結論

- 資料筆數少的時候，以 IQ 這種 Column-Base 的儲存架構來說，無須額外索引即可得到最佳效果。
- 資料量大時，若欄位 Unique 值少於 1500，優先選擇 LF Index。
- 資料量大時，若欄位 Unique 值大於 1500、且常進行 group by 與 join 運算，優先選擇 HG Index。
- 資料量大時，若欄位 Unique 值大於 1500、但不會進行 group by 與 join 運算，則可考慮 NHG Index。
- 即使欄位上已經有 LF 或 HG 等索引，但若該欄位經常用於 Range Search 或結合 Aggregate Function 作運算時，則可建立 HNG Index。
- 即使欄位上已經有 LF、HG 或 NHG 等索引，但若該欄位經常進行字串搜尋獲比對等動作，則可建立 WD Index。
- 即使欄位上已經有 LF、HG 或 NHG 等索引，但若該欄位經常透過日期與時間函數進行日期的一些搜尋及比對，則可建立 DATE、TIME、或 DTTM Index。
- 即使欄位上已經有 LF、HG 或 NHG 等索引，但若某兩欄位之間的比對結果經常被用來當作條件判斷，則可在此兩欄位上建立 CMP Index。