

## How to tune Sybase SQL statements

一般我們在作 SQL statement tuning 時會注意幾個方面：

1. 所執行的 SQL 跑的並不如預期的快。
2. 所建的 index，在執行時並未使用到。
3. 所建的 index，在執行時有使用到 index，但不是所預期的 index 被使用到。
4. 因大量新增或刪除 Table 資料，造成查詢 Table 速度不如往昔。

以下針對幾個查詢時需要注意地方作介紹：

- 查詢條件(Search Arguments)
- OR 條件
- 運算處理(Aggregate Processing)
- ORDER BY
- INDEX 選擇

### 1. 查詢條件(Search Arguments)

SARGs(Search Arguments)簡單的說就是在 Query 中 where 子句所陳述的條件式，包括欄位及透過>, <, =, >=, <=, is null 等操作符號進行搜尋，我們都知道 Optimizer 通常在取決於 where 子句及 having 子句條件是否有與 index 欄位吻合，但是如果條件欄位有加上運算式或是 function 就無法用到該欄位的 index 了。

例一：

```
Select * from books where price * 3 > 3000
```

例二：

```
Select * from authors where substring(au_lname,1,3)="Ger"
```

而建議寫法是：

例一：

```
Select * from books where price=3000/3
```

例二：

```
Select * from authors where au_lname like "Ger%"
```

另外，還有使用>=會比較只有使用>來的快，主要是因為 Optimizer 會知道在找 index 時從那一個區間開始或結束。

## 2. OR 條件

當同一 Table 一個欄位等於某幾個值，或是不同欄位等於某個值時會出現以下三種其中一種：

- Table Scan：當 Optimizer 發現所有 Index 的 Cost 會比 Table Scan 的 Cost 大時，或是有欄位沒有用到 index 時，則將採用 Full Table Scan。
- 多重 index scan：這種情形最多發生在 in 子句中，其實當 Optimizer 遇到 in 子句時，即會轉化成數個 or 子句
- 特殊 or 出現(dynamic index)

例如：

```
select * from titles_pridtitl
where price < 15 or title like "Assigned%"
```

首先，Optimizer 會對於每一個 or 子句挑出一個 index，再來將結果組合放入 tempdb 暫存的 worktable，最後再將重覆的部份刪除。

另外，還可將 or 改成 union 方式可能會較快，可以比較看看。

例如：

```
select * from authors_idstate where au_id =
    "A1024750512" or state = "WA"
```

改寫成

```
select * from authors_idstate where au_id =
    "A1024750512"
union
select * from authors_idstate where state =
    "WA"
```

### 3. 運算處理(Aggregate Processing)

Optimizer 在處理 queries 運算過程中會有二種；一是會將使用到的 indexes 或 table scan 結果存放在 worktable，例如有作 group by，另一是存放在記憶體變數中，例如取平均值功能 avg(price)。最後才整個取出最後結果。像這種需要運算處理最好能使用到 Index，才可以提供查詢效能。

例一：

```
Select count(*) from titles
```

這是我們最常看到的 SQL 之一，通常 Optimizer 會去找最短的 nonclustered index leaf level 計算出筆數。

例二：

```
Select count(*) from titles group by type
```

如果在 type 欄位上有一個 nonclustered index，那麼 Optimizer 即會掃描該 index leaf level。

例三：

```
Select type, avg(price) from titles group by type
```

同樣在 type 及 price 建一個 nonclustered index，那麼 Optimizer 即會掃描該 index leaf level。

從以上三個例子即表示當 APL Table 有需要作運算處理的最好能建立 nonclustered index，而在 DOL Table 上可建立 clustered index，均可充分使用到 Index。

另外，針對 min()及 max()有幾個須注意：

- 不要將 min 及 max 放在一起，最好分開二個 query 且有 nonclustered index。
- 不要將欄位作運算再取最大值或最小值。
- 不要同時使用 group by。
- 不要在 index 組合鍵中沒有排在第一個順位。
- 使用 max 功能如果有 where 子句是無法使用到最佳化。

#### 4. ORDER BY

當我們使用 order by 子句時，常面臨是否有用到 index，以下二個因素是決定能否使用到 index：

- Order by 子句欄位順序與 index 組合鍵欄位順序必需一致。

例如：

index 組合鍵為 ABC，那麼 order by 以下幾個會使用到該 index

order by A

order by A,B

order by A,B,C

而以下幾個不會使用到 index

Order by A,C

Order by B,C

#### 5. INDEX 選擇

一般來說 Optimizer 會使用到 statistics 資料來決定是否採用 index 以減少 I/O 使用，然而並非採用 index 就是好，Optimizer 會去比較 full table scan 與 index 何者 cost 最少，那麼如何讓 Optimizer 使用到 statistics 呢？

例如：

```
declare @city varchar(25)
```

```
select @city = city from publishers
```

```
where pub_name = "Brave Books"
```

```
select au_name from authors where city = @city
```

當一 stored procedure 執行時無法確切知道 @city 值時，Optimizer 則會採用預設值來達到最佳化，但往往這樣效能上卻打了折扣，所以遇到此種情形即可將 procedure 分開二個來執行：把變數放在第一個 procedure 然後再呼叫第二個 procedure 將查詢結果傳參數給第一個 procedure，這樣第二個 procedure 就可以達到最佳化處理了。

對於查詢來說我們最常看到二種：一是區間查詢(Range query)會回傳很多筆資料，另一種是單點查詢(Point query)回傳只有一二筆資料；在 index 選擇上會有些不同地方，以下作說明：

例如

```
select title
```

```
from titles
```

```
where price between $20.00 and $30.00
```

假設上述 Table 有 100 萬筆資料且採用 allpages locking scheme Table，大約有 135,000 個 pages，而在此一區間(price \$20~\$30)資料約有 19 萬筆資料其 index 影響如下

- A. 無任何 index：full table scan 約 135,000pages

- B. 建 price 爲 clustered index：大約 25,300pages
- C. 建 price 爲 nonclustered index：假設資料可能平均分散在 data page 中，所以該組 index 未被使用，爲求最少 cost 採用 full table scan 所以約 135,000pages
- D. 建 price,title 爲 nonclustered index：約 6,800pages

結論：針對此一 query 在 price, title 上建立 nonclustered index 會是最好的。而特別注意的是此一例子是針對區間查詢，如果採用單點查詢就不是這個結果了。

從以上林林總總相關 Sybase SQL Statement Tuning 中可瞭解良好的 SQL 對於資料庫效能來說是相當重要的，且所占的比重也相當多，不佳的執行效能和過長的反應時間，會讓你的關鍵應用系統變成使用者的夢魘，不管是 Web-based、自行開發、或購買的應用系統，都可能隱含效能上的問題，如此，在正式上線的環境(Production Environment)下的效能問題則可能更趨嚴重。爲了達到最佳的執行效能，並精確的指出問題所在，資料庫管理員須對應用系統及資料庫有所了解，並清楚兩者之間的關聯，同時能夠快速且自動的收集及解決有問題的指令，也是同樣重要；有藉於此如何將此一困難的 Tuning 工作化繁爲簡，有賴良好的 Tuning 工具協助，借助工具可以節省 Tuning 所投入的時間及人力，可達到事半功倍之效。

QUEST CENTRAL for Sybase 之 SQL Tuning 工具結合了強大的 Sybase tuning 環境，非侵入式的問題收集技術、影響分析能力、及專家級的調校建議，提供了一個完全的 SQL 效能解決方案。不管你是在開發階段或已正式上線，SQL Tuning 都能提供影響 Oracle 效能的原因，並引導你如何有效的解決效能問題。以下列出相關產品特色供讀者參考，並歡迎與我們聯繫。

- 可自動重編 SQL 語句，減少編排時間。
- 依具所指定資料庫找出最有效率 SQL 語句，確保最佳的 SQL。
- 可在問題發生之前，協助使用者快速找出不好的 SQL 語句。
- 提供產生虛擬 index 功能，可模擬建立 index 對 SQL 效能影響，且不影響目前正在存取的交易。
- 提供 SQL 語句應建立 index 之建議，且改善之效能是相關的 SQL 語句，而並非只針對單一 SQL。
- 可清楚列出那些 SQL 之執行計畫(Execution plan)已改變，且顯示改變後是提升效能或是降低。
- 提供其他影響 SQL 執行效能之因子，如 statistics 等。
- 在調校 SQL 時並不影響原有之 Table 或是原始編，而是另產生 SQL plan 作比較。
- 可整合 Benchmark Factory 模組，可綜合比較不同平台上效能差異。