第 1 部

# 宛如魔法的 SQL

# 1 CASE 陳述式的建議

**SQL 的條件分歧**

CASE 陳述式是要在 SQL 撰寫條件分歧之際，一定要學會的重要技術，對 CASE 陳述式的熟用度甚至可當成初學者與進階者的度量衡，本章將利用列欄轉換、程式碼體系再分類、與制約的組合、彙總結果這類條件分歧例題學習 CASE 陳述式的使用方法。

## 前言

CASE 陳述式是於 SQL-92 標準內建於標準 SQL。由於已經使用了 20 年以上，所以可在主要的 DBMS 正常使用。雖然 CASE 陳述式很便利，但多數人（尤其是初學者）不了解它真正的價值，或是以 DECODE（Oracle）、IF（MySQL）這類 CASE 陳述式的陽春版代替真正的 CASE 陳述式。一如知名 SQL 大師 Joe Celko 認為「CASE 陳述式或許是 SQL-92 最為有用的功能之一」，能熟悉 CASE 陳述式的使用方法，各位讀者就能在更廣泛的用途應用 SQL，也能將程式碼寫的的更加靈活，尤其是 CASE 陳述式是非規格相依的技術，所以程式碼具有高度相容性，可說是集好處於一身 *1。

只是 CASE 陳述式有些部分的確讓初學者覺得困難，但這是因為必須從有別於一般程式語言的條件分歧 IF 陳述句或 CASE 陳述句的觀點來看待 CASE 陳述式。本章將透過具體實例學習 CASE 陳述式的核心概念、便利的功能。

## [ 導入 ] CASE 陳述式的語法

首先介紹 **CASE 陳述式**的基本語法。CASE 陳述句的格式分成**單純 CASE 陳述式**（simple case）與**搜尋 CASE 陳述式**（searched case）這兩種，分別可寫成下列的格式。

*1 例如 Oracle 的 DECODE 在下列四點劣於 CASE 陳述式。
第一點是 DECODE 為 Oracle 的方言，所以不具相容性。
第二點是分歧條件只限 127 個（參數的上限為 255，但要呈現一個分歧條件需要兩個參數）。但 Oracle 的 CASE 陳述式參數數量上限為 65535，遠高於 DECODE。
第三點是一旦分歧條件增加，程式碼就會變得難以閱讀。
第四點是不方便撰寫。具體來說，無法在參數使用具有述詞的式子，當然也無法在參數使用子查詢。

與「< 數值 >+< 數值 >」是一樣的意思。「< 數值 >+< 數值 >」有時會得出數值或日期的結果，此時運算等於不成立。

### 注意事項 2　別忘了加上 END

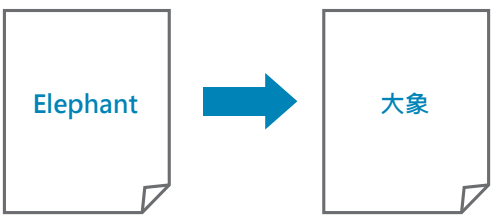
使用 CASE 陳述式時，最常犯的語法錯誤莫過於忘了加上 END。一旦忘了加上必要的 END，就會出現語法錯誤，而且通常會傳回比較簡單易懂的錯誤訊息，所以大多數不會造成太大的問題，不過，通常覺得「明明語法正確，怎麼無法執行時？」通常源自這個疏忽請大家多注意這點。

### 注意事項 3　一定要撰寫 ELSE 陳述句

與剛剛的 END 不同的是，ELSE 陳述句屬於可有可無的部分，不寫也不會造成錯誤，而此時通常會預設為「ELSE NULL」，只不過，「雖然不會造成錯誤，卻會出現不同結果」往往是程式臭蟲的溫床（即使 ELSE 真的是 NULL 也無所謂的情況），所以建議大家養成加註 ELSE 陳述句的習慣，才能明確指示程式產生 NULL，也能減少後續得修正的錯誤。

此外，或許有讀者看到 CASE 陳述式如此單純的使用方法之後，會有「CASE 陳述式終究只是解讀標籤的功能嗎？」的疑問，但實情的確是如此。

■ **圖 1.1　CASE 陳述式的功能就是解讀標籤**




單獨使用 CASE 陳述式時，只能將某欄的值解讀成另外一種值，但若只是這樣，就與規格相依的 IF 或 DECODE 這類函數沒有太大區別。CASE 陳述式必須與其他的 SQL 工具搭配才能發揮真正價值，尤其是與彙總函數（例如 SUM 或 AVG）以及 GROUP BY 陳述句一起使用，才能發揮絕大的力量。之後我們將會透過幾個實際的例子了解 CASE 陳述式的核心價值。

一如程式告訴我們的，這程式是在執行「是 a 就換成 b，是 b 就換成 a」的 UPDATE 條件分歧處理。除了主 Key 之外，特殊 Key 也能進行替換，重點也與剛剛的加減薪範例一樣，換言之，就是利用 CASE 陳述式「一口氣」完成更新，避免主 Key 造成的錯誤 [*5]。

不過，會需要對調資料，很有可能是資料表的設計出問題，所以請先檢視一下設計，拿掉不必要的限制。

## 資料表的比對

CASE 陳述式相對於 DECODE 函數的一大優勢在於能評估公式，換言之，CASE 陳述式具有 BETWEEN、LIKE、<、> 這類方便的述詞群可以使用，而且 IN 與 EXISTS 這類述詞還能以子查詢為參數，所以用途也非常廣泛。

接著，讓我們以證照補習班的課程一覽表以及每月課程管理表為例。

■ **課程大師**

**CourseMaster**

| course_id | course_name |
|---|---|
| 1 | 會計入門 |
| 2 | 財務知識 |
| 3 | 簿記檢定課程 |
| 4 | 稅務士 |

**OpenCourses**

| month | course_id |
|---|---|
| 201806 | 1 |
| 201806 | 3 |
| 201806 | 4 |
| 201807 | 4 |
| 201808 | 2 |
| 201808 | 4 |

接著，利用這張資料表製作能一眼看出每月開課情況的交叉分析表。

*5 這個查詢會在 PostgreSQL 或 MySQL 發生主 Key 重復的錯誤，例如 PostgreSQL 會顯示下列的錯誤訊息。

**ERROR：重複 Key 違反一致性限制 "sometable_pkey"**
**DETAIL：Key(p_key)=(b) 已存在**

這是要將主 Key 為 a 的列換成 b 之際，主 Key 的 b 還未變更所產生的錯誤，所謂的限制會在陳述句完成變更之後才出現，所以在執行途中暫時重復也不會有問題，而且這個 UPDATE 陳述句也能於 Oracle、Db2、SQL Server 正常執行。在 PostgreSQL 執行時，只需要加上延遲限制（DEFERRABLE）這個選項，就能正常執行了。

## 練習題

### ●練習題1-① 複數欄位的最大值

SQL 可從多個欄位快速篩選出最大值／最小值，只要於 GROUP BY 陳述句使用適當的 Key 彙整資料，再使用 MAX ／ MIN 函數篩選即可。不過，該怎麼做才能從多個欄位篩出最大值呢？範例資料如下：

**Greatests**

| key | x | y | z |
|---|---|---|---|
| A | 1 | 2 | 3 |
| B | 5 | 5 | 2 |
| C | 4 | 7 | 1 |
| D | 3 | 3 | 8 |

請試著從這張資料表取得 x 與 y 的最大值。下列是結果。

**結果**

```
key    greatest
-----  ---------
A             2
B             5
C             7
D             3
```

Oracle、PostgreSQL、MySQL 雖然內建了應付這種情況的 GREATEST 函數，但希望大家利用標準 SQL 篩選看看。

如果能順利解開此題，建議大家試著將欄位資料增加至三欄以上。這次是改從 x、y、z 篩選最大值，所以可得到下列的結果。

**結果**

```
key    greatest
-----  ---------
A             3
B             5
C             7
D             8
```

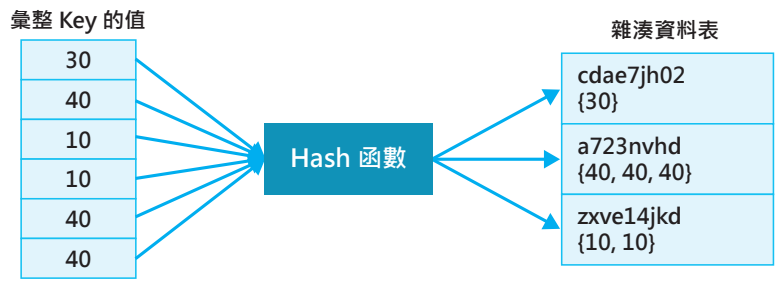
## 雜湊與排序

另一方面，也有人對視窗函數以排序寫成，是否真的最有效率這點提出異議。p.29 註解介紹的論文認為將 PARTITION BY 陳述句成雜湊計算，能提升效能，也提出了實測的結果。

> 這是因為假設輸入列數為 n，而分割數為 O(n)，此時雜湊將會是 O(n)，排序最多只能是 O(n log n)。
>
> — p.1062, 4.2 Determining the Window Frame Bounds [*10]

雜湊函數具有輸入值不同，輸出不同輸出值（值不會重複）的特性，此時的輸出值稱為「雜湊值」，大致上就是「30」→「cdae7jh02」這種轉換（圖 2.3）。記載成對的輸入值與雜湊值的是「雜湊資料表」，若使用這種資料表分組，就能跳過排序，直接彙整資料（不轉換成雜湊值也能分組，但雜湊值不用擔心欄數與資料類型之餘，還能當成各種函數的輸入值使用）。

■ 圖 2.3 雜湊的分組示意圖




*10 下列是筆者譯文的原文。
Efficient Processing of Window Functions in Analytical SQL Queries
http://www.vldb.org/pvldb/vol8/p1058-leis.pdf

## 刪除重複列

在關聯式資料庫的世界裡，重複列與 NULL 差不多令人討厭[*3]，所以也有許多排除重複列的方法因此問世。讓我們以剛剛例題的商品資料表為例，試著排除重複出現的「橘子」。最可怕的是，這張資料表沒有設定主 Key（應該說是無法設定），所以這類資料表有必要立刻進行「大掃除」。

■ 出現重複列的資料表

| name（商品名稱） | price（價格） |
|---|---|
| 蘋果 | 50 |
| 橘子 | 100 |
| 橘子 | 100 |
| 橘子 | 100 |
| 香蕉 | 80 |

重複

■ 刪從重複列之後的資料表

| name（商品名稱） | price（價格） |
|---|---|
| 蘋果 | 50 |
| 橘子 | 100 |
| 香蕉 | 80 |

接下來為大家介紹以自我關聯子查詢刪除重複列的方法。連結與關聯子查詢的運算方式雖然不同，但邏輯卻很類似，也有很多情況能等值轉換 SQL，所以在此一併介紹。

不管重複列只有 2 列還是有很多列，只要不包含主 Key，就能使用主 Key 排除，但像例題這種全欄位重複的情況，就必須使用規格依存的記錄 ID。請把記錄 ID 想成是擁有「任何資料表都可使用的主 Key」這類特徵的虛擬欄位，這次使用的是 Oracle 的 rowid[*4]。

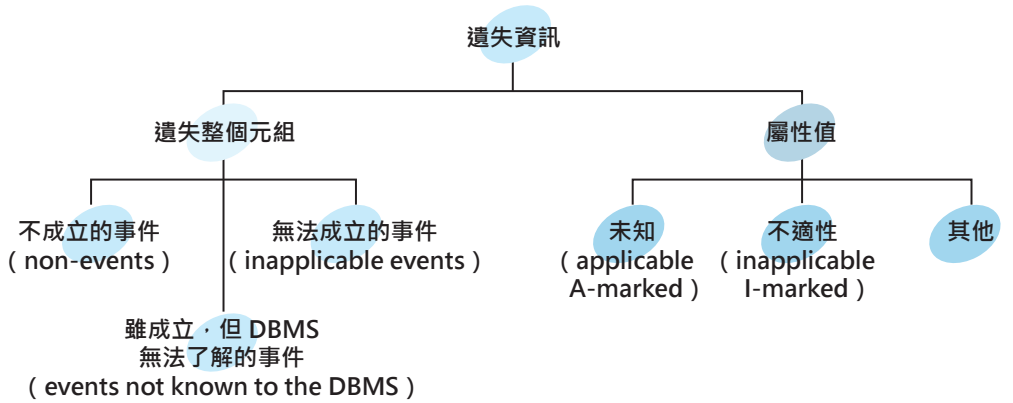
```
-- 刪除重複列的 SQL　之一：使用極值函數
DELETE FROM Products P1
 WHERE rowid < ( SELECT MAX(P2.rowid)
                   FROM Products P2
                  WHERE P1.name = P2.name
                    AND P1.price = P2.price );
```

乍見之下，這是執行過程難以理解的關聯子查詢，這也難怪，因為明明是基於描述兩張資

*3 允許重複列出現為何會導致資料表設計不良，請恕本書無法在此說明，詳情請參考 C.J.Date 的《データベース実践講義》（O'Reilly Japan，2006）的「3.5 禁止元組重複的理由」。

*4 能夠建立供使用者使用的記錄 ID 只有 Oracle（rowid）與 PostgreSQL（oid）。要在 PostgreSQL 使用 oid，必須事先在 CREATE TABLE 陳述文加上「WITH OIDS」選項。於其他 DBMS 刪除重複列的方法請參考練習題 3- ②。

■ 圖 4.1 關聯式資料庫的遺失資訊分類




Codd 過去曾提倡將 NULL 嚴謹分成兩種的四元邏輯 *2，但不知道是幸運還是不幸（筆者肯定是覺得幸運），這個點子並未得到普遍支持，目前所有的 DBMS 都採用只有一種 NULL 的三元邏輯。不過，這種分類實在很了不起，導致後續有許多學者沿用。

## 為什麼不寫成「=NULL」，必須寫成「IS NULL」？

想必不少人有此疑問，很多人在剛開始學習 SQL 時遇過，想以下列的查詢選擇某欄的 NULL 列，結果無法正確選取的經驗。

```
-- 無法正確偵測 NULL 的 SQL
SELECT *
  FROM tbl_A
 WHERE col_1 = NULL;
```

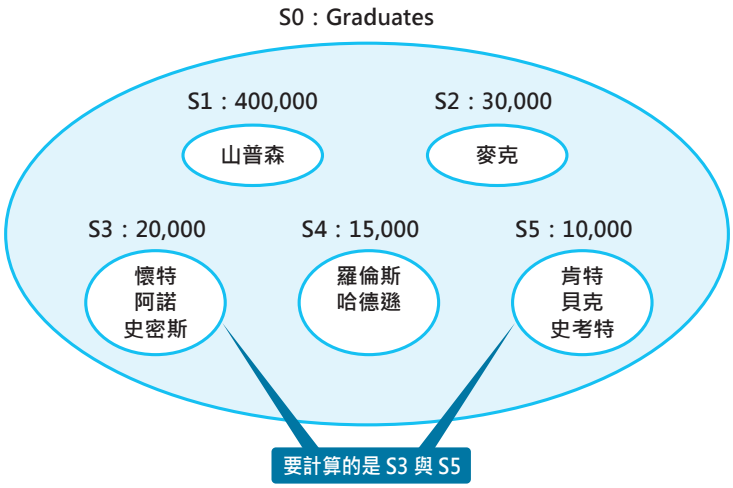
*2 筆者得知 Codd 曾提倡四元邏輯時，的確覺得很不可置信，也覺得沒有半個資料庫開發者對四元邏輯感到興趣是一件好事。下列介紹四元邏輯的真理表給大家參考。為 UNKNOWN 的 NULL 設計的布林值為 applicable，為 Not Applicable 的 NULL 設計的布林值為 inapplicable。這些表格終究只是參考，就算是偉大的 Codd 博士所提倡的事，我也無法接受這種四元邏輯。

| x | NOT x |
|---|---|
| t | f |
| a | a |
| i | i |
| f | t |

| AND | t | a | i | f |
|---|---|---|---|---|
| t | t | a | i | f |
| a | a | a | i | f |
| i | i | i | i | f |
| f | f | f | f | f |

| OR | t | a | i | f |
|---|---|---|---|---|
| t | t | t | t | t |
| a | t | a | a | a |
| i | t | a | i | f |
| f | t | a | f | f |

■ 圖 6.2 以收入（income）為 Key 的五個小集合




其中元素數最多的為 S3 與 S5，都擁有 3 個元素，因此，這兩個集合也被篩選出來。

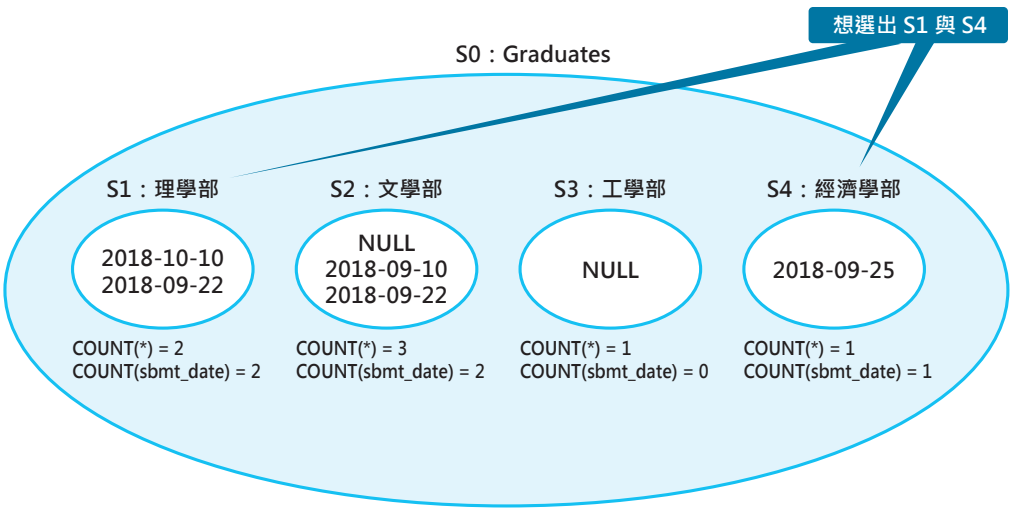
此外，一如「4　三元邏輯運算與 NULL」所述，若擔心使用 ALL 述詞時出現 NULL 或空集合，可利用極值函數代替 ALL 述詞。這次要計算的是「最多」，所以可改用 MAX 函數。

```
-- 計算眾數的 SQL　其二：使用極值函數
SELECT income, COUNT(*) AS cnt
  FROM Graduates
 GROUP BY income
HAVING COUNT(*) >= ( SELECT MAX(cnt)
                       FROM ( SELECT COUNT(*) AS cnt
                                FROM Graduates
                               GROUP BY income) TMP );
```

假設「Graduates」資料表為檔案，要利用程序語言計算眾值的話，又會怎麼做呢？恐怕會先以收入排序，接著，再以迴圈比較每一行，執行 Control Break，若相同收入的列數大於前面計算過的收入的列數，就將該收入存進變數。不過，一如所見，SQL 根本沒用到迴圈，也沒有將結果代入變數的部分。

學生一交報告，繳交日期欄位就輸入日期。在還未繳交之前，都為 NULL。接著，讓我們從這張資料表選出所屬學生已全數繳交報告的學部（理學部、經濟學部）。若單純以「WHERE sbmt_date IS NOT NULL」條件選取，就會連不該選的文學部都選到（文學部的 102 號學生還未繳交）。就邏輯而言，可先將學部當成 Key，再利用 GROUP BY 陳述句建立如圖 6.3 的部分集合。

■ **圖 6.3 所有學生都繳交報告的是哪個學部？**



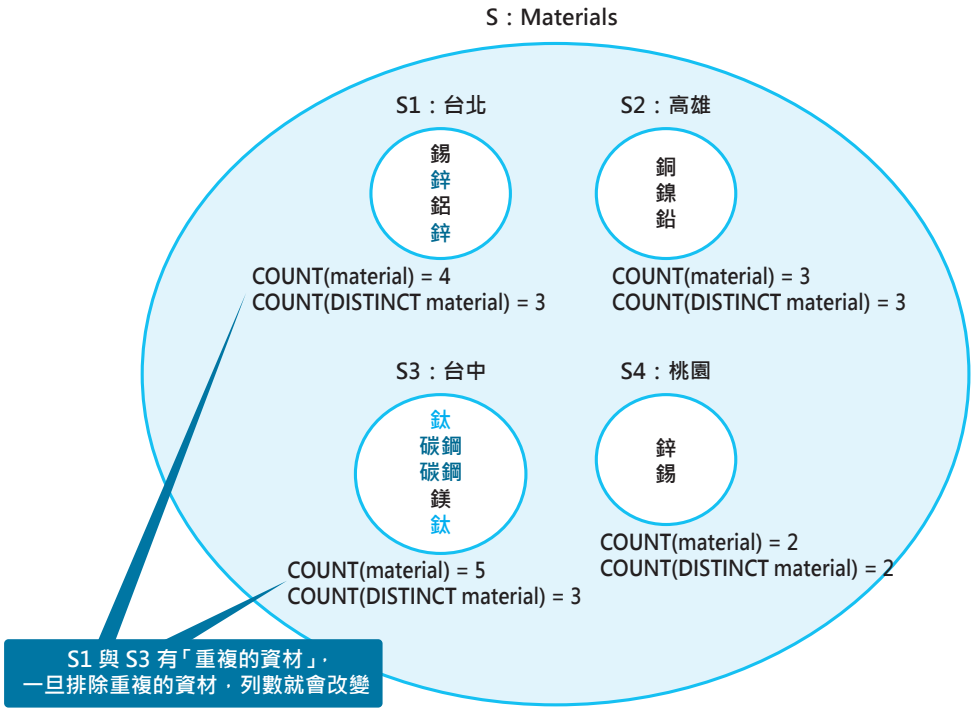

這次要從這四個部分集合選出的是 S1 與 S4。要問的是，這兩個集合共有，但其他集合沒有的性質為何？答案就是「COUNT(*) 與 COUNT(smbt_date) 一致」的性質，這個是因為 S2 與 S3 具有 NULL 才出現的現象，所以答案如下：

```
-- 選出繳交日期不含 NULL 的學部　其 1：使用 COUNT 函數
SELECT dpt
  FROM Students
 GROUP BY dpt
HAVING COUNT(*) = COUNT(sbmt_date);
```

**結果**

```
dpt
--------
理學部
經濟學部
```

■ 圖 6.5 將每個據點分割成集合




這次想要選取的是鋅這項資材重複的台北據點與鈦、碳鋼重複的台中據點。接著，要問，什麼是這兩個集合符合，但其他集合不符合的條件？

恐怕只有「排除重複之後的元素數，與尚未排除重複的元素數不同」這個條件了。只要元素沒有重複，加上 DISTINCT 這個選項再執行 COUNT，應該會得到相同的結果才對。

```
-- 選出資材重複的據點
SELECT center
  FROM Materials
 GROUP BY center
HAVING COUNT(material) <> COUNT(DISTINCT material);
```

**結果**

```
center
------
台北
台中
```

如此一來，只需要對這個「MASTER」視圖執行一次外部連結即可。換言之，若想將表側設計成巢狀結構，可先建立一張這種格式的主要資料表。即使是三層以上的巢狀結構，也只需要以相同的方式擴張。

此外，若是沒有 CROSS JOIN 語法的 DB，可利用「FROM TblAge, TblSex」這種在不指定連結條件的情況下排列資料表，就能得到與交叉連結處理時相同的結果。

## 相當於乘法的連結

大家是否還記得「6 HAVING 陳述句的力量」的專欄「關聯式除法」(p.133)曾介紹過「SQL 的連結相當於乘法」這件事呢？這不是什麼比喻，而是從列數來看，就真的是這樣。

這次要以下列的商品主資料表與商品業績履歷表，進一步探討當成乘法使用這一點。

**Items**

| item_no | item |
|---|---|
| 10 | SD 記憶卡 |
| 20 | CD-R |
| 30 | USB 隨身碟 |
| 40 | DVD |

**SalesHistory**

| sale_date | item_no | quantity |
|---|---|---|
| 2018-10-01 | 10 | 4 |
| 2018-10-01 | 20 | 10 |
| 2018-10-01 | 30 | 3 |
| 2018-10-03 | 10 | 32 |
| 2018-10-03 | 30 | 12 |
| 2018-10-04 | 20 | 22 |
| 2018-10-04 | 30 | 7 |

接著，要使用這兩張資料表調查各商品的總營業額，並且輸出成表格。目標是輸出下列這種格式的表格。

**結果**

```
item_no  total_qty
-------  ---------
     10         36
     20         32
     30         22
     40
```