第三章 系統設計

3-1 分而治之／Divide and Conquer

通常使用者點選一個動態網頁的連結後所需等待時間的長短，除了因為網路頻寬不足以短時間傳完所有內容以外，主要是在等待遠端 web server 的處理和回應，包括 PHP 之類的網頁程式執行與資料庫處理查詢或存取資料所花的時間。

在前一章的動態網頁運作圖中，Apache 所擔崗的角色只是中介傳送使用者的需求給 PHP module 處理，再將處理完的結果傳送給使用者接收，由此可知主要的等待時間都是在執行 PHP 程式與資料庫的資料查詢使用。因此本論文就以分而治之的方式，在「執行 PHP 程式」的階段提出一個改善方案，而在「資料庫的資料查詢使用」的階段提出兩個改善方案，且以一線上考試測驗系統為例。

3-2 方案一：除冗碼

PHP 是一種直譯式語言。當一個線上考試系統的一組功能的所有程式與網頁的介面，以及資料庫指令動作處理都寫在同一個 PHP 程式檔案裡面的狀況下，PHP 會把整個檔案的語法從頭到尾都檢查一次，然後再將整個檔案都直譯成 object code 後才真正進行執行的動作，即使這一次程式執行用到的只是這個檔案十分之一的程式碼。

所以從這裡可以先做分而治之的修改，讓 PHP 不用浪費 CPU 資源處理這些不會執行到的程式碼上。

不過，得先測試一下 PHP 的 parser 和 syntax checker 的行為，才能確定是否能用 PHP 內建的函式來進行相關的修改。

首先以下面這段有錯誤的程式碼，測試 PHP 的 parser 會不會跳過不會執行的區間。

```
if ( 0 ) { echo( "test "); }     //    '和  "沒有成對
```

表 2-1：方案一測試用程式碼，第一種。

結果，傳回語法錯誤的錯誤訊息，證明 parser 照樣會檢查不執行的區間。

再來測試 PHP 會不會對不執行的區間中, 指令的正確與否做檢查

```
if ( 0 ) { require( "abc.txt "); } // abc.txt 是不存在的檔案
```

表 2-2：方案一測試用程式碼，第二種。

結果，沒有回應找不到 abc.txt 的錯誤，所以 PHP 並不會在不執行區間中檢查並執行 require()所指定的檔案內容。

所以，以這種方式來做分而治之是可行的。以下是一個程式的修改範例，利用 PHP 的 require()函式改寫先後的程式差異比較。

if (isset($_GET['list_class']))
  { ……; } // N_1 lines code
if (isset($_GET['class_score']))
  { ……; } // N_2 lines code
if (isset($_GET['student_score']))
  { ……; } // N_3 lines code
 	
// display teacher function list;
……;
	if (isset($_GET['list_class']))
	  require("list_class.php ");
	if (isset($_GET['class_score']))
	  require("class_score.php ");
	if (isset($_GET['student_score']))
	  require("student_score.php ");
	// only 1 line require() in each block
	// call module to show teacher function
	show_teacher_function_list();

表 2-3：範例程式以方案一修改前後的差異比較。

3-3 方案二：減少 JOIN

在 relation database 的觀念下，資料的儲存並不是用單獨一個巨大的資料表儲存了所有的資料，在每一筆資料欄位中時常重複的部份就可以用 1NF-First Normal Form[26]來做資料的分散處理以節省空間。

可是只要用了 1NF 將資料分離，那麼在用 SELECT 取資料出來的時候，就一定得用 JOIN 把 1NF 分離出去的各個欄位資料以相對應的 primary key 關聯回來輸出結果[27]。當這種以 1NF 分離的資料欄位有不少的時候，那這個 SQL 指令執行的時間以及所耗用的記憶體空間就會增加許多。

```
SELECT * FROM student
LEFT JOIN school_list ON school_list.school_id=student.school_id
LEFT JOIN teacher_list ON teacher_list.teacher_id=student.teacher_id
WHERE student_id=1 LIMIT 0,1
```

表 3：學生個人資料以 1NF 分出去的部分關聯回來的 SQL 指令。

在這種考量下，就可以將這一個 SQL 指令拆開來分而治之。跟 Object Prefetching[28]類似的想法，把需要 JOIN 的各個以 1NF 產生的資料表都提前先用 SELECT 抓出來，然後以各自的 primary key 當作 PHP array 變數的索引值，放到 PHP 的記憶體空間裡暫存著。然後把 SQL 指令裡 1NF 用的 JOIN 指令都拿掉。在傳回來的資料裡，需要 JOIN 的欄位內容都是各項目 primary key 的數值，這時候再從 PHP array 變數裡直接拿出來套用即可達到同樣的效果。

原本的 SQL 指令執行的複雜度設想是 $O(N*T_1*T_2*\cdots*T_m)$，T_1 T_m是需要被 JOIN 的各資料表內的資料總數，有多少筆資料就得乘上去來產生原始的資料。然而在經過改寫之後，原本的複雜度就可以降低成 $O(N)+O(T_1)+O(T_2)+\cdots+O(T_m)$，跟原本需要運算的次數差異相當大。若有重複使用 T_m資料的話，連 $O(T_n)$這些都可以再減少，故值得用些許的記憶體空間來存放 T_n的內容，以換取更多的 CPU 運算資源。

舉例來說，考試系統中要叫出學生的詳細資料時，通常會去別的資料表中連帶找出學生所屬學校, 所屬班級, 所屬老師…之類的資料。

```
$sqlr = mysql_query( "SELECT * FROM student ".
        "LEFT JOIN school_list ON school_list.school_id=student.school_id ".
        "LEFT JOIN teacher_list ON teacher_list.teacher_id=student.teacher_id ".
        "WHERE student_id=1 LIMIT 0,1 ");
$student_array = mysql_fetch_array( $sqlr );
echo( "School Name: ". $student_array[ 'school_name' ]. "<br>\n ");
echo( "Teacher Name: ". $student_array[ 'teacher_name' ]. "<br>\n ");
```

```
session_register( 'school_list ');
session_register( 'teacher_list ');

function load_school_list() {
   $sqlr = mysql_query( "SELECT * FROM school ");
   for ( $i = 0; $i < mysql_num_rows( $sqlr ); $i++ ) {
      $tmp = mysql_fetch_array( $sqlr );
      $_SESSION[ 'school_list' ][$tmp[ 'school_id' ]] = $tmp;
   }
}

function load_teacher_list() { ……; }  // as load_school_list()

$sqlr = mysql_query( "SELECT * FROM student ".
        "WHERE student_id=1 LIMIT 0,1 ");
$student_array = mysql_fetch_array( $sqlr );
echo( "School Name: ".
    $_SESSION[ 'school_list' ][$student_array[ 'school_id' ]][ 'school_name' ].
    "<br>\n ");
echo( "Teacher Name: ".
    $_SESSION[ 'teacher_list' ][$student_array[ 'teacher_id' ]][ 'teacher_name' ].
    "<br>\n ");
```

表 4：範例程式以方案二修改 SQL 指令與程式的差異比較。

3-4 方案三：降低資料搜尋空間

考試系統中有一個巨大的題庫資料庫 Q_Library 總題數是 N，而且被應用在線上考試的數位教學上。每一次舉辦考試都得依照考卷的範圍條件，從題庫中找出所需的題目後輸出到畫面給學生做答。

在這個情況下，SQL 指令可能是這樣下的：

```
SELECT * FROM Q_Library WHERE q_id IN (1,250,4000,13579,…)
```

表 5：從題庫中選出指定題號的 SQL 指令寫法。

就算 Q_Library 的 q_id 有設定 INDEX 來加速搜尋的效果，每搜尋一個指定的 q_id 所花的時間，以一般常用的 binary search 來設想為 O(logN)[28]，那麼一份有 M 題的線上考卷，所花的搜尋時間就是 M*O(logN)，當這個題庫總題數 N 越大，所耗費的時間也就越久。

所以，在這種情況下就可以先做分而治之的應用。

首先，在建立這一份線上考卷的時候，給定一個考卷編號像是 Math060919E 這樣，順便再建立一個新的小題庫，資料表名稱設定為 Q_Math060919E 來儲存所要使用的題目 N_1題，並且在 Q_Library 中把原本題目的結構中，增加一個欄位儲存使用了題目的小題庫列表名稱，且將 Q_Math060919E 的記錄加進去。當原題庫需要修改題目內容或答案時，除了更新到題庫 Q_Library 以外，連小題庫 Q_Math060919E 裡面的題目也跟著更新。

當一份從 N_1題挑選出 M 題的線上考卷產生時，所花的搜尋時間就只會是 $M * O(\log N_1)$，甚至 $M = N_1$時就不用 WHERE 指定條件，直接將全部的題目都直接 SELECT 出來，時間就是 M * O(1)，在同時使用人數越多的時候，效能更有顯著的提昇。