本章內容：

- 從 Perl 與 SQL 伺服器互動
- 使用 DBI 架構
- 使用 ODBC 架構
- 伺服器文件
- 資料庫登入
- 監控一個SQL伺服器的CPU健康
- 本章模組的資料
- 詳細的參考資料

第七章

SQL 資料庫管理

在一本系統管理的書裡，要談什麼資料庫管理？有三項充分理由讓對 Perl 與系統管理感興趣的人成為資料庫專家：

1. 在本書好幾個章節裡有一項不難發現的趨勢，那就是資料庫在現代系統管理的重要性日趨增加。我們會使用資料庫（即便是很簡單的）來追蹤使用者與機器的資訊，而那只是冰山一角。通信論壇（mailing list）、密碼檔、甚至是 Windows NT/2000 的 registry，都是你可能每天會見到資料庫。所有大型的系統管理套件（例如，由 CA、Tivoli、HP以及 Microsoft 提供的產品），其背後皆有資料庫支撐。如果你打算進行任何嚴謹的系統管理，你終究會進入資料庫的領域。

2. 資料庫管理是系統管理的戲中之戲。資料庫管理者（DBAs）關心的不只是資料庫本身，還必須與其它事項奮鬥：
   - 使用者/登入
   - 日誌檔
   - 儲媒管理（諸如：硬碟空間）

- 程序管理
- 連結性的議題
- 備份
- 安全性

聽起來是否很熟悉？我們能、而且應該從兩種知識領域學習。

3. Perl 是最佳（或許有爭議）的整合語言（glue language）。Perl 與資料庫的整合工作已有相當成效，這大部份要歸功於 Web 發展模式的所帶來的驚人能量，讓我們可以享用這些成果。儘管 Perl 能整合多種不同的資料庫格式，諸如 Unix DBM、Berkeley DB ... 等等，但我在本章只想把注意力放在 Perl 與大型資料庫產品的介面上。其它格式會在本書的其它部分提及。

要當一位熟知資料庫的系統管理者，你至少需要學一點結構化查詢語言（Structured Query Language，SQL），它是絕大多數商用資料庫以及一些非商用資料庫的 " 共通語言 "。使用 Perl 撰寫資料庫管理程式，需要先對 SQL 有些基本的認識，因為這些程式中會含有 SQL 敘述。《附錄 D：十五分鐘 SQL 教學》提供了足夠讓你入門的 SQL 教材。為了一致性，本章範例所用的資料庫，將與先前章節一樣。

7.1 從 Perl 與 SQL 伺服器互動

要與 SQL 伺服器溝通，通常必須遵循兩套標準架構之一，分別是 DataBase Interface（DBI）以及 Open DataBase Connectivity（ODBC）。在過去曾有一段時間，DBI 是 Unix 的標準，而 ODBC 是 Win32 的標準；但隨著 ODBC 被移植到 Unix 系統，而 DBI 也被移植到 Win32 系統，這兩者的分野是日趨模糊。更甚者，`DBD::ODBC` 套件 —— 一個在 DBI 架構說 ODBC 語法的 DBD 模組【註】—— 的出現更加模糊了這兩者的界線。

註　除了我們要討論的標準之外，Perl 還有些 server 及 OS-specific 的優秀機制。Michael Peppler 為 Perl-Sybase 之間的溝通所設計的 *Sybperl* 就是一例。這些非標準的機制，有許多已有 DBI 化的模組。例如，*Sybperl* 的許多功能性，都可從 `DBD::Sybase` 獲得。在 Win32 平台，先前章節所提的 ActiveX Data Object（ADO）架構，也逐漸受到重視。

DBI 與 ODBC 的意圖與效果都相當接近，所以我們同時示範如何使用這兩者。DBI 與 ODBC 都可被視為“中介軟體”(middleware)。它們形成一層抽象概念，讓程式設計師只需使用一組通用的 DBI/ODBC 呼叫，而無須瞭解任何特殊資料庫的專用 API，就能寫出存取資料庫的程式。DBI/ODBC 軟體負責處理這些呼叫與特定資料庫之間的溝通。DBI 模組是呼叫 DBD driver 來完成這件事，而 ODBC Manager 則是呼叫資料來源專用的 ODBC driver 搞定。這個資料庫專屬的 driver，負責處理面對伺服器時所必須顧及的大大小小細節。圖 7-1 描繪出 DBI 與 ODBC 的架構。這兩者皆是歸納成一個三層式的模型（three-tiered model）：

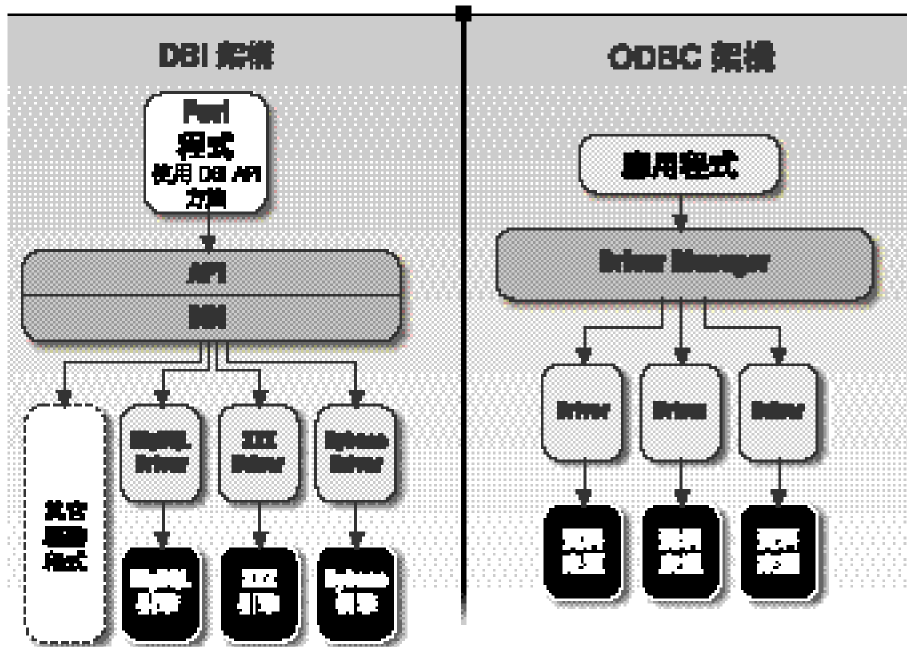

圖 7-1：DBI 與 ODBC 架構

1. **一個底層資料庫（Oracle、MySQL、Sybase、Microsoft SQL Server ... 等等），即圖 7-1 最底下的各種資料庫引擎（DBI 架構）與資料來源（ODBC 架構）。**

2. 底層資料庫驅動層。由於每種底層資料庫的實際存取方式不盡相同，因此必須存在此一層級，對上層提供共同的軟體界面，對下處理各種特定資料庫伺服器的差異。程式設計師不直接接觸這一層，而是透過第三層。在 DBI 架構下，這一層是一個與底層資料庫相關的 DBD 模組，對於 Oracle 資料庫而言，這模組就是 `DBD::Oracle`。在 ODBC 架構下，則是一個與資料來源相關的 ODBC Driver。不管是 DBI 或 ODBC，這一層的軟體都是由資料庫廠商提供的。

3. 第三層是跨資料庫的應用程式介面（API）層。很快地，我們將出能與這一層溝通的 Perl 程式。在 DBI架構下，這層稱為 DBI 層。在 ODBC 架構下，通常是透過 ODBC API 與 ODBC Driver Manager 互動。

這系統的漂亮之處，在於使用 DBI 或 ODBC 所寫出來的程式碼具備高度可移植性，程式碼幾乎不必修改，就能順利應用於不同廠商的伺服器。所有 API 呼叫都是一樣的，與底層的資料庫無關。對於大部份資料庫的"應用程式"設計而言，這理念的確如此；不幸地，對於我們要設計的"資料庫系統的管理程式"而言，則必須依據資料庫伺服器本身的特性來設計，因為不同伺服器之間的控管方式，實在找不到太多交集，即使它們的遠端遙控方式很相似【註】。有經驗的系統管理員喜歡可植移性高的方案，但他們不會有過份的期待。

先忘記這些麻煩事，讓我們看看如何使用 DBI 與 ODBC。這兩種技術都遵循相同的基本步驟，所以你在解說中將會看到若干的重複，或至少在標題是這樣。

在下一節，我假設你已經安裝了一個資料庫伺服器以及所需的 Perl 模組。在我們的 DBI 範例程式中，我將採用 MySQL 伺服器；在 ODBC 的範例中，我使用 MS-SQL 伺服器。

註　MS-SQL 當初是從 Sybase 的原始程式碼衍生出來的，這算是相當罕見的特例。

搭建 Unix 與 NT/2000 資料庫的橋樑

多平台系統管理者常問的一個共同問題是：『如何從我的 Unix 機器與 Microsoft SQL Server 溝通？』。如果作業環境的中控監督系統是 Unix，新安裝的 MS-SQL Server 將會是個挑戰。我知道三種處理這問題的方案。以下的第二與第三選擇，並非針對特定 SQL 伺服器，所以，即使你採用伺服器的並非 MS-SQL Server，這些技術也可能派上用場。

1. **建立並使用** `DBD::Sybase`。`DBD::Sybase` **將需要一些底層資料庫的通訊程式庫。有兩組這樣的程式庫可用。第一組是 Sybase OpenClient libraries，這應該有適合你的平台之版本（某些 Linux distribution 把 Sybase Adaptive Server Enterprise 當成它們發行套件的一部份免費奉送）。如果你用的是 MS-SQL Server 6.5（或更舊的版本），用這些程式庫所建立出來的** `DBD::Sybase` **模組就可以使用。對於 7.0 及其之後的版本，你需要從 Microsoft 取得一份相容性修正。在 http://support.microsoft.com/support/kb/articles/q239/8/83.asp 可找到這項修正的相關資訊（KB 文章編號 Q239883）。你第二項選擇是安裝 FreeTDS 程式庫（在 http://www.freetds.org 可以找到），請參閱該網站上的說明，瞭解如何建立你所使用伺服器的正確協定版本。**

2. **使用** `DBD::Proxy`。**有一個隨著 DBI 發佈的 DBD 模組稱為** `DBD::Proxy`，**它允許你在你的 MS-SQL Server 機器上跑一個小型網路伺服程式，這伺服程式能把來自 Unix client 的 request 無形地轉接到 MS-SQL Server。**

3. **經由** `DBD::ODBC` **取得並使用 Unix ODBC 軟體。包括 MERANT（http://www.merant.com）與 OpenLink Software（http://www.openlinksw.com）在內的幾家廠商出售這樣的軟體。或者，你也可以嘗試使用 Open Source 開發者的一些工作成果，關於進一步的資訊，請參考 Brian Jepson 在 http://users.ids.net/~bjepson/freeODBC 的 freeODBC 網頁。你會同時需要適合你的 Unix 平台的 ODBC Driver（由資料庫公司提供）以及一個 ODBC Manager（像是** `unixODBC` **或** `iODBC`**）。**

7.2 使用 DBI 架構

以下是使用 DBI 的基本步驟，關於更詳細的資訊，請參考 Alligator Descartes 與 Tim Bunce 合著的《Programming the Perl DBI》（O'Reilly）。

步驟一：載入必需的 Perl 模組

這裡沒什麼特別的，你只須要這樣：

```
use DBI;
```

步驟二：連接到資料庫，並取得代碼

與 MySQL 資料庫達成 DBI 連線、並取得其資料庫代碼（database handle）的 Perl 程式碼如下：

```
# 以所給的使用者名稱、密碼連接到 $database 資料庫
# 傳回代表該連線的資料庫代碼
$database = "sysadm";
$dbh = DBI->connect("DBI:mysql:$database",$username,$pw);
die "Unable to connect: $DBI::errstr\n" unless (defined $dbh);
```

DBI 會在真正連接到伺服器之前，先幫我們載入底層的 DBD driver（在此例中是 DBD::mysql）。接著我們檢查 connnect() 是否成功，以決定是否繼續下一個動作。DBI 為 connect() 提供了 RaiseError 與 PrintError 這兩選項，我們藉此決定是否讓 DBI 行使這項測試，或是發生錯誤時自動發出抱怨。例如，假如我們這樣做：

```
$dbh = DBI->connect("DBI:mysql:$database",
                    $username,$pw,{RaiseError => 1});
```

那麼當 connect() 失敗時，DBI 將會幫我們呼叫 die。

步驟三：傳送 SQL 命令到伺服器

當我們的 Perl 模組已載入，而且也成功連線到資料庫之後，好戲就可以上場了；讓我們傳送一些 SQL 命令到資料庫伺服器去。我們將使用附錄 D 的一些 SQL 查詢句（SQL query）為例。這些查詢句將使用 Perl 的 q 代替引號（亦即，*something* 是寫成 q{*something*}），這樣我們就不必擔心查詢句裡的單引號或雙引號【譯註】。以下是在 DBI 傳送命令的兩種方法之一：

```
$results=$dbh->do(q{UPDATE hosts
                    SET bldg = 'Main'
                    WHERE name = 'bendir'});
die "Unable to perform update:$DBI::errstr\n" unless (defined $results);
```

若執行成功，$results 將會收到已更新的列數，若發生錯誤，則它會收到 undef。儘管知道有多少列受影響頗為有用，但對於類似 SELECT 這種我們想看到實際資料的查詢句，就顯然不符合我們所需，這就是引進第二種方法的原因。

要使用第二種方法，你必須先準備（prepare）一道 SQL 敘述以供使用，並要求伺服器執行（execute）它，像這樣：

```
$sth = $dbh->prepare(q{SELECT * from hosts}) or
  die "Unable to prep our query:".$dbh->errstr."\n";
$rc = $sth->execute or
  die "Unable to execute our query:".$dbh->errstr."\n";
```

prepare() 傳回我們沒見過的新物種：敘述代碼（statement handle）。就像「資料庫代碼」代表已開啟的「資料庫連線」一樣，「敘述代碼」指的是我們用 prepare() 準備好的特定「SQL 敘述句」。一旦我們有了這敘述代碼，就可用 execute() 將該查詢句傳送到伺服器。稍後，我們將使用同一個敘述代碼取回我們的查詢結果。

譯註　如果不使用 q，那麼查詢句裡的單引號要寫成 \'，而雙引號要寫成 \"，這使整個查詢句看起來比較雜亂。

你或許會覺得奇怪，為何我們不能直接執行 SQL 敘述，而必須先用 prepare() 準備它？這個“準備”的用意何在？其實 prepare() 是為了給 DBD driver（或是它所呼叫的資料庫用戶端程式庫）一個分析該 SQL 查詢句的機會。藉由 prepared() 傳回的敘述代碼，我們就可以重複執行同一個 SQL 敘述，而不必每次都重新分析一模一樣的 SQL 查詢句，如此一來，將能有效提昇程式與資料庫的運作效率。事實上，do() 的內定行為，會暗自將你要它執行的每個敘述都先 prepare() 過，然後才 execute()。

就像我們先前看過的 do，execute() 傳回受影響的列數。如果查詢句不改動任何資料（0 列），將傳回字串 0E0，以便讓隨後的邏輯測試會成功。如果該 driver 無法探知到底影響了多少列，則傳回 -1。

在我們進到 ODBC 之前，有個大多數 DBD 模組都支援的 prepare() 技巧值得一提：placeholder（佔位符號），又稱為位置記號（positional marker）。利用 placeholder，你可以 prepare() 一個“不完整”的 SQL 敘述，然後在 execute() 時才補齊這個 SQL 敘述。藉此，我們能動態產生查詢句，但又省去每次都要分析查詢句的時間。用問號（?）當成 placeholder，代表它是一個純量值。以下示範如何在 Perl 程式中使用 placeholder：

```
@machines = qw(bendir shimmer sander);
$sth = $dbh->prepare(q{SELECT name, ipaddr FROM hosts WHERE name = ?});
foreach $name (@machines){
  $sth->execute($name);
      ... 處理所得的結果 ...
}
```

每回合的 foreach 迴圈，都會產生不同 WHERE 子句的 SELECT 查詢句。多個 placeholder 的用法也很直接：

```
$sth->prepare(
   q{SELECT name, ipaddr FROM hosts
     WHERE (name = ? AND bldg = ? AND dept = ?)});
$sth->execute($name,$bldg,$dept);
```

現在我們知道如何取得 non-SELECT SQL 查詢句所影響的列數，讓我們看看如何取得我們的 SELECT 結果。

步驟四：取得 SELECT 結果

這裡的機制類似我們在附錄 D 介紹的 cursor 功能。當我們使用 execute() 送一個 SELECT 敘述到伺服器時，我們使用的正是可讓我們一次取得一行結果的機制。

在 DBI，我們可呼叫表 7-1 的任一方法傳回 result set 中的資料。

表 7-1 傳回資料的 DBI 方法

名稱	傳回值	沒有後續資料時的傳回值
fetchrow_arrayref()	一個指向匿名陣列的參照，這陣列內容是 result set 內下一筆紀錄各欄的值。	undef
fetchrow_array()	一個陣列，這陣列內容是 result set 內下一筆紀錄各欄的值	一個空的陣列
fetchrow_hashref()	一個指向匿名雜湊表的參照，這雜湊表以欄名為索引，其內容是 result set 內下一筆紀錄各欄的值	undef
fetchall_arrayref()	一個指向陣列的參照指標，這陣列內容是一組陣列的資料結構。	一個空的陣列

我打算以範例解釋這些方法。為了方便說明，我假設每個範例程式在執行之前，已先跑過一次以下這段程式碼：

```
$sth = $dbh->prepare(q{SELECT name,ipaddr,dept from hosts}) or
  die "Unable to prepare our query: ".$dbh->errstr."\n";
$sth->execute or die "Unable to execute our query: ".$dbh->errstr."\n";
```

這是 fetchrow_arrayref() 的用法：

```
while ($aref = $sth->fetchrow_arrayref){
  print "name: "   .  $aref->[0] . "\n";
  print "ipaddr: " .  $aref->[1] . "\n";
  print "dept: "   .  $aref->[2] . "\n";
}
```

在 DBI 說明文件中提到，由於 fetchrow_hashref() 需要處理一些額外的細節，以致於其效率不如 fetchrow_arrayref()。但是，使用 fetchrow_hashref() 會使得程式更具可讀性，像這樣子：

```
while ($href = $sth->fetchrow_hashref){
  print "name: "   .  $href->{name}  . "\n";
  print "ipaddr: " .  $href->{ipaddr}. "\n";
  print "dept: "   .  $href->{dept}  . "\n";
}
```

最後，讓我們看一下 fetchall_arrayref() 這個 " 方便 " 的方法。這方法將整個 result set 塞入一個資料結構，然後傳回該陣列的參照指標。使用這方法時要小心，對於會產生大量 result set 的查詢，最好不要用此方法，因為這方法會把完整的 result set 都讀進記憶體。倘若你有一個 100GB 的 result set，這肯定會出問題。

每個傳回的參照指標，看起來都很像我們從 fetchrow_arrayref() 取得的東西。請參考圖 7-2。

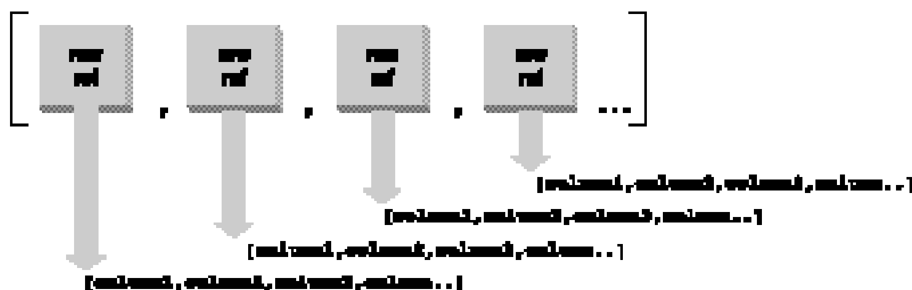

圖 7-2：fetchrow_arrayref 所傳回的資料結構

這裡是印出整個查詢結果的程式碼：

```
$aref_aref = $sth->fetchall_arrayref;
foreach $rowref (@$aref_aref){
  print "name: "   .  $rowref->[0] . "\n";
  print "ipaddr: " .  $rowref->[1] . "\n";
  print "dept: "   .  $rowref->[2] . "\n";
  print '-'x30,"\n";
}
```

這段程式碼是針對我們特殊的資料集合而設計的，因為它假定了欄的數量以及各欄所代表的意義。例如，我們假設機器名稱放在查詢結果的第一欄 ($rowref->[0])。

若要讓擷取查詢結果的程式碼與資料結構“脫勾”，也就是說，我們希望能有一套與資料結構無關的通用作法，不管查詢結果的各欄位意義如何，都可用同一段程式碼秀出來。那麼我們將需要利用敘述代碼的一些“魔術屬性”(magic attributes；通常稱為“中介資料”，metadata)。更具體地說，如果我們在查詢之後看看 $sth->{NUM_OF_FIELDS} 的值，它將告訴我們 result set 裡的欄位數量。$sth->{NAME} 含有一個參照指標，它指向一個內有各欄名稱的陣列。讓我們利用這些技巧改寫前一段程式，使它更具彈性：

```
$aref_aref = $sth->fetchall_arrayref;
foreach $rowref (@$aref_aref){
  for ($i=0; $i < $sth->{NUM_OF_FIELDS};i++;){
    print $sth->{NAME}->[$i].": ".$rowref->[$i]."\n";
  }
  print '-'x30,"\n";
}
```

關於其它 metadata 屬性的細節，請參考 DBI 文件。

步驟五：關閉伺服器連線

在 DBI，兩下子就搞定了：

```
# 讓伺服器知道，我們已不需要從這個敘述代碼取得其它資料了
# （這是可有可無的步驟，因為我們下一步就是切斷連線）
$sth->finish;
# 切斷通往資料庫伺服器的連線
$dbh->disconnect;
```

7.2.1 DBI 的其餘注意事項

在我們進到 ODBC 之前，還有兩項值得一提的 DBI 議題。首先是一組我稱為“捷徑”的方法，也就是表 7-2 所列的方法，再加上先前步驟三與步驟四所提的那些方法。

表 7-2：DBI 捷徑方法

名稱	所結合的方法
selectrow_arrayref($stmnt)	prepare($stmnt), execute(), fetchrow_arrayref()
selectcol_arrayref($stmnt)	prepare($stmnt), execute(), (@{fetchrow_arrayref()})[0] （也就是傳回每一列的第一欄）
selectrow_array($stmnt)	prepare($stmnt), execute(), fetchrow_array()

DBI 第二項值得一提的能力，是它能將變數與查詢結果聯繫在一起。我們可用 bind_col() 與 ind_columns() 這兩個方法告訴 DBI，要求它自動將查詢結果放在一個特定變數，或是特定的一連串變數。這可讓我們的程式碼減少一兩個步驟。這裡示範 bind_columns() 的用法：

```
$sth = $dbh->prepare(q{SELECT name,ipaddr,dept from hosts}) or
  die "Unable to prep our query:".$dbh->errstr".\n";
$rc = $sth->execute or
  die "Unable to execute our query:".$dbh->errstr".\n";

# 這些變數會依序收到 SELECT 查詢結果的第一、二、三欄。

$rc = $sth->bind_columns(\$name,\$ipaddr,\$dept);

while ($sth->fetchrow_arrayref){
    # 從查詢結果取出的列，其各欄已經神奇地自動填入 $name、$ipaddr 與 $dept
    ... 處理查詢結果 ...
}
```

7.3 使用 ODBC 架構

使用 ODBC 的基本步驟，類似於先前討論過 DBI 使用步驟。

步驟一：載入必需的 Perl 模組

```
use Win32::ODBC;
```

步驟二：連線到資料庫，並取得其代碼

ODBC 在連線之前，還需要一項預備動作：產生「資料來源名稱（Data Source Name，DSN）」。DSN 是當你想觸及資訊來源（像是 MS-SQL Server）時，所需的一組組態資訊（伺服器、資料庫名稱 ... 等等）的名稱。DSN 分成兩種，分別是 user 與 system，這兩者之間的主要差異，在於資料庫連線的服務對象；user DSN 只服務一台機器上的一位使用者，而 system DSN 則供連線機器上的任何人或伺服程式【註】使用。

DSN 的建立方式有兩種：透過 Windows NT/2000 控制台（control panel）裡的「ODBC 資料來源管理員」，或是利用 Perl 在程式裡建立。很顯然，第一種作法只適用於 Windows 系統，而第二種作法則可以跨平台，兼顧 UNIX 的使用者，因此我們採取寫程式的作法。以下程式碼在對我們在 MS-SQL server上的資料庫建立一個 user DSN：

```
# 對 MS-SQL Server 建立一個 user DSN
# 注意：若要建立 system DSN，要把 ODBC_ADD_DSN 換成 ODBC_ADD_SYS_DSN
if (Win32::ODBC::ConfigDSN(
                    ODBC_ADD_DSN,
                    "SQL Server",
                    ("DSN=PerlSysAdm",
                     "DESCRIPTION=DSN for PerlSysAdm",
                     "SERVER=mssql.happy.edu",  # 伺服器名稱
                     "ADDRESS=192.168.1.4",     # 伺服器 IP 位址
                     "DATABASE=sysadm",         # 我們的資料庫
                     "NETWORK=DBMSSOCN",        # TCP/IP Socket Lib
                    ))){
                         print "DSN created\n";
                       }
else {
   die "Unable to create DSN:" . Win32::ODBC::Error() . "\n";
}
```

註　其實還有第三種：檔案。這是把 DSN 組態資訊寫到一個檔案，讓多台電腦分享。但它不是我們所用的 Win32::ODBC 建立的。

拿到 DSN 之後，就可用它連線到我們的資料庫。

```
# 連線到具名 DSN，傳回一個資料庫代碼。
$dbh=new Win32::ODBC("DSN=PerlSysAdm;UID=$username;PWD=$pw;");
die "Unable to connect to DSN PerlSysAdm:" . Win32::ODBC::Error() . "\n"
  unless (defined $dbh);
```

步驟三：傳送 SQL 命令到伺服器

與 DBI 的 do()、prepare() 與 execute() 對等的 ODBC 方法相當簡單，因為 Win32::ODBC 模組裡唯一能將命令送到伺服器的方法是 Sql()。雖然 ODBC 在理論上也有 "prepared statement" 以及 "placeholder"，但是目前版本的 Win32::ODBC 模組並沒有實現這些功能【註】。Win32::ODBC 也沒使用敘述代碼，所有通訊都是透過在初始化資料庫時、由 new 所傳回的資料庫代碼（參考步驟二最後的程式碼）。因此，留給我們的是最簡單的命令結構：

```
$rc = $dbh->Sql(q{SELECT * from hosts});
```

在 ODBC 與 DBI 方法之間的一項重要差異是：ODBC 的 Sql() 在「成功」時是傳回 undef，而失敗時則傳回某個非零數值。而 DBI 的 do() 若傳回 undef，那是代表「失敗」。

如果你想知道一次 INSERT、DELETE 或 UPDATE 查詢影響了多少列，你可以用 RowCount() 方法。在 Win32::ODBC 的說明文件中指出，並非所有 ODBC driver 都支援此方法（或並非所有 SQL 作業皆實作它），因此，在你依賴這方法之前，最好先測試你的 ODBC driver 是否支援。如同 DBI 的 execute() 方法，如果 driver 無法傳回受影響的列數，RowCount() 將會傳回 -1。

註　當我為文至此，Dave Roth 正在測試新版的 Win32::ODBC 模組，這個版本支援「參數繫結」(parameter binding)。它使用類似於 DBI 的語法（先 prepare() 然後 Sql()）。請參考 http://www.roth.net。

以下是先前 DBI do() 範例的等效 ODBC 程式碼：

```
if (defined $dbh->Sql(q{UPDATE hosts
                        SET bldg = 'Main'
                        WHERE name = 'bendir'})){
  die "Unable to perform update: ".Win32::ODBC::Error()."\n"
}
else {
  $results = $dbh->RowCount();
}
```

步驟四：擷取 SELECT 的查詢結果

用 ODBC 取得 SELECT 查詢結果的方式，與 DBI 的作法大同小異：必須先把資料從伺服器“取出”(fetching) 來，然後才能“取用”(accessing) 這些資料。在 Win32::ODBC 架構下，第一個動作是靠 FetchRow() 完成，當它傳回 1，表示已經成功取得下一列資料，否則傳回 undef 代表失敗。在“取出”我們所要的一列資料之後，就可用以下兩種方法之一來“取用”資料。

在串列環境 (list context) 中呼叫 Data() 方法時，它傳回一列 FetchRow() 所傳回的欄。若在純量環境 (scalar context) 呼叫 Data()，它傳回的這些欄首尾相連在一起的樣子。若想指定這些欄的排列順序，以及哪些欄是要的，而哪些欄又是不要的，你可以另外指定一個串列引數給 Data() 方法，要求它依據這串列挑出你所要的欄，並依照你定義的順序排列，否則，依照說明文件，這些欄將以“未指明”(unspecified) 的順序傳回。

DataHash() 傳回一個以欄名為索引、欄值為內容的雜湊表。就像 DBI 的 fetchrow_hashref() 方法一樣，只差 DataHash() 傳回的是「雜湊表」，而 fetchrow_hashref() 傳回的是「雜湊表的參照指標」。此外，DataHash 也接受一個可有可無的串列引數，讓你指定要傳回哪些欄，以及這些欄的順序。

在串列環境，“取出”與“取用”的程式碼看起來會像是這樣：

```
if ($dbh->FetchRow()){
  @ar = $dbh->Data();
   ... 運用 @ar 的值做你想做的事 ...
}
```

在純量環境則是這樣：

```
if ($dbh->FetchRow()){
   $ha = $dbh->DataHash('name','ipaddr');
   ... 運用 $ha{name} 以及 $ha{ipaddr} 做你想做的事 ...
}
```

為了讓我們的討論更完整，順便一提，DBI 的敘述代碼屬性，{NAME}，所代表的資訊，在 Win32::ODBC 架構下，可利用 FieldNames() 找到。如果你想知道欄位的數量（像是 {NUM_OF_FIELDS}），你必須計算 FieldNames() 所傳回串列之元素個數。

步驟五：關閉伺服器連線

這動作很簡單：

```
$dbh->close();
```

如果你建立一個 DSN，而且想要自己刪除它，其步驟與當初在建立它時很像：

```
# 對於 system DSN，你必須將 ODBC_REMOVE_DSN 換成 ODBC_REMOVE_SYS_DSN
if (Win32::ODBC::ConfigDSN(ODBC_REMOVE_DSN,
                           "SQL Server","DSN=PerlSysAdm")){
  print "DSN deleted\n";
}
else {
  die "Unable to delete DSN:".Win32::ODBC::Error()."\n";
}
```

你現在知道如何使用 Perl 透過 DBI 與 ODBC 來操作資料庫。現在讓我們將你所學到的，發揮到系統管理的領域。

7.4 記錄伺服器組態

設定 SQL 伺服器，並管理資料庫中的各種物件，是一件相當耗時勞心的工作。若有辦法將這類資訊記錄下來，某些情況下將會很方便。例如，若資料庫損毀而且沒有備份，你可能會被要求重建該資料庫的所有表格，或者，你可能需要把資料搬到另一個伺服器；在這些情況下，知道來處和去處的伺服器組態，就顯得相當重要。另一方面，倘若你自己要設計資料庫程式，能準備一份所有表格的對照表，將會有相當的幫助。

資料庫的管理工作，其細節與技巧隨著所用伺服器的不同，而有相當程度的差異。為了讓你體會一下資料庫管理工作“不具可植移性”的本質，讓我以相同的一件工作為例，分別在三個不同的伺服器上使用 DBI 與 ODBC 作示範。以下每一個程式做的事都完全相同：印出伺服器上的所有資料庫、這些資料庫的所有表格、以及每個表格的基本結構。你可以輕易擴充這些範例程式碼，秀出各個物件的詳細資訊，像是秀出表格的哪些欄允許設定成 NULL，諸如此類。先讓我們看一下這三個程式的輸出大概是什麼樣子：

```
---sysadm---
        hosts
                name [char(30)]
                ipaddr [char(15)]
                aliases [char(50)]
                owner [char(40)]
                dept [char(15)]
                bldg [char(10)]
                room [char(4)]
                manuf [char(10)]
                model [char(10)]
---hpotter---
        customers
                cid [char(4)]
                cname [varchar(13)]
                city [varchar(20)]
                discnt [real(7)]
        agents
                aid [char(3)]
                aname [varchar(13)]
```

```
                city [varchar(20)]
                percent [int(10)]
        products
                pid [char(3)]
                pname [varchar(13)]
                city [varchar(20)]
                quantity [int(10)]
                price [real(7)]
        orders
                ordno [int(10)]
                month [char(3)]
                cid [char(4)]
                aid [char(3)]
                pid [char(3)]
                qty [int(10)]
                dollars [real(7)]
...
```

7.4.1 透過 DBI 存取 MySQL Server

以下是用 DBI 從 MySQL 伺服器提取資訊的作法，MySQL 額外提供的 *SHOW* 命令，使得這項工作變得相當簡單：

```
use DBI;

print "Enter user for connect: ";
chomp($user = <STDIN>);
print "Enter passwd for $user: ";
chomp($pw = <STDIN>);

$start= "mysql"; # 一開始先連接這資料庫

# 連接到 $start 所代表的 MySQL 資料庫
$dbh = DBI->connect("DBI:mysql:$start",$user,$pw);
die "Unable to connect: ".$DBI::errstr."\n" unless (defined $dbh);

# 找尋伺服器上的資料庫
$sth=$dbh->prepare(q{SHOW DATABASES}) or
  die "Unable to prepare show databases: ". $dbh->errstr."\n";
```

```
$sth->execute or
  die "Unable to exec show databases: ". $dbh->errstr."\n";
while ($aref = $sth->fetchrow_arrayref) {
    push(@dbs,$aref->[0]);
}
$sth->finish;

# 找出每個資料庫的表格
foreach $db (@dbs) {
    print "---$db---\n";

    $sth=$dbh->prepare(qq{SHOW TABLES FROM $db}) or
     die "Unable to prepare show tables: ". $dbh->errstr."\n";
    $sth->execute or
      die "Unable to exec show tables: ". $dbh->errstr."\n";

    @tables=();
    while ($aref = $sth->fetchrow_arrayref) {
        push(@tables,$aref->[0]);
    }

    $sth->finish;

   # 找出每個表格的各欄資訊
   foreach $table (@tables) {
        print "\t$table\n";

        $sth=$dbh->prepare(qq{SHOW COLUMNS FROM $table FROM $db}) or
          die "Unable to prepare show columns: ". $dbh->errstr."\n";
        $sth->execute or
          die "Unable to exec show columns: ". $dbh->errstr."\n";

        while ($aref = $sth->fetchrow_arrayref) {
            print "\t\t",$aref->[0]," [",$aref->[1],"]\n";
        }

       $sth->finish;
   }
}
$dbh->disconnect;
```

簡單說明一下這段程式碼：

- **由於我們在這環境中運用了** SHOW **命令，所以在一開始連接到** $start **資料庫的動作，其實是不必要的，我們之所以這樣做，純粹只是為了配合 DBI** connect() **的語法，在隨後的兩個例子中，我們不會這樣做。**
- **如果你認為我們應該在** SHOW TABLES **與** SHOW COLUMNS **的“準備”與“執行”敘述中運用 placeholders 才對，沒錯，你絕對是正確的；不幸的是，這一組 DBD driver/server 的組合，在這環境下並不支援 placeholder (至少在寫作本文時尚未支援)。在我們下個範例會看到類似的情形。**
- **我們以互動的方式，提示使用者輸入其名稱與密碼，那是因為我們沒有更好的替代作法（直接把名稱與密碼寫死在程式碼中）。在使用者輸入密碼時，他所鍵入的字元會出現在螢幕上，這並不妥當，謹慎一點的作法，應該利用類似** Term::Readkey **之類的東西關掉字元回應。**

7.4.2 透過 DBI 存取 Sybase Server

以下是同樣的事在 Sybase Server 上的作法，照例，稍後會說明這段程式碼的注意事項：

```
use DBI;

print "Enter user for connect: ";
chomp($user = <STDIN>);
print "Enter passwd for $user: ";
chomp($pw = <STDIN>);

$dbh = DBI->connect('dbi:Sybase:',$user,$pw);
die "Unable to connect: $DBI::errstr\n"
  unless (defined $dbh);

# 找出伺服器上的資料庫
$sth = $dbh->prepare(q{SELECT name from master.dbo.sysdatabases}) or
  die "Unable to prepare sysdatabases query: ".$dbh->errstr."\n";
$sth->execute or
  die "Unable to execute sysdatabases query: ".$dbh->errstr."\n";
```

```
while ($aref = $sth->fetchrow_arrayref) {
    push(@dbs, $aref->[0]);
}
$sth->finish;

foreach $db (@dbs) {
    $dbh->do("USE $db") or
      die "Unable to use $db: ".$dbh->errstr."\n";
    print "---$db---\n";

    # 找出每個資料庫裡的表格
    $sth=$dbh->prepare(q{SELECT name FROM sysobjects WHERE type="U"}) or
      die "Unable to prepare sysobjects query: ".$dbh->errstr."\n";
    $sth->execute or
      die "Unable to exec sysobjects query: ".$dbh->errstr."\n";

    @tables=();
    while ($aref = $sth->fetchrow_arrayref) {
        push(@tables,$aref->[0]);
    }
    $sth->finish;

    # 必須先 "進入" 資料庫，才能繼續下一步
    $dbh->do("use $db") or
          die "Unable to change to $db: ".$dbh->errstr."\n";

    # 查詢各表格中的各欄資訊
    foreach $table (@tables) {
        print "\t$table\n";

       $sth=$dbh->prepare(qq{EXEC sp_columns $table}) or
            die "Unable to prepare sp_columns query: ".$dbh->errstr."\n";
        $sth->execute or
            die "Unable to execute sp_columns query: ".$dbh->errstr."\n";

        while ($aref = $sth->fetchrow_arrayref) {
            print "\t\t",$aref->[3]," [",$aref->[5],"(",
                  $aref->[6],")]\n";
```

```
        }
        $sth->finish;
    }
}
$dbh->disconnect or
  warn "Unable to disconnect: ".$dbh->errstr."\n";
```

以下是先前承諾的相關說明：

- **Sybase 將其資料庫與表格的資訊，另外放在一組特殊的系統表格，分別是 *sysdatabases* 與 *sysobjects*。每個資料庫都有一個 *sysobjects* 表格，而伺服器裡每個資料庫的資訊，則記錄在主資料庫裡的 *sysdatabases*。我們在第一個** `SELECT` **敘述中使用** *`databases.owner.table`* **這種比較明確的語法來代表此表格，以免產生模擬兩可的狀況。若要取得每個資料庫的 *sysobjects*，我們可以直接使用這樣的語法，而不必以** `USE` **另外切換到資料庫環境（database context）。但就像 *cd* 進入目錄一樣，切換到資料庫環境，可讓你在構思其它查詢句時，比較簡單些。**

- **使用** `SELECT` **從 *sysobjects* 選擇表格時，我們運用了** `WHERE` **子句限定它只傳回使用者定義的表格（user-defined tables）【譯註】，這樣做是為了不想輸出太多不必要的東西，倘若想你當真想連同系統性表格也一起輸出，可以把** `WHERE` **子句換成這樣：**

  ```
  WHERE type="U" AND type="S"
  ```

- `DBD::Sybase` **支援 placeholder 的用法，但不能運用在預儲程序（stored procedure）；若非如此，我們會在** `EXEC sp_columns` **使用 placeholder。**

譯註　**即** `$sth=$dbh->prepare(q{SELECT name FROM sysobjects WHERE type="U"})` **這行程式碼，其中的** `"U"` **代表 User-defined**

7.4.3 透過 ODBC 存取 MS-SQL Server

最後，以下是透過 ODBC 將同樣的資訊從 MS-SQL Server 抓出來的程式。多虧了 Sybase 與 MS-SQL Server "一脈相承"，這裡所用的 SQL 敘述與前一節如出一轍。本節與前一節範例的差異之處，歸納如下：

- **由於 ODBC 的 DSN 就已經提供了內定的資料庫環境，所以我們無須另外查詢 *sysdatabases* 表格。**
- **以** `$dbh->DropCursor()` **取代等效的** `$sth->finish`。
- **你需要以比較惱人的語法才能執行預儲程序。關於如何處理預儲程序的細節，請參考** `Win32::ODBC` **的網頁。**

程式碼如下：

```
use Win32::ODBC;

print "Enter user for connect: ";
chomp($user = <STDIN>);
print "Enter passwd for $user: ";
chomp($pw = <STDIN>);

$dsn="sysadm"; # 我們將使用的 DNS 名稱

# 找出可用的 DSNs，若對應的 $dsn 不存在，則建立之
die "Unable to query available DSN's".Win32::ODBC::Error()."\n"
  unless (%dsnavail = Win32::ODBC::DataSources());
if (!defined $dsnavail{$dsn}) {
    die "unable to create DSN:".Win32::ODBC::Error()."\n"
      unless (Win32::ODBC::ConfigDSN(ODBC_ADD_DSN,
                   "SQL Server",
                   ("DSN=$dsn",
                   "DESCRIPTION=DSN for PerlSysAdm",
                   "SERVER=mssql.happy.edu",
                   "DATABASE=master",
                   "NETWORK=DBMSSOCN", # TCP/IP Socket Lib
                   )));
}
```

```
# 連接到主資料庫
$dbh = new Win32::ODBC("DSN=$dsn;UID=$user;PWD=$pw;");
die "Unable to connect to DSN $dsn:".Win32::ODBC::Error()."\n"
    unless (defined $dbh);

# 找出伺服器上的所有資料庫
if (defined $dbh->Sql(q{SELECT name from sysdatabases})){
    die "Unable to query databases:".Win32::ODBC::Error()."\n";
}

while ($dbh->FetchRow()){
    push(@dbs, $dbh->Data("name"));
}
$dbh->DropCursor();

# 找出每個資料庫的 user-defined 表格
foreach $db (@dbs) {
    if (defined $dbh->Sql("use $db")){
         die "Unable to change to database $db:" .
            Win32::ODBC::Error() . "\n";
    }
    print "---$db---\n";
    @tables=();
    if (defined $dbh->Sql(q{SELECT name from sysobjects
                            WHERE type="U"})){
        die "Unable to query tables in $db:" .
            Win32::ODBC::Error() . "\n";
    }
    while ($dbh->FetchRow()) {
        push(@tables,$dbh->Data("name"));
    }
    $dbh->DropCursor();

    # 找出各表格每一欄的資訊
    foreach $table (@tables) {
        print "\t$table\n";
        if (defined $dbh->Sql(" {call sp_columns (\'$table\')} ")){
            die "Unable to query columns in
```

```
                $table:".Win32::ODBC::Error() . "\n";
        }
        while ($dbh->FetchRow()) {
            @cols=();
            @cols=$dbh->Data("COLUMN_NAME","TYPE_NAME","PRECISION");
            print "\t\t",$cols[0]," [",$cols[1],"(",$cols[2],")]\n";
        }
        $dbh->DropCursor();
    }
}
$dbh->Close();

die "Unable to delete DSN:".Win32::ODBC::Error()."\n"
    unless (Win32::ODBC::ConfigDSN(ODBC_REMOVE_DSN,
                                "SQL Server","DSN=$dsn"));
```

7.5 資料庫登入

如前所述，資料庫管理者必須處理一些系統管理者所面對的問題，像是維護使用者的登入活動，以及管理使用者帳號 ...等等。我日常教授的資料庫程式設計課程為例，每位來上課的學生，都各自需要一個能登入我們的 Sybase Server 的帳號，而且在伺服器上各自有一個專用的小資料庫以供練習。以下是我們用來建立這些資料庫及登入帳號的程式（這是簡化過的版本）：

```
use DBI;

# 用法：syaccreate <使用者名稱>

$admin = 'sa';
print "Enter passwd for $admin: ";
chomp($pw = <STDIN>);
$user=$ARGV[0];

# 將使用者名稱反轉，其後加上六個減號字元，
# 藉此產生 *假的* 密碼
$genpass = reverse join('',reverse split(//,$user));
$genpass .= "-" x (6-length($genpass));
```

```
# 以下列出我們將依序執行的 SQL 命令：
#【譯註：為了方便對照後面程式碼所產生的 SQL 敘述，以下註解保留原文】
#      1) create the database on the USER_DISK device,
          with the log on USER_LOG
#      2) add a login to the server for the user,
          making the new database the default.
#      3) switch to the newly created database
#      4) change its owner to be this user
@commands = ("create database $user on USER_DISK=5 log on USER_LOG=5",
             "sp_addlogin $user,\"$genpass\",$user",
             "use $user",
             "sp_changedbowner $user");

# 連接到伺服器
$dbh = DBI->connect('dbi:Sybase:',$admin,$pw);
die "Unable to connect: $DBI::errstr\n"
  unless (defined $dbh);

# 逐一依序執行命令陣列中的各個命令
for (@commands) {
    $dbh->do($_) or die "Unable to $_: " . $dbh->errstr . "\n";
}

$dbh->disconnect;
```

因為這工作只需執行一組不會傳回資料的命令，因此我們可用一個相當簡潔的迴圈，反覆執行 `$dbh->do()` 即可。我們可以在課程結束時，使用幾乎一樣的程式碼刪除些帳號與資料庫。

```
use DBI;

# 用法：syacdelete <使用者名稱>

$admin = 'sa';
print "Enter passwd for $admin: ";
chomp($pw = <STDIN>);
$user=$ARGV[0];

# 以下列出刪除使用者資料庫與帳號所需的 SQL 命令：
```

```
#【譯註：照例，為了方便對照，以下註解保留原文】
#   drop the user's server login
@commands = ("drop database $user",
             "sp_droplogin $user");

# 連接到伺服器
$dbh = DBI->connect('dbi:Sybase:',$admin,$pw);
die "Unable to connect: $DBI::errstr\n"
  unless(defined $dbh);

# 逐一執行陣列中的命令
for (@commands) {
    $dbh->do($_) or die "Unable to $_: " . $dbh->errstr . "\n";
}

$dbh->disconnect or
  warn "Unable to disconnect: " . $dbh->errstr . "\n";
```

許多跟帳號有關的功能，都可以寫成程式，以下是一些構想：

密碼稽核程式

連接到伺服器後，取得資料庫與帳號的清單。然後試著以一些差勁的密碼（帳號名稱、空白密碼、預設密碼 ... 等等）嘗試登入資料庫，藉此揪出危險的帳號。

使用者對照表

產生一個哪些使用者能存取哪些資料庫的清單。

密碼控制

設計一個虛構密碼（pseudo-password）的期限控制系統。

7.6 監控伺服器的健康

我們打算以一些監控 SQL 伺服器健康狀態的手法，作為本章的壓軸範例。這類例行性監控工作，其本質類似我們在第五章看到的網路服務監控。

7.6.1 空間容量監控

就技術上的觀點而言，資料庫是一個讓你放置東西的地方，如果你耗盡了放置東西的空間，這就是一件 "糟糕"（嗯，應該說 "相當糟糕"）的事。結論就是，我們需要有一套能幫我們查看伺服器已經配置了多少空間、以及耗用了多少空間的程式。讓我們以一個 DBI 程式為例，看看它如何監控 Sybase 伺服器。

照例，讓我們先看看程式的輸出狀況，知道它如何以圖形化的圖表呈現伺服器上每個資料庫的耗用空間情形。在此例中，你可以看到這伺服器上有五個資料庫，各個資料庫本身及其日誌檔所占用的空間比例，都是以長條圖表示（d 表示資料空間，l 表示日誌檔空間），如下：

```
               |ddddddd                                           |15.23%/5MB
hpotter--------|                                                  |
               |                                                  |0.90%/5MB

               |ddddddd                                           |15.23%/5MB
dumbledore-----|                                                  |
               |                                                  |1.52%/5MB

               |dddddddd                                          |16.48%/5MB
hgranger-------|                                                  |
               |                                                  |1.52%/5MB

               |ddddddd                                           |15.23%/5MB
rweasley-------|                                                  |
               |l                                                 |3.40%/5MB

               |ddddddddddddddddddddddddddd                       |54.39%/2MB
hagrid---------|                                                  |
               |- no log                                          |
```

以下是產生這份輸出報表的程式：

```
use DBI;

$admin = 'sa';
print "Enter passwd for $admin: ";
```

```
chomp($pw = <STDIN>);
$pages = 2; # 資料以 2k 為單位存放。

# 連接到伺服器
$dbh = DBI->connect('dbi:Sybase:',$admin,$pw);
die "Unable to connect: $DBI::errstr\n"
  unless (defined $dbh);

# 取得伺服器上的資料庫名稱
$sth = $dbh->prepare(q{SELECT name from sysdatabases}) or
  die "Unable to prepare sysdatabases query: ".$dbh->errstr."\n";
$sth->execute or
  die "Unable to execute sysdatabases query: ".$dbh->errstr."\n";

while ($aref = $sth->fetchrow_arrayref) {
    push(@dbs, $aref->[0]);
}
$sth->finish;

# 取得每個資料庫的狀態
foreach $db (@dbs) {

    # 取得並加總所有 size 欄（非日誌部分）
    $size    = &querysum(qq{SELECT size FROM master.dbo.sysusages
                          WHERE  dbid = db_id(\'$db\')
                          AND    segmap != 4});
    # 取得並加總所有日誌的 size 欄
    $logsize = &querysum(qq{SELECT size FROM master.dbo.sysusages
                          WHERE  dbid = db_id(\'$db\')
                          AND    segmap = 4});

    # 切換到該資料庫，並取得用量狀態
    $dbh->do(q{use $db}) or
      die "Unable to change to $db: ".$dbh->errstr."\n";

    # 使用 reserved_pgs 函式傳回資料庫中，
    # "資料"（doampg）與 "索引"（ioampg）所占用的頁數
```

```
    # 【譯註】一頁是 2KB。

    $used=&querysum(q{SELECT reserved_pgs(id,doampg)+reserved_pgs(id,ioampg)
                      FROM sysindexes
                      WHERE id != 8});

    # 一樣，但這次看的是日誌檔的用量
    $logused=&querysum(q{SELECT reserved_pgs(id, doampg)
                         FROM sysindexes
                         WHERE id=8});

    # 以長條圖顯示所得資訊
    &graph($db,$size,$logsize,$used,$logused);
}
$dbh->disconnect;

# 準備/執行 所給的單欄 SELECT 查詢，傳回查詢結果的加總值
sub querysum {
    my($query) = shift;
    my($sth,$aref,$sum);

    $sth = $dbh->prepare($query) or
      die "Unable to prepare $query: ".$dbh->errstr."\n";
    $sth->execute or
      die "Unable to exec $query: ".$dbh->errstr."\n";

    while ($aref=$sth->fetchrow_arrayref) {
        $sum += $aref->[0];
    }
    $sth->finish;

    $sum;
}

# 把所給的資料庫名稱、大小、日誌大小、用量資訊繪製成圖表

sub graph {
    my($dbname,$size,$logsize,$used,$logused) = @_;
```

```
    # 資料空間用量的長條圖
    print ' 'x15 . '|'.'d'x (50 *($used/$size)) .
          ' 'x (50-(50*($used/$size))) . '|';

    # 資料空間的使用百分比，以及資料量的總和（以 MB 為單位）
    printf("%.2f",($used/$size*100));
    print "%/". (($size * $pages)/1024)."MB\n";
    print $dbname.'-'x(14-length($dbname)).'-|'.(' 'x 49)."|\n";

    if (defined $logsize) { # 日誌空間用量的長條圖
        print ' 'x15 . '|' . 'l'x (50 *($logused/$logsize)) .
               ' 'x (50-(50*($logused/$logsize))) . '|';
        # 日誌空間的使用百分比，以及資料量的總和（以 MB 為單位）
        printf("%.2f",($logused/$logsize*100));
        print "%/". (($logsize * $pages)/1024)."MB\n";
    }
    else { # 有些資料庫沒有分離的日誌空間
         print ' 'x15 . "|- no log".(' 'x 41)."|\n";
    }
    print "\n";
}
```

SQL 高手可能會好奇，為什麼我用一個特別的函式（querysum()）來計算單欄內容的總和，而不使用 SQL 的 SUM 這個可以更完美達成任務的運算子。沒錯，就此例而言，使用 SUM 會比較簡潔些，而且 querysum() 看起來的確多餘，不過，我是為了提供一個更通用的解法，以便應付更複雜的狀況；舉例來說，如果我們需要依據正規式（regular expression）的結果另外計算一個持續變動的總和，這樣的事直接交給 Perl 處理，應該會比要求伺服器來執行要好（即使伺服器支援）。

在何處完成工作？

在 Perl 裡寫 SQL 程式時，常會出現一個問題：『這功能應該寫成 SQL 敘述，然後交給伺服器完成，或應該在用戶端以 Perl 運算？』，因為伺服器所提供的某些 SQL 函式（例如：SUM()），其功能用 Perl 的運算子往往也能達成。

舉例而言，使用 DISTINCT 關鍵字讓伺服器在傳回資料到 Perl 之前，先刪除掉有重複的部份，這樣看來似乎比較有效率，即使這樣的動作可用 Perl 輕易達成。

不幸的是，在決定要用那種作法時，有太多變因以及法則必須考量。以下是一些你或許需要考慮的因素：

- **伺服器處理某特定查詢的效率？**
- **涉及的資料量多寡？**
- **該項作業會動到多少資料量？以及該項作業的複雜度？**
- **伺服器、用戶端、網路（如果有的話）這三者的速度各有多快？**
- **你打算把你的程式移植到其它資料庫伺服器嗎？**

或許，與其耗費時間精力評估上述各種因素，不如花時間實際測試哪種作法比較好。

7.6.2 監控一個 SQL Server 的 CPU 健康狀況

本章末節的範例程式，將示範如何使用 DBI 秀出 SQL Server 上 CPU 的每分鐘使用率。為了使程式更有趣些，我打算在同一個程式裡同時監控兩個不同的伺服器。現在先看看程式碼再說，稍後會有這程式的說明：

```
use DBI;

$syadmin = "sa";
print "Sybase admin passwd: ";
chomp($sypw = <STDIN>);

$msadmin = "sa";
print "MS-SQL admin passwd: ";
chomp($mspw = <STDIN>);

# 連接到 Sybase Server
```

```
$sydbh = DBI->connect("dbi:Sybase:server=SYBASE",$syadmin,$sypw);
die "Unable to connect to sybase server: $DBI::errstr\n"
  unless (defined $sydbh);
# 打開 ChopBlanks 選項，移除各欄末端的空白。
$sydbh->{ChopBlanks} = 1;

# 連接到 MS-SQL Server（要偷懶的話，其實直接用 DBD::Sybase 也可以）
$msdbh = DBI->connect("dbi:Sybase:server=MSSQL",$msadmin,$mspw);
die "Unable to connect to mssql server: $DBI::errstr\n"
  unless (defined $msdbh);
# 打開 ChopBlanks 選項，移除各欄末端的空白。
$msdbh->{ChopBlanks} = 1;

$|=1; # 關閉 STDOUT I/O 緩衝

# 初始化 signal handler，以便有一個 "乾淨" 的開始
$SIG{INT} = sub {$byebye = 1;};

# 無限迴圈，直到我們的中斷旗標被設定為止
while (1) {
    last if ($byebye);

    # 執行 sp_monitor 預儲程序
    $systh = $sydbh->prepare(q{sp_monitor}) or
       die "Unable to prepare sy sp_monitor:".$sydbh->errstr."\n";
    $systh->execute or
       die "Unable to execute sy sp_monitor:".$sydbh->errstr."\n";
    # 用迴圈從輸出取得我們所需的行
    # 看到 cpu_busy 時，代表已取得所有資訊
    while($href = $systh->fetchrow_hashref or
          $systh->{syb_more_results}) {
             # 已取得所需要資料，停止要求
             last if (defined $href->{cpu_busy});
    }
    $systh->finish;

    # 除了 % 數字外，將我們收到的數值全部代換掉
    for (keys %{$href}) {
```

```
        $href->{$_} =~ s/.*-(\d+%)/\1/;
    }

    # 將我們所需要的資料收集到一行
    $info = "Sybase: (".$href->{cpu_busy}." CPU), ".
                "(".$href->{io_busy}." IO), ".
                "(".$href->{idle}." idle)   ";

    # 好了，現在對第二個伺服器（MS-SQL Server）再做一次同樣的事
    $mssth = $msdbh->prepare(q{sp_monitor}) or
       die "Unable to prepare ms sp_monitor:".$msdbh->errstr."\n";
    $mssth->execute or
       die "Unable to execute ms sp_monitor:".$msdbh->errstr."\n";
    while($href = $mssth->fetchrow_hashref or
            $mssth->{syb_more_results}) {
              # 夠了，可以停了
              last if (defined $href->{cpu_busy});
    }
    $mssth->finish;

    # 除了 % 數字外，將我們收到的數值全部代換掉
    for (keys %{$href}) {
        $href->{$_} =~ s/.*-(\d+%)/\1/;
    }

    $info .= "MSSQL: (" . $href->{'cpu_busy'}." CPU), ".
                "(".$href->{'io_busy'}." IO), ".
                "(".$href->{'idle'}." idle)";
    print " "x78,"\r";
    print $info,"\r";

    sleep(5) unless ($byebye);
}

# 會走到這裡，表示收到了中斷訊號而跳離迴圈
$sydbh->disconnect;
$msdbh->disconnect;
```

這程式會持續顯示一行類似這樣的字樣，每五分鐘更新一次：

```
Sybase: (33% CPU), (33% IO), (0% idle)  MSSQL: (0% CPU), (0% IO), (100% idle)
```

這個程式的關鍵是 sp_monitor **這個預儲程序，Sybase 與 MS-SQL Server 這兩者都有它。**sp_monitor **的輸出看起來像這樣：**

```
last_run                    current_run                 seconds
--------------------------- --------------------------- -----------
Aug 3 1998 12:05AM          Aug 3 1998 12:05AM          1

cpu_busy                  io_busy                   idle
------------------------- ------------------------- -------------------------
0(0)-0%                   0(0)-0%                   40335(0)-0%

packets_received          packets_sent              packet_errors
------------------------- ------------------------- -------------------------
1648(0)                   1635(0)                   0(0)

total_read          total_write         total_errors        connections
------------------- ------------------- ------------------- ------------------
391(0)              180(0)              0(0)                11(0)
```

不幸地，sp_monitor **有一項不具移植性的 Sybase 主義被延續到 MS-SQL：多組查詢結果集（multiple result set），也就是說，**sp_monitor **所傳回的每一行，都是以一組單獨 result set 形式呈現。**DBD::Sybase **藉由設定一項特殊的敘述屬性（statement attribute）來處理這問題，這也就是你會看到這樣測試的原因：**

```
while($href = $systh->fetchrow_hashref or
      $systh->{syb_more_results}) {
```

以及為何我們在一看到所要的欄時，就離開迴圈：

```
last if (defined $href->{cpu_busy});
```

這程式本身會持續轉迴圈，直到收到中斷信號（通常因為按下 Ctrl-C ）為止。當我們一收到此信號，我們在 signal handler 內做一件所能處理的最安全的事，並設定 exit 旗標。這是 *perlipc* manpage 對安全訊號處置（safe signal handling）所建議的技術。接收 INT **信號會設定一個旗標，導致迴圈在下一回合結束，攔截這信號讓程式在結束之前，還有機會先關掉它的資料庫代碼。**

這小程式勾勒出我們可以對伺服器進行怎樣的監控，運用從 sp_monitor 得到的結果繪出伺服器忙碌程度的的分時狀態圖，只是簡單的應用而已，至於如何應用到其它用途，就看你如何發揚光大了。

7.7 本章模組的資訊

模組	CPAN ID	版本
DBI	TIMB	1.13
Msql-Mysql 模組（DBD::mysql）	JWIED	1.2210
DBD::Sybase	MEWP	0.21
Win32::ODBC（由 http://www.roth.net 取得）	GBARR	970208

7.8 詳細資訊的參考來源

7.8.1 SQL

James Hoffman 在 http://w3.one.net/~jhoffman/sqltut.htm 發表了一篇極佳的 SQL 入門教材。該網站也提供了許多重要 SQL 網站的連結。

7.8.2 DBI

Sriram Srinivasan 所著的《Advanced Perl Programming》（中文版《Perl 高等程式設計》），（O'Reilly 於 1997 年出版）。

http://dbi.symbolstone.org/index.html 是 DBI 的正式網頁；這應該是你的第一站。

Alligator Descartes 與 Tim Bunce 合著的《Programming the Perl DBI》（O'Reilly 於 2000 出版）。

7.8.3 ODBC

http://www.microsoft.com/data/odbc/default.htm 提供了 Microsoft 的 ODBC 資訊。或者，你可以到 http://msdn.microsoft.com 找尋 ODBC 的資訊，仔細看看 MDAC SDK 關於 ODBC 程式庫方面的題材。

http://www.roth.net/perl/odbc/ 是 `Win32::ODBC` 的正式網頁。

《Win32 Perl Programming: The Standard Extensions》，作者是 Dave Roth（Macmillan Technical Publishing，1999）。這本書的作者也就是 `Win32::ODBC` 的原創者。這本書應該是目前最好的 Win32 環境 Perl 模組程式設計參考文件。

7.8.4 其它題材

Sybase 將它所有的線上文件都公佈在 http://www.sybase.com/support/manuals/，它有一個容易使用的搜尋與導覽界面。這不只對於 Sybase/MS-SQL Server 相關的問題有用，對於一般的 SQL 問題也很有用。

SybPerl 與 `DBD::Sybase` 的作者，Michael Peppler，他的個人網頁放在 http://www.mbay.net/~mpeppler/。這裡不僅提供與 Sybase 相關的有用資訊，而且對於 SQL 與一般資料庫程式設計方面的題材，也提供了相當豐富的資訊。