玩轉 SQL

從 MS-SQL 快速學習 OpenSource 資料庫

作者：陳威均

目錄

第一章
緣起及目標

是故百戰百勝，非善之善也；不戰而屈人之兵，善之善者也。

孫子兵法 謀攻篇

第一章：緣起及目標

這本書要談的主題，主要是筆者大力提倡的「以資料庫為開發核心」的開發理念。簡單的說，所謂的「以資料庫為開發核心」有 2 個主軸

1. 開發工具及程式語言，主要是負責使用者操作介面（User Interface）
2. 商業邏輯全部寫在資料庫中的 Store Procedure 及 function

開發企業資訊系統的方法非常多，每一個架構師都會有自己的一套設計思惟，可以說是百花齊放。筆者親身參與至少不同版本的企業資訊系統（ERP、CRM 等），從 DOS → Window 95 → Client Server → Web → Web 2.0 …
每一次更換開發語言（Clipper、VB5、Delphi、Visual Studio 2008…）
或是作業系統平台（DOS、Window 95、Web）
都必須被迫重寫所有的程式碼，重新經歷

圖 Visio

一開始還不覺得有什麼不對，反正也只能任由工具大廠擺佈，該重寫就重寫。當然，部份原因也是為整個開發工具集還不夠完善，經驗也還累積的不夠多。但是內心深處總隱隱覺得不對，不過也還是只能隨波逐流。即便到了 Client/Server 架構的時代，資料庫也都已經升級使用 MS-SQL，但也僅限於利用它來讀取及寫入資料，Store Procedure 那是什麼？好像有聽過，User Define Fujnction（UDF）？好像聽過，但跟它不熟。

這種情況持續了很多年，突然有一天，「當 Michael 坐在蘋果樹下沉思，被天上掉下來的蘋果砸到頭…」，當然，這是牛頓的故事。真實的情況是 Michael（就是筆者小弟我啦）有一天碰到了一個程式設計的天才，跟他討論一個困擾 Michael 很久的問題，

「如何大幅提昇 MRP 的展算速度」？

這個傢伙想了一下，和 Michael 仔細討論了計算的邏輯，隔天就展示了一段 Store Procedure，不試就算了，一試之下驚為天人，速度是原先寫程式跑迴圈的舊方法的 10 倍速不止，當下大為折服，從此就成了 SQL 的信徒。

「從此公主與王子就過著幸福快樂的日子…」

No、No、No，這不是結束，故事才剛開始。領教了 SQL 的高效，但由於績習難改，而且原系統已經完工，所以筆者並沒有全部改寫，而只是將幾段效率較差的程式碼（如月加權成本計算等）改寫為 Store Procedure，只算是入了門，但離大成還有漫漫長路。

如此，又過了幾年，筆者又參與了一個新專案的開發，這個專案計劃在一年中，將原本的 ERP 系統所有模組，移植到 Web 平台。由於在開發團隊中，熟悉 Web 開發的程式設計師嚴重不足，但是有幾位熟悉 SQL 的資深程式設計師，所以筆者被迫要想出最有效率的團隊工作模式，於是筆者從最根源思考，ERP 是什麼？

圖

經過仔細的思考、反覆推敲，最後就得出「以資料庫為開發核心」的設計思惟，並開始在新專案中實作。詳細的說明，請參考拙著「ERP 開發勝經」。簡單的說，就是把所有的商業邏輯及報表的取值都寫成 Store Procedure、View 及 User Define Function，在這樣的架構下，程式開發工具主要的任務就是開發使用者操作介面，複雜的商業邏輯則交給資深的系統設計或分析師寫成 Store Procedure 供程式設計師呼叫使用。

這樣切割的最大優點，就是前端介面可以任意更換，您可以用 VB6，Delphi7，也可以用 Visual Studio 2012、Java、PHP 等你所慣用（擅長）的任何語言，只要該工具提供和後端資料庫連結的方式即可。而且依目前資料庫發展的態勢來看，至少 20 年內（甚至再久一點），資料庫一定還是資料儲存的要角。投資在上面，比起每 5-10 年就要全面重新改寫的舊思維，看起來有前途的多。

因為筆者個人的 SQL 相關經驗都是以 MS-SQL 的 T-SQL（Since MS-SQL7.0）為主所以對 T-SQL 的了解當然就會比較深入，加上後來 MicroSoft 佛心來著，推出了 express 的免費版本，雖有限制，但對小型企業和測試、展示來說已經足夠，這的確是很大的福音。但是，如果是有大資料量的需求及效能考慮的企業，還是建議購買商業版本，這就有了成本的考量，當然，如果客戶的情況許可，筆者都會建議購買商業授權版本。

隨著 Open Source 各項專案的風起雲湧，各式各樣的應用程式如雨後春筍般的冒出來，其中不乏另人讚不絕口的傑出產品。而在資料庫管理程式的領域，也出現了許多優秀的產品，其中以 MySQL、PostgreSQL 及 FireBird 等更是其中的翹楚。很多人都以為 Open Source = FREE（免費），這事實上是個誤解。的確，有一部份的 Open Source 產品是完全免費的（即使是商業使用），但大部份的 Open Source 都有商業版本或開發版本的授權使用費用，只是通常和大型商業軟體比起來，費用相對低很多，在使用前請詳細的閱讀了解該產品的授權方式。本書所介紹的三個產品，其授權方式列示如下，詳細的內容，還是以各產品的說明文準。

	開發組織、公司	Licence 方式	商業使用
MySQL	MySQL AB（Oracle 旗下子公司）	GPL	企業授權
PostgreSQL	PostgreSQL Global Development Group	BSD	免費
FireBird	Firebird Foundation	Initial Developer's Public License (variants of the Mozilla Public Licence V.1.1)	免費

至於其他的細節，請參閱第二章的說明。

由於 Open Source 開源碼的資料庫管理程式日漸成熟，功能甚至不輸一般的商業版本，基於雞蛋不應該只是放在同一個籃子的觀念，更重要的是，筆者提暢的「以資料庫為開發核心」思惟，既然是核心主軸，就不應該只限定在單一 SQL 管理程式，而應該是通用的概念。基於以上的原因，就有必要一併了解其它的資料庫管理程式。

不同的程式開發工具，如 VB6，Delphi7，VS2012 等，各有其各自不同的程式語法，你不可能把 Delphi 的程式碼移到 VB6 去使用、執行，當然您可以試著手動修改，不過難度頗高。然而，資料庫中的 SQL 語法相較之下，是有標準的，基本上，只要能遵守 ANSI-92 以上的標準，大部份的 SQL 語法都可以相通。各家資料庫產品的 SQL 語法都會參考這個標準，再擴充進階的功能而成。這也就是說，只要寫 SQL 程式時，盡量都使用標準語法，在不同的資料庫管理程式中，幾乎都可以直接執行。

說實在的，這真是一個絕妙的主意，因為一旦有了標準，相關廠商就必須遵詢，否則就會被市場淘汰，如此就可以確保 SQL 程式的可移執性。程式設計是個非常勞心的工作，不斷的有學習不完的新技術會冒出來，同時又得面臨急迫的專案時程。有時候必須在很短的時間內熟悉一些重要的技術。本書就因此應運而生。希望能提供一個快速理解 Open Source 資料庫的捷徑，當然，如果要深入的探討，還是要參考進階的相關書籍。

要熟悉一種新的開發語言，除了基本語法的學習必不可免，透過大量閱讀精湛的範例程式碼，更是非常有效的學習方法。SQL 語法非常的簡潔，並不複雜，一般的程式設計人員通常都有一些基本的認識，但是要精通就不是件容易的事，觀念的轉換是最難打通的一關，需要長時間經驗的累績。所謂「觀念的打通」，主要指的是「捨迴圈而用批次」。大部份的程式設計師在處理大量資料的運算時，習慣性的會使用迴圈一筆一筆的累計計算，這在資料量少時，不會有問題，但隨著輸入資料的日漸增多，執行的效率就會越來越差，通常系統上線的初期，使用者還沒有感覺，但使用一段時間後，就會開始反應系統越來越慢，有時還會像當機一樣。而當程式師檢查程式碼時，程式通常也沒有錯誤，常會歸咎於網路不穩定，使用者操作習慣不良… 等未知的原因。

SQL 的最大優勢是在處理大量資料的運算處理，只要 SQL 語法寫的好，百萬筆甚至千萬筆的資料都不是太大的問題（當然，硬體配備也不能太差）。本書並不是一本 SQL 的入門書籍，筆者的主要目的，是要提供一個快速有效的學習 Open Source 開源碼資料庫管理程式的方法。但前提是已經對 MS-SQL 已經有一定程度理解的相關開發人員。如果您對 T-SQL 的撰寫並不精通，您可以參考一下本書第七章及第八章的內容，或是先找一些相關的介紹書籍，否則閱讀本書會有一些障礙。筆者個人推薦楊志強老師的相關著作，楊老師的著作文筆簡斂、範例說明清楚明白，筆者也從中獲益良多。

本書所用的方法，是將一個 MS-SQL 的範例資料庫，內含 ERP 系統具代表性的 Table、Store Procedure、Function、Trigger 等，將其完整的轉換為其他的資料庫（MySQL、PostgreSQL、FireBird）

並包含其資料的轉入。並在這個轉換的過程中，介紹不同資料庫 SQL 語法及欄位定義的差異。最後會簡單的實測各資料庫的效能。

這樣的方式，可以透過範例資料庫的實際內容及程式碼，了解實務上的程式寫法，讀者可以再自行延伸為各種不同的應用方式。對於沒有大量運用 SQL 經驗的程式開發人員，應該會有很大的幫助。

當然，誠如本書先前所說，這是一個快速理解的捷徑，重點在介紹主要的 SQL 語法及基本的操作，至於各自資料庫的更深層應用，及其專有的擴充語法，則不在本書的討論內容。對實際運用而言，這應該已經足夠，不足的部份只要在配合 Google 大神的提示即可。

企業資訊系統有三個重要的評估指標
Performance
Security
User Friendly
其中效能的部份，絕大部份都和資料庫相關，包含
Tables 的設計及關連
SQL 語法的正確撰寫
本書的最後一章會以轉換好的資料庫（完全相同資料結構及內容），實際測試各資料庫的效能，這當然不是非常得精確，但是應該還有一定的參考價值。

越是深入的了解開放源碼的幾個優秀的資料庫管理程式，就越是讚嘆。經過多年的發展，開放源碼已經成為軟體產業不可忽視的重要角色，和傳統的商業軟體已成分庭抗禮之勢。一些優秀的產品早就已經廣泛的實際運用在各個不同的領域。本書所介紹的只是跟資料庫相關的幾個代表性產品，日後若有因緣，筆者也計劃多介紹一些其他類型的開源碼產品。

如果這本書能在您評估開源碼資料庫管理軟體程式時，提供一些幫助及參考，甚至認真的考慮採用它為企業的核心架構之一，那就不枉筆者提筆寫這本書的初衷。

第二章
開源碼資料庫簡介及發展史

故兵聞拙速，未睹巧之久也。
夫兵久而國利者，未之有也。
故不盡知用兵之害者，則不能盡知用兵之利也。

孫子兵法 作戰篇

第二章－開源碼資料庫簡介及發展史

要談關聯式資料庫，就不能不提「關係資料庫之父」埃德加·弗蘭克·科德（EdgarF.Codd，1923－2003）。1970 年，科德發表題為「大型共享資料庫的關係模型」的論文，文中首次提出了資料庫的關係模型。由於關係模型簡單明瞭、具有堅實的數學理論基礎，所以一經推出就受到了學術界和產業界的高度重視和廣泛響應，並很快成為資料庫市場的主流。20 世紀 80 年代以來，計算機廠商推出的資料庫管理系統幾乎都支持關係模型，資料庫領域當前的研究工作大都以關係模型為基礎。

科德十二定律

資料來源：http://zh.wikipedia.org/wiki/科德十二定律

科德十二定律（Codd's 12 rules）是由資料庫的關係模型的先驅埃德加·科德（Edgar F. Codd）提出的，使資料庫管理系統關係化需滿足的十三條（從 0 至 12）準則。又稱為「黃金十二定律」。

全關係系統十二準則

全關係系統應該完全支持關係模型的所有特徵。關係模型的奠基人埃德加·科德具體地給出了全關係系統應遵循的基本準則。

準則 0

一個關係形的關係資料庫系統必須能完全通過它的關係能力來管理資料庫

準則 1 信息準則

關係資料庫系統的所有信息都應該在邏輯一級上用表中的值這一種方法顯式的表示。

準則 2 保證訪問準則

依靠表名、主碼和列名的組合，保證能以邏輯方式訪問關係資料庫中的每個數據項。

準則 3 空值的系統化處理

全關係的關係資料庫系統支持空值的概念，並用系統化的方法處理空值。

準則 4 基於關係模型的動態的聯機數據字典

資料庫的描述在邏輯級上和普通數據採用同樣的表述方式。

準則 5 統一的數據子語言

一個關係資料庫系統可以具有幾種語言和多種終端訪問方式，但必須有一種語言，它的語句可以表示為嚴格語法規定的字元串，並能全面的支持各種規則。

準則 6 視圖更新準則

所有理論上可更新的視圖也應該允許由系統更新。

準則 7 高級的插入、修改和刪除操作

系統應該對各種操作進行查詢優化。

準則 8 數據的物理獨立性

無論資料庫的數據在存儲表示或存取方法上作任何變化，應用程序和終端活動都保持邏輯上的不變性。

準則 9 數據邏輯獨立性

當對基本關係進行理論上信息不受損害的任何改變時，應用程序和終端活動都保持邏輯上的不變性。

準則 10 數據完整的獨立性

關係資料庫的完整性約束條件必須是用資料庫語言定義並存儲在數據字典中的。

準則 11 分布獨立性

關係資料庫系統在引入分布數據或數據重新分布時保持邏輯不變。

準則 12 無破壞準則

如果一個關係資料庫系統具有一個低級語言，那麼這個低級語言不能違背或繞過完整性準則。

關於開放原始碼的說明，本書就不再贅述，如果有興趣多瞭解，請參見

http://www.twwiki.com/wiki/開放源代碼軟體

有詳細的說明。

簡單的說，開放原始碼當然是說該軟體會隨附原始程式碼，但並不表示它是免費的，要看它的授權條款，一般而言，除非是屬於基礎系統（如作業系統 Linux、瀏覽器 FireFox 等），否則一般的商業應用軟體，通常還是會有商業授權的費用。否則僅憑開發團隊的開發熱情，是很難長久維運的。

以下簡單的談談本書提到的三套開源碼資料庫管理系統，它們的歷史、開發團隊及發展史。

MySQL

http://www.mysql.com/

原本是一個開放原始碼的關聯式資料庫管理系統，原開發者為瑞典的 MySQL AB 公司，該公司於 2008 年被昇陽微系統（Sun Microsystems）收購。2009 年，甲骨文公司（Oracle）收購昇陽微系統公司，MySQL 成為 Oracle 旗下產品。MySQL 被廣泛地應用在 Internet 上的中小型網站中。由於其體積小、速度快、總體擁有成本低，尤其是開放源碼這一特點，許多中小型網站為了降低網站總體擁有成本而選擇了 MySQL 作為網站資料庫。隨著 MySQL 的不斷成熟，它也逐漸用於更多大規模網站和應用，比如維基百科、Google 和 Facebook 等網站。非常流行的開源軟體組合 LAMP 中的「M」指的就是 MySQL。

MySQL 使用 C 和 C++編寫，並使用了多種編譯器進行測試，保證原始碼的可移植性。支援 AIX、BSDi、FreeBSD、HP-UX、Linux、Mac OS、Novell NetWare、NetBSD、OpenBSD、OS/2 Wrap、Solaris、Windows 等多種作業系統。為多種程式語言提供了 API。這些程式語言包括 C、C++、C#、VB.NET、Delphi、Eiffel、Java、Perl、PHP、Python、Ruby 和 Tcl 等。

MySQL 原先被人所詬病的一些缺點，如不支援 Store Procedure 等必要的進階功能，在 V5.0 以後的版本都已經提供。但自從被甲骨文公司收購後，Oracle 大幅調漲 MySQL 商業版的售價，

且甲骨文公司不再支援另一個自由軟體計畫 OpenSolaris 的發展，因此導致自由軟體社群們對於 Oracle 是否還會持續支援 MySQL 社群版（MySQL 之中唯一的免費版本）有所隱憂，因此原先一些使用 MySQL 的開源軟體逐漸轉向其它的資料庫。例如維基百科已於 2013 年正式宣布將從 MySQL 遷移到 MariaDB 資料庫。

PostgreSQL
http://www.postgresql.org/
https://sites.google.com/site/mammoth/

PostgreSQL 經歷了長時間的演變，開始於在 UC Berkeley（**柏克萊加州大學**）的 Ingres 計劃。這個計劃的領導者 Michael Stonebraker 在 1982 年離開 Berkeley 去商業化 Ingres，但是最後還是返回了學術界。在 1985 年返回 Berkeley 之後，Stonebraker 開始了 post-Ingres 計劃來致力於在 1980 年代早期變得日益清楚的、當代資料庫系統的問題。Postgres 和 Ingres 的代碼庫開始（並保持）完全分離了。

從 1986 年開始計畫組發表了一些描述系統基本原理的論文，並在 1988 年這項計劃建成並執行了一個原型版本。計畫組在 1989 年六月向少數用戶發行了版本 1，隨後在 1990 年六月發行了帶有重寫後的規則系統的版本 2。1991 年的版本 3 再次重寫了規則系統，並增加了對多個儲存管理器和改進的查詢引擎的支援。在 1993 年就有大量的用戶存在了，並開始用對支援和特徵的要求淹沒這個計劃。在發行了主要作為最後清理的版本 4 之後計劃就終止了。

儘管 Postgres 計劃正式的終止了，BSD 授權條款（Berkeley 在其下發行的 Postgres）卻使開放原始碼開發者獲得複本並進一步開發系統。在 1994 年兩個 UC Berkeley 大學的研究生，Andrew Yu 和 Jolly Chen，增加了一個 SQL 語言直譯器來替代早先的基於 Ingres 的 QUEL 系統，建立了 Postgres95。代碼隨後被發行到 web 上來在世界上找尋它自己的出路。在 1996 年計劃被重新命名了：為了反映資料庫的新 SQL 查詢語言，Postgres95 變成了 PostgreSQL。

第一次 PostgreSQL 發行形成了版本 6.0。隨後來自世界各地的一組資料庫開發者和志願者，透過 Internet 協作起來，維護著這套軟體。自從版本 6.0 之後，出現了很多後續發行，在系統中也出現了很多改進；在 2005 年 1 月 19 日，版本 8.0 成為當前發行。由 8.0 後，PostgreSQL 以原生（Native）的方式，執行於 Windows 視窗系統。

在 2005 年中，兩間其他的公司宣佈商業化 PostgreSQL，分別進入不同的利基市場。EnterpriseDB 宣布將專注於讓使用 Oracle 的應用程式能更容易的在 PostgreSQL 上運行。Greenplum 則專注貢獻在資料倉儲和商業智慧的應用程式，尤其以 BizGres 計畫著稱。

至於 PostgreSQL 計畫本身，他繼續著每年一個主要版本發佈，以及次要的除

錯版本發佈，全都可以在 BSD 授權底下取得。這些都是基於商業化廠商、支援公司、和開放原始碼駭客。

FireBird
http://www.firebirdsql.org/

FireBird 的前身是 Borland 推出的 Interbase 資料庫，並將其在 2000 年開放原始碼的免費資料庫軟體，目前版本是 2.5.3。它比 MySQL, PostgreSQL 更適合於 Windows 作業系統，它並不是使用 cygwin 的方式移植到 windows 上，而是真正的 Windows 軟體，我們一般將其比較於 MS-SQL，是一個小而美資料庫，最大的優點在於它是免費的，其實它也支援不同作業平台(含 Linux)，不過對於 Linux 筆者還是建議大家儘量還是多考慮 PostgreSQL 或是 MySQL，因為 Firebird 若要用 php 的話就有點麻煩了。

FireBird 的長期發展取決於使用者的自願捐款。可以通過贊助 FireBird 火鳥基金會成員，以贊助它的長期發展。

以下簡單的表列特色及差異
http://db-engines.com/en/system/Firebird%3BMySQL%3BPostgreSQL

	Firebird	MySQL	PostgreSQL
簡述	FireBird 的前身是 Borland 推出的 Interbase 資料庫	Widely used open source RDBMS	Based on the object relational DBMS Postgres
網站	www.firebirdsql.org	www.mysql.com	www.postgresql.org
技術文檔	www.firebirdsql.org/en/reference-manuals	dev.mysql.com/doc	www.postgresql.org/docs/manuals
開發組織	Firebird Foundation	Oracle	PostgreSQL Global Development Group
初次發佈	2000	1995	1989
開發語言	C and C++	C and C++	C
支援 OS	AIX FreeBSD HP-UX Linux OS X server-less Solaris Unix	FreeBSD Linux OS X Solaris Windows	HP-UX Linux OS X Solaris Unix Windows

	Windows		
API	C/C++ API OLE DB ADO.NET JDBC ODBC	ADO.NET JDBC ODBC	native C library streaming API for large objects ADO.NET JDBC ODBC
支援程式語言	C C++ Delphi Java JavaScript Perl PHP Python Ruby	Ada C C# C++ Java Objective-C OCaml Perl PHP Python Ruby Scheme Tcl	.Net C C++ Java Perl Python Tcl

由於 FireBird 相關的資訊及使用人數較少，所以稍後的章節會較少提及此資料庫，有興趣的童鞋請自行上官網查詢。

第三章
DataBase.NET 的操作說明

知人者智，自知者明；
勝人者有力，自勝者強；
知足者富，強行者有志；
不失其所者久，死而不亡者壽。

老子道德經

第三章 DataBase.NET 的操作說明

在後面幾個章節，筆者會分別介紹三套開源碼資料庫管理軟體的安裝及操作，在這之前，先介紹一款好用的多資料庫操作軟體，這款軟體神奇之處在於

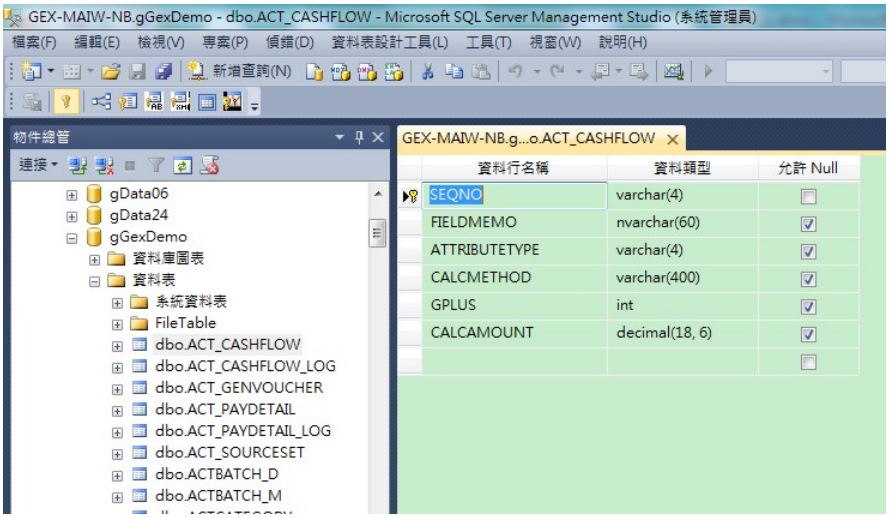
1. 作者是優秀的台灣子弟。
2. 它是綠色軟體，不必安裝直接就可以使用。
3. 它可以用單一介面，操作管理超過 20 個以上不同的資料庫。
4. 操作非常的簡單，幾乎不必學習，下載後就可以立刻上手。

由於本書的目標是要快速學習多套開源碼的重要概念及 SQL 語法差異，原本每一套資料庫管理軟體都有各自的操作介面，例如

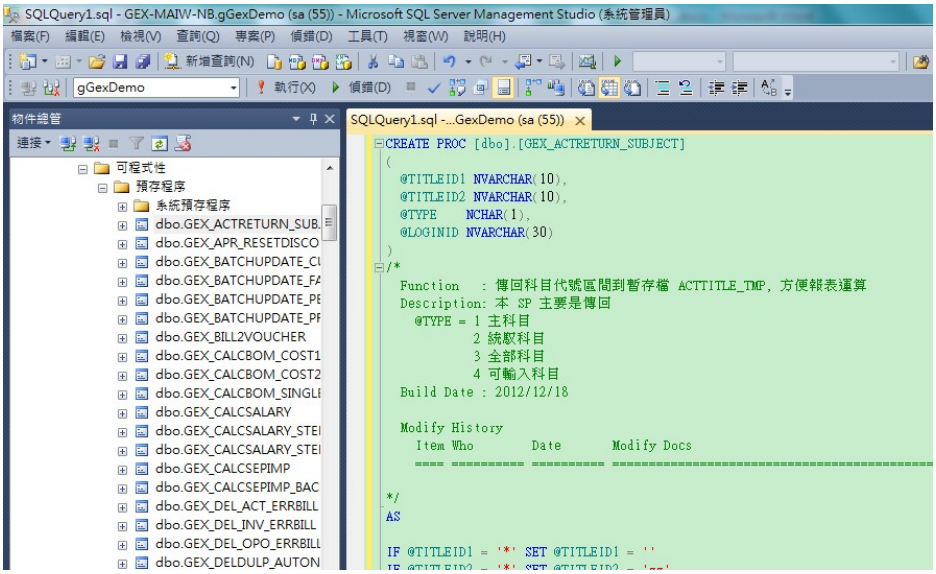
MS-SQL

連接資料庫

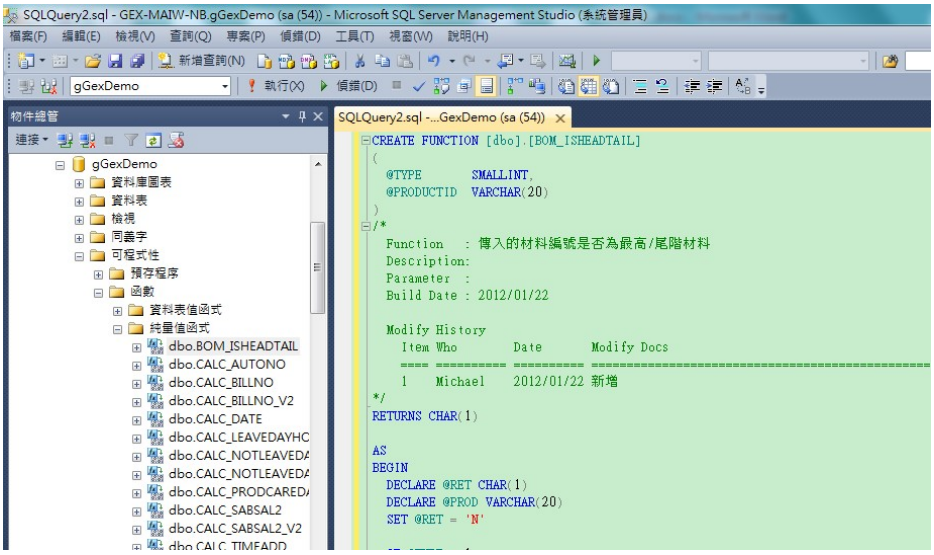
資料庫欄位的設計及維護



Store Procedure （預存程序）的維護



自訂函式的維護（User Defune Function）

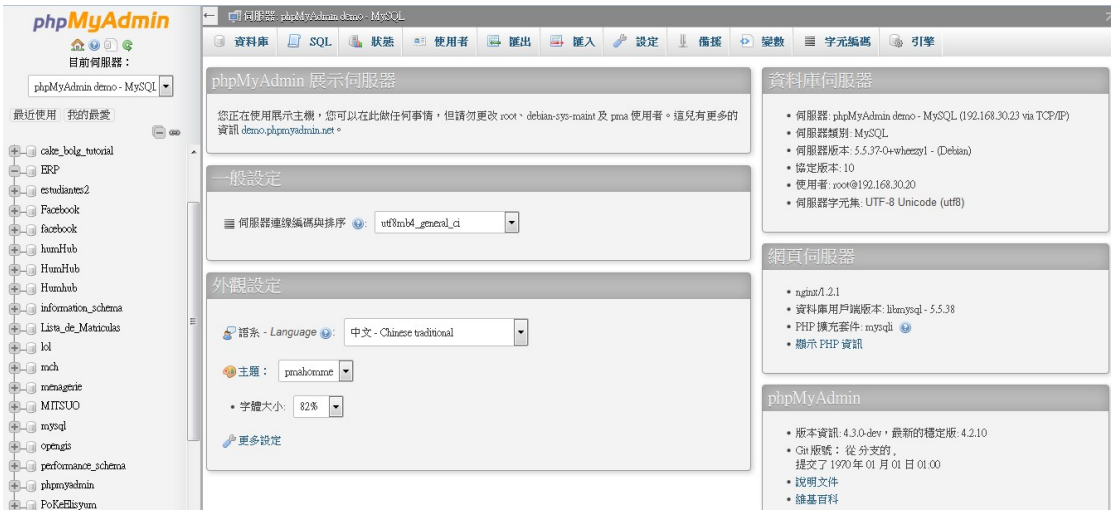


MySQL

1. 圖形管理工具 MySQL Workbench（官方版本）
2. Navicat 導航貓 for MySQL 是一套專為 MySQL 設計的強大資料庫管理及開發工具。

3. phpMyAdmin 是由 PHP 寫成的 MySQL 資料庫系統管理程式，讓管理者可用 Web 介面管理 MySQL 資料庫。

phpMyAdmin 的網頁式操作首頁



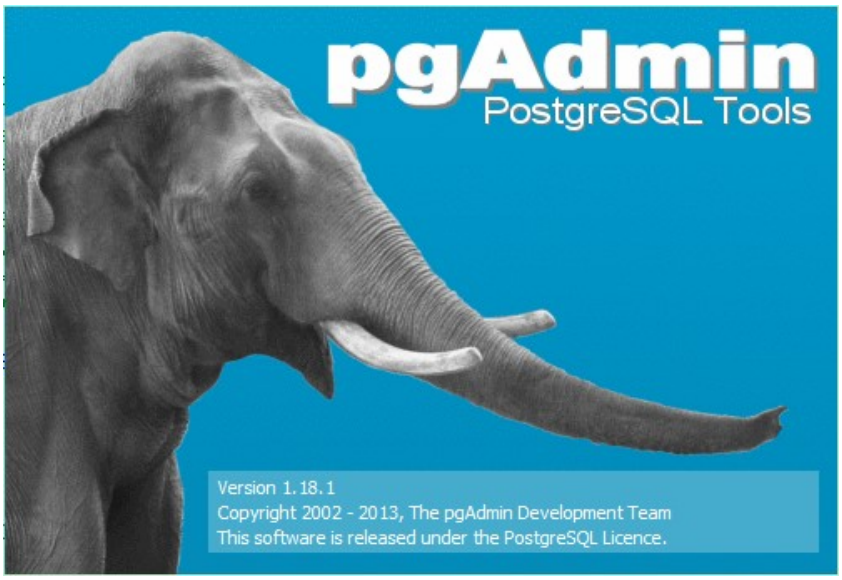
建立新的資料庫

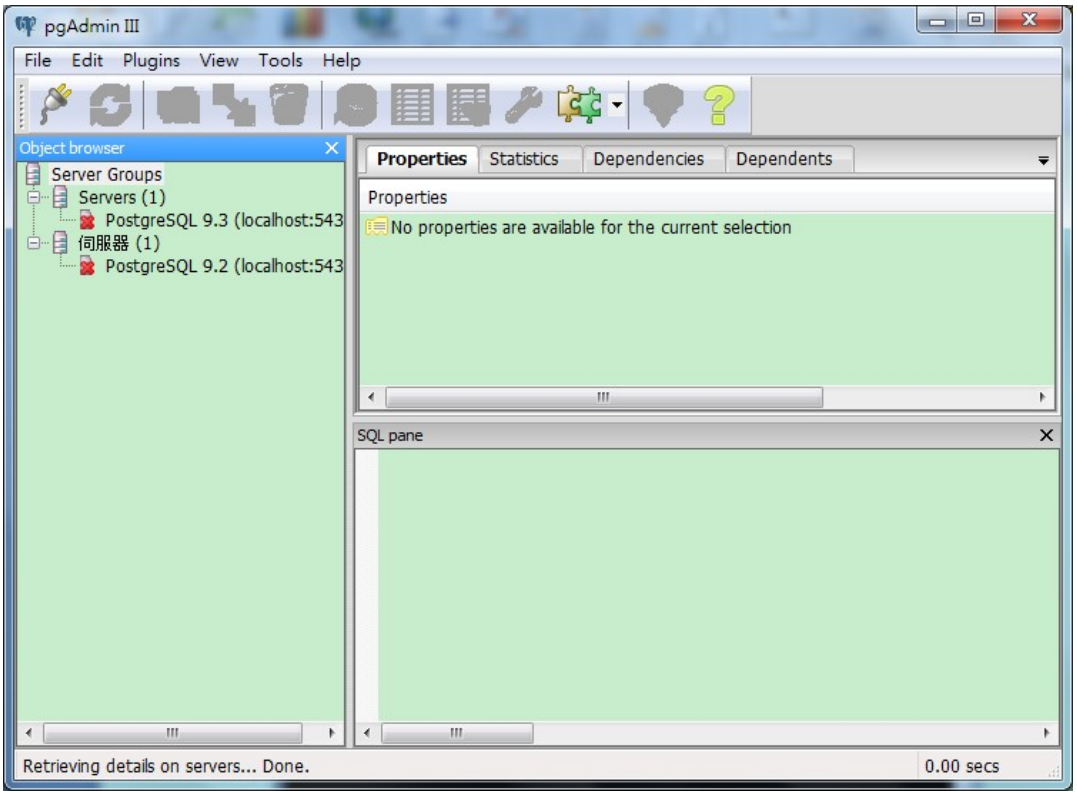
建立新的 Table

phpMyAdmin 應該是最多人使用的圖形化的管理介面，這當然和 php 配合 MySQL 是最多網站的首選開發組合有關。

PostgreSQL

1. Navicat 導航貓 for PostgreSQL 是一套專為 PostgreSQL 設計的強大資料庫管理及開發工具。
2. phpPgAdmin 基於 php 語言寫的用於管理 PostgreSQL 資料庫的程式，它類似於 phpMyAdmin 在 MySQL 的角色。
3. PgAdmin 另外一個用於管理 PostgreSQL 資料庫的軟體。

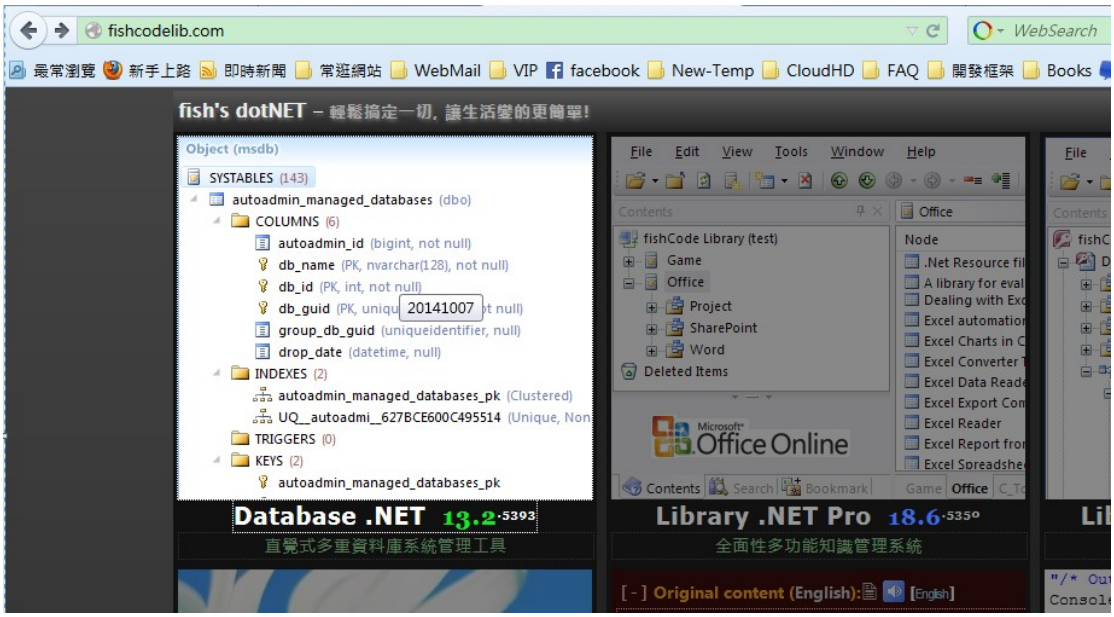


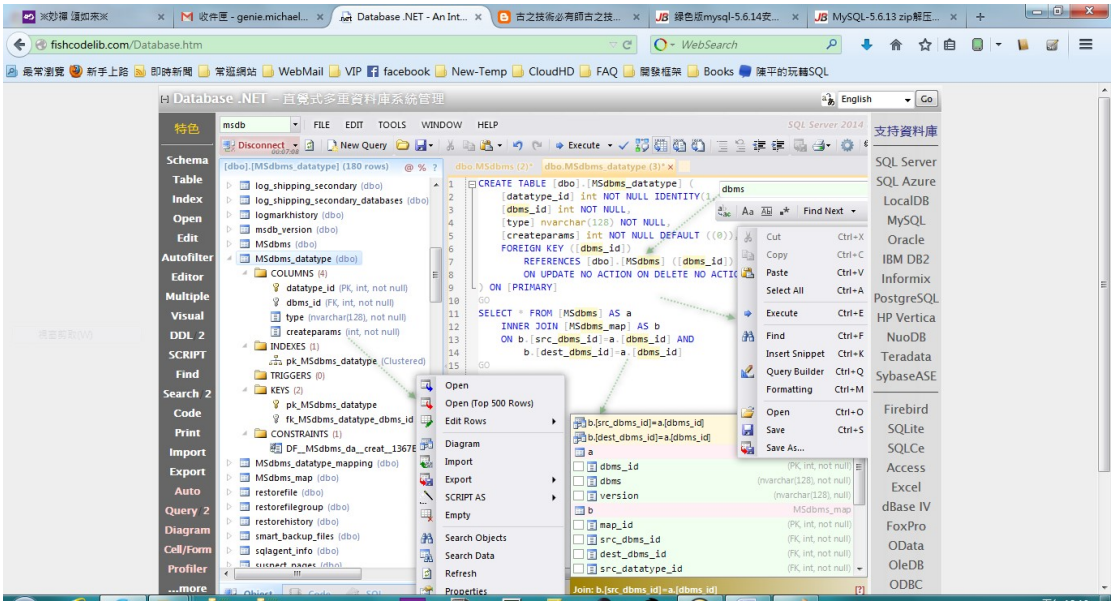


每一套資料庫程式都會有一套（甚至多套）資料庫的圖形管理程式，老是用 Console 模式實在是很累人。這些程式雖然都寫的蠻好的的，不過又是安裝，又要學習使用，像筆者這樣沒事只是抓幾個頁面，書又灌水了好幾頁，自己都覺得不好意思。後來，筆者在一次衝浪的過程中，看到了 DataBase.NET 這個程式，真是覺得 fish 老大真是佛心來著，一支程式就搞定所有的資料庫，仔細再看看所有的操作超直覺，跟本書的主題簡直是完美的搭配。個人非商業使用免費，即使是購買授權，價格也是物超所值，這樣的好物，不用會遭天譴的。

接下來，不囉嗦，就來看看這個超棒的工具
網址　http://fishcodelib.com/
看來 fish 老大的好東西不少，除了 DataBase.NET 外還有

Library.NET Pro
Library.NET Free
Capture.NET
等等，請自己逛逛

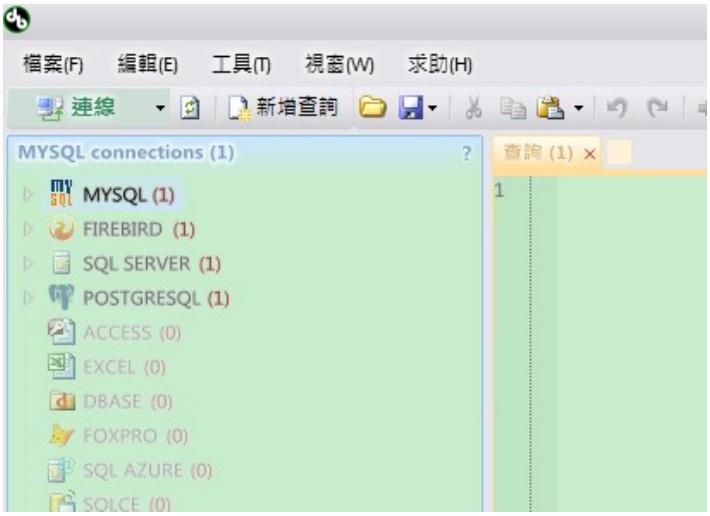




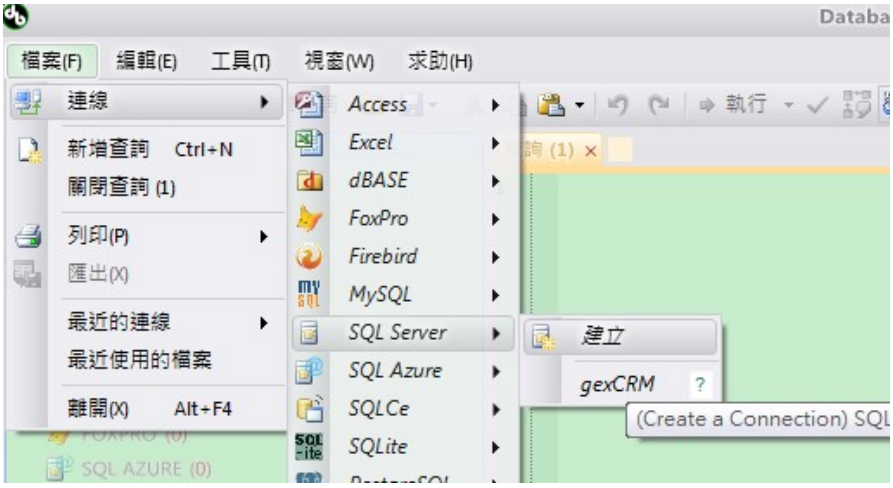
下載後，解壓縮到任一個目錄，就可以開始使用。為了方便操作，我在把捷徑拉到工具列。

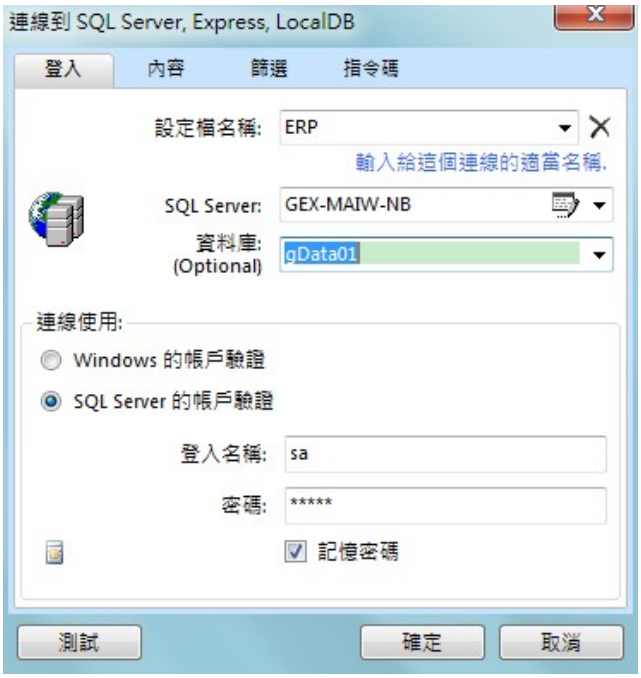
啟動 DataBase.NET

然後就進到主頁面了

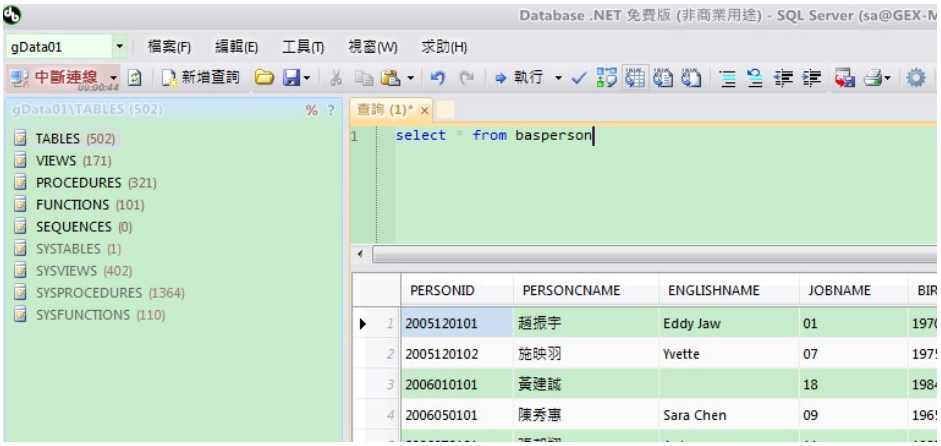


如果要連 MS-SQL，就選 SQL Server－建立 （會新增一個連線）

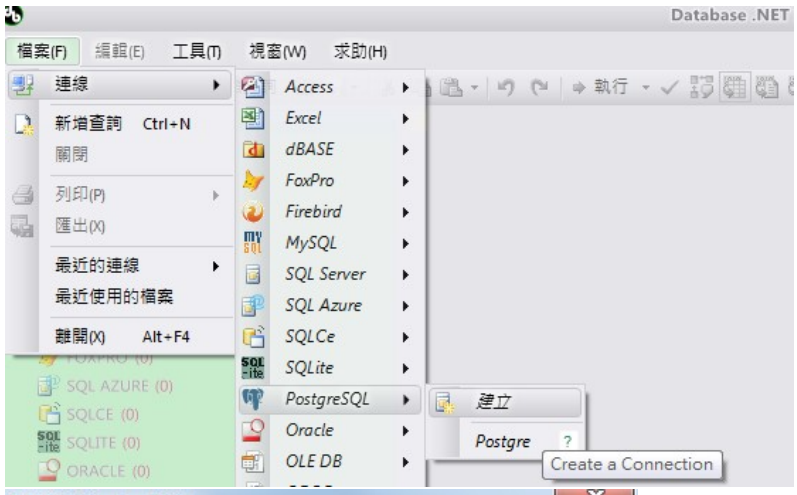


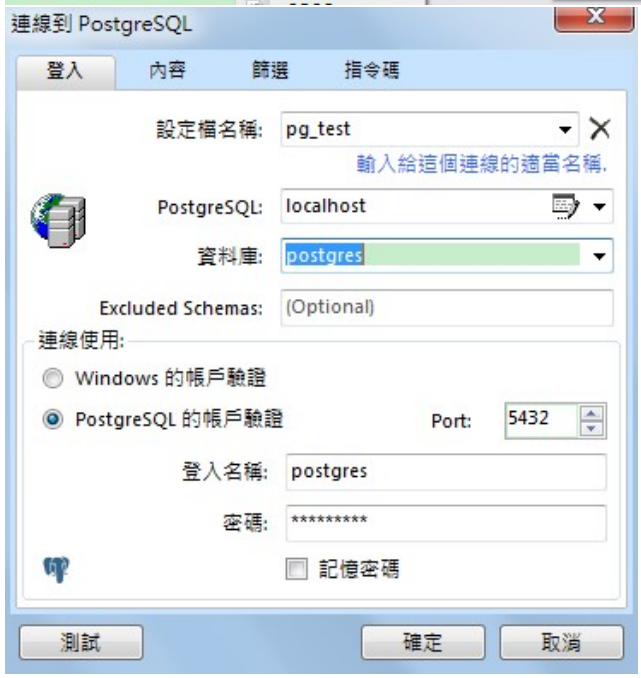


連接好 Database 後，其他的操作就一切如昔

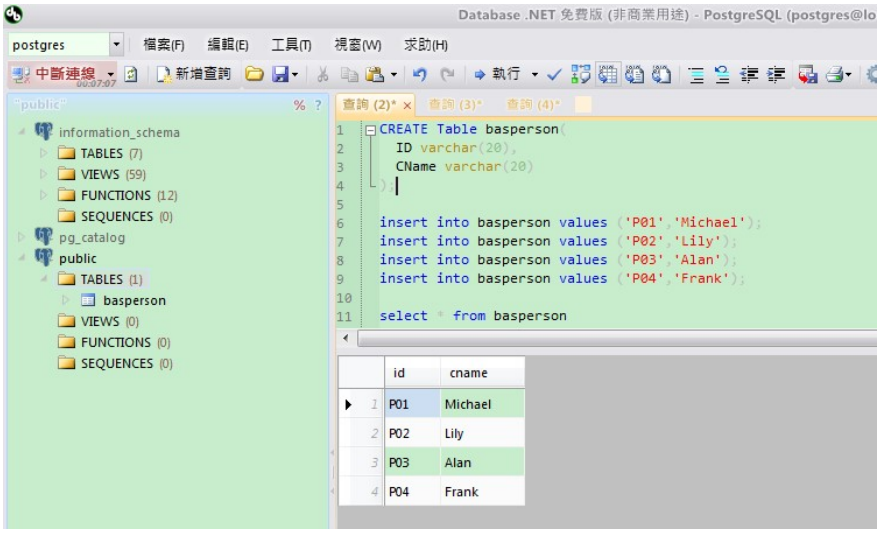


連接 PostgreSQL 基本上同前面的步驟，只有帳號、密碼會有點差異

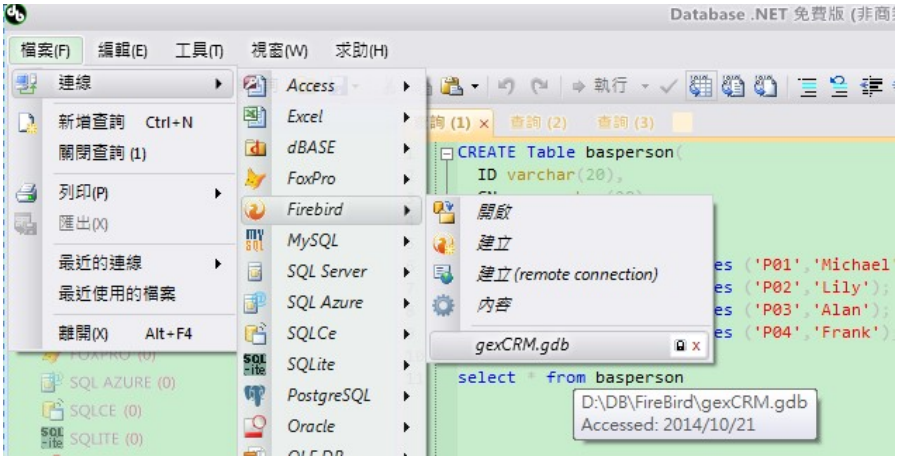

Database .NET
檔案(F)
編輯(E)
工具(T)
視窗(W)
求助(H)
連線
新增查詢 Ctrl+N
關閉
列印(P)
匯出(X)
最近的連線
最近使用的檔案
離開(X) Alt+F4
Access
Excel
dBASE
FoxPro
Firebird
MySQL
SQL Server
SQL Azure
SQLCe
SQLite
PostgreSQL
Oracle
OLE DB
建立
Postgre
Create a Connection
SQL AZURE (0)
SQLCE (0)
SQLITE (0)
ORACLE (0)

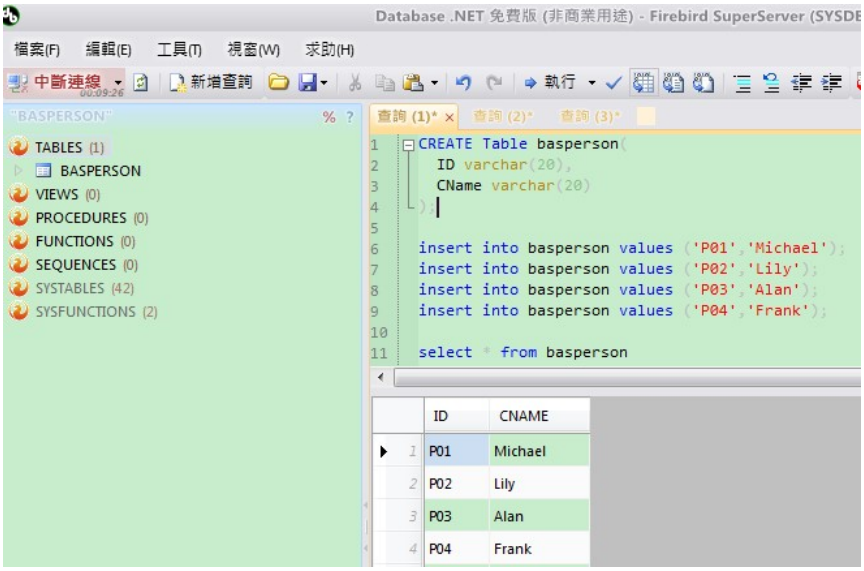

連線到 PostgreSQL
登入
內容
篩選
指令碼
設定檔名稱:
pg_test
輸入給這個連線的適當名稱.
PostgreSQL:
localhost
資料庫:
postgres
Excluded Schemas:
(Optional)
連線使用:
Windows 的帳戶驗證
PostgreSQL 的帳戶驗證
Port:
5432
登入名稱:
postgres
密碼:
****
記憶密碼
測試
確定
取消

測試一下幾個簡單的 SQL 指令

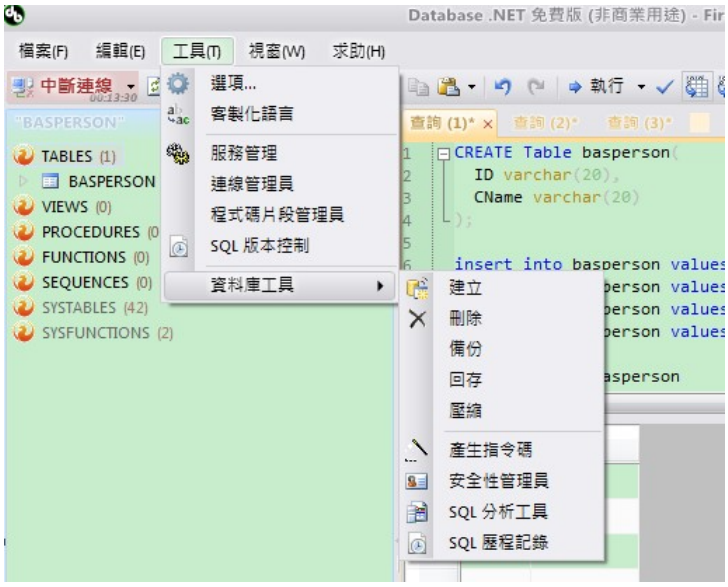


連接 MySQL 的步驟和程序大致都是一樣的，都是先建立一個資料庫的連線，連線後就看要執行哪些資料庫的操作。





另外，Database.NET 除了基本的連線資料，執行 SQL 外，還有提供諸多常用的工具，例如備份資料庫、創建新資料庫等功能，請自行測試使用。



因為都是透過 SQL 語法，所以基本上只要遵循標準的 SQL 語法（ANSI-92 以後的標準），基本上可以通吃。但是天下事總不盡如人意，有時後總會碰到一些特殊的需求，要用到一些特殊的語法，這時就只好認真的 K 一下各資料庫的使用手冊了。本書的目的是快速的瞭解不同資料庫的差異，但是不同的資料庫總會有不同的特色及優缺點，所以太特殊的語法就不在討論的範圍內。

Database.NET 是一個優秀的工具，基本上只要對資料庫有一點基本的概念，應該就可以應用自如。當然，要使用這個程式前，必須先將資料庫管理程式安裝好，在接下來的幾個章節，筆者會簡單的帶一下安裝的注意事項，基本上都是 Next → Next 就可以了。如果您之前沒有裝過 MySQL，筆者建議您先看一下說明再安裝，MySQL 的安裝程式防錯能力極差，不曉得 Oracle 是故意的還是怎樣，一旦出錯，連重新安裝都很麻煩。

可能是老天聽到了廣大使用者的心聲，原 MySQL 的開發團隊，又另起了一個 MariaDB 的新專案，號稱幾乎完全相容於 MySQL，經筆者測試使用後，發現此言不虛，大部分的應用程式都不必修改程式，就可以直接使用。連在 Node.js 平台，資料庫安裝 MariaDB，然後其他的 npm 相關的連線程式(套件)都仍安裝 MySQL，到目前為止還沒發現問題。更棒的是，筆者安裝 MySQL 測試時，常常會莫名其妙就裝到當機，但是 MariaDB 到目前為止還沒發生裝不起來的情況。再加上 Oracle 未來的商業授權等不確定性，筆者會建議各位可以大膽的升級到 MariaDB，您會發現真的是無痛升級。

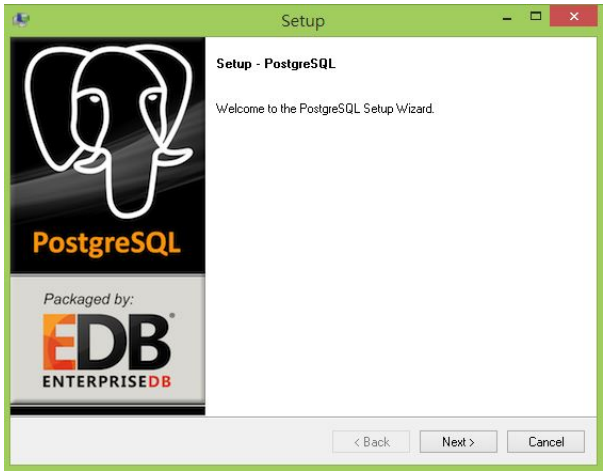
第四章
PostgreSQL 的安裝及基本操作

夫兵形象水，水之形避高而趨下，兵之形，避實而擊虛，
水因地而制流，兵應敵而制勝。故兵無常勢，水無常形，
能因敵變化而取勝者，謂之神。

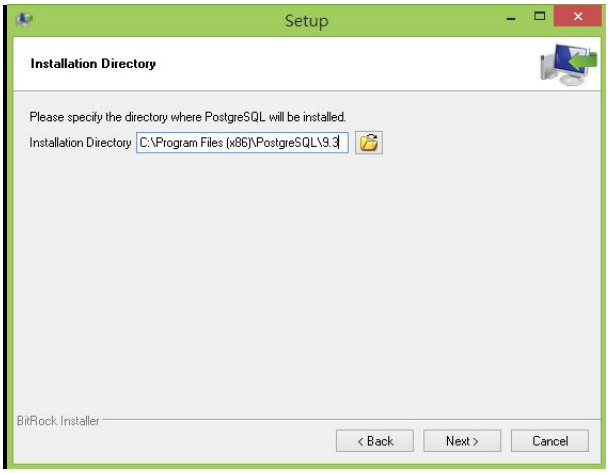
孫子兵法 虛實篇

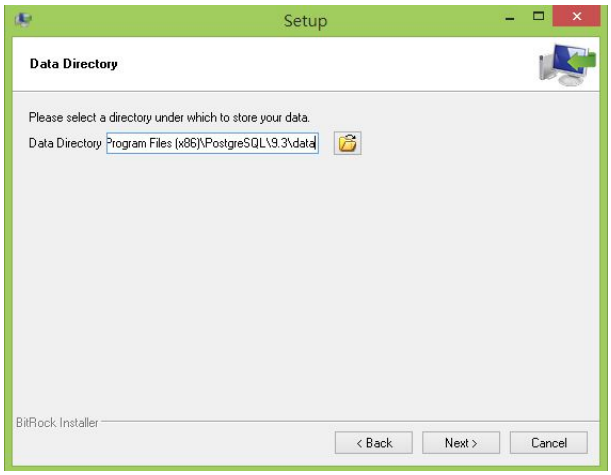
第四章 PostgreSQL 的安裝及基本操作

1. 下載 PostgreSQL
   http://www.postgresql.org/download/windows/
2. 開始安裝
   a. 執行安裝檔　postgresql-9.3.5-1-windows.exe

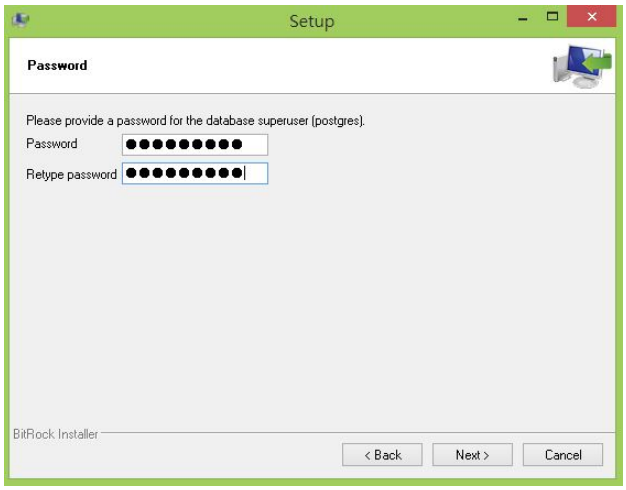


   b. 指定程式安裝路徑

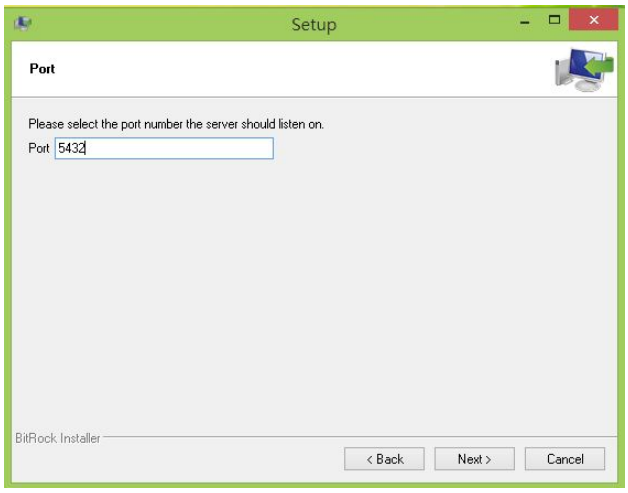




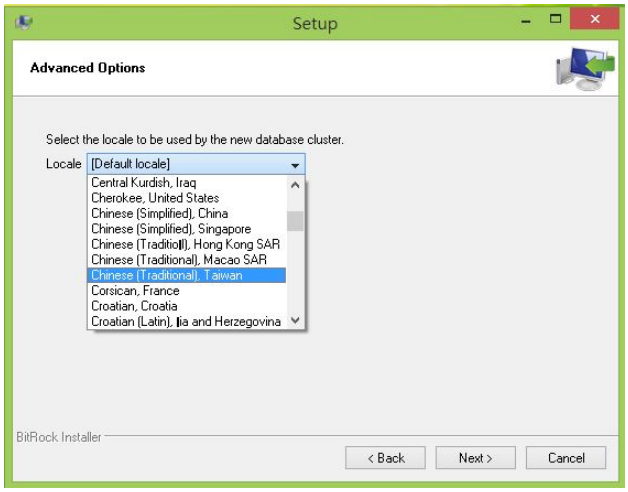
   c. 設定 postgres（管理者）的密碼



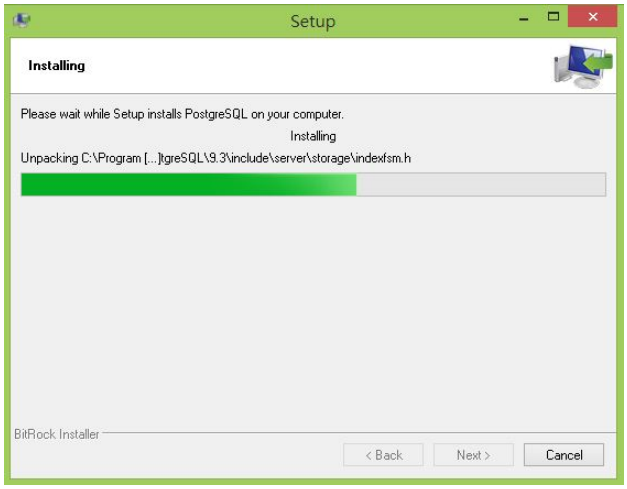
使用的 Port

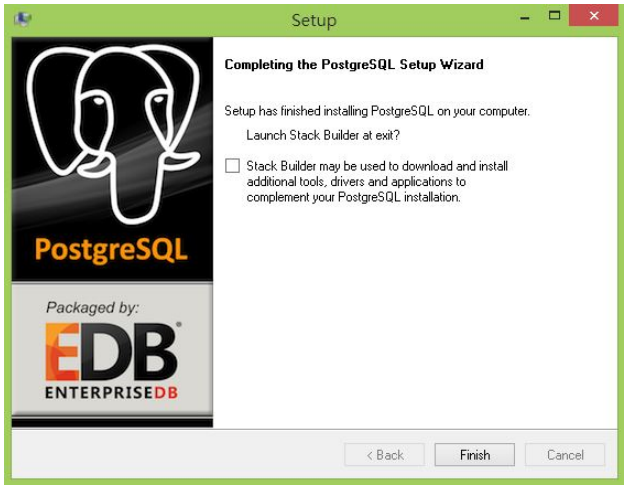


進階的語系設定

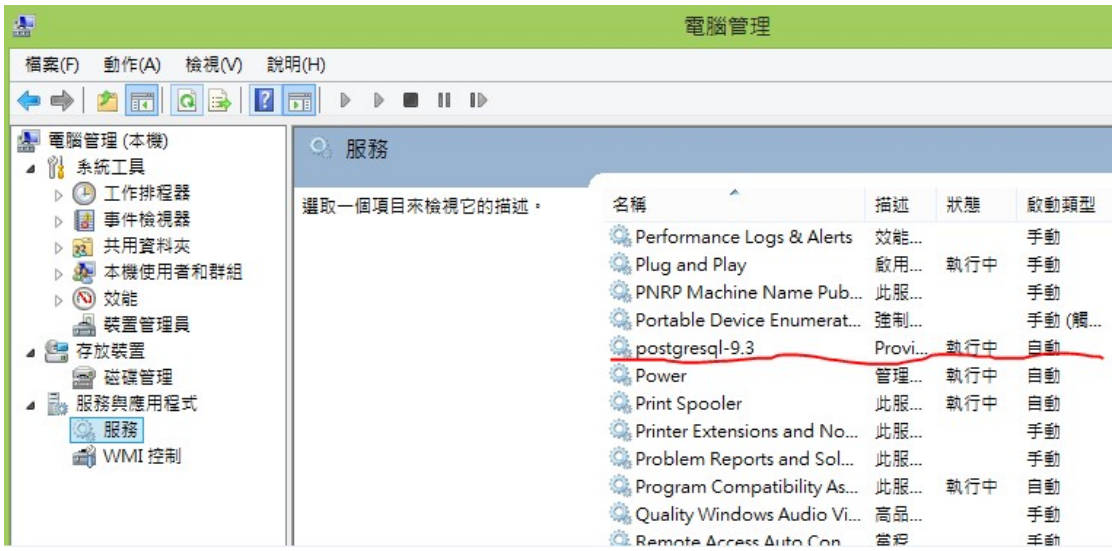


d. 開始安裝



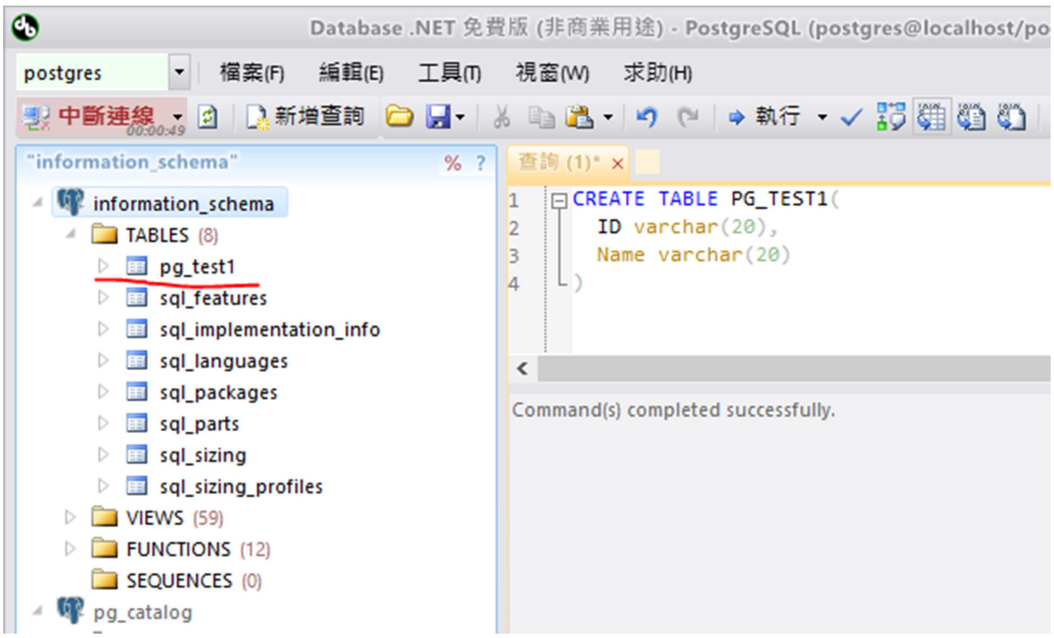


e. 安裝後，已經自啟動 Windows 服務



f. 測試一下，用 Database.NET 連線

用簡單的 SQL 實際測試



第五章
MySQL 的安裝及基本操作

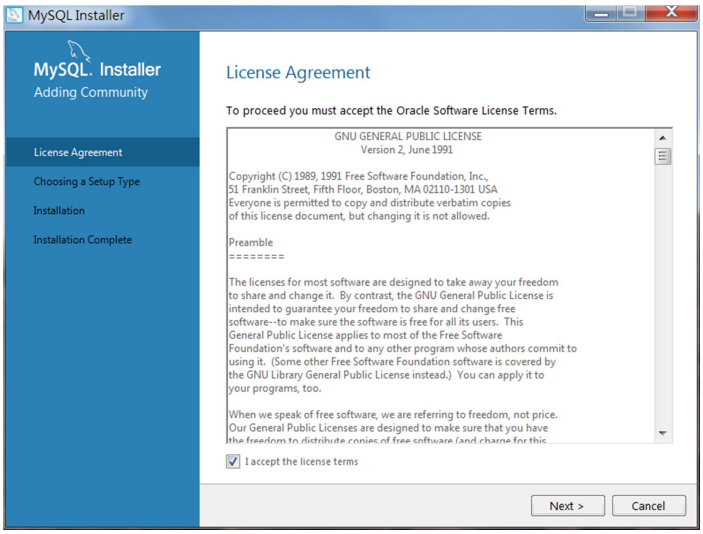
故其疾如風，其徐如林，
侵掠如火，不動如山，
難知如陰，動如雷震。

孫子兵法 軍爭篇

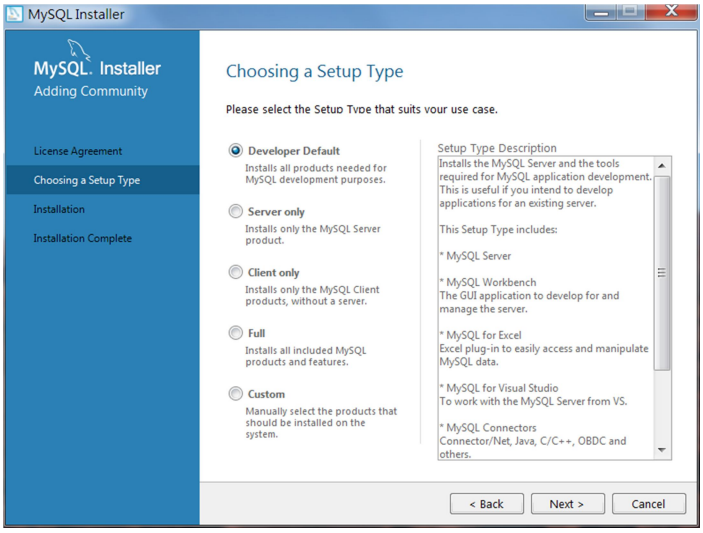
第五章 MySQL 的安裝及基本操作

1. 下載 MySQL
   http://dev.mysql.com/downloads/mysql/

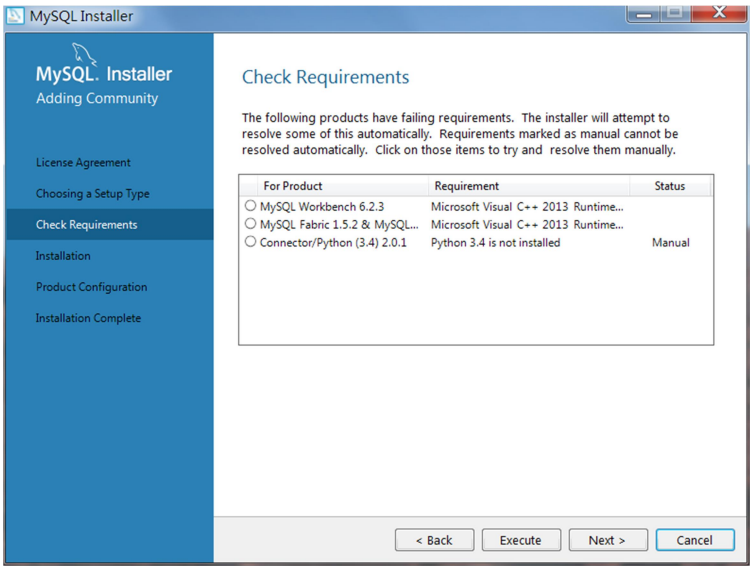
2. 開始安裝
   a. 執行安裝檔 mysql-installer-community-5.6.21.1.msi
   b. 版權宣告

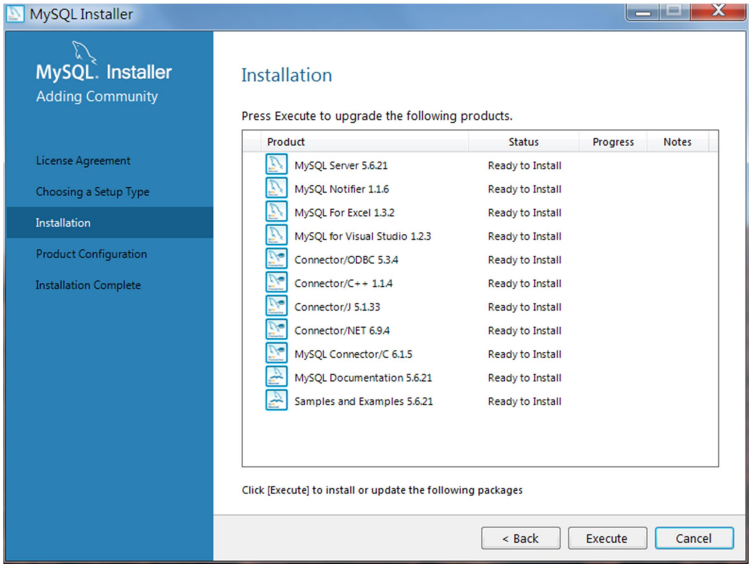


   安裝模式選擇
   Development Default 開發環境，開發人員請選此項
   Server Only 只安裝資料庫主程式，通常是發佈主機會選此項

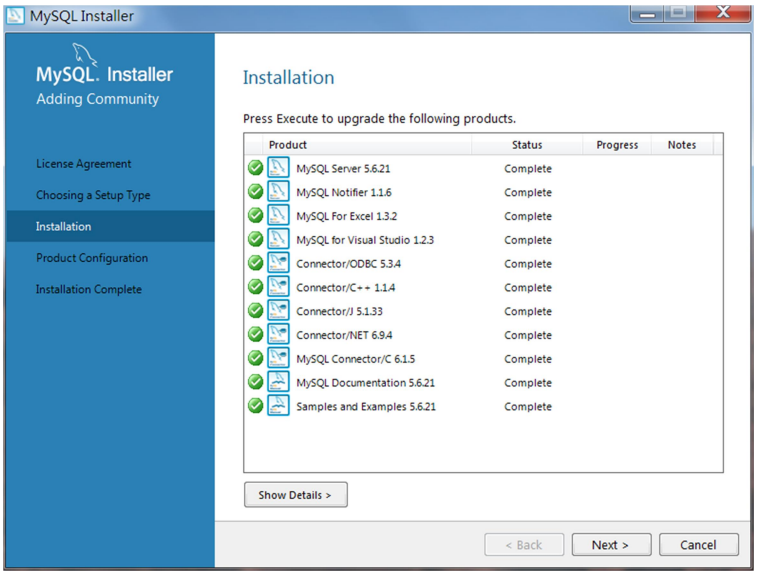


   c. 檢查缺少的元件並安裝

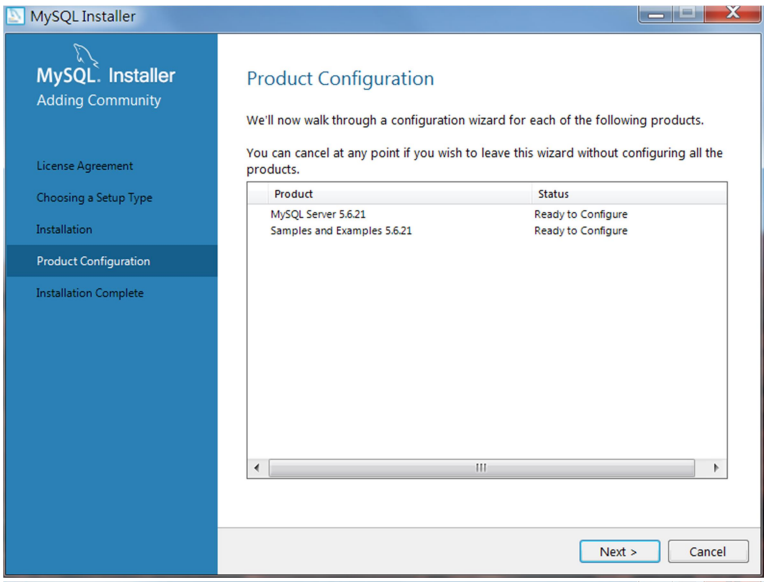


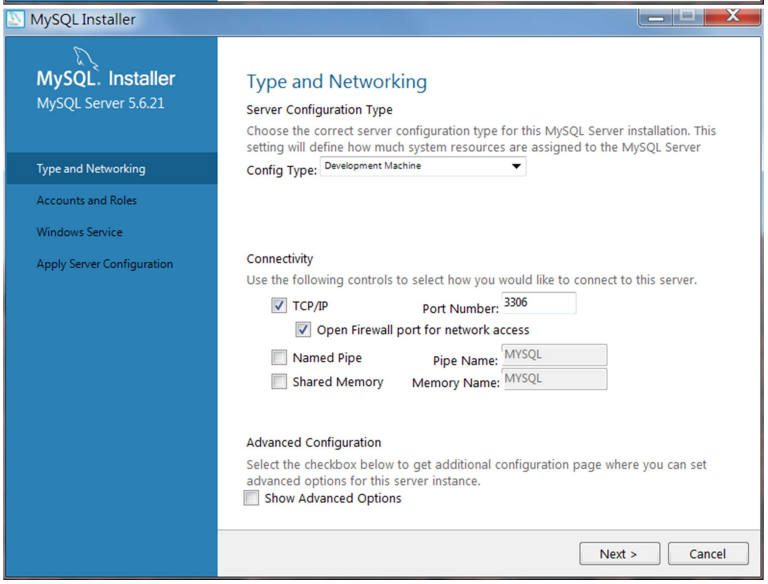


開始安裝相關的程式

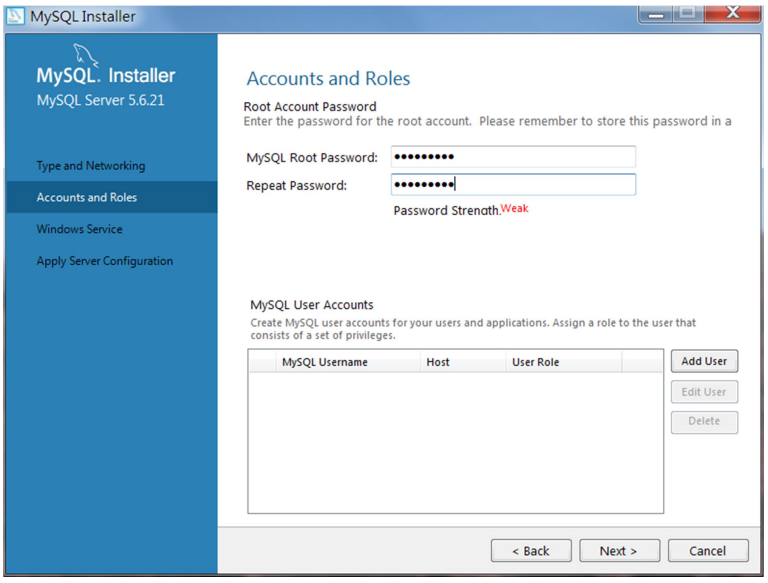


d. 安裝完成後，開始設定

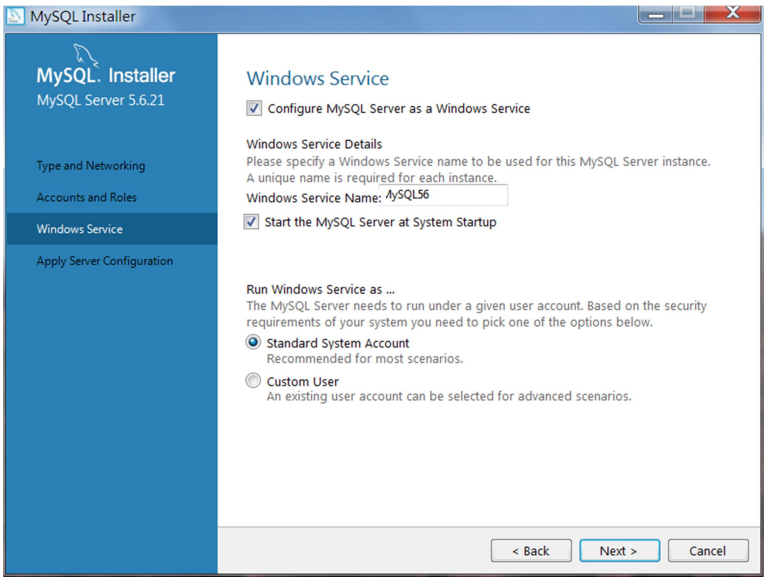


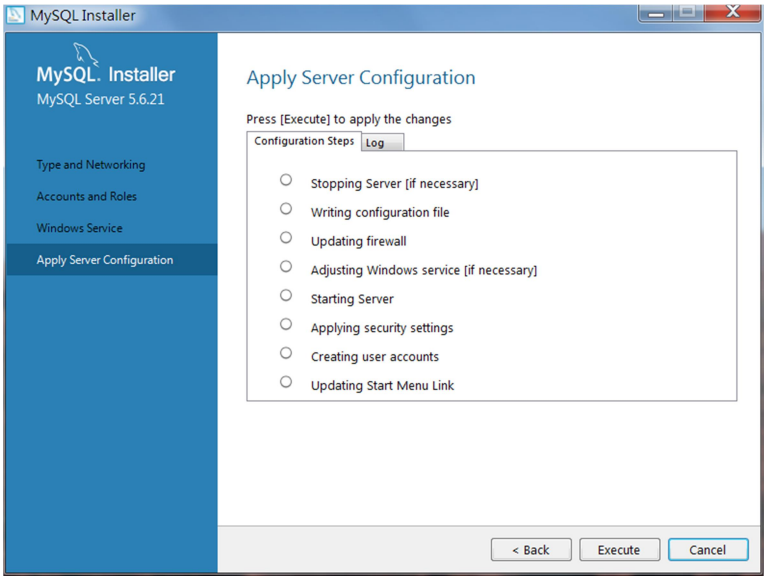


設定管理者 root 的密碼

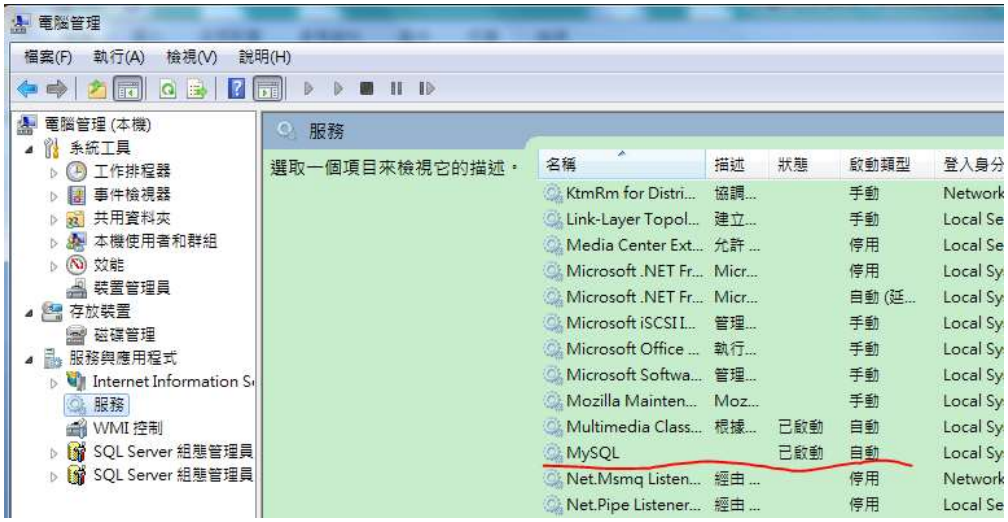


e. 設定為 Windows 的服務





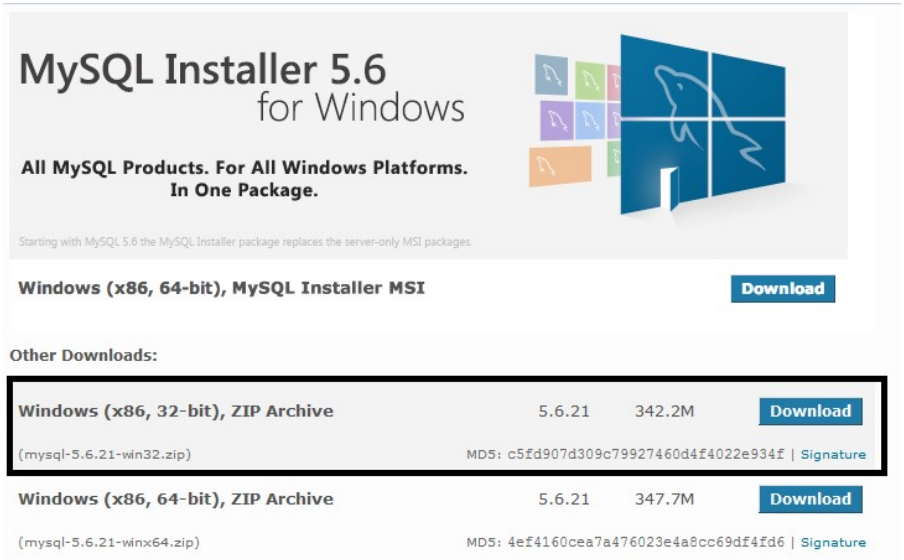
正確設定完成後，MySQL 在 Windows 的 Service 也已經啟動



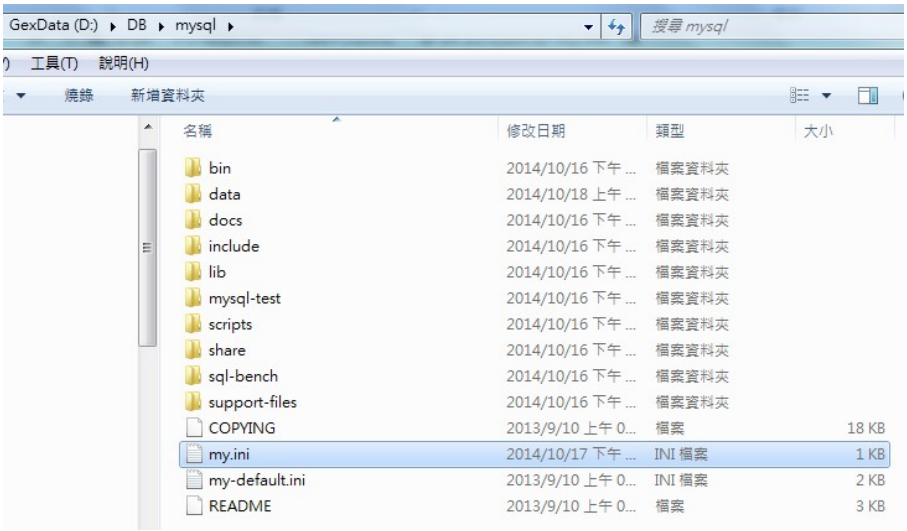
前面的說明，是用 Install 程式安裝的過程。
事實上，MySQL 還有另一種安裝模式，看起來比較笨，可是步驟簡單多了。

安裝步驟說明如下

1. 下載 zip 檔案
   一樣在 http://dev.mysql.com/downloads/mysql/



2. 解壓縮後，開始設定。所謂的設定，事實上是設定 my.ini 檔
   本書的範例，是將 MySQL 解壓縮到 D:/DB/mysql



   將下面的內容貼到 my.ini 中，主要是路徑和 Port 要設定

```
# Save as "my.ini" in your MySQL installed directory
[mysqld]
# Set MySQL base (installed) directory
# @@ Change to your MySQL installed directory @@
basedir=D:/DB/mysql

# Set MySQL data directory
# @@ Change to sub-directory "data" of your MySQL installed directory @@
```

datadir=D:/DB/mysql/data

Run the server on this TCP port number

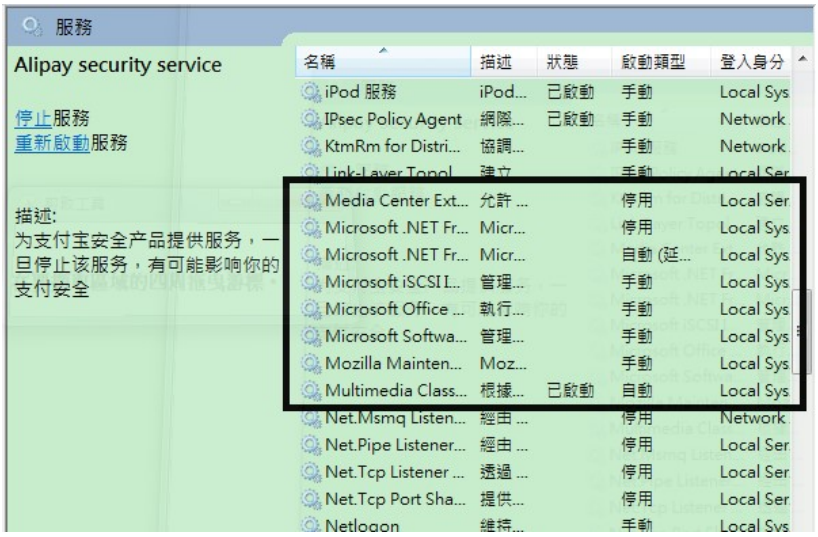
port=3306

[client]

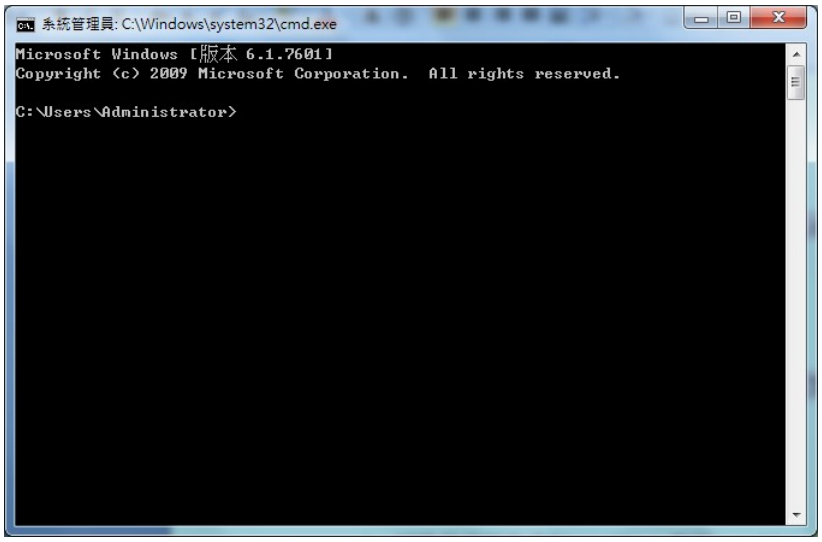
MySQL client connects to the server running on this TCP port number

port=3306

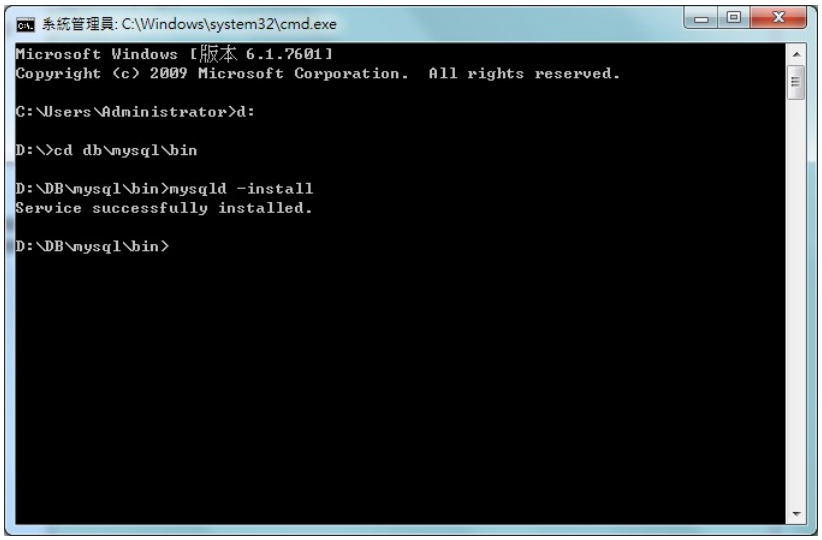
3. 在主控台模式安裝並啟動 Windows 服務
   a. 安裝前無 MySQL 服務



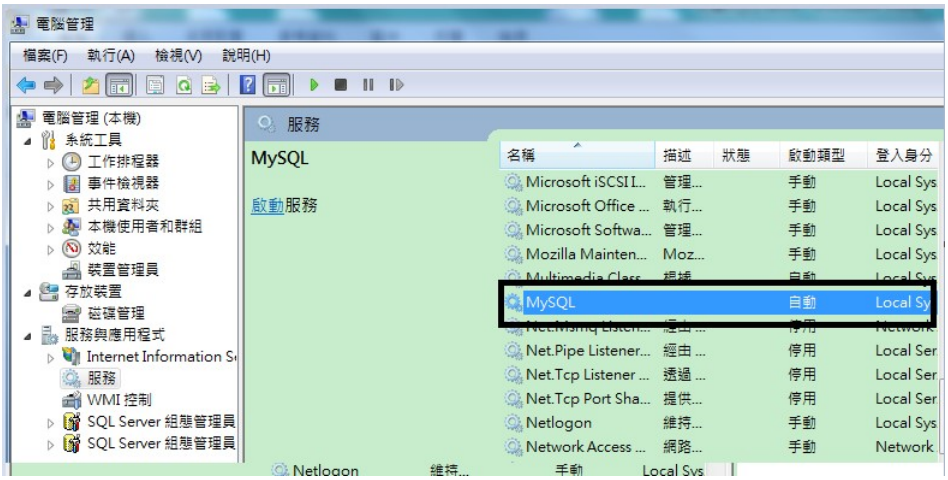
   b. 進入 console



c. 切換目錄到 D:/DB/mysql/bin，執行 mysqld －install



Windows 的服務就被啟動了

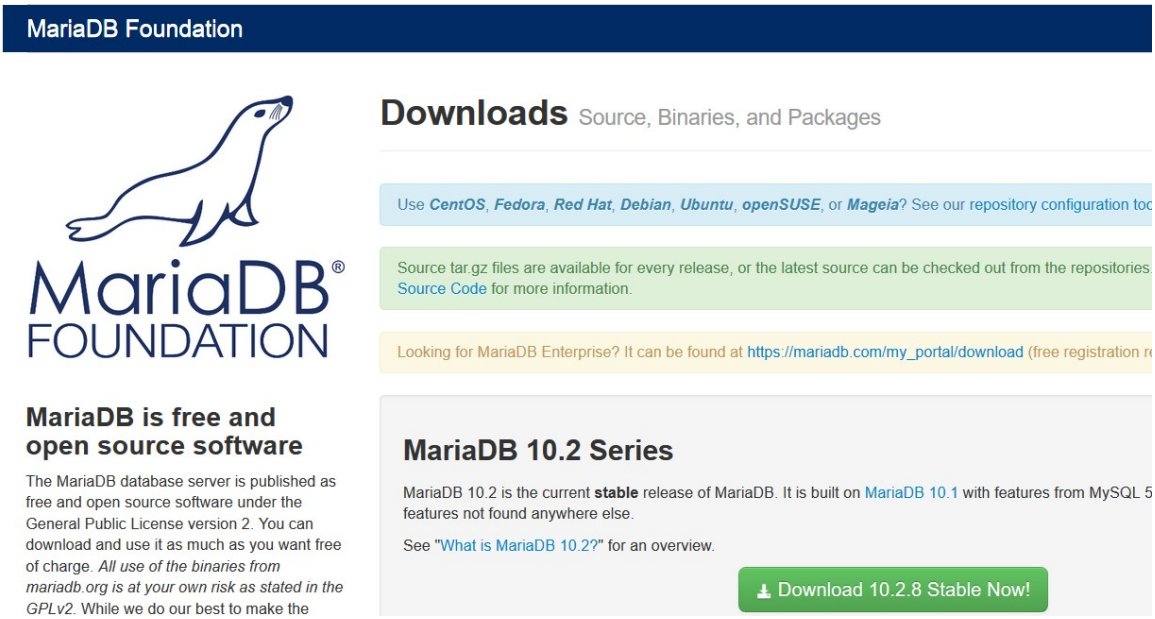


用這個方式安裝的 MySQL,管理員的帳號 root 的密碼預設是空白，一定要記得更改密碼，否則會有安全性的問題。以上是安裝 MySQL 的 2 種方式，一般而言，沒有好壞，只要能正確安裝執行就可以，您可以自行決定用哪一種方式。

接下來，我們也簡單的介紹 MariaDB 的安裝過程

1. 下載 MariaDB

   https://downloads.mariadb.org/

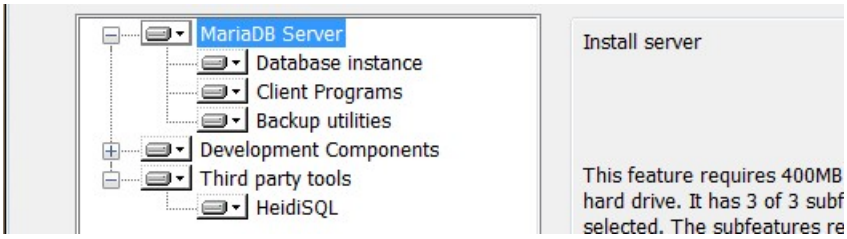


2. 開始安裝

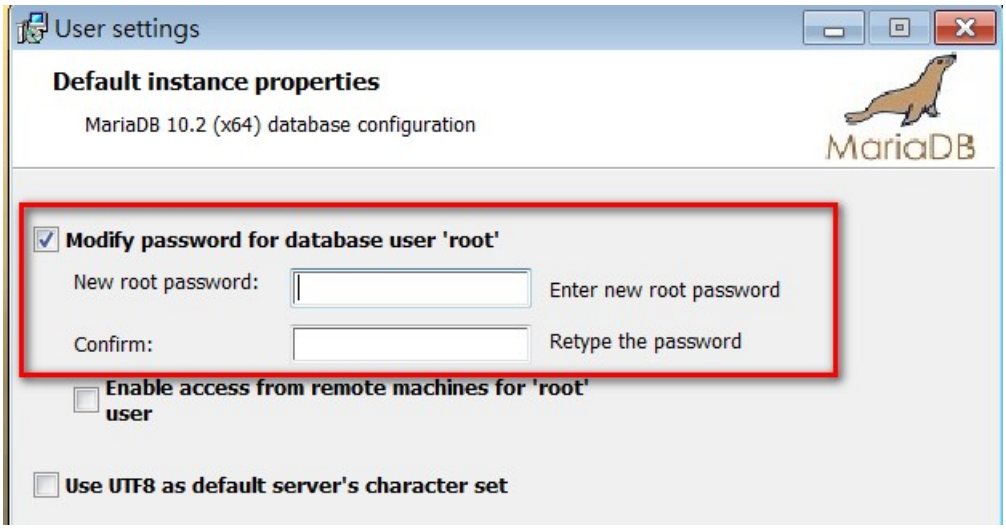
   f. 執行安裝檔 mariadb-10.2.8-winx64.msi
   (視您的作業系統環境下載，筆者測試環境是 Win7-x64 專業版)

   g. 版權宣告

常用的工具都會自動預設安裝



h. 當然，root 的密碼一定是要設的拉，設定好後要記好，程式設定連線時會用到。



i. 相對於 MySQL 的繁複設定及檢查，基本上 MariaDB 的安裝包輕鬆多了，通常只要用一般的預設值安裝就可以。

j. 安裝完後就可以使用 HeidiSQL 或是之前介紹的 Database.NET 程式測試連線或常用 SQL 語法，請自行測試。

第六章
欄位型態比較

天地不仁，以萬物為芻狗；
聖人不仁，以百姓為芻狗。
天地之間，其猶橐籥乎！
虛而不屈、動而愈出，多言數窮，不如守中。

老子道德經

第六章欄位型態比較

在說明各資料庫的 SQL 語法前，還是必須先來談談各資料庫提供的欄位型態，關聯式資料庫儲存資料時會將資料存放在 Table 中，Talbe 再新增多個欄位，而每個欄位都會再依資料的內容分為多個不同的類型，而所謂資料類型是指定物件所能保留之資料類型的屬性。

類型包括整數、精確位數、浮點數、字元、貨幣資料、日期和時間資料、二進位字串等。一般分為四大類，不同的資料庫程式多少都會有點差異，而且版本升級時可能也會變更，以下僅列出較常使用的類型供參考。

1. 數值資料 (Numeric Data)
   數值型態有 integer, float, money 使用數值資料能夠搭配內建的數值函數來做資料處理該數值欄位。
2. 字串(元)資料 (Character & Strings Data)
   儲存字元或符號之資料型別。
3. 日期/時間資料 (Date Data)
   記錄日期/時間的資料型別，類型 Date, Time, Timestamp。
4. 布林值 (Boolean Data)
   True、False, Yes、No, 1、0

以下就表列出 MS-SQL 的常用資料類型及每個型態的描述說明，不同的資料庫會有些許的差異，其他開源碼資料庫的細項描述也列示如下，在本章最後，會整理一張比對表，幫助您更容易理解不同資料庫的欄位資料類型差異。

MS SQL Server 常用資料類型

字串(元)資料 (Character & Strings Data)：

資料類型	描述
char(n)	固定長度的字串。最多 4,000 個字元。
varchar(n)	可變長度的字串。最多 4,000 個字元。
varchar(max)	可變長度的字串。最多 1,073,741,824 個字元。
text	可變長度的字串。最多 2GB 字元資料。

Unicode 字串：

資料類型	描述

nchar(n)	固定長度的 Unicode 資料。最多 4,000 個字元。
nvarchar(n)	可變長度的 Unicode 數據。最多 4,000 個字元。
nvarchar(max)	可變長度的 Unicode 數據。最多 536,870,912 個字元。
ntext	可變長度的 Unicode 數據。最多 2GB 字元資料。

Number 類型：

資料類型	描述	存儲
tinyint	允許從 0 到 255 的所有數字。	1 位元組
smallint	允許從 -32,768 到 32,767 的所有數字。	2 位元組
int	允許從 -2,147,483,648 到 2,147,483,647 的所有數字。	4 位元組
bigint	允許介於 -9,223,372,036,854,775,808 和 9,223,372,036,854,775,807 之間的所有數字。	8 位元組
decimal(p,s)	固定精度和比例的數字。允許從 -10^38 +1 到 10^38 -1 間的數字。 p 參數指示可以存儲的最大位元數（小數點左側和右側）。p 必須是 1 到 38 之間的值。默認是 18。 s 參數指示小數點右側存儲的最大位元數。s 必須是 0 到 p 之間的值。默認是 0。	5-17 位元
numeric(p,s)	固定精度和比例的數字。允許從 -10^38 +1 到 10^38 -1 間的數字。 p 參數指示可以存儲的最大位元數（小數點左側和右側）。p 必須是 1 到 38 之間的值。默認是 18。 s 參數指示小數點右側存儲的最大位元數。s 必須是 0 到 p 之間的值。默認是 0。	5-17 位元組
smallmoney	介於 -214,748.3648 和 214,748.3647 之間的貨幣資料。	4 位元組
money	介於 -922,337,203,685,477.5808 和 922,337,203,685,477.5807 之間的貨幣資料。	8 位元組

float(n)	從 -1.79E + 308 到 1.79E + 308 的浮動精度數位資料。 參數 n 指示該欄位保存 4 位元組還是 8 位元組。float(24) 保存 4 位元組，而 float(53) 保存 8 位元組。n 的預設值是 53。	4 或 8 位元組
real	從 -3.40E + 38 到 3.40E + 38 的浮動精度數位資料。	4 位元組

Date 類型：

資料類型	描述	存儲
datetime	從 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度為 3.33 毫秒。	8 bytes
datetime2	從 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度為 100 納秒。	6-8 bytes
smalldatetime	從 1900 年 1 月 1 日 到 2079 年 6 月 6 日，精度為 1 分鐘。	4 bytes
date	僅存儲日期。從 0001 年 1 月 1 日 到 9999 年 12 月 31 日。	3 bytes
time	僅存儲時間。精度為 100 納秒。	3-5 bytes
datetimeoffset	與 datetime2 相同，外加時區偏移。	8-10 bytes
timestamp	存儲唯一的數位，每當創建或修改某行時，該數字會更新。timestamp 基於內部時鐘，不對應真實時間。每個表只能有一個 timestamp 變數。	

Binary 類型：

資料類型	描述
bit	允許 0、1 或 NULL
binary(n)	固定長度的二進位資料。最多 8,000 位元組。
varbinary(n)	可變長度的二進位資料。最多 8,000 位元組。
varbinary(max)	可變長度的二進位資料。最多 2GB 位元組。
image	可變長度的二進位資料。最多 2GB。

MySQL 常用資料類型

字串(元)資料 (Character & Strings Data)：

資料類型	描述
char(n)	固定長度的字串。最多 255 個字元。
varchar(n)	可變長度的字串。最多 65,535 個字元。
Text	可變長度的字串。最多 2^16 - 1 字元資料。

Unicode 字串：

資料類型	描述
nchar(n)	固定長度的 Unicode 資料。同 char(n) CHARSET utf8。
nvarchar(n)	可變長度的 Unicode 數據。同 varchar(n) CHARSET utf8。
ntext	可變長度的 Unicode 數據。同 text CHARSET utf8。

Number 類型：

資料類型	描述	存儲
tinyint	允許從 -128~127 的所有數字。	1 位元組
smallint	允許從 -32,768 到 32,767 的所有數字。	2 位元組
int	允許從 -2,147,483,648 到 2,147,483,647 的所有數字。	4 位元組
bigint	允許介於 -9,223,372,036,854,775,808 和 9,223,372,036,854,775,807 之間的所有數字。	8 位元組
decimal(p,s)	固定精度和比例的數字。允許從 -10^38 +1 到 10^38 -1 間的數字。 p 參數指示可以存儲的最大位元數（小數點左側和右側）。p 必須是 1 到 38 之間的值。默認是 18。 s 參數指示小數點右側存儲的最大位元數。s 必須是 0 到 p 之間的值。默認是 0。	5-17 位元組
numeric(p,s)	固定精度和比例的數字。允許從 -10^38 +1 到 10^38 -1 間的數字。	5-17 位元組

	p 參數指示可以存儲的最大位元數（小數點左側和右側）。p 必須是 1 到 38 之間的值。默認是 18。 s 參數指示小數點右側存儲的最大位元數。s 必須是 0 到 p 之間的值。默認是 0。	
float(n)	從 -1.79E + 308 到 1.79E + 308 的浮動精度數位資料。 參數 n 指示該欄位保存 4 位元組還是 8 位元組。float(24) 保存 4 位元組，而 float(53) 保存 8 位元組。n 的預設值是 53。	4 或 8 位元組
real	從 -3.40E + 38 到 3.40E + 38 的浮動精度數位資料。	4 位元組

Date 類型：

資料類型	描述	存儲
datetime	1000-01-01 00:00:00 ~ 9999-12-31 23:59:59。	8 bytes
date	僅存儲日期。從 1000 年 1 月 1 日 到 9999 年 12 月 31 日。	3 bytes
time	僅存儲時間。精度為 100 納秒。	3-5 bytes
timestamp	1970-01-01 00:00:00 ~ 2037-12-31 23:59:59	4 bytes

Binary 類型：

資料類型	描述
TINYBLOB	255 bytes。
BLOB	65535 bytes。
LONGBLOB	4,294,967,295 bytes。

PostgreSQL 常用資料類型

字串(元)資料 (**Character & Strings Data**)：

資料類型	描述
char(n)	fixed-length, blank padded。

varchar(n)	variable-length with limit。
text	variable unlimited length。

Unicode 字串：

PostgreSQL 並沒有類似 MS-SQL 或 MySQL的nvarchar(n)資料類型，而是依資料庫創建時的編碼決定。如果編碼是 utf-8 的資料庫，varchar 之類的資料欄位就會存為雙位元。

Number 類型：

資料類型	描述	存儲
smallint	允許從 -32,768 到 32,767 的所有數字。	2 位元組
int	允許從 -2,147,483,648 到 2,147,483,647 的所有數字。	4 位元組
bigint	允許介於 -9,223,372,036,854,775,808 和 9,223,372,036,854,775,807 之間的所有數字。	8 位元組
decimal(p,s)	固定精度和比例的數字。允許從 -10^38 +1 到 10^38 -1 間的數字。 p 參數指示可以存儲的最大位元數（小數點左側和右側）。p 必須是 1 到 38 之間的值。默認是 18。 s 參數指示小數點右側存儲的最大位元數。s 必須是 0 到 p 之間的值。默認是 0。	5-17 位元組
numeric(p,s)	固定精度和比例的數字。允許從 -10^38 +1 到 10^38 -1 間的數字。 p 參數指示可以存儲的最大位元數（小數點左側和右側）。p 必須是 1 到 38 之間的值。默認是 18。 s 參數指示小數點右側存儲的最大位元數。s 必須是 0 到 p 之間的值。默認是 0。	5-17 位元組
double precision	15 decimal digits precision。	8 位元組
real	從 -3.40E + 38 到 3.40E + 38 的浮動精度數位資料。	4 位元組

Date 類型：

資料類型	描述	存儲
timestamp [(p)] with time zone	both date and time, with time zone。4713BC – 294276 AD	8 bytes
timestamp [(p)] [without time zone	both date and time (no time zone) 。4713BC – 294276 AD	8 bytes
date	僅存儲日期。	3 bytes
time[(p)] [with time zone]	time of day (no date) 。00:00:00+1459 - 24:00:00-1459	3-5 bytes
time [(p)] [without time zone]	time of day (no date) 。00:00:00-24:00:00	8 bytes

上面詳細列出了各資料庫的資料型別，事實上，大部份的資料型別都差異不大，除非有特別的原因，否則不建議在資料庫存入大型資料，那一定會影響資料庫的效能。一般都是存該大型檔案（如圖片或 word 檔等）的路徑即可。如果是一般企業內部使用的系統，如 ERP、CRM 或官網的一些動態資料，一般都只會使用通用的字串、數字、日期等型別。那本書所介紹的幾個開源資料庫應該都足夠使用，特別是依目前的硬體成本大幅下滑，早期為了節省硬碟空間，會設定一些較不佔空間的資料型別，事實上現在看來意義不大。使用一般通用的型別，不論是資料庫轉換或是程式開發工具的變更，都比較不會發生問題，所以不必為了節省有限的空間，特別去使用特殊的型別。

在下一頁，筆者簡單的整理一下 MS-SQL、MySQL、PostgreSQL 的常用的資料型別方便各位讀者參考，資料型別夠用就好，不是支援的型別較多就特別好，這是特別再提醒各位的。

關聯式資料庫常見資料類型（DATA TYPE）比較

DATA TYPE	MS SQL	MySQL	POSTGRE
布林			BOOL
字串	CHAR VARCHAR TEXT	CHAR VARCHAR BLOB TEXT ENUM SET	CHAR VARCHAR TEXT NAME CHARACTER
Unicode 字串	NCHAR NVARCHAR NTEXT	NCHAR NVARCHAR	設定資料庫 utf-8
精確數字	INTEGER BIGINT SMALLINT TINYINT NUMERIC DECIMAL	INTEGER BIGINT SMALLINT DECIMAL NUMERIC	INTEGER BIGINT SMALLINT DECIMAL NUMERIC SERIAL
近似數字（浮點運算）	FLOAT REAL DOUBLE	FLOAT REAL DOUBLE	FLOAT REAL DOUBLE
日期時間	DATE TIME SMALLDATATIME TIMESTAMP UNIQUEIDENTIFIER	DATE TIME DATETIME TIMESTAMP YEAR	DATE TIME TIMESTAMP CURRENT INFINITY INVALID NOW
BINARY	BINARY VARBINARY IMAGE		

除了欄位型態之外，下面也簡單的整理了不同資料庫的 SQL 差異，方便讀者能盡速掌握不同資料庫間的差異，盡速入門。

較常用的 udf(user define function sql ：自定義函數）對照表

	MS-SQL	MySQL	PostgreSQL
	聚合函數		
.取某欄位的平均值 .計算筆數 .某個欄位值最大值 .某個欄位值的最小值 .某個欄位取值的總和	avg() count() max() min() sum()	avg(col) count(col) max(col) min(col) sum(col)	avg(expression) count(*) max(expression) min(expression) sum(expression)
	字串類		
.取 ascii 碼 .根據 ascii 碼取字元 .轉化為指定的資料類型 .將 s1,s2...,sn 連接成字串 .返回字串的位元長度 .返回字串長度 .取子字串 .返回字串右邊 x 個字元 .返回字串左邊 x 個字元 .轉爲大寫 .轉爲小寫 .生成 n 個空格 .複制字串 n 次 .反轉字串 .字串代替 .指定位置插入指定字串 .取掉左方空格 .取掉右方空格 .返回 str 在 substr 的起始位置 .條件式判斷 .若 null 取第 2 個引數 .用反斜線轉義 str 中的單引號 .去除字串前後的所有空格 .日期格式化 .比較字串 s1 和 s2 .固定字串長度(左或右)	ascii(char) char(ascii) cast() + datalength(Char_expr) len() substring(exp,start,len) right(str,x) left(str,x) upper(str) lower(str) space(n) replicate(str,n) reverse(str) replace() stuff(str,x,y,instr) ltrim(char_expr) rtrim(char_expr) charindex(substr,str) iif(bool,val_t,val_f) isnull(bool,val_null) convert()	ascii(char) chr() cast() concat(s1,s2...,sn) data_length(str) length(s) substring() right(str,x) left(str,x) upper(str) lower(str) repeat(str,src,rplcstr) reverse(str) replace() insert(str,x,y,instr) ltrim(str) rtrim(str) position(substr,str) iif(bool,val_t,val_f) ifnull() quote(str) trim(str) date_format(date,fmt) strcmp(s1,s2) lpad or rpad	ascii(string) chr(int) to_char() ‖ or concat(x,y,x) char_length(string) length(string) substring() right(str text, n int) left(str text, n int) upper(string) lower(string) repeat() reverse(str) replace() ltrim(str) rtrim(str) strpos(str, substr) trim(str) to_date(text, text) lpad or rpad

	數學類		
.求 num 絕對值 .取指數 .返回 x/y 的模（餘數） .隨機數產生器 .四捨五入 .平方根 .返回數字 x 截短為 y 位小數	abs(num) exp(float_expr) mod() rand([int_expr]) round(x,y) sqrt(float_expr)	abs(x) exp(x) mod(x,y) rand() round(x,y) sqrt(x) truncate(x,y)	abs(x) exp(x) mod(y, x) random() round(x,y) sqrt(x) trunc(x,y)
	日期類		
.返回當前日期及時間 .返回日期 .返回當前的時間 .返回日期加上 number .日期差 .傳回日期欄的西元年 .傳回日期欄的月 .傳回日期欄的日 .傳回日期欄是星期幾(1~7) .返回日期為一年中第幾(0~53)	now() getdate() dateadd(type,int,d_exp) datediff(type,d1,d2) year(date) month(date) day(date)	Now() current_date() current_time() date_add()/date_sub() datediff(date1,date2) year(date) dayofmonth(date) dayname(date) dayofweek(date) week(date)	now() current_date current_time date_part(text, times)
	系統函數		
.數據庫名 .伺服器的版本	db_name()	database() version()	current_database() version()

預存程序(store procedure)的語法差異對照表

	MS-SQL	MySQL	PostgreSQL

創建程序
(更新)
	Create procedure sp_xxx
	(
	  Para1 xxx
	)
	begin
	  xxxx…
	end
		create
		[definer = { user |
		current_user }]
		procedure sp_name
		([proc_parameter[,...]])
		[characteristic ...]
		routine_body
		
		其中的"DEFINEER"是定義誰創建了此儲存步驟，預設是 CURRENT_USER
			create or replace function
			sp_xxx()
			Return text as
			$$
			Declare
			Begin
			  xxx….
			
			  Return rets;
			End
			$$
			Language plpgsql;

呼叫程序	Exec sp_xxx()	Call sp_xxx()	Select sp_xxx()
刪除程序	Drop proc sp_xxx;	drop procedure if exists sp_xxx;	drop function sp_xxx;

If-else	if bool1
	begin
	  xxx
	end
	else if bool2
	begin
	  xxx
	end
		if search_condition then
		statement_list
		[elseif search_condition then
		statement_list] ...
		[else statement_list]
		end if
			if bool-exp then
			  statements
			[elsif bool-exp then
			  statements
			[elsif bool-exp then
			  statements
			  ...]]
			[else
			  statements]
			end if;

case		case case_value
		  when w_v1 then s_list1
		  [when w_v2 then s_list2]
		  ...
		  [else statement_list]
		end case
			case
			  when cond_1 then r_1
			  when cond_2 then r_2
			  [when ...]
			  [else result_n]
			end

For loop	使用 while 語法	xxx_loop: LOOP
		xxx1….
		if bool1 then
		Leave xxx_loop;
		else
		xxx2;
		end if;
		End LOOP xxx_loop;
			for i in 1..10 loop
			xxxx
			end loop;
			
			loop
			-- some computations
			if count > 0 then
			exit; -- exit loop
			end if;
			end loop;
			
			loop
			-- some computations
			exit when count > 0;
			end loop;
			
			<<ablock>>
			begin
			-- some computations
			if stocks > 100000 then
			exit ablock;
			end if;
			end;

While loop	declare @i int;
	set @i=0
	while @i<10
	begin
	xxx1…
	
	set @i=@i+1
	end;
		declare i int;
		set i=1;
		while i<100 do
		xxxx;
		set i = i + 1;
		end while;
			while bool1 and bool2 loop
			-- some codes here
			end loop;
			
			while not done loop
			-- some codes here
			end loop;

宣告變數	declare @xxx;	declare max_count int default 10; declare n varchar(20) default '';	declare r foo%rowtype; begin for r in select * from foo where fooid > 0 loop

			-- do some processing here return next r; -- current row end loop; return; end

設定變數	set @xxx = ‘’ ;	set xxx = ‘’ ;	if number = 0 then
			result := 'zero';
			elsif number > 0 then
			result := 'positive';
			elsif number < 0 then
			result := 'negative';
			else
			-- hmm, xxx is null
			result := 'null';
			end if;

取某欄位內容存到變數	select @xxx= xxx_field from xxx_table where filters	select xxx_field into xxx from xxx_table where filters	select xxx_field into xxx from xxx_table where filters
回傳值否	最後一句 select 語法的內容	最後一句 select 語法的內容	RETURN
執行動態 SQL 的語法	Exec sp_executesql @sql	Prepare sql_name from sql_text Execute sql_name using var Deallocate prepare sql_name	execute command-string [into [strict] target] [using expression [, ...]];
暫存檔	1.select into # Select * Into #tmp1 From xxx_table 2.create #tmp1 Create table #tmp1(f1 varchar(10), f2 int) 3.declare table declare @act_tmp1 table	Drop temporary table if exists tmp_xxx; Create temporary table if not exists tmp_xxx (fld1 int,fld2 varchar(20),fld3 numeric(8,2))	If exists (select relname from pg_class where relname='tmp_xxx') Then Truncate table tmp_xxx; Else Create temp table tmp_xxx (like basperson_xxx)

	(titleid varchar(10),prevalue float)	Create temporary table tmp_xxx as select * from basarea where filters	

udf(user define function)的語法差異對照表

	MS-SQL	MySQL	PostgreSQL
創建 udf	Create function	create [definer = { user | current_user }] function sp_name ([func_parameter[,...]]) returns type [characteristic ...] routine_body	Create function pgSQL 並沒有區分 udf 和 store procedure，都是 create function xxx
呼叫程序	Select xxx()	Select xxx()	Select xxx()
If-else	程式寫法差異，請參考 Store Procedure 的說明		
For loop			
While loop			
回傳值	return	return	return

第七章
常用的 SQL 語法簡介

咬定青山不放鬆，
立根原在破岩中。
千磨萬擊還堅勁，
任爾東西南北風。

鄭板橋。竹石

第七章 常用的 SQL 語法簡介

結構化查詢語言（Structured Query Language，SQL），一種程式語言，用於資料庫中的標準資料查詢語言，IBM 公司最早使用在其開發的資料庫系統中。1986 年 10 月，美國國家標準學會（ANSI）對 SQL 進行規範後，以此作為關聯式資料庫管理系統的標準語言（ANSI X3.135-1986），1987 年得到國際標準組織的支援下成為國際標準。不過各種通行的資料庫系統在其實踐過程中都對 SQL 規範作了某些修改和擴充。所以，實際上不同資料庫系統之間的 SQL 不能完全相互通用。本書的主要目的是要就是要了解 MS-SQL 和三套開源碼資料庫的 SQL 差異何在？儘量找出差異處，並儘量依詢 ANSI-92 標準，讓資料庫的轉換難度降低。推廣筆者所強調主張的「以資料庫為開發核心」的開發理念。

SQL 包含 3 個部分：

「資料定義語言」（DDL : Data Definition Language）

「資料操縱語言」（DML : Data Manipulation Language）

「資料控制語言」（DCL : Data Control Language）

在本章中，主要的重點是前 2 個部份，並以一個完整系列的實際案例，讓各位能通過實作，熟悉基礎的 SQL 語法。SQL 的資料定義語言 (DDL) 部分主要的功能是創建、維護或刪除表格。我們也可以定義索引（鍵），規定表之間的連結，以及施加表間的約束。

SQL 中最重要的 DDL 語句：

- CREATE DATABASE -創建新資料庫
- ALTER DATABASE -修改資料庫
- CREATE TABLE - 創建新表
- ALTER TABLE - 變更（改變）資料庫表
- DROP TABLE - 刪除表
- CREATE INDEX - 創建索引（搜索鍵）
- DROP INDEX - 刪除索引

SQL 也包含用於更新、插入和刪除記錄的語法。查詢/更新/插入/刪除 指令構成了 SQL 的 DML 部分：

- SELECT - 從資料庫表中獲取資料
- UPDATE - 更新資料庫表中的資料
- DELETE - 從資料庫表中刪除資料
- INSERT INTO - 向資料庫表中插入資料

接下來，我們就實際的來操作一下常用的 SQL

1. 建立新資料庫

   語法：CREATE DATABASE database-name

   實例：CREATE DATABASE SampleDB

2. 刪除資料庫

   語法：DROP DATABASE database-name

   實例：DROP DATABASE SampleDB

3. 建立新表(Table)

   語法：CREATE TABLE tabname(

   col1 type1 [not null] [primary key],col2 type2 [not null],..

)

   實例：

```
/*
  本章內容主要是方便解說範例中的 SQL 語法，只取部份欄位，
  並非完整的 ERP 資料表內容，特此說明
*/
CREATE TABLE MEMBER (
  MEMBERID              varchar(20) not null,
  MEMBERNAME            nvarchar(30) DEFAULT '',
  MEMBERLAVEL           varchar(4) DEFAULT '',
  HOME_TEL              varchar(30) DEFAULT '',
  CELLPHONE             varchar(20) DEFAULT '',
  EMAIL                 varchar(100) DEFAULT '',
  ADDRESSCODE           varchar(10) DEFAULT '',
  DELIVERADDRESS        nvarchar(250) DEFAULT '',
  PRICELEVEL            smallint DEFAULT 0,
  DISCOUNTRATE          decimal(6,2) DEFAULT 0,
  CREDITTOTAL           decimal(18,6) DEFAULT 0,
  AREAID                varchar(4) DEFAULT '',
  CURRENCYID            varchar(4) DEFAULT '',
  MEMODOC               nvarchar(2000) DEFAULT '',
  DATAEXTEND1           nvarchar(200) DEFAULT '',
  DATAEXTEND2           nvarchar(200) DEFAULT '',
  DATAEXTEND3           nvarchar(200) DEFAULT '',
  DATAEXTEND4           nvarchar(200) DEFAULT '',
  DATAEXTEND5           nvarchar(200) DEFAULT '',
  DATAEXTEND6           nvarchar(200) DEFAULT '',
  DATAEXTEND7           decimal(12,2) DEFAULT 0,
```

```
  DATAEXTEND8           decimal(12,2) DEFAULT 0,
  CREATE_USER           nvarchar(20) DEFAULT '',
  CREATE_DATE           datetime NULL,
  UPDATE_USER           nvarchar(20) DEFAULT '',
  UPDATE_DATE           datetime,
  SIGN_USER             nvarchar(20) DEFAULT '',
  FLOWFLAG              char(1) DEFAULT '',
  CONSTRAINT PK_MEMBER PRIMARY KEY(MEMBERID)
)
```

```
CREATE TABLE OPOORDER1_M (
  BILLNO                varchar(12) not null,
  BILLDATE              datetime,
  BILLTYPE              smallint DEFAULT 0,
  ONHANDDATE            datetime,
  CUSTID                varchar(10) DEFAULT '',
  DEPARTMENTID          varchar(4) DEFAULT '',
  CONNECTPERSON         nvarchar(60) DEFAULT '',
  PERSONID              varchar(10) DEFAULT '',
  WAREID                varchar(4) DEFAULT '',
  CURRENCYID            varchar(4) DEFAULT '',
  CURRRATE              decimal(12,6) DEFAULT 0,
  MEMODOC               nvarchar(2000) DEFAULT '',
  TOTALAMOUNT           decimal(18,6) DEFAULT 0,
  TAXAMOUNT             decimal(12,2) DEFAULT 0,
  FLOWFLAG              char(1) DEFAULT '',
  CONSTRAINT PK_OPOORDER1_M PRIMARY KEY(BILLNO)
)
```

```
CREATE TABLE OPOORDER1_D (
  BILLNO                varchar(20) NOT NULL,
  SEQNO                 int DEFAULT 0,
  WAREID                varchar(4) DEFAULT '',
  PRODUCTID             varchar(20) DEFAULT '',
  PRODCNAME             nvarchar(30) DEFAULT '',
  PRODSTRUCTURE         nvarchar(30) DEFAULT '',
  QUANTITY              decimal(18,6) DEFAULT 0,
  UNIT                  varchar(10) DEFAULT '',
  PRICE                 decimal(18,6) DEFAULT 0,
  SUBAMOUNT             decimal(18,6) DEFAULT 0,
  UNIQUENO              int IDENTITY(1,1)
  CONSTRAINT PK_OPOORDER1_D PRIMARY KEY(BILLNO,UNIQUENO)
```

```
)
```

```
CREATE TABLE OPOORDER2_M (
  BILLNO                varchar(12) not null,
  BILLDATE              datetime,
  BILLTYPE              smallint DEFAULT 0,
  ONHANDDATE            datetime,
  CUSTID                varchar(10) DEFAULT '',
  DEPARTMENTID          varchar(4) DEFAULT '',
  CONNECTPERSON         nvarchar(60) DEFAULT '',
  PERSONID              varchar(10) DEFAULT '',
  WAREID                varchar(4) DEFAULT '',
  CURRENCYID            varchar(4) DEFAULT '',
  CURRRATE              decimal(12,6) DEFAULT 0,
  MEMODOC               nvarchar(2000) DEFAULT '',
  TOTALAMOUNT           decimal(18,6) DEFAULT 0,
  TAXAMOUNT             decimal(12,2) DEFAULT 0,
  FLOWFLAG              char(1) DEFAULT '',
  CONSTRAINT PK_OPOORDER2_M PRIMARY KEY(BILLNO)
)
```

```
CREATE TABLE OPOORDER2_D (
  BILLNO                varchar(20) NOT NULL,
  SEQNO                 int DEFAULT 0,
  WAREID                varchar(4) DEFAULT '',
  PRODUCTID             varchar(20) DEFAULT '',
  PRODCNAME             nvarchar(30) DEFAULT '',
  PRODSTRUCTURE         nvarchar(30) DEFAULT '',
  QUANTITY              decimal(18,6) DEFAULT 0,
  UNIT                  varchar(10) DEFAULT '',
  PRICE                 decimal(18,6) DEFAULT 0,
  SUBAMOUNT             decimal(18,6) DEFAULT 0,
  UNIQUENO              int IDENTITY(1,1)
  CONSTRAINT PK_OPOORDER2_D PRIMARY KEY(BILLNO,UNIQUENO)
)
```

以上是 MS-SQL 的語法，其他資料庫要修改處說明如下，相較之下，MySQL 和 MS-SQL 的語法最接近。

資料庫	報錯誤	正確
MySQL	不認識行末的注解 --	
	int IDENTITY(1,1)	int auto_increment 請注意 PK 不能用複合鍵值

		PRIMARY KEY(UNIQUENO)
PostgreSQL	無 nvarchar	改用 varchar
	無 datetime	改用 timestamp(或 date)
	int IDENTITY(1,1)	SERIAL UNIQUE, PK 和 MySQL 不同，可以用複合鍵值 (同 MS-SQL)
FireBird	無 nvarchar	改用 varchar
	無 datetime	改用 timestamp(或 date)
	DEFAULT 後不能(要)加 null	
	int IDENTITY(1,1)	V2.x 版只能利用 trigger 處理 V3.0(beta 版測試中) id integer generated by default as identity primary key,

4. 刪除表(Table)
   語法：DROP TABLE tabname
   實例：DROP TABLE MEMBER

5. 增加一個欄位
   語法：Alter table tabname add column col type
   實例：ALTER TABLE MEMBER ADD NEWFIELD VARCHAR(20) DEFAULT "

6. 添加主鍵： Alter table tabname add primary key(col)
   刪除主鍵： Alter table tabname drop primary key(col)

7. 新增一個 Store Procedure CreateDemoData 自動寫入 1000 筆資料

```
/*
  以下為 MySQL 的程式寫法
*/
CREATE PROCEDURE CreateDemoData(iCnt int)
BEGIN
  DECLARE i INT DEFAULT 1;
  DECLARE sNo varchar(100);
  WHILE (i<=iCnt) DO
   SET sNo = LPAD(LTRIM(CAST(i AS CHAR)),9,'0');
   INSERT INTO member (memberid)
   VALUES (CONCAT('M',sNo));

   INSERT INTO OPOORDER1_M (billno,billdate,personid)
   VALUES (CONCAT('B',sNo), CURDATE() , CONCAT('M',sNo));
```

```
    INSERT INTO OPOORDER1_D (billno,seqno,uniqueno)
    VALUES (CONCAT('B',sNo), 1,i);

    SET i=i+1;
  END WHILE;
END;
```

```
/*
  以下為 PostgreSQL 的程式寫法
*/
CREATE OR REPLACE FUNCTION CreateDemoData(iCnt integer) returns int4 as $$
DECLARE
  i integer ;
  sNo varchar(100);
  s1   varchar(100);
  s2   varchar(100);
BEGIN
  i = 1;
  WHILE i <= iCnt LOOP
    sNo = LPAD(LTRIM(CAST(i AS VARCHAR)),9,'0');
    s1 = 'M' || sNo;
    s2 = 'B' || sNo;
    INSERT INTO member (memberid) VALUES (s1);
    INSERT INTO OPOORDER1_M (billno,billdate,personid) VALUES (s2, GETDATE() , s1);
    INSERT INTO OPOORDER1_D (billno,seqno,uniqueno) VALUES (s2, 1,i);

    i = i + 1;
  END LOOP;
  RETURN i;
END;
$$ language plpgsql;
```

```
CREATE FUNCTION GETDATE() RETURNS date
/*
  資料庫別: MySQL
  目的: 確保 ms-sql 的 Store Procedure 轉換時的相容性
  Author: Michael
  Date:    2014/10/22
*/
BEGIN
    RETURN CURDATE();
```

```
END
```

```
CREATE function GETDATE() RETURNS timestamp as $$
/*
  資料庫別: PostgreSQL
  目的: 確保 PostgreSQL 的 Store Procedure 轉換時的相容性
  Author: Michael
  Date:   2014/10/30
*/
BEGIN
    RETURN current_timestamp;
END
$$ language plpgsql;
```

```
CREATE function GETDATE() RETURNS timestamp
/*
  資料庫別: FireBird
  目的: 確保 ms-sql 的 Store Procedure 轉換時的相容性
  說明: FireBird V3.0 大改版後, 才支援這種寫法, 之前的版本要用dll, 非db標準作法
  Author: Michael
  Date:   2014/10/22
*/
AS
BEGIN
    RETURN current_timestamp;
END
```

呼叫 Store Procedure 的方法

資料庫	呼叫方法	執行時間
MS-SQL	EXEC CreateDemoData	
MySQL	Call CreateDemoData()	1 min 40sec
PostgreSQL	Select CreateDemoData(100)	0.356 sec
FireBird	EXECUTE PROCEDURE CreateDemoData;	Psql 功能有限，尚無法執行

Store Procedure 語法差異

	MS-SQL	MySQL	PostgreSQL	FireBird
字串相加	'A'+'B'+'C'	CONCAT('A', 'B', 'C')	'A'||'B'||'C' 在預存程序時，有和 FireBird 的相同問題，不過	'A'||'B'||'C' 但是下列 SQL 無法執行 INSERT INTO member

			可以透過變數解決，FireBird 連變數都無解	(memberid) VALUES ('M' || s);
今天日期	GETDATE()	CURDATE() 可以寫個 func 代替		V3.0 作法同 MySQL

8. 常用 SQL 語法

選取資料	SELECT * FROM MEMBER WHERE MEMBERID >= '0' AND MEMBERID <= '0000000100'
	select * from MEMBER where MEMBERID like '%10%'
新增資料	insert into MEMBER (memberid,membername) values('M099', 'Michael Chen')
	insert into MEMBER(memberid,membername) select custid,coprcname from bascustomer where custid like 'M%'
刪除資料	delete from MEMBER WHERE MEMBERID > '0000000900'
更新資料	update MEMBER set MEMBERLAVEL = '00' WHERE MEMBERID <= '0000000100'
排序	select * from MEMBER order by MEMBERID
計算筆數	select count(*) as totalcount from MEMBER
合計	select productid,sum(Quantity) as sumvalue from OPOORDER1_D group by productid
求平均值	select billno,avg(TotalAmount) as avgvalue from OPOORDER1_M group by billno
取最大值	select custid,max(TotalAmount) as maxvalue from OPOORDER1_M group by custid
取最小值	select custid,min(billdate) as maxvalue from OPOORDER1_M group by custid

9. 多表聯合、組合 SQL 語法

UNION [ALL]	UNION 運算子通過組合其他兩個結果表（例如 TABLE1 和 TABLE2）並消去表中任何重複行而派生出一個結果表。當 ALL 隨 UNION 一起使用時（即 UNION ALL），不消除重複行。
	Select 1 as flag,A.billno,B.billdate,A.subaqmount from opoorder1_d A left join opoorder1_m B on A.billno = b.billno UNION ALL

	Select 2 as flag,A.billno,B.billdate,A.subaqmount from opoorder2_d A left join opoorder2_m B on A.billno = b.billno
EXCEPT [ALL]	EXCEPT 運算子通過包括所有在 TABLE1 中但不在 TABLE2 中的行並消除所有重複行而派生出一個結果表。當 ALL 隨 EXCEPT 一起使用時 (EXCEPT ALL)，不消除重複行。
	Select 1 as flag,A.billno,B.billdate,A.subaqmount from opoorder1_d A left join opoorder1_m B on A.billno = b.billno EXCEPT ALL Select 2 as flag,A.billno,B.billdate,A.subaqmount from opoorder2_d A left join opoorder2_m B on A.billno = b.billno
INTERSECT [ALL]	INTERSECT 運算子通過只包括 TABLE1 和 TABLE2 中都有的行並消除所有重複行而派生出一個結果表。當 ALL 隨 INTERSECT 一起使用時 (INTERSECT ALL)，不消除重複行。
	Select 1 as flag,A.billno,B.billdate,A.subaqmount from opoorder1_d A left join opoorder1_m B on A.billno = b.billno INTERSECT ALL Select 2 as flag,A.billno,B.billdate,A.subaqmount from opoorder2_d A left join opoorder2_m B on A.billno = b.billno

10. 進階 SQL 語法

子查詢	select * from member where memberid IN (select custid from bascustomer where custid like 'M0%')
	select * from member where memberid IN ('M001','M009','M012')
	select a.title,a.username,b.adddate from table a,(select max(adddate) adddate from table where table.title=a.title) b
	select * from (SELECT memberid,membername FROM member) T where T.memberid >= 'M001'
	/*兩張關聯表，刪除主表中已經在副表中沒有的資訊 */ delete from table1 where not exists (select * from table2 where table1.field1=table2.field1)
資料庫分頁	select top 10 b.* from (select top 20 主鍵欄位,排序欄位 from 表名 order by 排序欄位 desc) a,表名 b where b.主鍵欄位 = a.主鍵欄位 order by a.排序欄位
	/* 選擇從 10 到 15 的記錄 */ select top 5 * from (select top 15 * from table order by id asc) table_別名 order by id desc
	declare @start int,@end int @sql nvarchar(600) set @sql='select top'+str(@end-@start+1)+'+from T where rid not in(select top'+str(@str-1)+'Rid from T where Rid>-1)'

	exec sp_executesql @sql
刪除重複記錄	delete from tablename where id not in (select max(id) from tablename group by col1,col2,...)
條件式加總	select type,sum(case vender when 'A' then pcs else 0 end),sum(case vender when 'C' then pcs else 0 end),sum(case vender when 'B' then pcs else 0 end) FROM tablename group by type

11. End

第八章
進階的 SQL 語法

故兵聞拙速，未睹巧之久也。
夫兵久而國利者，未之有也。
故不盡知用兵之害者，則不能盡知用兵之利也。

孫子兵法 作戰篇

第八章進階的 SQL 語法

在第一章中，筆者就曾經提醒各位，SQL 的語法一點都不難。事實上，跟越來越複雜的程式開發工具相較，他簡直就是超級美模的化身－健美、苗條，充滿速度和知性美。但是，觀念是最難跨越的門檻，一個腦袋中只有迴圈的研發工程師，是學不好 SQL 的，即使學了也僅學到皮毛而已。必須用「批量」的觀念來學習，習慣性的從上而下的俯瞰整個資料庫的檔案結構，如此必有所得。

各位同學老是聽我這麼說，耳朵都生繭了，心中難免嘀嘀咕咕，Michael 你就不能直接上菜嗎？聽你說的母豬都飛上天了，直接給個例子瞧瞧吧。行，我就先拿一個常見的 Store Procedure 來跟各位實際說明。

下面這個預存程序（Store Procedure）是進銷存系統中，重置產品(材料)期末數量的方法，通常經過一段時間的使用，有時後會發生期末數量不正確的情況發生，此時就可以透過執行本程式修正錯誤。關於這段程式的詳細說明，我會在下一章的實例詳細說明，這裏只是先搶先嘗嘗鮮而已。

```
CREATE PROC dbo.GEX_RESETQTY
(
   @PROD1    NVARCHAR(20),
   @PROD2    NVARCHAR(20),
   @RESETTYPE NCHAR(1),
   @DATE2    VARCHAR(10)
)
/*
   Function    : 進銷存數量重置主程式
   Description:
      目前的想法是將所有歷史重置的資料都統一寫到 FIVINVFINAL_HISTORY, 用 RESETDATE 區分不同時期的重置
      風險是單一 Table 會有過大的疑慮, 但好處是固定的 Table 名稱, 管理及 SQL 語法都較方便處理
      不含生管的數量重置作業, 可以支援單一產品或全產品重置
      執行範例    EXEC GEX_RESETQTY '','ZZ','1','2099/12/31'

      RESETTYPE = 1 全重置
                  2 從頭到指定重置日
                  3 從前次重置日到指定重置日
                  4 從前次重置日到最後
                  9 期初期末存量表等報表要臨時使用的期初存量(取消)
      1,4 寫到 FIVINVFINAL
      2,3 寫到 FIVINVFINAL_HISTORY
```

```
    目前只要使用 1 和 2 即可, 期末數量和單據筆數關連不大
    但是成本重置的 SP, 就要分段計算, 這是效能問題
  Build Date : 2011/07/13

  Modify History
    Item Who          Date          Modify Docs
    ==== ========== ========== ============================================
    1    Michael    2011/07/13 加入註解
    2    Michael    2011/11/10 修正產品引數傳入星號(zz)的 SQL 問題
    3    Michael    2013/03/18 將數量欄位為 null 更新為 0

*/
AS

--要先停掉 Trigger
DISABLE TRIGGER DBO.TRI_INVSTKBILL1_CALCCOST ON dbo.INVSTKBILL1_D;
DISABLE TRIGGER DBO.TRI_INVSTKBILL2_CALCCOST ON dbo.INVSTKBILL2_D;
DISABLE TRIGGER DBO.TRI_INVSTKBILL3_CALCCOST ON dbo.INVSTKBILL3_D;

IF @RESETTYPE = 1 OR @RESETTYPE = 4
BEGIN
  SET @DATE2 = '2199/12/31'
END

IF @RESETTYPE = '' OR @RESETTYPE = '*' BEGIN SET @RESETTYPE = '1' END
IF @DATE2 = '' BEGIN SET @DATE2 = '2199/12/31' END

--FOR 擴充機制使用
DECLARE @LOGINID VARCHAR(60)
DECLARE @FUNCTAG VARCHAR(30)

SELECT @LOGINID=GETDATE()+Round(RAND() * 10000, 0)
SET @FUNCTAG = 'GEX_RESETQTY'

--擴充機制
EXEC GEXRPT_INVRPT_EXTEND @PROD1,@PROD2,'','zz','1900/01/01',@DATE2,@FUNCTAG,'INV_INIT',@LOGINID,1

SELECT A.PRODUCTID, B.WAREID INTO #INVALL FROM BASPRODUCT A LEFT JOIN BASWAREHOUSE B ON 1 = 1
WHERE CASE WHEN @PROD1 = '*' THEN @PROD1 ELSE A.PRODUCTID END >= @PROD1
  AND CASE WHEN @PROD2 = '*' THEN @PROD2 ELSE A.PRODUCTID END <= @PROD2
```

```
IF @RESETTYPE = 1 OR @RESETTYPE = 4
BEGIN
   DELETE FROM FIVINVFINAL WHERE NOT EXISTS
   (
      SELECT * FROM BASPRODUCT
      WHERE PRODUCTID = FIVINVFINAL.PRODUCTID
   )

   DELETE FROM FIV_LPS WHERE LOWPOINTSTOCK IS NULL

   INSERT INTO FIVINVFINAL
(PRODUCTID,WAREID,LOWPOINTSTOCK,NOWQUANTITY,NOWCOST,BORROWOUTQTY,BORROWINQTY,RETURNINQTY,RETURNOUT
QTY)
   SELECT A.PRODUCTID,A.WAREID,0,0,0,0,0,0,0 FROM #INVALL A
   LEFT JOIN FIVINVFINAL B ON A.PRODUCTID = B.PRODUCTID AND A.WAREID = B.WAREID
   WHERE B.NOWQUANTITY IS NULL
   AND CASE WHEN @PROD1 = '*' THEN @PROD1 ELSE A.PRODUCTID END >= @PROD1
   AND CASE WHEN @PROD2 = '*' THEN @PROD2 ELSE A.PRODUCTID END <= @PROD2

   --保留安全庫存設定
   INSERT INTO FIV_LPS (PRODUCTID,WAREID,LOWPOINTSTOCK)
   SELECT A.PRODUCTID,A.WAREID,ISNULL(A.LOWPOINTSTOCK,0) FROM FIVINVFINAL A
   LEFT JOIN FIV_LPS B ON A.PRODUCTID = B.PRODUCTID AND A.WAREID = B.WAREID
   WHERE B.LOWPOINTSTOCK IS NULL
   AND CASE WHEN @PROD1 = '*' THEN @PROD1 ELSE A.PRODUCTID END >= @PROD1
   AND CASE WHEN @PROD2 = '*' THEN @PROD2 ELSE A.PRODUCTID END <= @PROD2

   UPDATE FIV_LPS SET LOWPOINTSTOCK = (SELECT LOWPOINTSTOCK FROM FIVINVFINAL WHERE FIV_LPS.PRODUCTID =
FIVINVFINAL.PRODUCTID AND FIV_LPS.WAREID = FIVINVFINAL.WAREID)
END
ELSE IF @RESETTYPE = 2 OR @RESETTYPE = 3
BEGIN
   INSERT INTO FIVINVFINAL_HISTORY
(RESETDATE,PRODUCTID,WAREID,LOWPOINTSTOCK,NOWQUANTITY,NOWCOST,BORROWOUTQTY,BORROWINQTY,RETURNINQTY,
RETURNOUTQTY)
   SELECT @DATE2,A.PRODUCTID,A.WAREID,0,0,0,0,0,0,0 FROM #INVALL A
   LEFT JOIN FIVINVFINAL_HISTORY B ON A.PRODUCTID = B.PRODUCTID AND A.WAREID = B.WAREID AND RESETDATE = @DATE2
   WHERE B.NOWQUANTITY IS NULL
   AND CASE WHEN @PROD1 = '*' THEN @PROD1 ELSE A.PRODUCTID END >= @PROD1
```

```
   AND CASE WHEN @PROD2 = '*' THEN @PROD2 ELSE A.PRODUCTID END <= @PROD2
END;

WITH INV1
AS
(
   SELECT A.PRODUCTID, A.WAREID, SUM(A.QUANTITY) AS QTY1 FROM INVSTKBILL1_D A
   LEFT JOIN INVSTKBILL1_M B ON A.BILLNO = B.BILLNO
   WHERE B.BILLDATE <= @DATE2 GROUP BY A.PRODUCTID, A.WAREID
), INV2
AS
(
   SELECT A.PRODUCTID, A.WAREID, SUM(A.QUANTITY) AS QTY2 FROM INVSTKBILL2_D A
   LEFT JOIN INVSTKBILL2_M B ON A.BILLNO = B.BILLNO
   WHERE B.BILLDATE <= @DATE2 GROUP BY A.PRODUCTID, A.WAREID
), INV3
AS
(
   SELECT A.PRODUCTID, A.WAREID, SUM(A.QUANTITY) AS QTY3 FROM INVSTKBILL3_D A
   LEFT JOIN INVSTKBILL3_M B ON A.BILLNO = B.BILLNO
   WHERE B.BILLDATE <= @DATE2 GROUP BY A.PRODUCTID, A.WAREID
), INV41
AS
(
   SELECT A.PRODUCTID, A.WAREINWAREHOUSE, SUM(A.QUANTITY) AS QTY41 FROM INVSTKBILL4_D A
   LEFT JOIN INVSTKBILL4_M B ON A.BILLNO = B.BILLNO
   WHERE B.BILLDATE <= @DATE2 GROUP BY A.PRODUCTID, A.WAREINWAREHOUSE
), INV42
AS
(
   SELECT A.PRODUCTID, A.WAREID, SUM(A.QUANTITY) AS QTY42 FROM INVSTKBILL4_D A
   LEFT JOIN INVSTKBILL4_M B ON A.BILLNO = B.BILLNO
   WHERE B.BILLDATE <= @DATE2 GROUP BY A.PRODUCTID, A.WAREID
), INV5
AS
(
   SELECT A.PRODUCTID, A.WAREID, SUM(A.QUANTITY) AS QTY5 FROM INVSTKBILL5_D A
   LEFT JOIN INVSTKBILL5_M B ON A.BILLNO = B.BILLNO
   WHERE B.BILLDATE <= @DATE2 GROUP BY A.PRODUCTID, A.WAREID
), INV6
```

```
AS
(
   SELECT A.PRODUCTID, A.WAREID, SUM(A.QUANTITY) AS QTY6 FROM INVSTKBILL6_D A
   LEFT JOIN INVSTKBILL6_M B ON A.BILLNO = B.BILLNO
   WHERE B.BILLDATE <= @DATE2 GROUP BY A.PRODUCTID, A.WAREID
), INV71
AS
(
   SELECT A.M_PRODID, A.WAREID, SUM(A.STANDARDBATCHQTY) AS QTY71 FROM INVSTKBILL7_M A
   WHERE A.BILLDATE <= @DATE2 GROUP BY A.M_PRODID, A.WAREID
), INV72
AS
(
   SELECT A.PRODUCTID, A.WAREID, SUM(A.QUANTITY) AS QTY72 FROM INVSTKBILL7_D A
   LEFT JOIN INVSTKBILL7_M B ON A.BILLNO = B.BILLNO
   WHERE B.BILLDATE <= @DATE2 GROUP BY A.PRODUCTID, A.WAREID
), INV77
AS
(
   --擴充機制
   SELECT A.PRODUCTID, A.WAREID, SUM(A.QUANTITY) AS QTY77 FROM INVEXTEND_INIT A
   WHERE A.LOGINID = @LOGINID AND FUNCTAG = @FUNCTAG
   GROUP BY A.PRODUCTID, A.WAREID
), FINAL99
AS
(
   SELECT A.PRODUCTID, A.WAREID, ISNULL(B.QTY1,0) AS QTY1, ISNULL(C.QTY2,0) AS QTY2, ISNULL(D.QTY3,0) AS QTY3,
   ISNULL(E.QTY41,0) AS QTY41, ISNULL(F.QTY42,0) AS QTY42, ISNULL(G.QTY5,0) AS QTY5, ISNULL(H.QTY6,0) AS QTY6,
   ISNULL(I.QTY71,0) AS QTY71, ISNULL(J.QTY72,0) AS QTY72, ISNULL(K.QTY77,0) AS QTY77
   FROM #INVALL A
   LEFT JOIN INV1 B ON A.PRODUCTID = B.PRODUCTID AND A.WAREID = B.WAREID
   LEFT JOIN INV2 C ON A.PRODUCTID = C.PRODUCTID AND A.WAREID = C.WAREID
   LEFT JOIN INV3 D ON A.PRODUCTID = D.PRODUCTID AND A.WAREID = D.WAREID
   LEFT JOIN INV41 E ON A.PRODUCTID = E.PRODUCTID AND A.WAREID = E.WAREINWAREHOUSE
   LEFT JOIN INV42 F ON A.PRODUCTID = F.PRODUCTID AND A.WAREID = F.WAREID
   LEFT JOIN INV5 G ON A.PRODUCTID = G.PRODUCTID AND A.WAREID = G.WAREID
   LEFT JOIN INV6 H ON A.PRODUCTID = H.PRODUCTID AND A.WAREID = H.WAREID
   LEFT JOIN INV71 I ON A.PRODUCTID = I.M_PRODID AND A.WAREID = I.WAREID
   LEFT JOIN INV72 J ON A.PRODUCTID = J.PRODUCTID AND A.WAREID = J.WAREID
   LEFT JOIN INV77 K ON A.PRODUCTID = K.PRODUCTID AND A.WAREID = K.WAREID
```

```
), FINALOK
AS
(
   SELECT PRODUCTID, WAREID, (QTY1-QTY2+QTY3+QTY41-QTY42-QTY5+QTY6+QTY71-QTY72+QTY77) AS NOWQTY
   FROM FINAL99
)

--用 WITH 就一定要在下一句 SQL 使用, 只好先寫到 TempDB
SELECT * INTO #FINALOK FROM FINALOK

IF @RESETTYPE = 1 OR @RESETTYPE = 4
BEGIN
   UPDATE FIVINVFINAL SET NOWQUANTITY = B.NOWQTY FROM #FINALOK B WHERE B.PRODUCTID = FIVINVFINAL.PRODUCTID
AND B.WAREID= FIVINVFINAL.WAREID
   UPDATE FIVINVFINAL SET LOWPOINTSTOCK = (SELECT LOWPOINTSTOCK FROM FIV_LPS WHERE FIV_LPS.PRODUCTID =
FIVINVFINAL.PRODUCTID AND FIV_LPS.WAREID = FIVINVFINAL.WAREID)

   DELETE FROM FIVINVFINAL WHERE NOWQUANTITY = 0 AND LOWPOINTSTOCK = 0 AND BORROWOUTQTY = 0 AND
BORROWINQTY = 0 AND RETURNINQTY = 0 AND RETURNOUTQTY = 0
END
ELSE IF @RESETTYPE = 2 OR @RESETTYPE = 3
BEGIN
   UPDATE FIVINVFINAL_HISTORY SET NOWQUANTITY = B.NOWQTY FROM #FINALOK B WHERE B.PRODUCTID =
FIVINVFINAL_HISTORY.PRODUCTID AND B.WAREID= FIVINVFINAL_HISTORY.WAREID
   AND FIVINVFINAL_HISTORY.RESETDATE = @DATE2

   DELETE FROM FIVINVFINAL_HISTORY WHERE NOWQUANTITY = 0 AND LOWPOINTSTOCK = 0 AND BORROWOUTQTY = 0 AND
BORROWINQTY = 0 AND RETURNINQTY = 0 AND RETURNOUTQTY = 0
END

--刪除暫存內容
DELETE FROM INVEXTEND_INIT WHERE LOGINID = @LOGINID AND FUNCTAG = @FUNCTAG

--重置借還貨數量
IF @RESETTYPE = 1 OR @RESETTYPE = 4 BEGIN
   EXEC GEX_RESET_BORRQTY @PROD1,@PROD2
END

--將 NULL 重設為 0
UPDATE FIVINVFINAL SET LOWPOINTSTOCK = 0 WHERE LOWPOINTSTOCK IS NULL
```

```
UPDATE FIVINVFINAL SET BORROWOUTQTY = 0 WHERE BORROWOUTQTY IS NULL
UPDATE FIVINVFINAL SET BORROWINQTY = 0 WHERE BORROWINQTY IS NULL
UPDATE FIVINVFINAL SET RETURNINQTY = 0 WHERE RETURNINQTY IS NULL
UPDATE FIVINVFINAL SET RETURNOUTQTY = 0 WHERE RETURNOUTQTY IS NULL

DROP TABLE #INVALL
DROP TABLE #FINALOK;

--重啟 Trigger
ENABLE TRIGGER DBO.TRI_INVSTKBILL1_CALCCOST ON dbo.INVSTKBILL1_D;
ENABLE TRIGGER DBO.TRI_INVSTKBILL2_CALCCOST ON dbo.INVSTKBILL2_D;
ENABLE TRIGGER DBO.TRI_INVSTKBILL3_CALCCOST ON dbo.INVSTKBILL3_D;
```

欣賞完上面賞心悅目但完全不曉得在幹嘛的程式碼後，接下來，我們在本章中，會繼續介紹幾個進階的 SQL 語法，以及寫 SQL 語法時，和效能有關的幾個議題和注意事項。

下面就是本章要說明的較常用且重要 SQL 進階語法及用法

- CASE WHEN
- 三種暫存表的寫法
- 配合 DB Func 讓 SQL 更靈活、更強大
- Store Procedure 呼叫 Store Procedure 如何傳遞計算暫存內容

1. CASE WHEN

   什麼？有沒有搞錯，CASE WHEN 這麼普遍常用的語法，是哪裡進階了？

   一般而言，我們常用的 SQL 語法，通常都是使用在 Select 欄位時的判斷

   例如

```
--跟據某欄位值，判斷顯示男/女
Select EmpID,Case When Sex='1' Then '男生' Else '女' END 性別 From Employee
```

```
--跟據某欄位有值否，取得不同的欄位內容
SELECT
PERSONID USERID,
CASE WHEN ISNULL(DATAEXTEND5,'') = '' THEN DEPARTMENTID ELSE DATAEXTEND5 END GROUPID
FROM BASPERSON
```

   進階的原因是在於，連 Where 條件都可以用 CASE WHEN 取不同的欄位來判斷，之所以會有這種奇特的需求，是因為這張報表有一個超強的選項

   查詢對帳單時，取得資料的日期區間可以依客戶要求自選（有圖有真相）

   @DATETYPE 就是畫面上的選項，傳給 Store Procedure 的參數值


```
), INV5
AS
(
   SELECT A.IDNO,A.BILLDATE,A.BILLNO,'出貨單-'+A.BILLNO BILLDESC,A.CURRENCYID,A.PERSONID,
   A.TOTALAMOUNT,A.TAXAMOUNT,(A.TOTALAMOUNT+A.TAXAMOUNT) SUMAMT,
   B.QUANTITY,B.UNIT,B.PRICE,B.TAXPRICE,B.SUBAMOUNT,B.BATCHNO
   FROM INVSTKBILL5_M A
   LEFT JOIN INVSTKBILL5_D B ON A.BILLNO = B.BILLNO
   WHERE CASE
      WHEN @DATETYPE = 1 THEN A.BILLDATE
      WHEN @DATETYPE = 2 THEN A.DEFAULTPAYDATE
      WHEN @DATETYPE = 3 THEN A.INVOICEDATE
      ELSE A.BILLDATE
   END >= @BILLDATE1
   AND CASE
      WHEN @DATETYPE = 1 THEN A.BILLDATE
      WHEN @DATETYPE = 2 THEN A.DEFAULTPAYDATE
      WHEN @DATETYPE = 3 THEN A.INVOICEDATE
      ELSE A.BILLDATE
   END <= @BILLDATE2
   --
   AND CASE WHEN @IDNO1 = '*' THEN @IDNO1 ELSE A.IDNO END >= @IDNO1
   AND CASE WHEN @IDNO2 = '*' THEN @IDNO2 ELSE A.IDNO END <= @IDNO2
   AND A.CURRENCYID >= @CURRENCYID1 AND A.CURRENCYID <= @CURRENCYID2
   --
   AND (@MS_CUSTID = 'N' OR EXISTS
   (
      SELECT * FROM DBO.GEXFUNC_GETBAS(@FUNCTAG,'BASCUSTOMER',@LOGINID) GMS
      WHERE A.IDNO = GMS.ID
   ))
)
```

如果能很好的掌握 CASE WHEN 很多客戶希奇古怪的需求，彈彈手指間就輕鬆的解決了，這能更大的擴充 SQL 的能力。在沒有找到這個方式前，我一直都認為只能在前端程式跟據條件組好 SQL 再執行。但是誠如我一再強調的，要讓 SQL 能夠自主的解決所有的問題，前端呼

叫時，永遠只是傳入參數，所有的需求都要用 SQL 語法解決，不要依靠前端程式處理，這樣才有辦法將前端 UI 和後端資料庫獨立，可以隨時切換、重組。如果能做到這樣，才算真正的掌握 SQL 的精髓。

2. 暫存表
   - 實體暫存檔(如 ESS_TMP)
   - WITH GEX01 (XX,XX) AS
   - DECLARE @ACT_TMP1 TABLE
   - #TmpDB、##TmpDB

3. 自定義函式(udf：user define function)
4. Store Procedure 呼叫 Store Procedure