數據被譽為「新時代黑金」，而資料庫是其最核心的基礎架構，能掌握這個關鍵技能，將會是最大贏家。

玩轉SQL

陳威均 著

超過一百個由淺入深的實際案例，帶您快速遨遊MS-SQL、MariaDB 及 PostgreSQL 三大資料庫的SQL 語法天地

目錄

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，重新經歷全部的開發流程。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」？

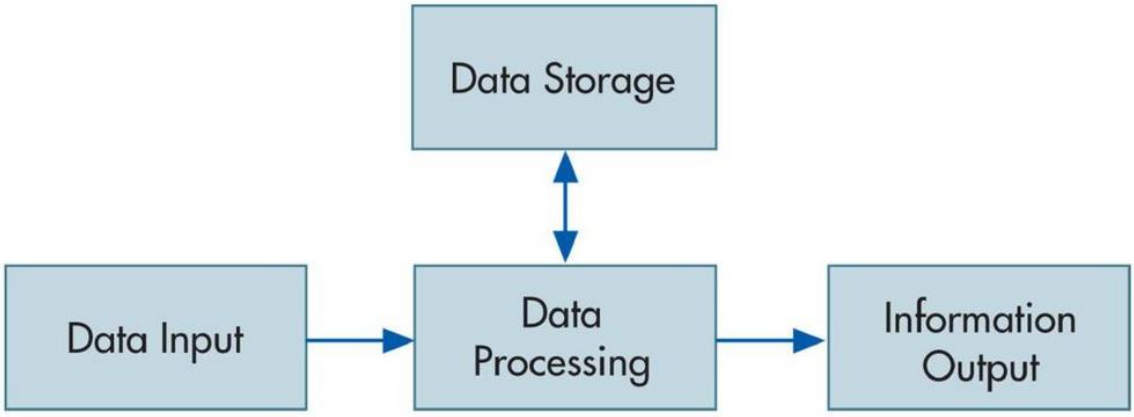
這個傢伙想了一下，和 Michael 仔細討論了計算的邏輯，隔天就展示了一段 Store Procedure，不試就算了，一試之下驚為天人，速度是原先寫程式跑迴圈的舊方法的 10 倍速不止，當下大為折服，從此就成了 SQL 的信徒。

「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？



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運算、取值等，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。本書所介紹的幾個產品，其授權方式列示如下，詳細的內容，請以各產品的官方說明為準。

DataBase	說明
MS-SQL	Microsoft
	Licence：非 Open Source
	商業使用：企業授權
MySQL	MySQL AB(Oracle 子公司)
	Licence：GPL
	商業使用：企業授權
PostgreSQL	PostgreSQL Global Development Group
	Licence：BSD
	商業使用：免費
FireBird	Firebird Foundation
	Licence：Initial Developer's Public License
	商業使用：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，筆者提倡的「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 程式的可移植性。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積。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。

本書採用的是快、狠、準的純實戰方法，以微軟提供的 Northwind(北風)資料庫為基礎，這是一個麻雀雖小但五臟俱全的小型銷售訂單範例資料庫，內含 ERP 系統具代表性的一些 Table，再依本書的需求擴充而成。有了這個整理後的資料庫，在測試 SQL 語法時才能有的放矢，更直觀的感受執行後的結果及效能。因為這是微軟為其 MS-SQL 所提供，其他 OpenSource 資料庫當然沒有，但是網路是個神奇的領域，沒有什麼不可能，所以作者順利的在 gitHub 上找到 PostgreSQL 和 MariaDB(MySQL)的 Northwind 版本(含測試資料)，雖然有發現部分差異，但是修正後仍然可以正常使用。

有了這個統一的資料庫平台，對於我們來說是非常重要的里程碑。因為我們就可以在幾乎完全相同的基礎上，解決相同的問題，這樣就可以一目了然的觀察各資料庫 SQL 的優缺點。本書第十章會以超過 100 個由淺入深的案例，介紹不同資料庫 SQL 語法的差異，雖不能說涵蓋全部的範圍，但常用的 SQL 應該都已包含在內。

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(效能)
- Security(安全性)
- User Friendly(易用性)

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

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

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。

資料庫是什麼? 資料庫（Database）是按照資料結構來組織、存儲和管理資料的電子檔案，它產生於距今五十年前，隨著資訊技術和市場的發展，特別是二十世紀九十年代以後，資料管理不再僅僅是存儲和管理資料，而轉變成使用者所需要的各種資料管理的方式。資料庫有很多種類型，從最簡單的存儲有各種資料的表格到能夠進行海量資料存儲的大型資料庫系統都在各個方面得到了廣泛的應用。

在日常工作中，常常需要把某些相關的資料放進資料庫，並根據管理的需要進行相應的處理。例如，企業的人事部門常要把員工的基本情況(工號、姓名、年齡、性別、籍貫、工資、簡歷等)存放在表中，這張表就可以看成是一個資料庫。有了這個"資料倉庫"我們就可以根據需要隨時查詢員工的基本資料及薪資、保險等相關資訊。此外，在財務管理、倉庫管理、生產管理中也需要建立眾多的這種「資料庫」，使其可以利用電腦實現財務、倉庫、生產的自動化管理。

根據 DB-Engines Ranking，常見的關聯式資料庫列表如下

Rank			DBMS	Database Model	Score		
Dec 2020	Nov 2020	Dec 2019			Dec 2020	Nov 2020	Dec 2019
1.	1.	1.	Oracle	Relational, Multi-model	1325.60	-19.40	-20.80
2.	2.	2.	MySQL	Relational, Multi-model	1255.45	+13.81	-20.21
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1038.09	+0.45	-58.11
4.	4.	4.	PostgreSQL	Relational, Multi-model	547.57	-7.49	+44.20
5.	5.	5.	IBM Db2	Relational, Multi-model	160.43	-1.19	-10.91
6.	6.	↑ 7.	SQLite	Relational	121.68	-1.63	+1.32
7.	7.	↓ 6.	Microsoft Access	Relational	116.74	-0.50	-12.73
8.	8.	8.	MariaDB	Relational, Multi-model	93.61	+1.31	+6.82
9.	9.	↑ 10.	Teradata	Relational, Multi-model	73.83	-1.77	-4.66
10.	10.	↓ 9.	Hive	Relational	70.27	+0.01	-15.78
11.	11.	↑ 14.	Microsoft Azure SQL Database	Relational, Multi-model	69.49	+2.50	+41.60
12.	12.	↓ 11.	SAP Adaptive Server	Relational	54.88	-0.51	-0.66
13.	13.	13.	SAP HANA	Relational, Multi-model	52.50	-1.08	-1.67
14.	14.	↓ 12.	FileMaker	Relational	47.70	+1.03	-7.44
15.	15.	↑ 16.	Google BigQuery	Relational	35.76	+0.68	+10.26
16.	↑ 18.	↑ 17.	Firebird	Relational	22.83	+0.56	+0.39
17.	17.	↓ 15.	Informix	Relational, Multi-model	22.65	+0.08	-2.87
18.	↓ 16.	↑ 19.	Amazon Redshift	Relational	22.42	-0.58	+1.02
19.	19.	↓ 18.	Vertica	Relational, Multi-model	22.21	+0.13	+0.04
20.	20.	↑ 21.	Spark SQL	Relational	19.29	+0.08	+2.59

其中可以看出開源資料庫的使用已經日漸蓬勃，關於開源軟體的說明，請參考如下 Wiki 的相關說明。

> 開源軟體（open source software，縮寫：OSS）又稱開放原始碼軟體，是一種原始碼可以任意取用的電腦軟體，這種軟體的著作權持有人在軟體協定的規定之下保留一部分權利並允許使用者學習、修改以及以任何目的向任何人分發該軟體。開源協定通常符合開放原始碼的定義的要求。一些開源軟體被釋出到公有領域。開源軟體常被公開和合作地開發。開源軟體是開放原始碼開發的最常見的例子，也經常與使用者創作內容做比較。
>
> 開源軟體同時也是一種軟體散布模式。一般的軟體僅可取得已經過編譯的二進位可執行檔，通常只有軟體的作者或著作權所有者等擁有程式的原始碼。

簡單的說，開放原始碼當然是說該軟體會隨附原始程式碼，但並不表示它就一定是免費的，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幾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。

MariaDB

https://mariadb.org/
https://mariadb.com/

MariaDB 是 MySQL 關聯式資料庫管理系統的一個復刻，由社群開發，有商業支援，旨在繼續保持在 GNU GPL 下開源。MariaDB 的開發是由 MySQL 的一些原始開發者領導的，他們擔心甲骨文公司收購 MySQL 後會有一些隱患。

MariaDB 打算保持與 MySQL 的高度相容性，確保具有核心主要功能的直接替換功能，以及與 MySQL API 和命令的精確匹配。MariaDB 內建了一個新的儲存引擎 Aria，它可以替代 MyISAM，成為預設的事務和非事務引擎。它最初使用 XtraDB 作為預設儲存引擎，並從 10.2 版本切換回 InnoDB。

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 去開展商業化的 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，但隨之而來的大量用戶要求淹沒這個計劃。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 年開放原始碼為免費資料庫軟體，我們一般將其類比於 MS-SQL 的簡化版本，是一個小而美資料庫，最大的優點在於它是免費的，但是筆者實際下載 v3.0.x 版測試後認為他和 MariaDB、PostgreSQL 並不是同等級的產品，有太多的非標準作法及限制，加上 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。

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

DataBase	相關的說明
MariaDB	Widely used open source RDBMS(branch from MySQL)
	開發語言：C and C++
	主要支援 OS： FreeBSD Linux OS X Windows
PostgreSQL	Based on the object relational DBMS Postgres
	開發語言：C
	主要支援 OS： HP-UX Linux OS X Unix Windows
Firebird	FireBird 的前身是 Borland 推出的 Interbase 資料庫
	開發語言：C and C++
	主要支援 OS： AIX FreeBSD Linux OS X Unix Windows

SQLLite

https://www.sqlite.org/index.html

SQLite 是一個軟體庫，實現了自給自足的、無伺服器的、零配置的、交易性的 SQL 資料庫引擎。SQLite 是一個很小的 C 語言程式庫，且本身就完全包含資料庫引擎的功能，而且可以嵌入至其他程式中，完全不用額外的設定。

SQLite 與其他一般資料庫差異不大，一般的 SQL-92 語法都能夠使用，而且不需要建立一個資料庫系統，要使用的時候，只要在編譯程式的時候將 SQLite 程式庫一起編入就可以使用。另外，SQLite 的資料庫（database）都是以單一檔案的形式存於磁碟中，不需要再安裝資料庫伺服器軟體，所以要把資料庫複製或建立在你的電腦上是相單簡單快速。

由於本書特別側重 Store Procedure 的開發，但是 SQLLite 因其定位及特性，並不支援 Store Procedure，所以沒有將 SQLLite 列入轉換比較的對象。

MongoDB

https://www.mongodb.com

除了前面提到的關聯式資料庫之外，近幾年 NoSQL 資料庫成為大家時常討論的技術主題，而 MongoDB 則是最廣為人知的開源 NoSQL 資料庫。在 MongoDB 的世界中，資料不像類 Excel 一樣的方正。相反的，Document 的形式就像 JSON 一般， 由多組 key-value 組成，且每筆 Document 的長相可以不同。由於本文重點不在 NoSQL 資料庫，所以就不多談論，有興趣的同學可以自行參考相關的資訊。

第三章

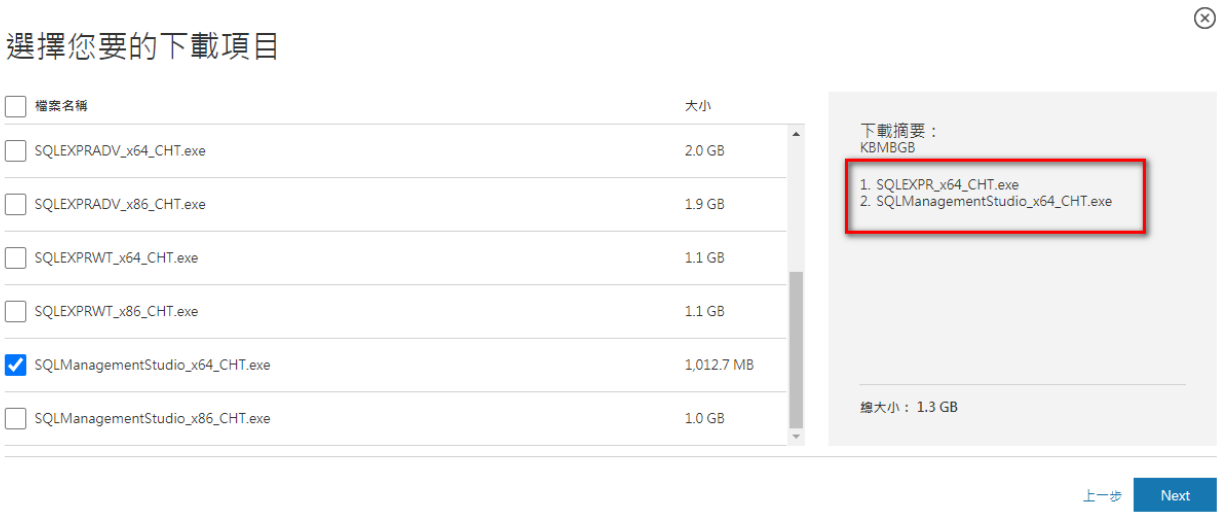
MS-SQL 的安裝及管理工具介紹

在接下來的幾個章節，筆者會分別說明商業版本的 MS-SQL(Express 版) 及 PostgreSQL 及 MariaDB、FireBird 這三套開源碼資料庫管理軟體的安裝及操作。完整的安裝程序，除了資料庫管理程式本身外，還必須安裝配套的管理程式，每一種資料庫程式，通常都會有多套管理工具可以使用，筆者會在稍後一併說明，您可以自行測試並決定要使用哪個管理工具。

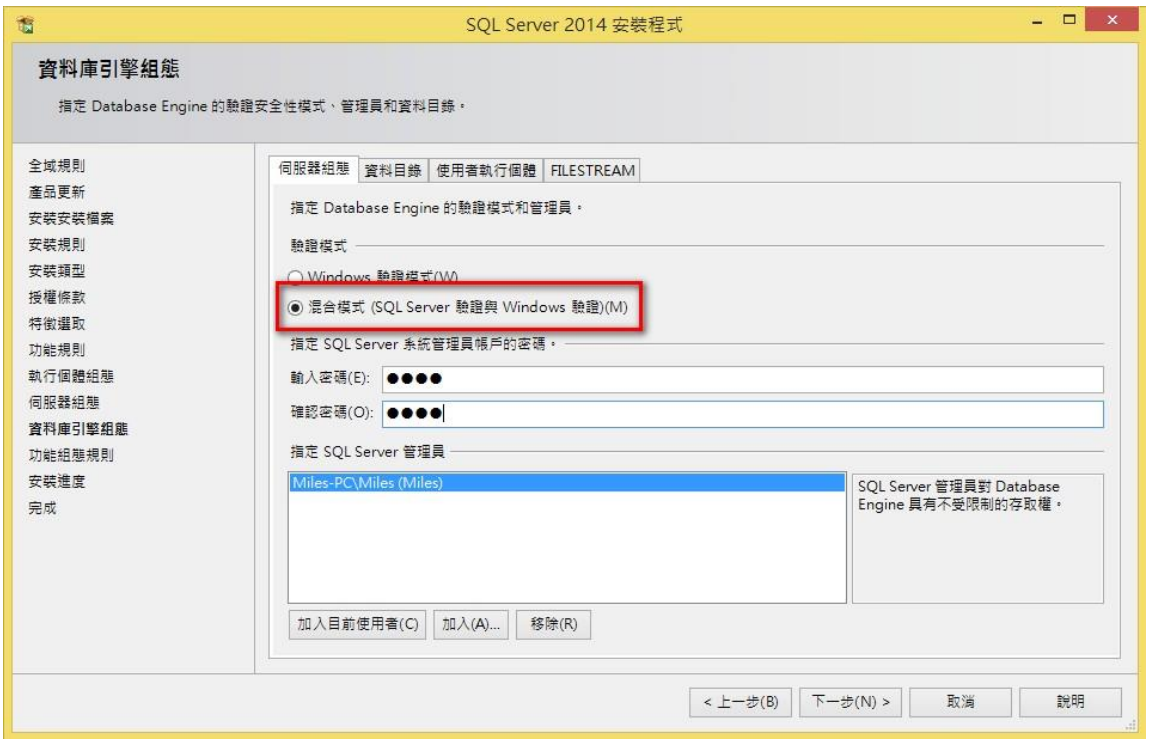
首先，我們就從 MS-SQL 開始介紹。MS-SQL 是由微軟公司在 1989 年推出，根據 DB-Engine 在 2020 年 10 月的排行，它在關聯式資料庫排名第三，僅次於 Oracle 和 MySQL。MS-SQL 有提供不同定位的版本，我們目前要安裝的是它的 Express 版本，雖功能受限但免費，針對資料量筆數不多的中小型應用也足敷使用。(本書安裝的版本是 SQL2012 SP4 的版本)。

下載網址：
SQL2019 Express：https://www.microsoft.com/zh-tw/sql-server/sql-server-downloads
SQL2012 Express：https://www.microsoft.com/zh-tw/download/details.aspx?id=56042

可以根據您的環境選擇安裝的版本，其中 SQLManagementStudio_x64_CHT.exe 這就是 MS-SQL 官方提供的管理工具。



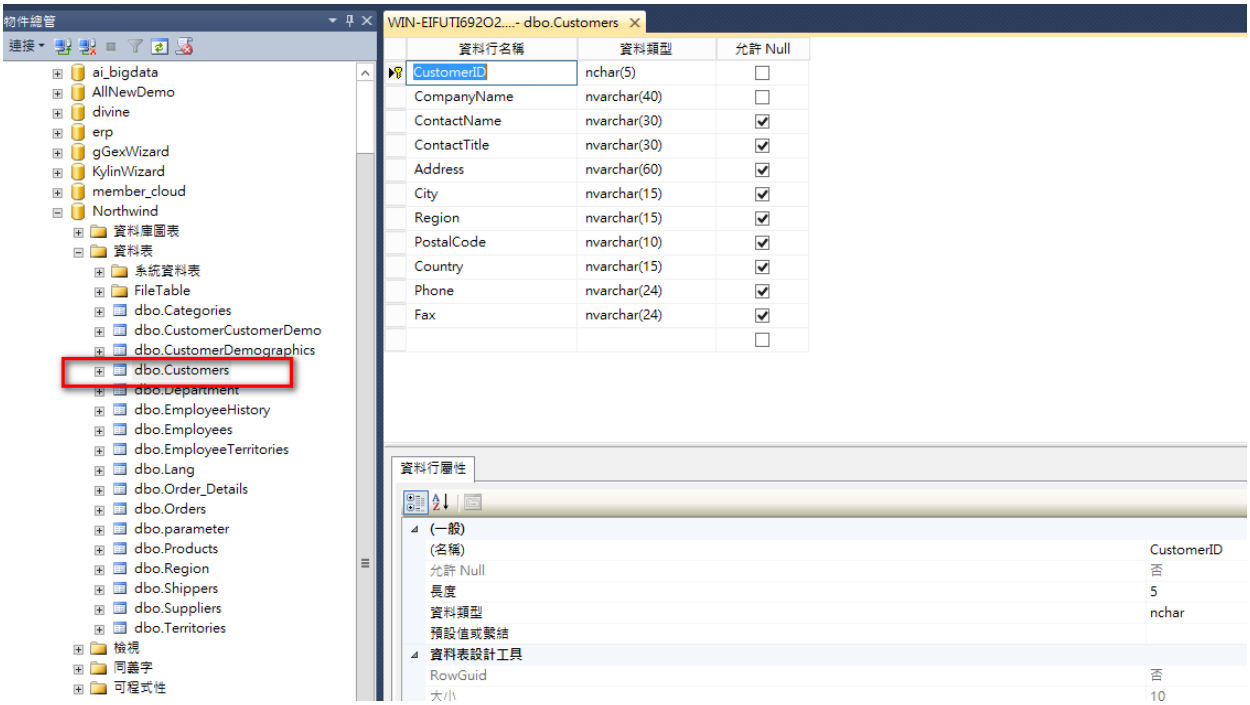
基本上目前的安裝程式都寫得非常智能，過程中只有一個地方要稍加注意，就是設定登入 SQL 時的帳號 sa 的密碼，請參見下圖。



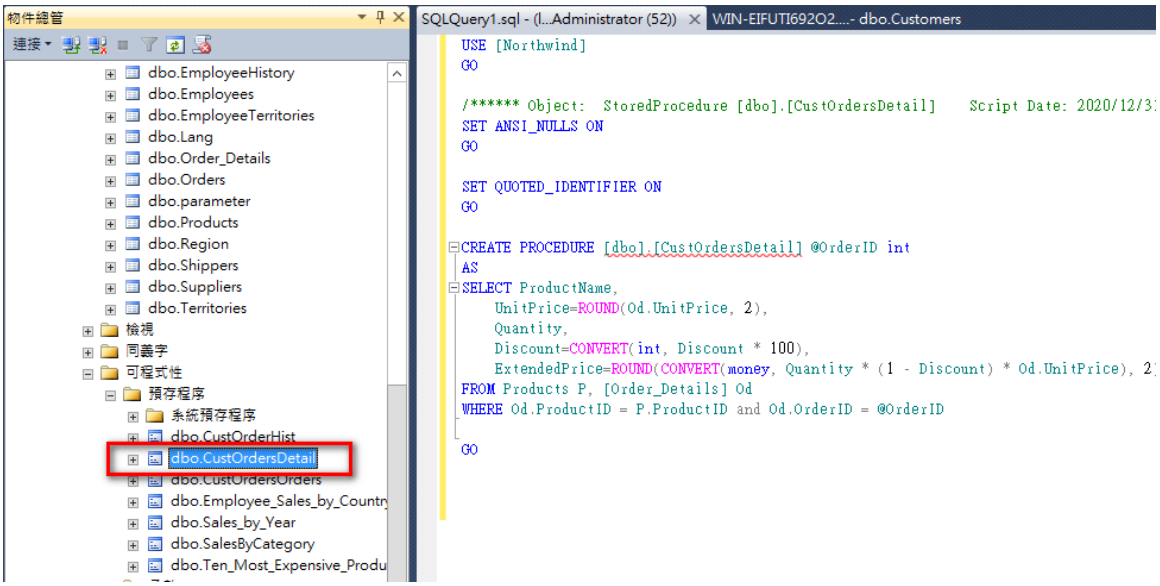
接下來要安裝管理程式 SQLManagementStudio_x64_CHT.exe，一般都是 Step-By-Step 安裝即可。安裝後的執行畫面如下。

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



因為這是官方版本，所以大部分的情況下使用都沒有任何問題，但是如果您的電腦等級不高，您可能會覺得這支程式的執行效能不佳。所以筆者接下來會推薦另外一支我個人日常使用的管理程式 Database.NET。

Database.NET 是一款好用的多資料庫管理軟體，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(但只支援 Windows 作業系統)。
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。

筆者是在一次網路衝浪的過程中，看到了 DataBase.NET 這個程式，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，不用會沒朋友的。

接下來，不囉嗦，就來看看這個超棒的工具

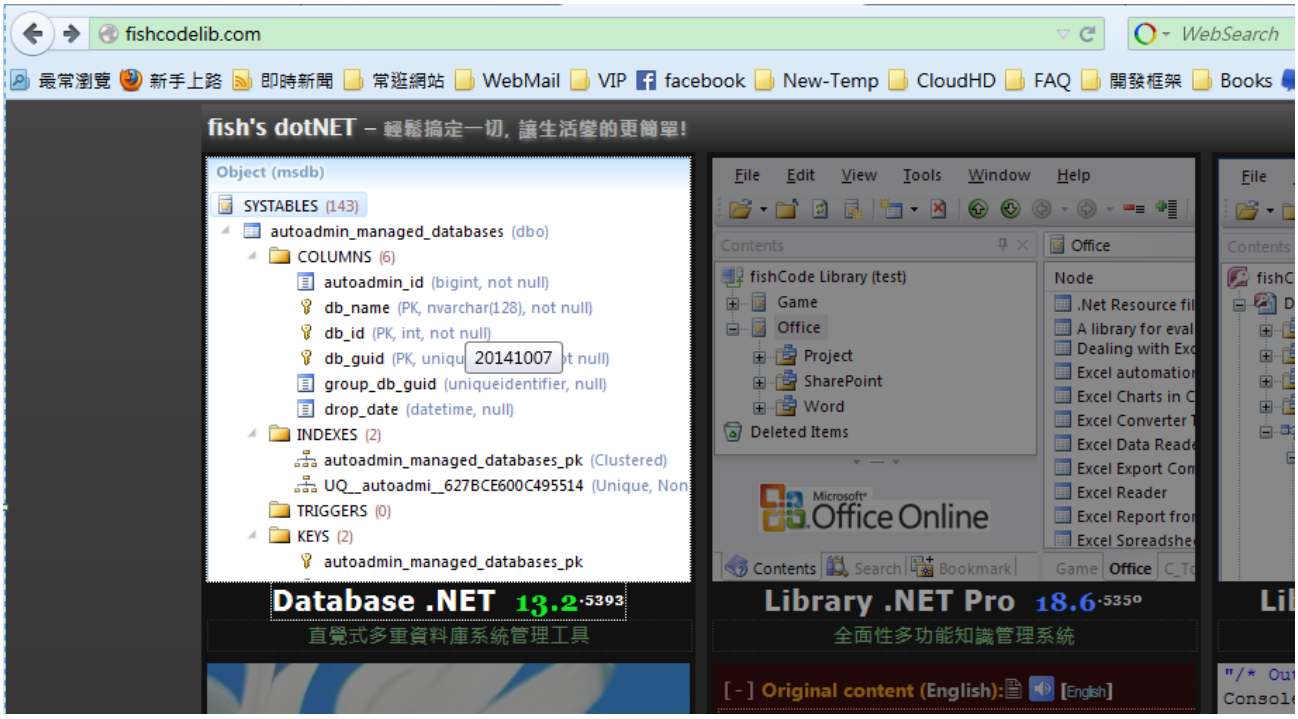
網址：http://fishcodelib.com/

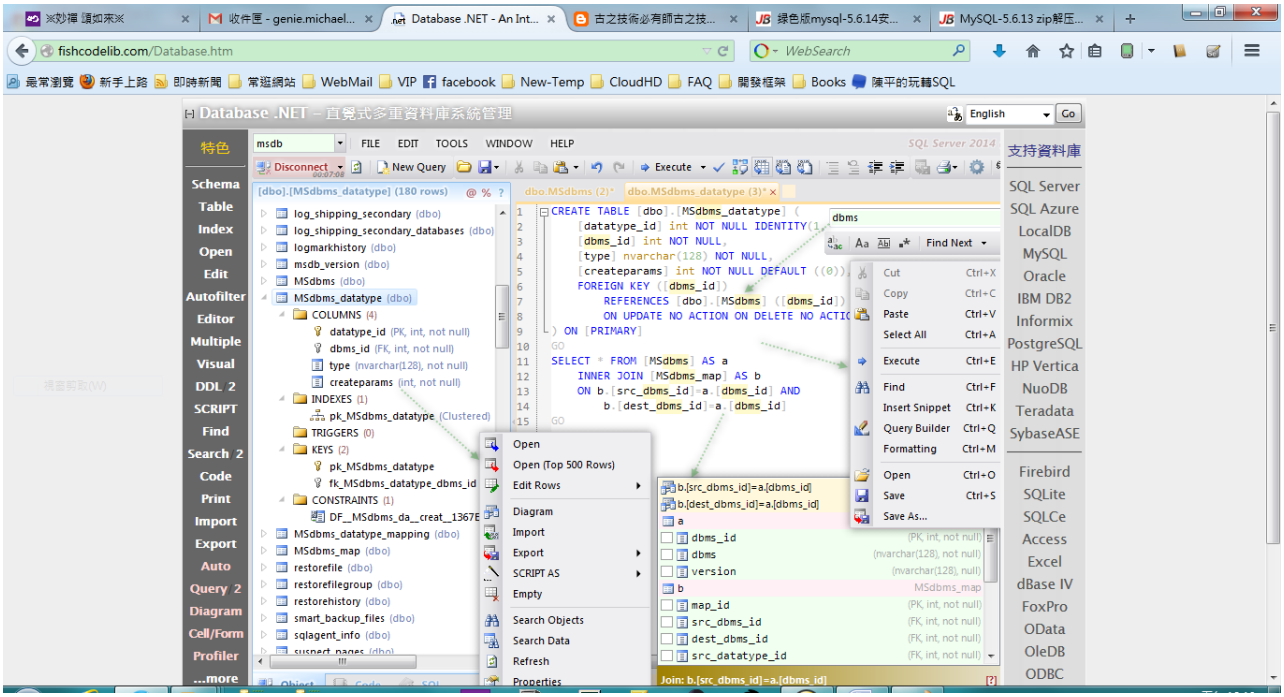
看來 fish 老大的好東西不少，除了 DataBase.NET 外還有

Library.NET Pro

Library.NET Free

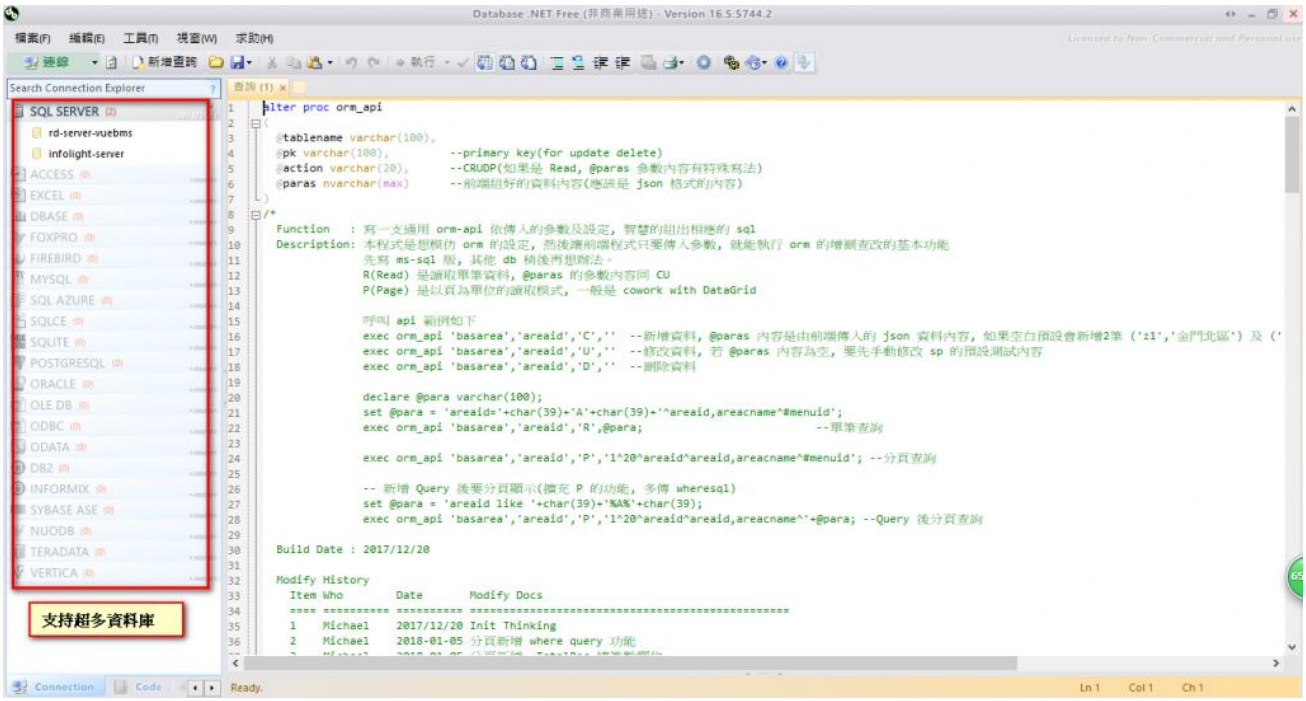
Capture.NET 等等，請自己逛逛



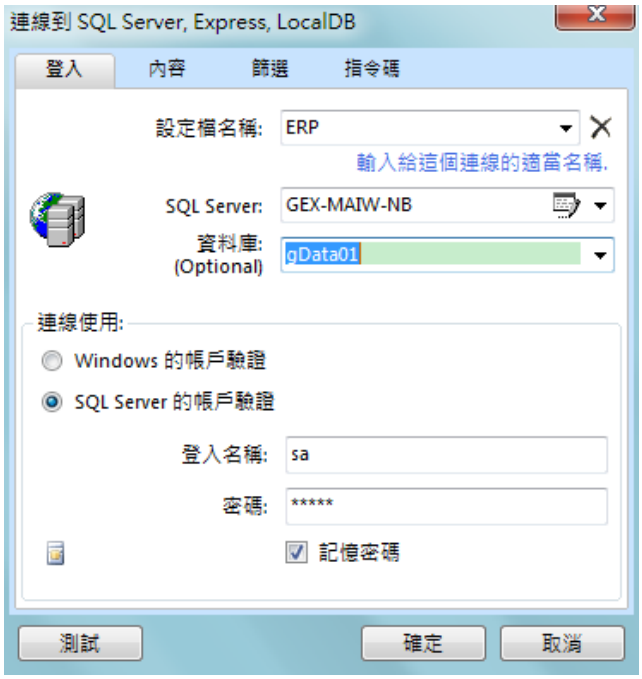


下載後，解壓縮到任一個目錄，就可以開始使用。啟動 DataBase.NET

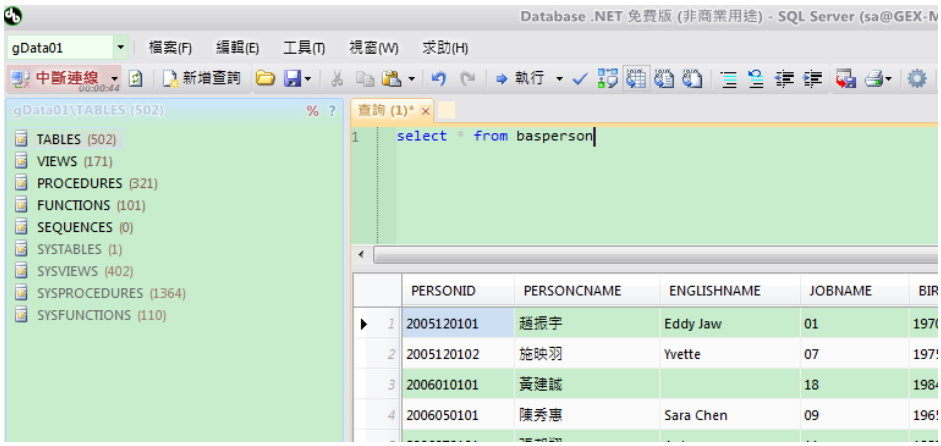
然後就進到主頁面了



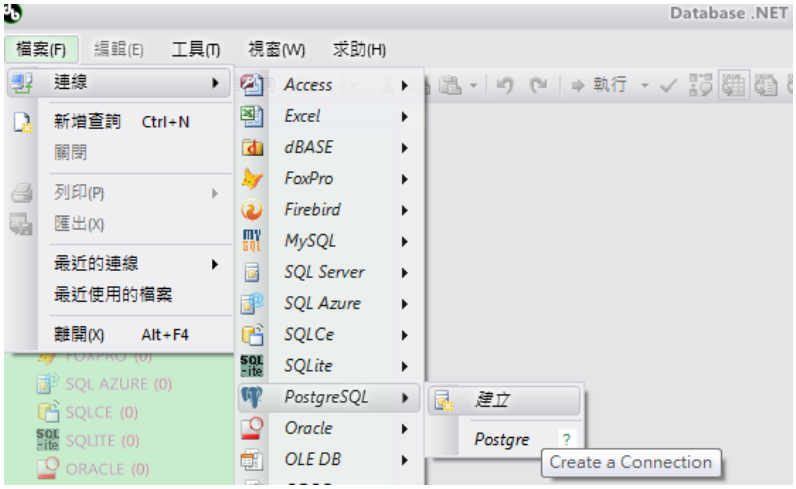
如果要連 MS-SQL，就選 SQL Server－建立 （會新增一個連線）

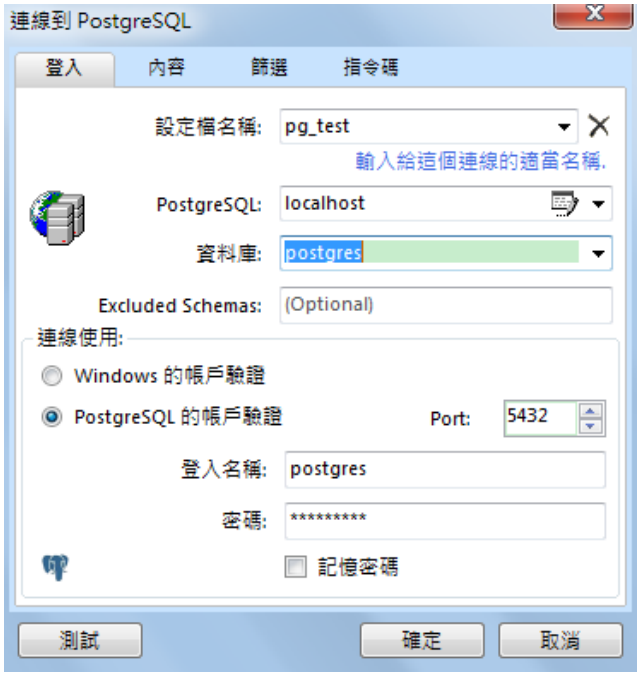


連接好 Database 後，其他的操作就一切如昔

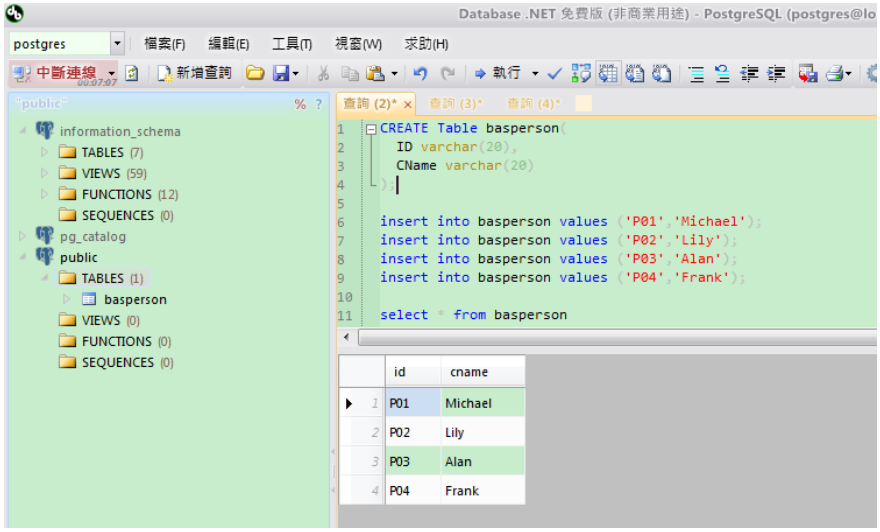


連接其他的資料庫，只要有支援，步驟都差不多，PostgreSQL 基本上同前面的步驟，只有帳號、密碼會有點差異

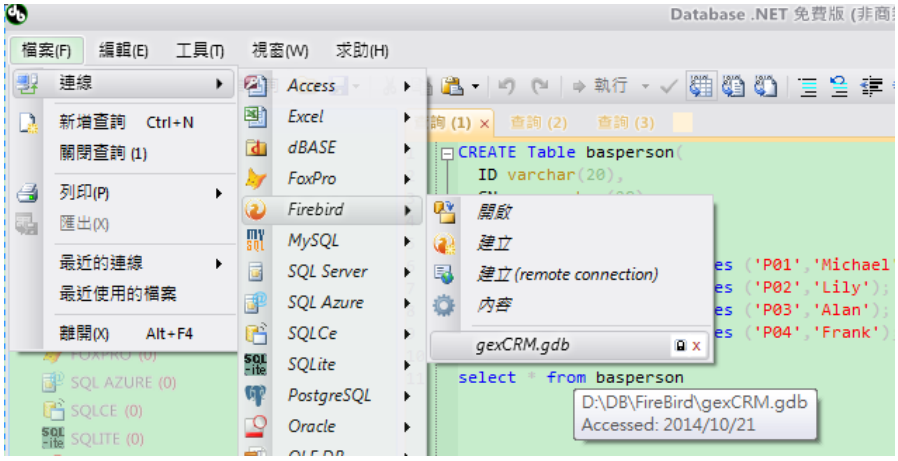


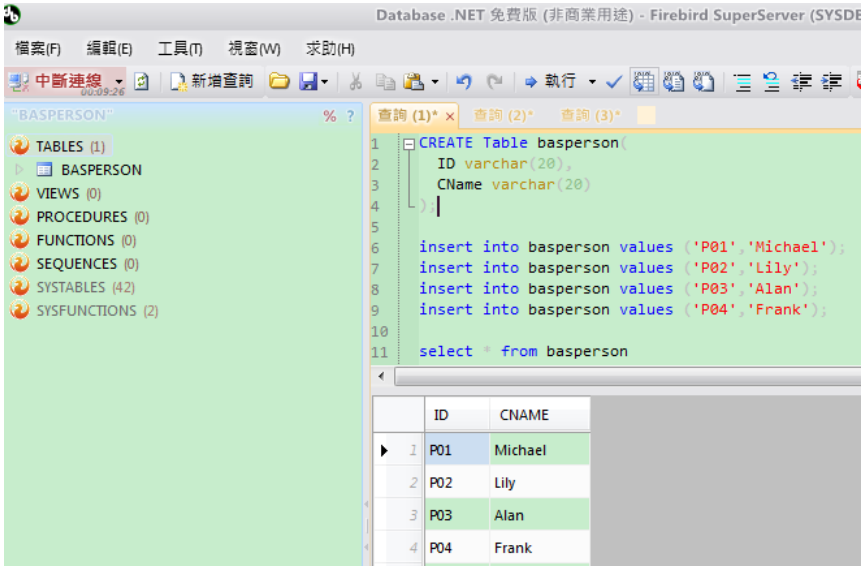


測試一下幾個簡單的 SQL 指令

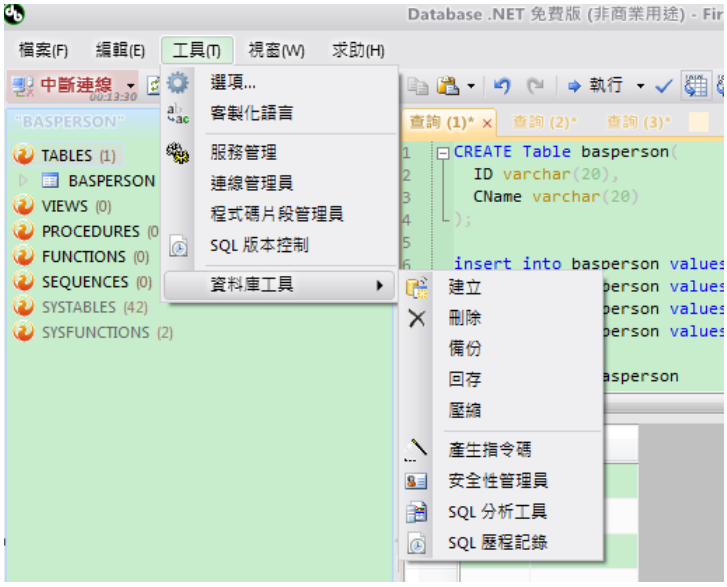


連接 MariaDB 的步驟和程序大致都是一樣的，都是先建立一個資料庫的連線，連線後就看要執行哪些資料庫的操作。





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。

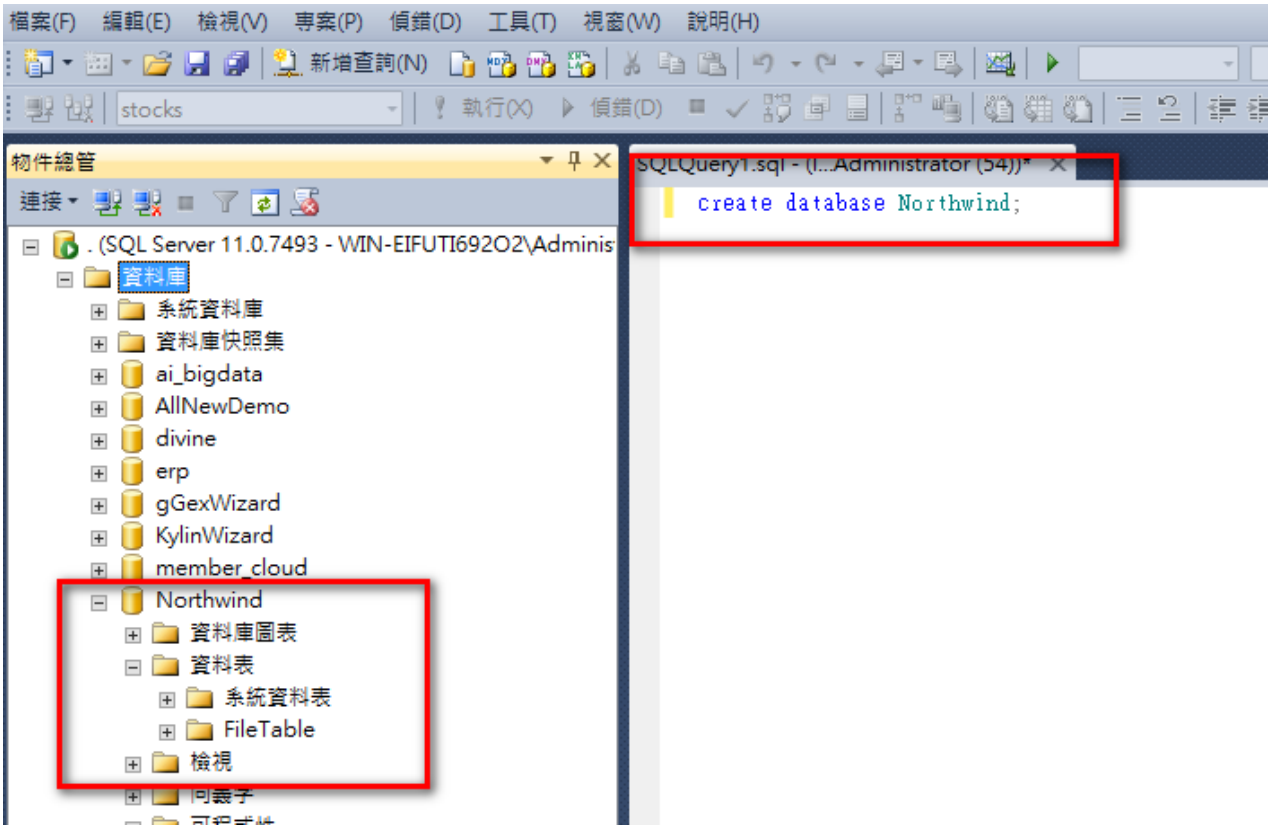


因為所有的資料庫存取資料內容都是透過 SQL 語法，所以基本上只要遵循標準的 SQL 語法（ANSI-92 以後的標準），基本上可以通吃。但是天下事總不盡如人意，有時後總會碰到一些特殊的需求，要用到一些特殊的語法，這時就只好認真的 K 一下各資料庫的使用手冊了。本書的目的是快速的瞭解不同資料庫的差異，但是不同的資料庫總會有不同的特色及優缺點，所以太特殊的語法就不在討論的範圍內。

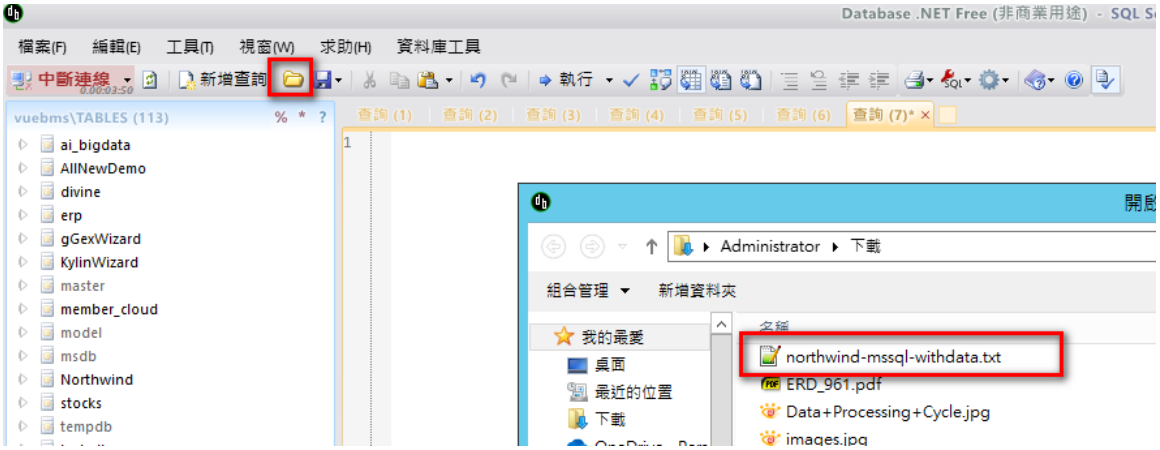
Database.NET 是一個優秀的工具，基本上只要對資料庫有一點基本的概念，應該就可以應用自如。所以接下來，我們就開始實作，目標是新建立一個 Northwind 資料庫，並利用已經準備好的 SQL Script 建立完整的資料庫內容。這個部分主要是第 10 章實例開發的前置作業，稍後再安裝 PostgreSQL 及 MariaDB 時，也會重複相同的步驟，一旦我們準備好這些資料，才能盡情的在不同資料庫間，利用相同的 Table 及 Data(資料)，透過精心準備的測試題庫，比對各資料庫的 SQL 語法差異，並能進一步測試效能等進階問題。

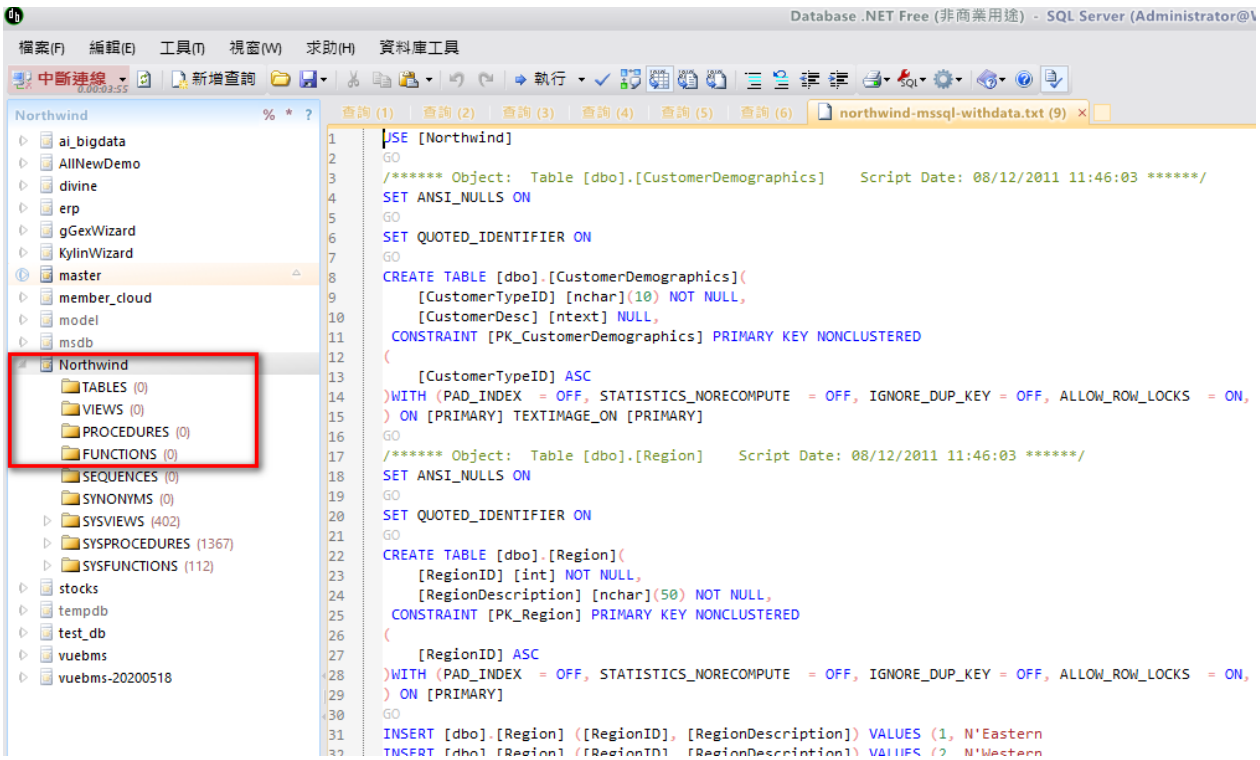
請先下載 ms-sql 版的 Northwind 的 Script，下載網址如下

- northwind-mssql-withdata.txt
  https://github.com/gexMichael/Play-Learn-SQL/blob/main/northwind-mssql-withdata.txt
- 建立 Northwind 資料庫(請先確認資料庫尚未建立)

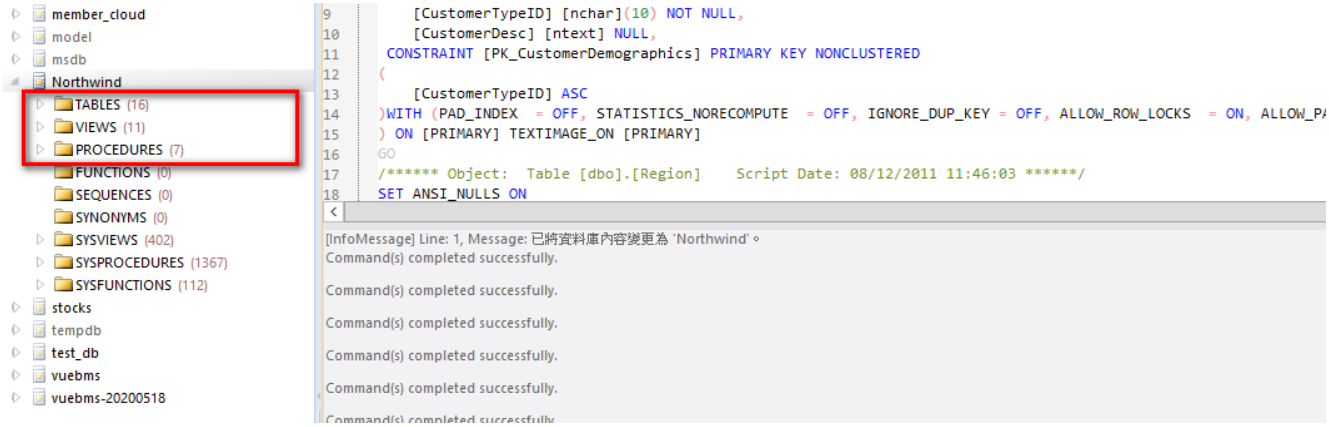


- 執行 DataBase.NET，並開啟下載的文字檔





- 執行後就會匯入完整的資料庫內容



- 查詢訂單資料

```
Select * from Orders;
```

```
1  select * from orders;
```

Ln 1 Ch 22

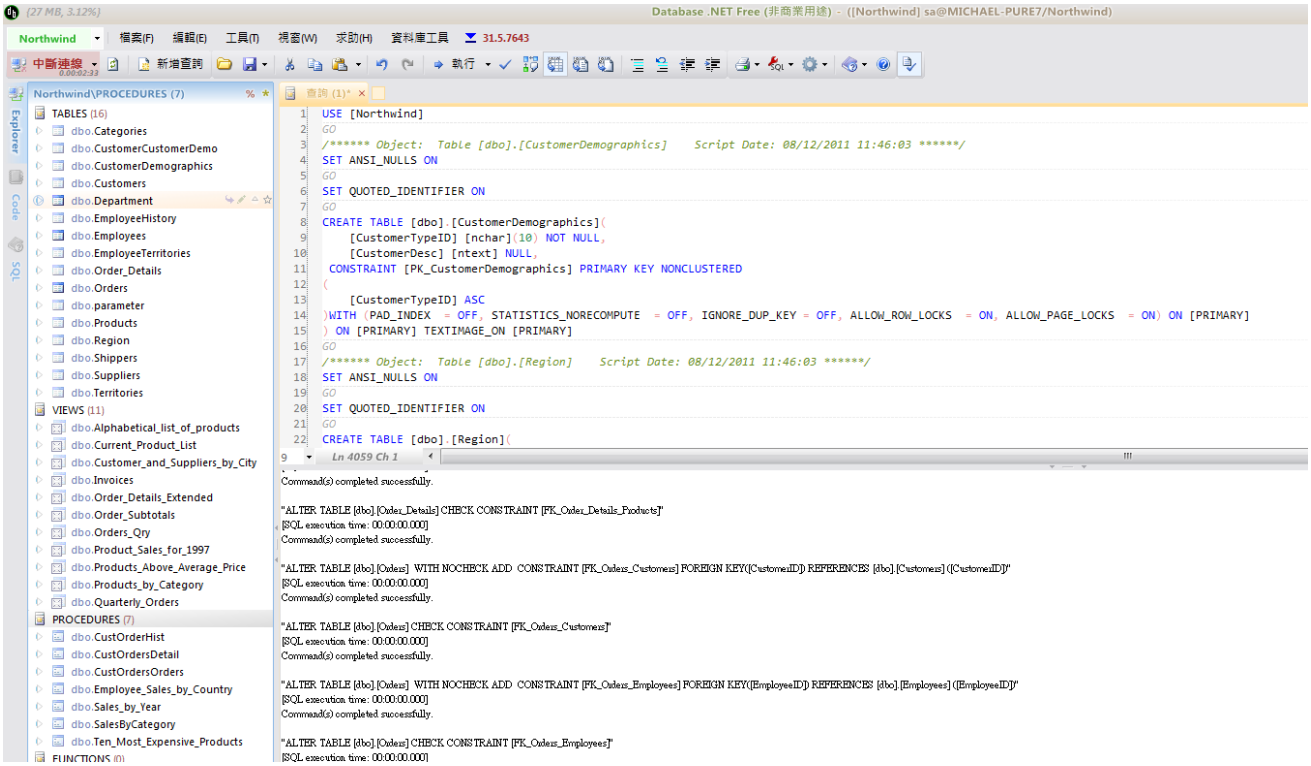
	OrderID int	CustomerID nchar(5)	EmployeeID int	OrderDate datetime	RequiredDate datetime	ShippedDate datetime	ShipVia int	Freight money	ShipName nvarchar(40)	ShipAddress
1	10248	VINET	5	1996-07-04	1996-08-01	[redacted]	3	32.3800	Vins et alcools Cheva...	59 rue de l'Abbaye
2	10249	TOMSP	6	1996-07-05	1996-08-16	1996-07-10	1	11.6100	Toms Spezialitäten	Luisenstr. 48
3	10250	HANAR	4	1996-07-08	1996-08-05	1996-07-12	2	65.8300	Hanari Carnes	Rua do Paço, 67
4	10251	VICTE	3	1996-07-08	1996-08-05	1996-07-15	1	41.3400	Victuailles en stock	2, rue du Commerc
5	10252	SUPRD	4	1996-07-09	1996-08-06	1996-07-11	2	51.3000	Suprêmes délices	Boulevard Tirou, 25
6	10253	HANAR	3	1996-07-10	1996-07-24	1996-07-16	2	58.1700	Hanari Carnes	Rua do Paço, 67
7	10254	CHOPS	5	1996-07-11	1996-08-08	1996-07-23	2	22.9800	Chop-suey Chinese	Hauptstr. 31
8	10255	RICSU	9	1996-07-12	1996-08-09	1996-07-15	3	148.3300	Richter Supermarkt	Starenweg 5
9	10256	WELLI	3	1996-07-15	1996-08-12	1996-07-17	2	13.9700	Wellington Importad...	Rua do Mercado, 1
10	10257	HILAA	4	1996-07-16	1996-08-13	1996-07-22	3	81.9100	HILARION-Abastos	Carrera 22 con Ave.
11	10258	ERNSH	1	1996-07-17	1996-08-14	1996-07-23	1	140.5100	Ernst Handel	Kirchgasse 6
12	10259	CENTC	4	1996-07-18	1996-08-15	1996-07-25	3	3.2500	Centro comercial Mo...	Sierras de Granada
13	10260	OTTIK	4	1996-07-19	1996-08-16	1996-07-29	1	55.0900	Ottilies Käseladen	Mehrheimerstr. 369
14	10261	QUEDE	4	1996-07-19	1996-08-16	1996-07-30	2	3.0500	Que Delícia	Rua da Panificador

- 關於 Northwind 的內容，請參考第 9 章的詳細說明，本章的主要目的是完成相關程式的安裝，並將資料庫內容都設定完成。到此步驟，已完成 MS-SQL 版本的相關作業，稍後會繼續說明另外 3 個資料庫的類似安裝說明。

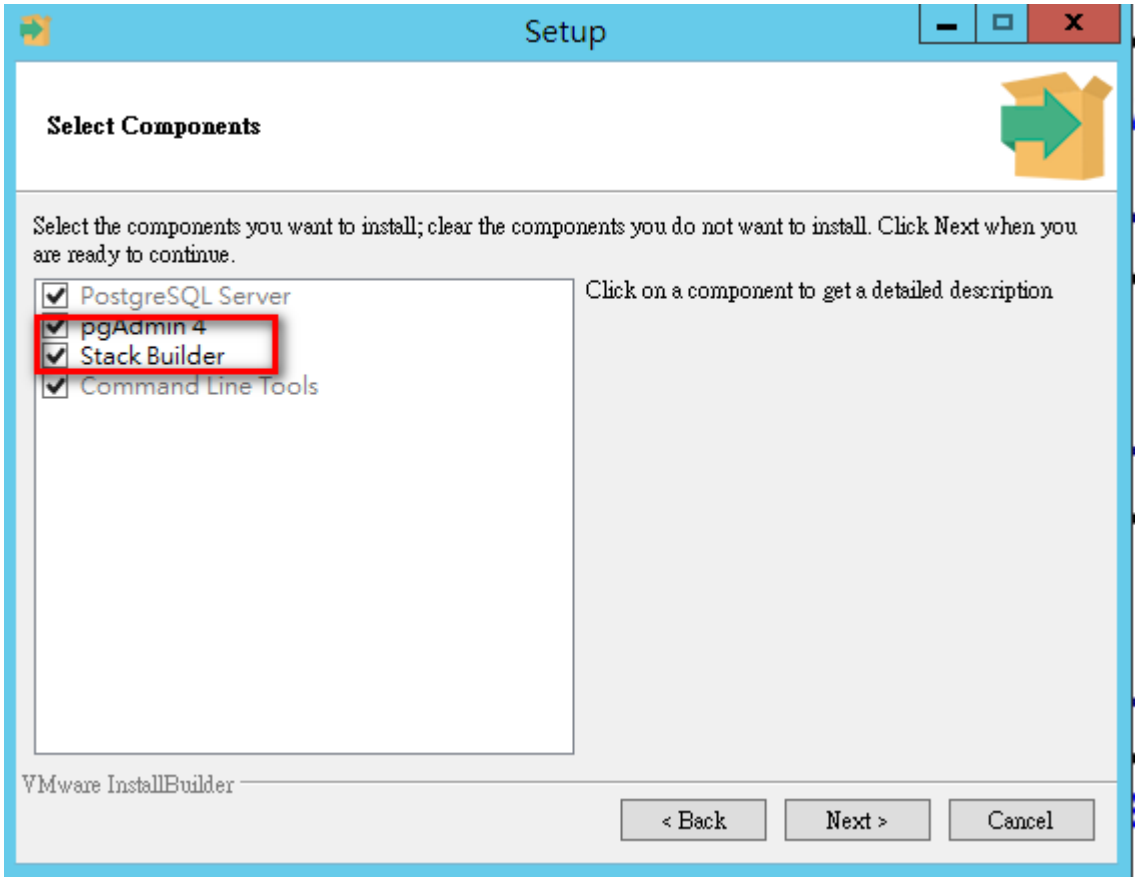
第四章

PostgreSQL 的安裝及基本操作

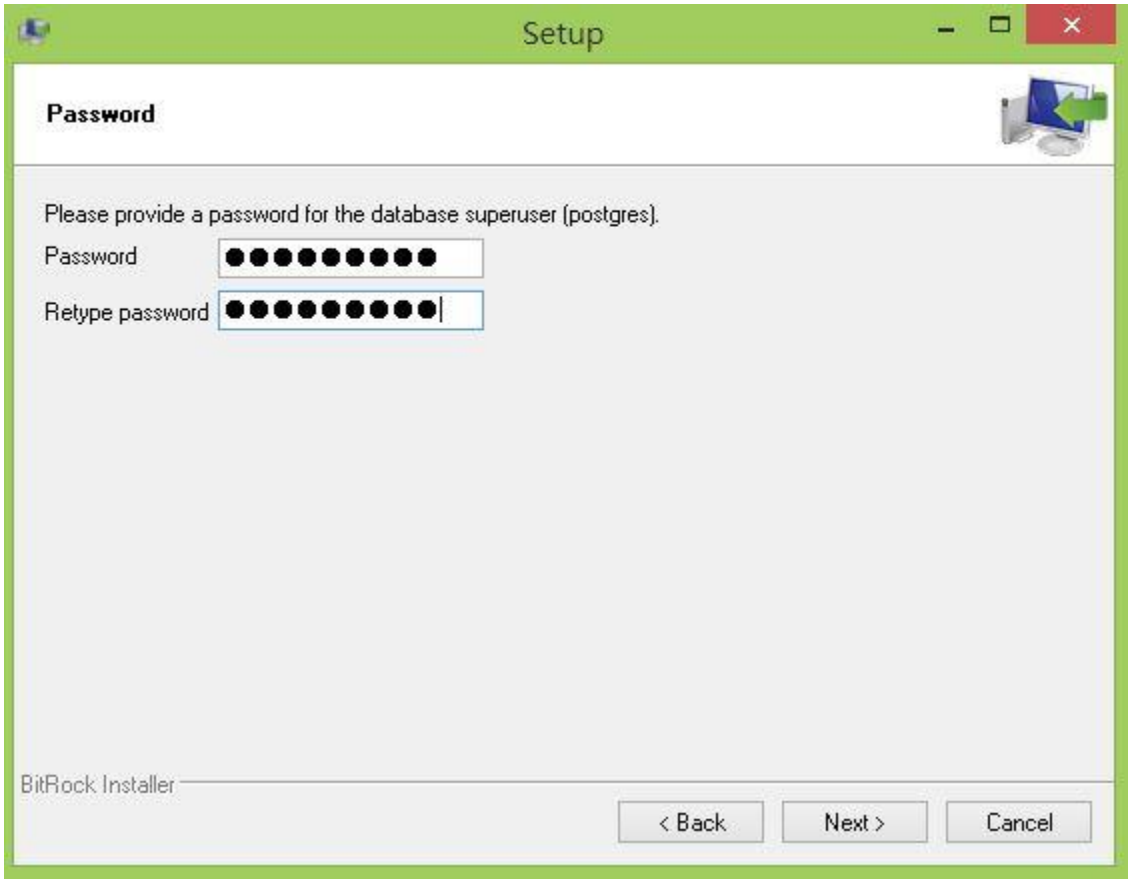
前一章我們實際的安裝了 MS-SQL 及 DataBase.NET 等程式及 Northwind 資料庫的 Script，本章會延續前章的流程，完整的安裝 PostgreSQL 資料庫。



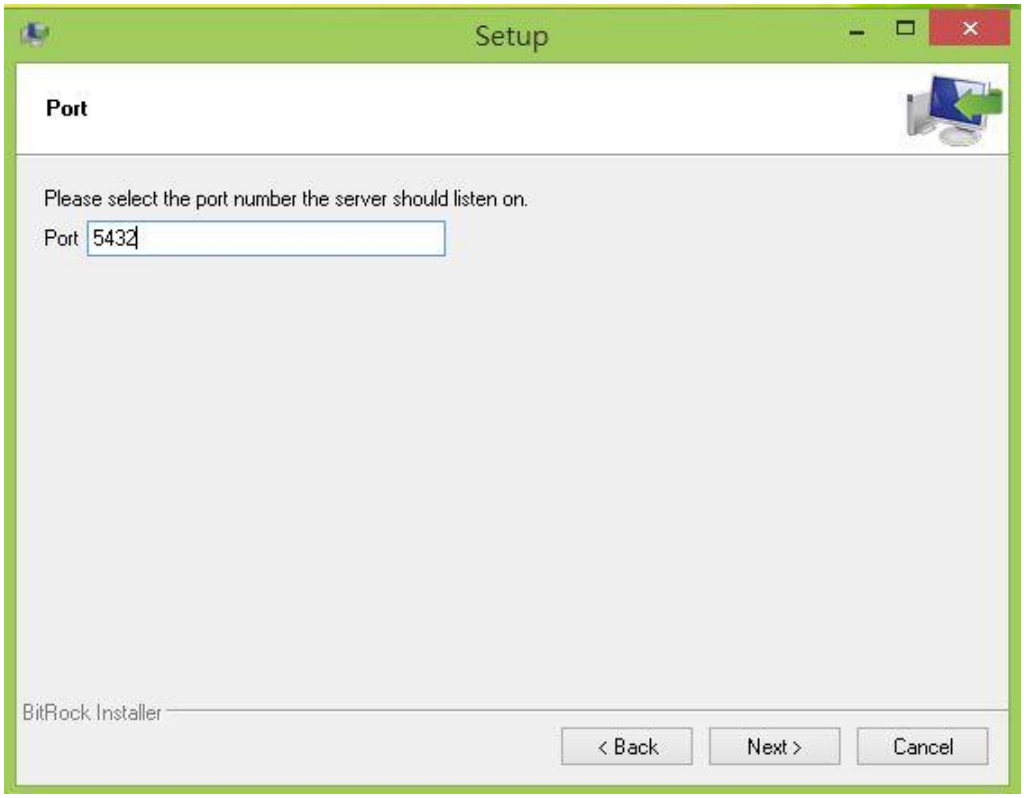
- 下載 PostgreSQL
  http://www.postgresql.org/download/windows/
- 開始安裝，請根據安裝環境選擇適合的版本
  - ✧ 執行安裝檔 postgresql-10.4-1-windows-x64
  - ✧ pgAdmin 4 是官方推薦的管理程式，我個人用不慣所以沒安裝。
  - ✧ Stack Builder 尚不清楚用途，可暫不安裝。
  - ✧ 安裝過程特別注意登入 Admin 帳號 postgres 的密碼設定及 Port(建議不要改動)



Admin 帳號 postgres 的密碼



使用的 Port

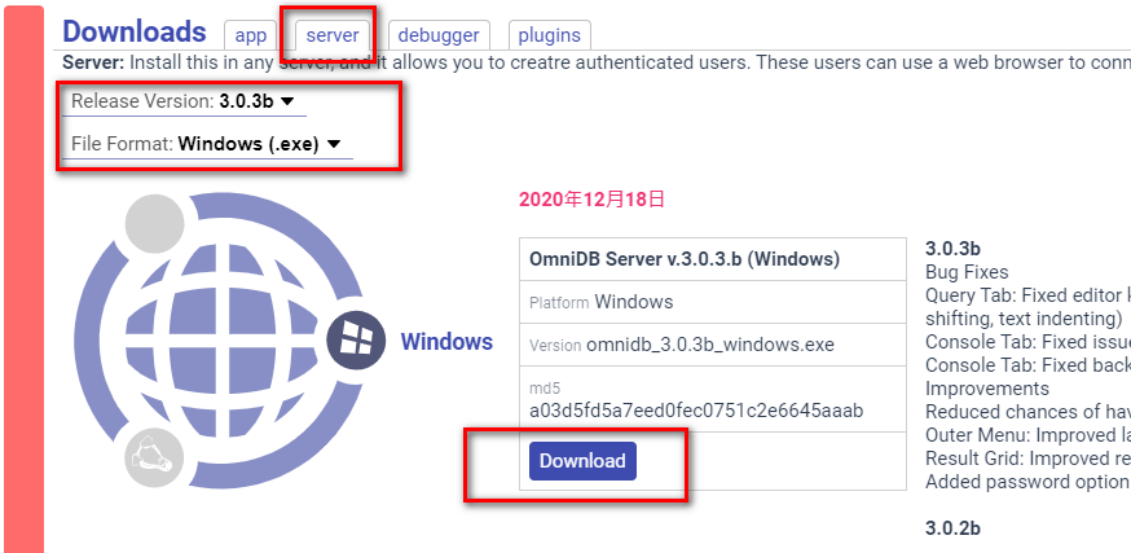


測試用 Database.NET 連線

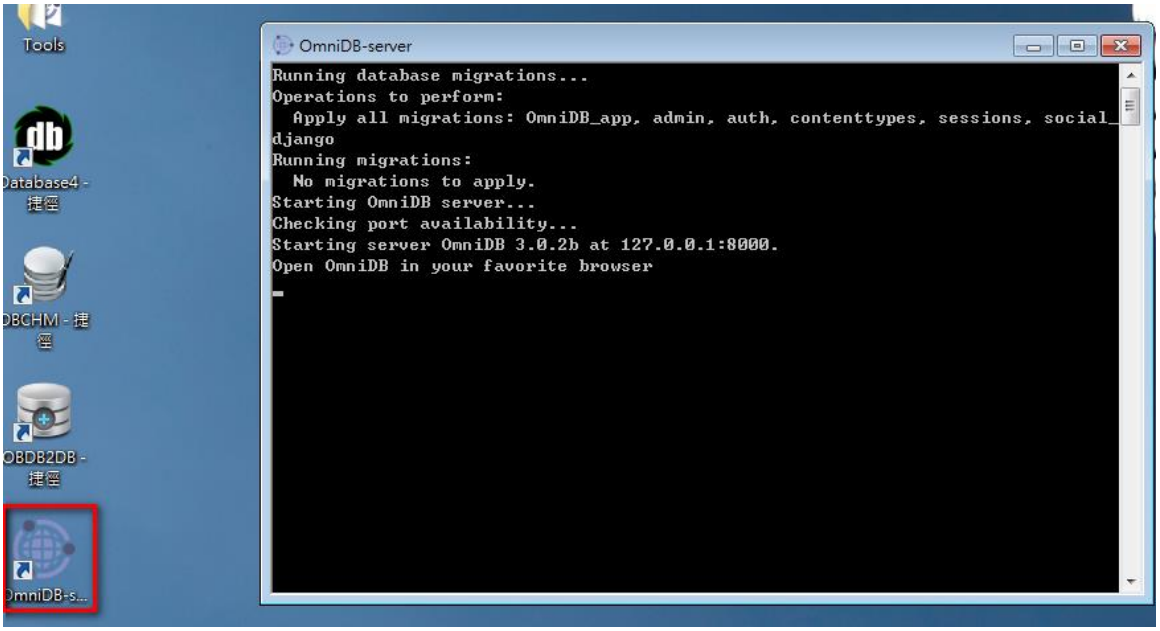
- 安裝完 PostgreSQL 後，SQL 管理工具有以下幾支程式可以選擇
  - 官方標配的 pgAdmin 4
  - 前章提到的 DataBase.NET
  - 個人推薦的專用工具 OmniDB
  - 功能齊全的 DBeaver(但執行效能稍差)

OmniDB 安裝及使用

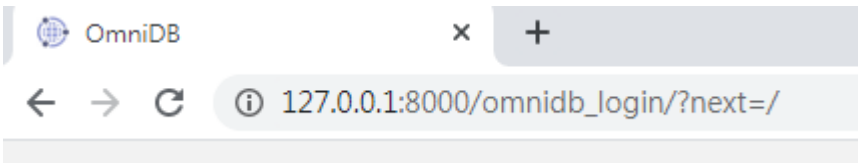
- https://omnidb.org/#downloads
- 建議安裝 server 版的 Windows 最新版本，如果選擇 app 版本，會同時安裝 Server 及 Client 端程式，但是我個人測試 Client 都一直無法執行，所以習慣只安裝 server 端程式。



- 安裝完 OmniDB 的 Server 端程式後，會在桌面建立一個 ShortCut，請先執行 Console

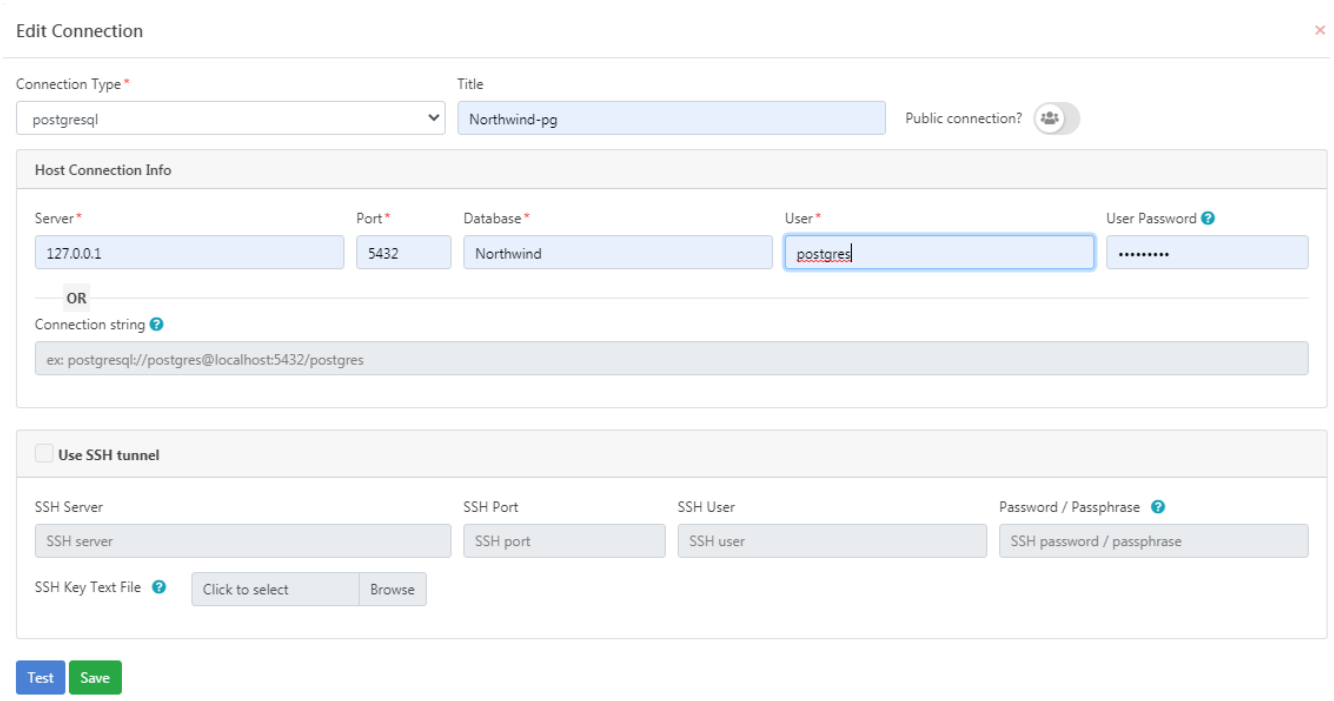


- 然後請開啟 Chrome 或 Firefox 瀏覽器，網址輸入 127.0.0.1:8000。預設的帳號及密碼都是 admin，登入後可以變更密碼。

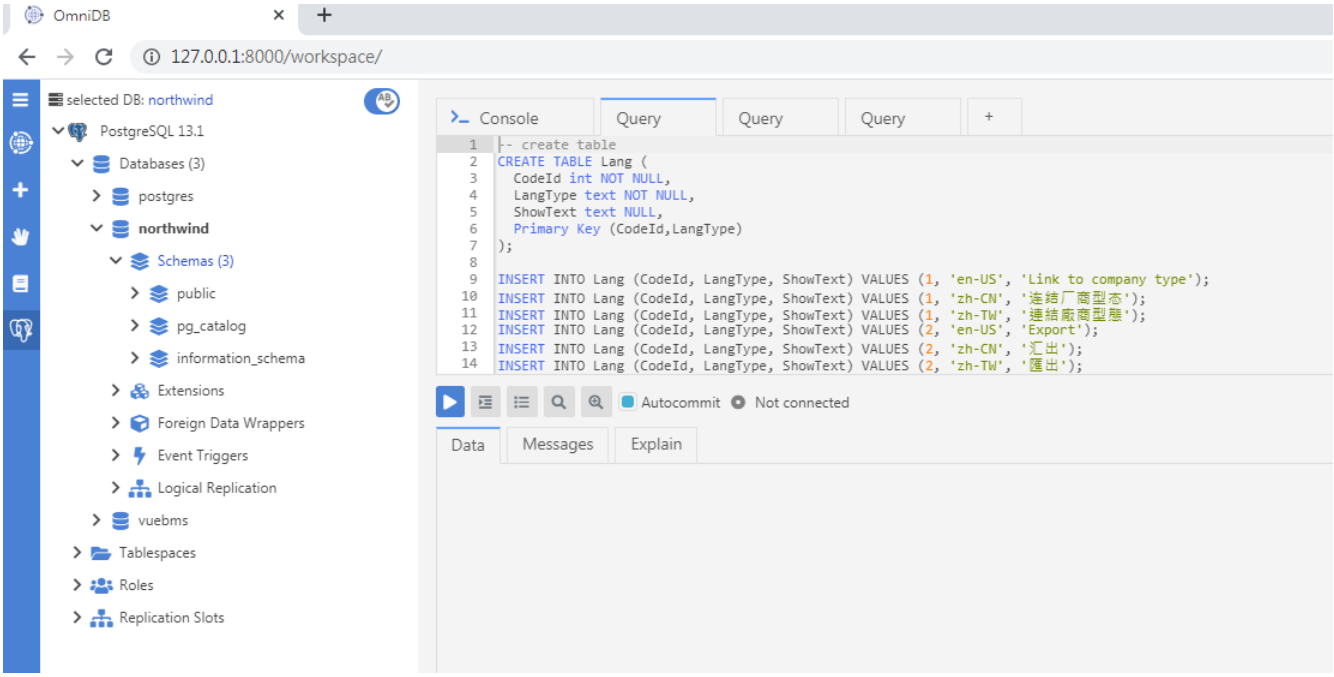


- 目前 OmniDB 可以支援 PostgreSQL、MySQL、MariaDB 及 Oracle 等 4 種資料庫。

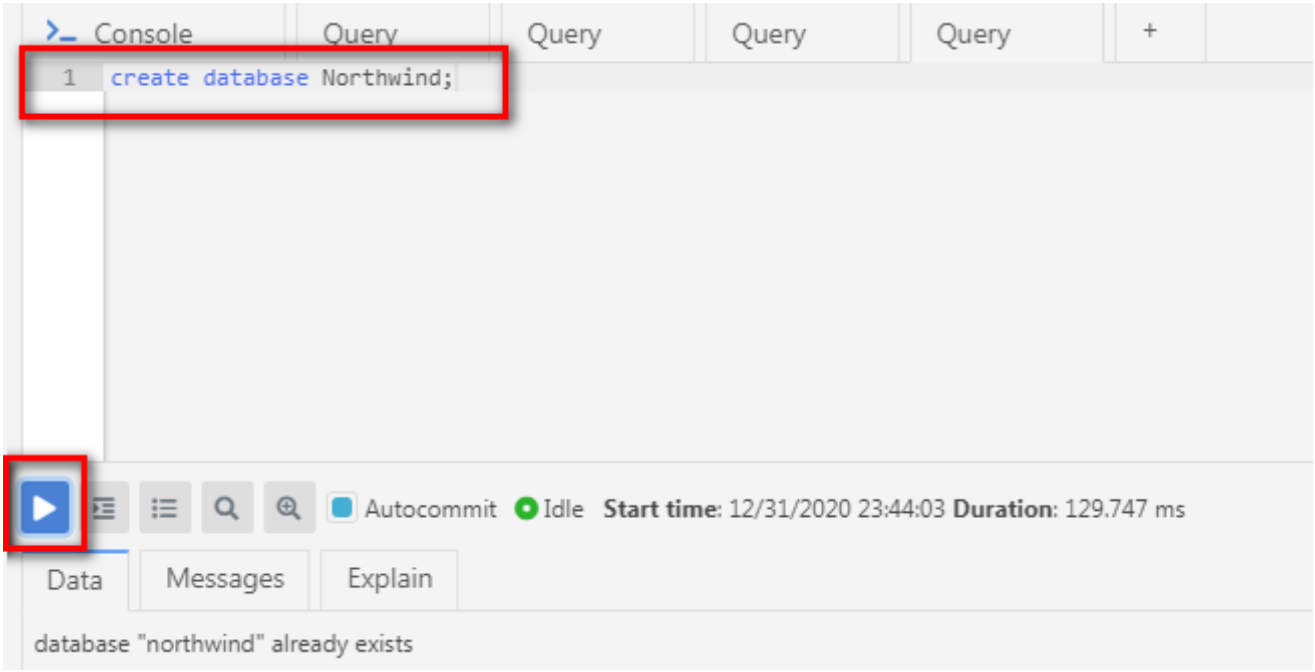
- 連線設定的參考畫面如下



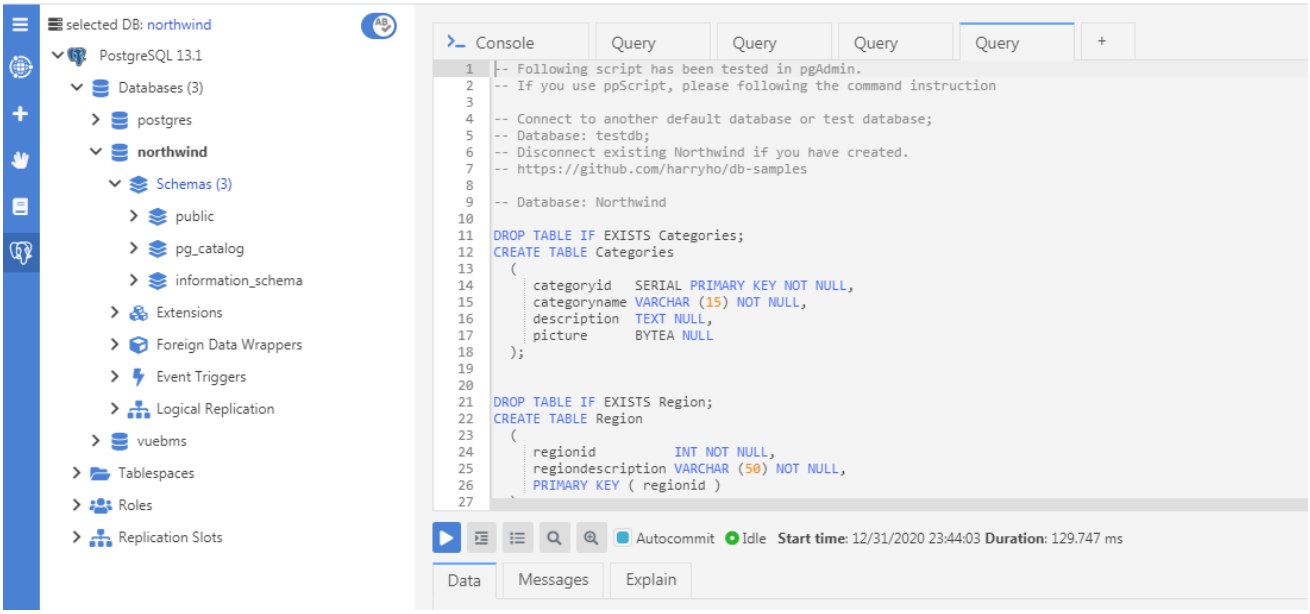
- 設定好連線號後的執行畫面如下



- 安裝 Northwind 資料庫的流程同 MS-SQL
  - 下載 Northwind-Script-postgre.sql
    https://github.com/gexMichael/Play-Learn-SQL/blob/main/Northwind-Script-postgre.sql
  - 下載 sp&view_postgre.txt
    https://github.com/gexMichael/Play-Learn-SQL/blob/main/sp%26view_postgre.txt
  - Create DataBase Northwind;



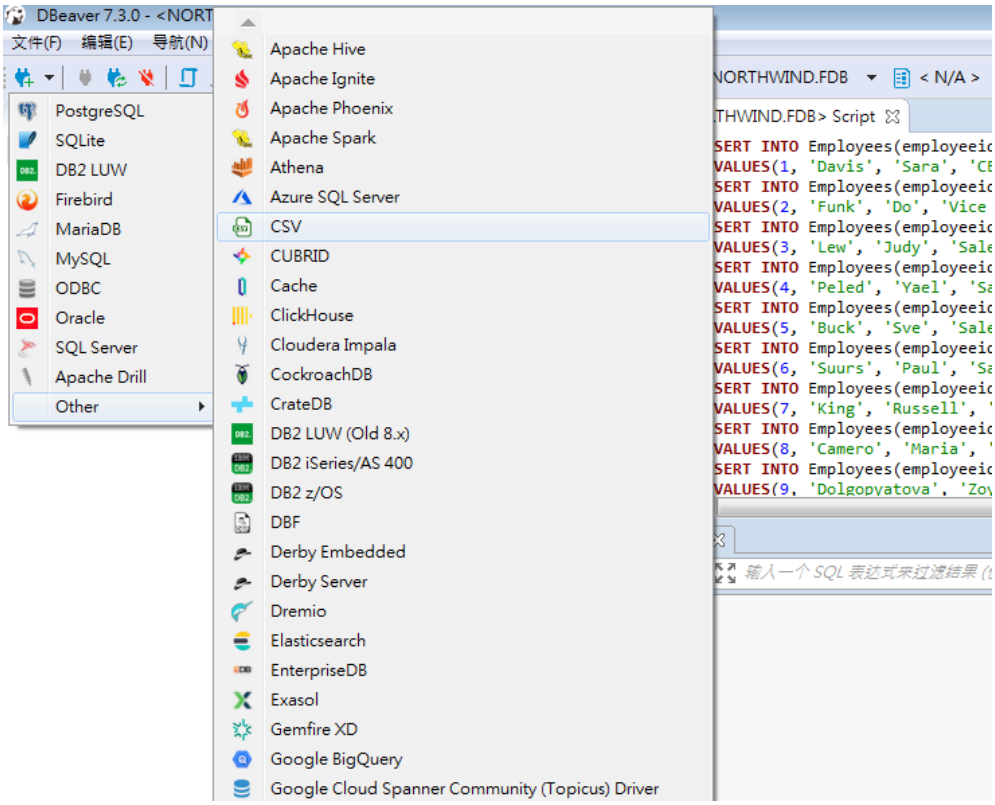
  - 將 SQL Script 內容貼入程式區並執行



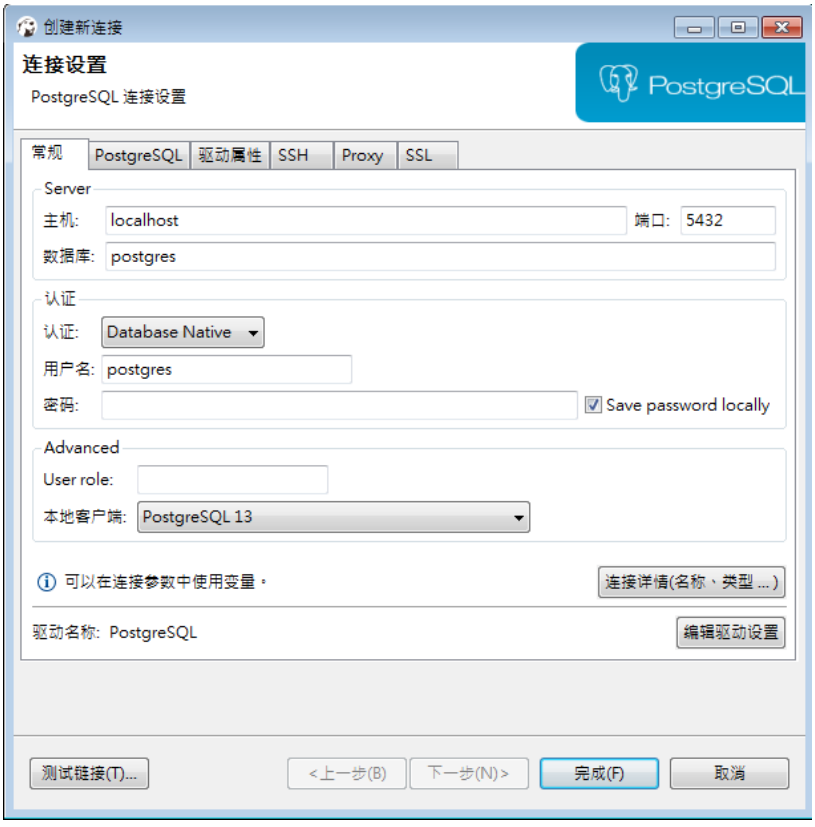
如此就完成了 Northwind 的安裝程序。

DBeaver 的安裝及使用(安裝社區版即可)

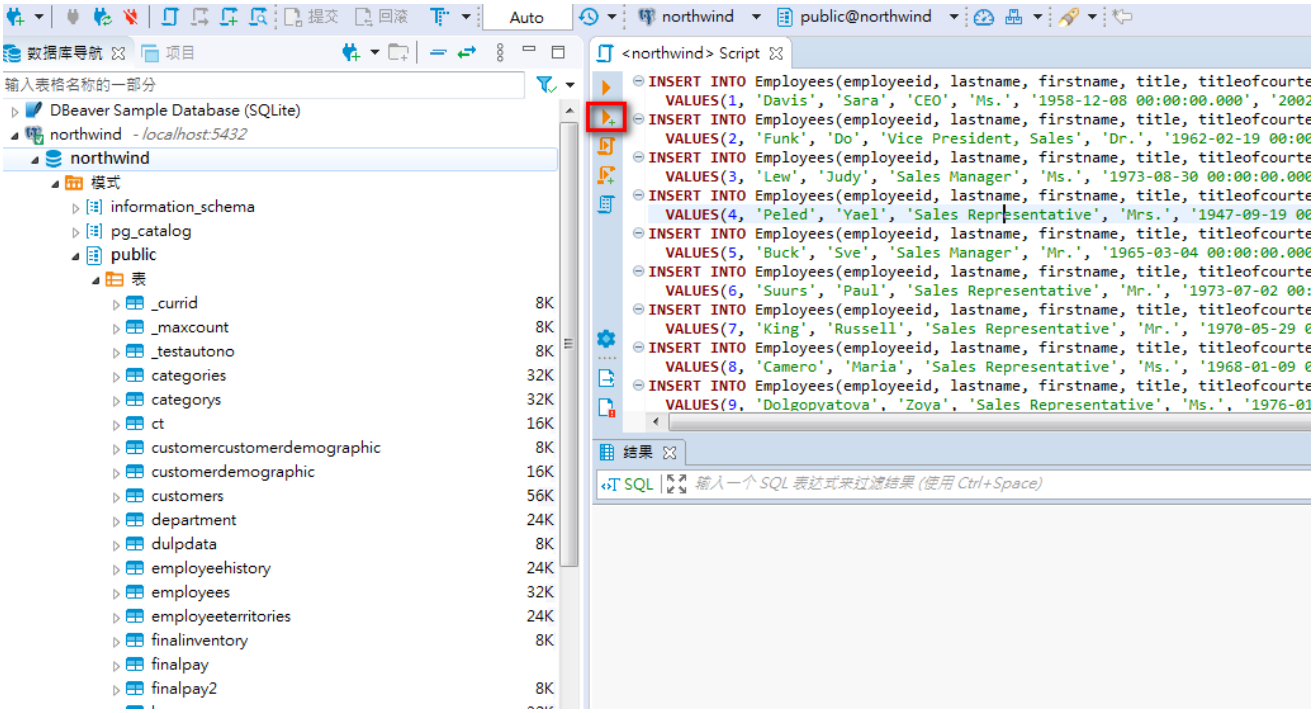
- https://dbeaver.io/download/
- 幾乎支援所有已知的資料庫



- 連接設定



- 連接成功後的執行畫面



- 如果要用 DBeaver 安裝 Northwind 資料庫，基本上程序和 OmniDB 類似。請自行測試。

介紹了這面多管理工具，除了官方的 pgAdmin 4 我個人用不慣外，事實上這幾套都是頗受好評的工具，蘿蔔青菜各有所愛，我偏好 OmniDB 主要是喜歡它的小巧及快速反應。您只要找到您用的習慣的工具就好，畢竟這些工具都只是和資料庫溝通的工具。關於 PostgreSQL 的安裝說明就先到這邊，請您找時間熟悉工具的使用，下一章會繼續討論 MariaDB 及簡單的聊聊 FireBird。

第五章

MariaDB 的安裝及基本操作

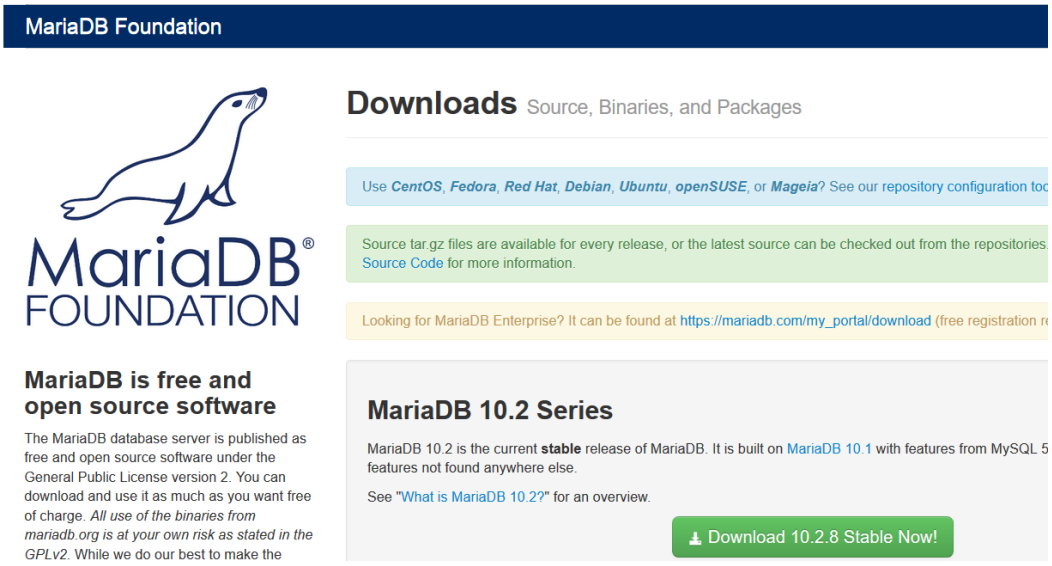
LAMP（Linux-Apache-MySQL-PHP）網站架構是目前國際流行的 Web 框架，該框架包括：Linux 操作系統，Apache 網絡服務器，MySQL 資料庫，PHP 或 Python 等程式語言，所有組成產品均是開源軟體，是非常成熟的架構框架，很多流行的商業應用都是採取這個架構，和 Java/J2EE 架構相比，LAMP 具有 Web 資源豐富、輕量、快速開發等特點，微軟的.NET 架構相比，LAMP 具有通用、跨平台、高性能、低價格的優勢，因此 LAMP 無論是性能、質量還是價格都是企業搭建網站的首選平台。

由於 LAMP 架構的普及，使得 MySQL 在開發者世界廣受歡迎，根據 DB-Engine Ranking 的調查統計，在 2020 年底，MySQL 穩居關聯式資料庫的第二名寶座，而他的分支版本 MariaDB 則是排在第八名。早期的 MySQL 強調效能，因此沒有提供預存程式(Store Procedure) 等進階的資料庫功能，但從 MySQL 5.0 版開始，加入了對 Store Procedure 及 Trugger 等進階功能的支援，更符合大型資料庫的開發需求，所以更是勢不可擋。

由於擔心 Oracle 可能對 MySQL 閉源(相對於 Open Source)，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，所以本章內容我們會以 MariaDB 為討論主題，關於 MySQL 的部分，請您自行參閱相關文檔。

接下來，我們簡單的介紹 MariaDB 的安裝過程

- 下載 MariaDB
  https://downloads.mariadb.org/

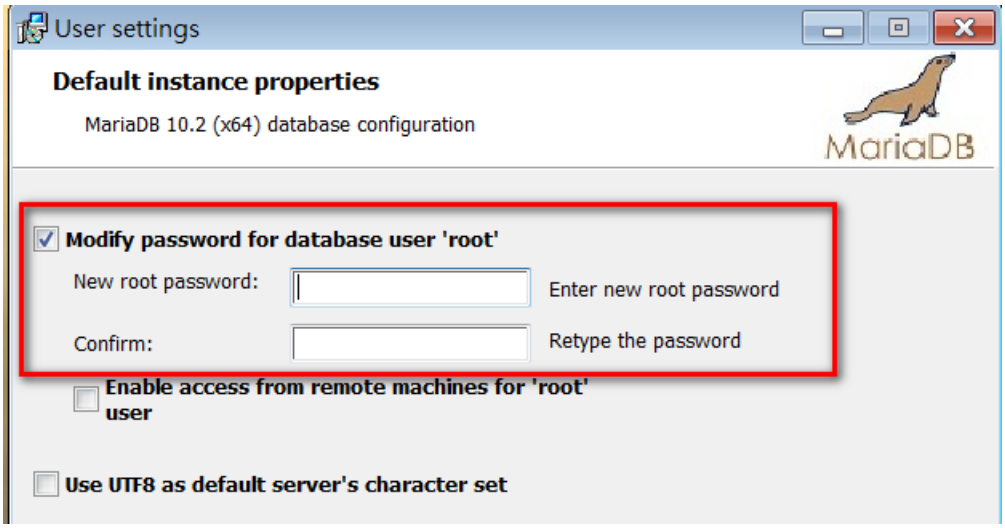


- 開始安裝

  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)

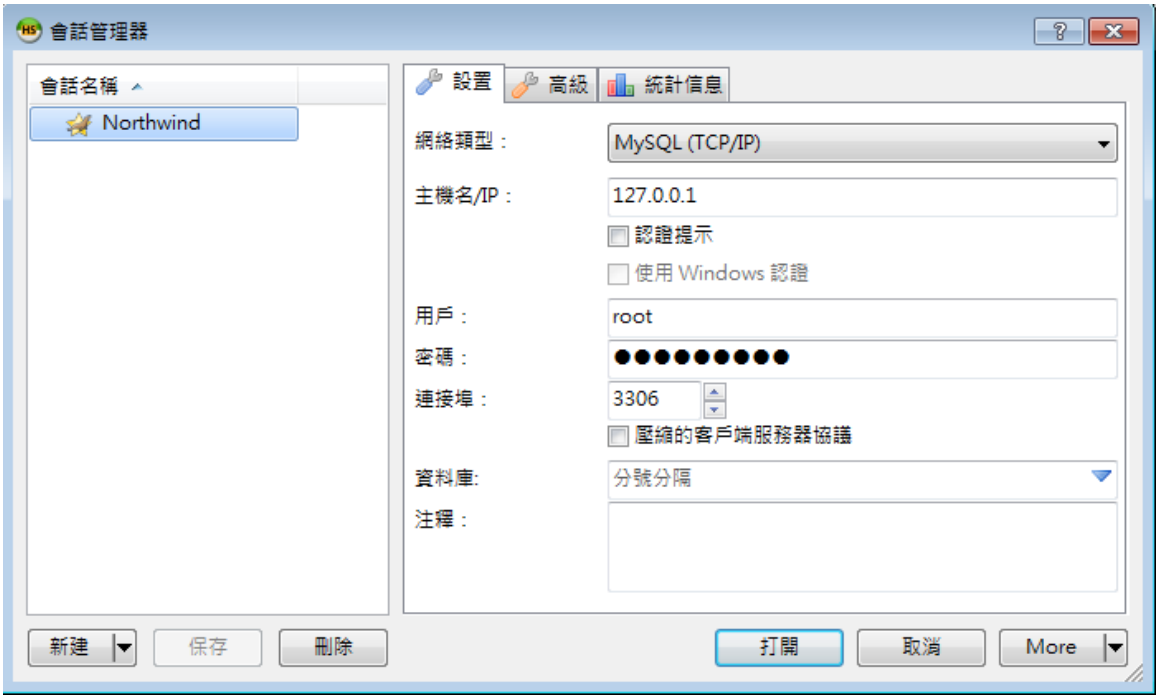
  - 當然，root 的密碼一定是要設的拉，設定好後要記好，程式設定連線時會用到。



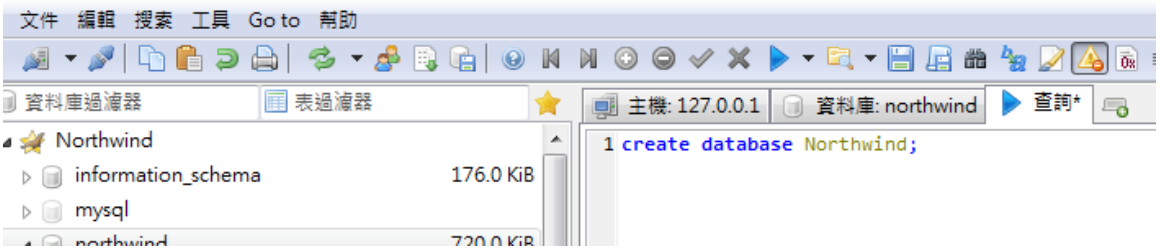
  - 相對於 MySQL 的繁複設定及檢查，基本上 MariaDB 的安裝包輕鬆多了，通常只要用一般的預設值安裝就可以。
  - 安裝完後就可以使用 HeidiSQL (官方標準的管理程式) 連線測試。除了 HeidiSQL 外，我們之前介紹的 OmniDB 或 DBeaver、Database.NET 都有支援 MariaDB 的連線。請自行測試。接下來筆者會用 HeidiSQL 來執行 Northwind 資料庫的設定安裝。

- 安裝 Northwind 資料庫的流程同 MS-SQL
  - 下載 Northwind-Script-mysql.sql
    https://github.com/gexMichael/Play-Learn-SQL/blob/main/Northwind-Script-mysql.sql
  - 下載 sp&view_mysql.txt
    https://github.com/gexMichael/Play-Learn-SQL/blob/main/sp%26view_mysql.txt

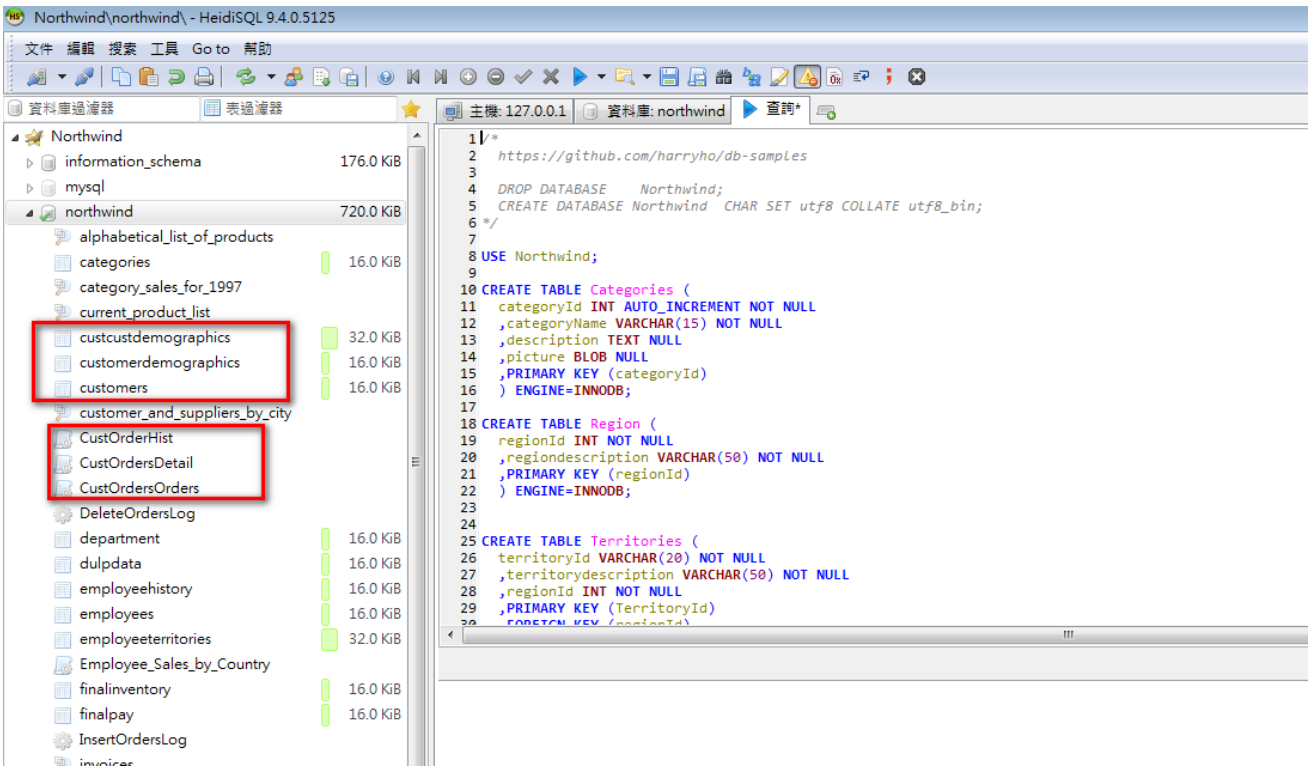
- 連線資料庫



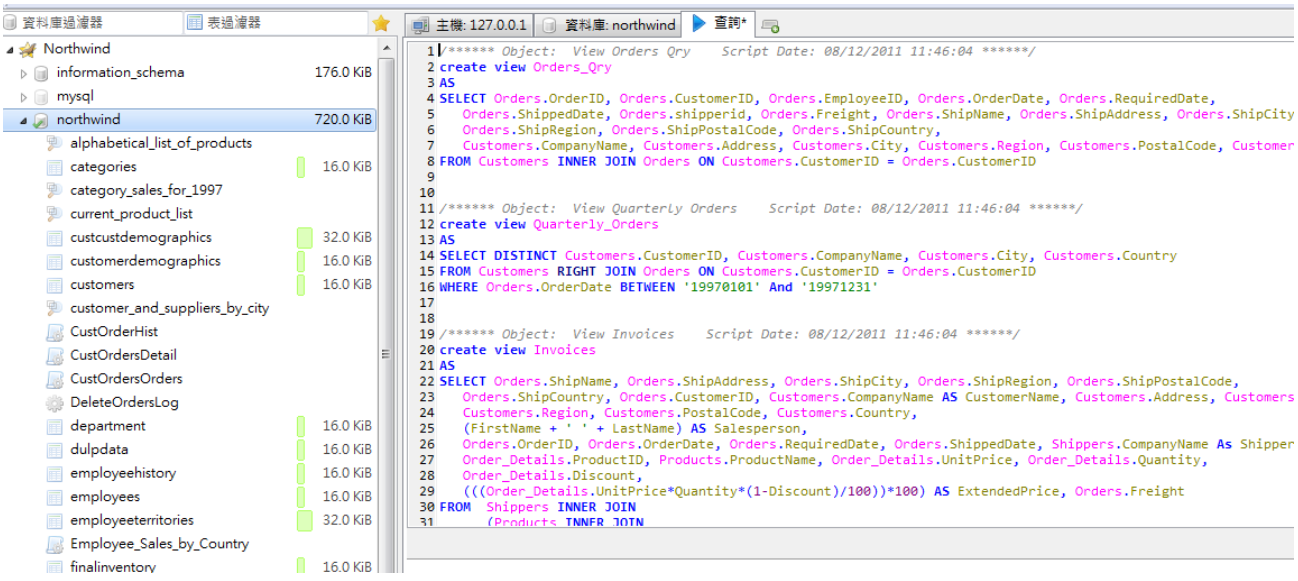
- Create DataBase Northwind;



- 將 SQL Script 內容貼入程式區並執行



- 再將 store procedure 及 udf、Trigger 也加入後，Northwind 的安裝就完成。其他之前曾經介紹的管理程式如 OmniDB、DBeaver、DataBase.NET 等都有支援 MariaDB 的連線，操作事實上也大同小異，還是老話一句，需要您自行安裝、測試然後找出您的最愛，用習慣就好了。以我個人而言，對每個資料庫都會有主要的管理工具，然後會有一個備用的程式。

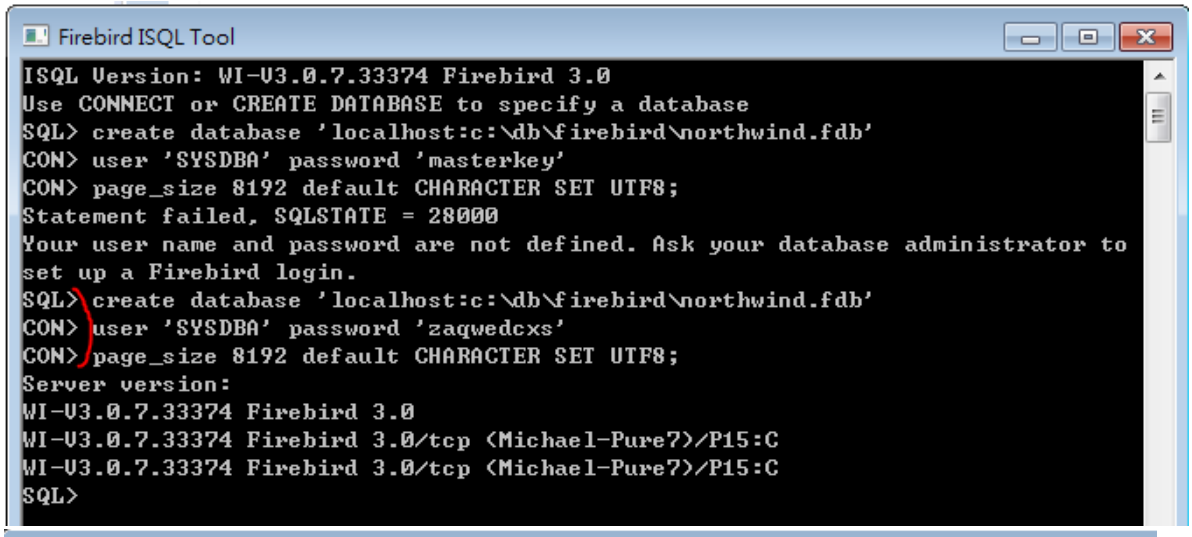


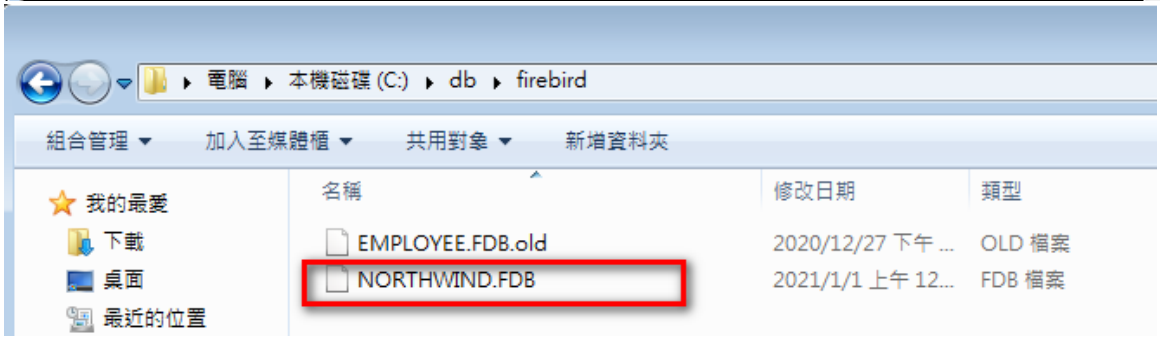
- 本章最後，我簡單的介紹 FireBird 一下這個資料庫，FireBird 的前身是 Borland 的 InterBase，原先對這個資料庫持重點關注，因為她在很早以前就是跨多平台的關聯式資料庫，但是經過測試其最新版本後，和另外 2 套開源資料庫相比已遠遠不如，稍後會有說明。
  - 下載並安裝 Firebird V3.0.7

https://firebirdsql.org/en/firebird-3-0/

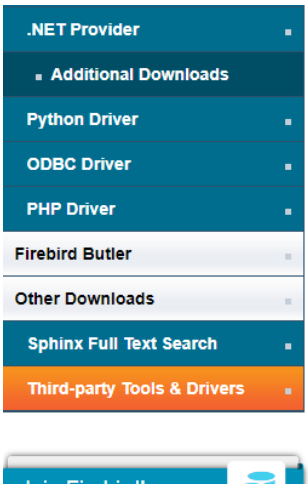
- 新增資料庫，並不是用 Create DataBase 指令來執行
  關於 Create Database 的方法，要在 Console Mode 的 isql 下指令

請參閱官方文件的說明
https://firebirdsql.org/file/documentation/reference_manuals/fbdevgd-en/html/fbdevg30-db-create.html

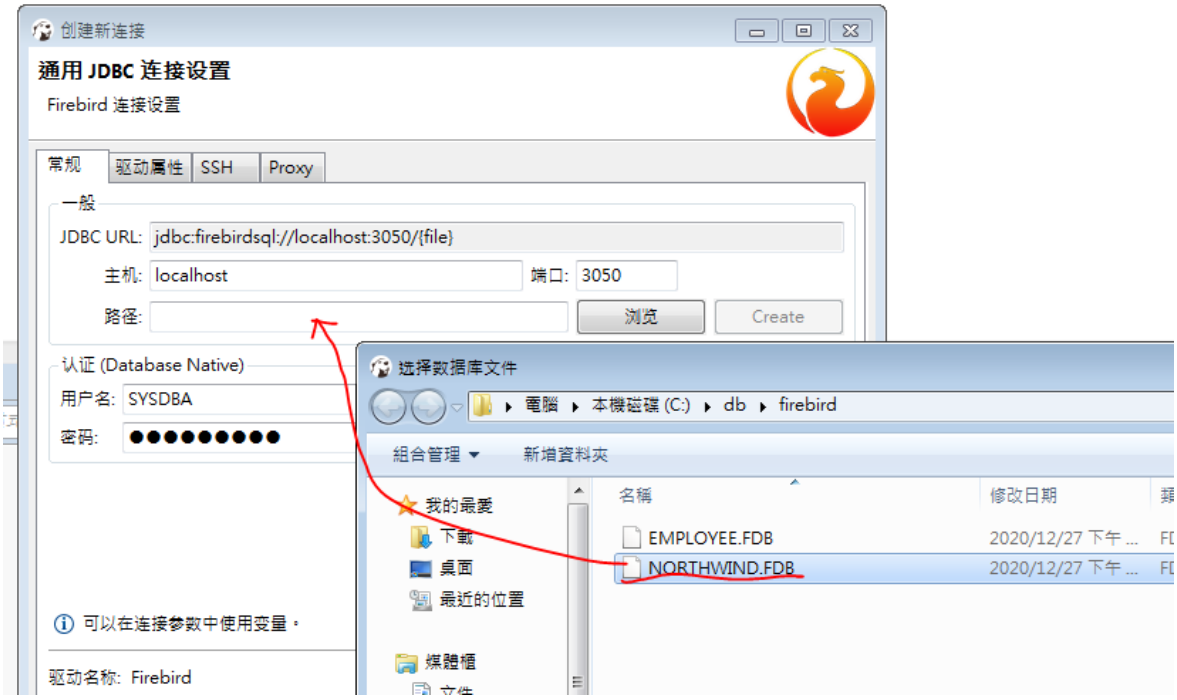


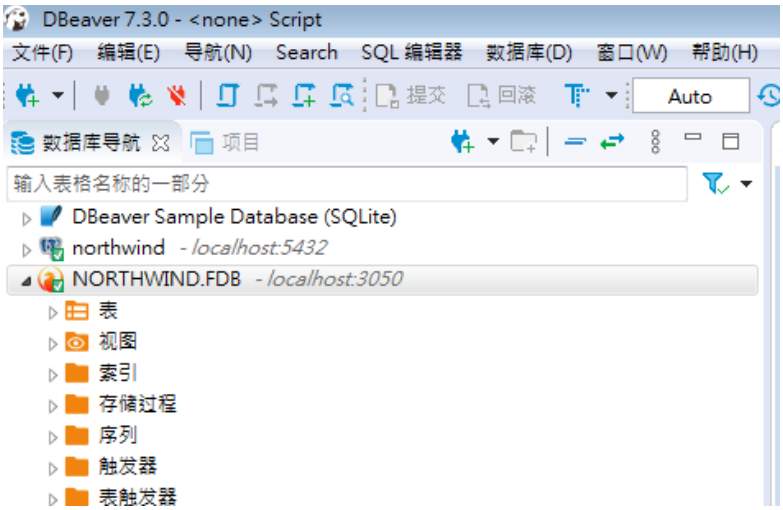


- 官網有列出支持的管理工具列表，包括商用和免費版，看到一個熟悉的名字，就先用 DBeaver 來試試。



IBExpert	Commercial
RazorSQL	Commercial
SQL Maestro	Commercial
SQLLY Studio	Commercial
SQL Management Studio for InterBase/Firebird	Commercial
Database.NET	Commercial
RedExpert	**Free**
Dbeaver	Free
DB Explorer For Firebird Databases	Free





- 實際執行 FireBird 安裝 Northwind 資料庫的整個過程，事實上是個很彆扭的過程，非常的不順暢，因為太多的非標準作法及限制，連要 Insert Into 多筆資料到 Northwind 竟然都有限制，和 MariaDB 和 PostgreSQL 根本不能算是同等級的產品，除非後續有推出大幅度的改版，短期內不必特別關注。

第六章

資料庫的欄位型態比較

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

＊＊ＳＱＬ小常識＊＊
不管在撰寫程式或是設定資料表，都少不了資料型別，它描述了數值、字元等表示法，可以讓這些變數或是欄位可以更容易管理自己的資料，不容易讓使用者出差錯。所以應當好好妥善運用才對！！

型別	描述
字串資料型別	
character(n) char(n)	固定長度的字申，最大的字元數為 n

character varying(n) varchar(n)	可變動儲存字串長度的字串，最大字元數為 n
text	可變動儲存字串長度的字串，最大字元數沒有限制
數值資料型別	
integer	整數型態，長度為 4Bytes
int8	長整數型態，長度為 8Bytes
numeric(p,d)	使用者自行訂義的數值型態
float4	浮點數精確度為 6 位
float8	浮點數精確度為 15 位
日期時間資料型別	
date	日期資料型別
time	時間資料型別
timestamp	日期時間資料型別
interval	時間差距資料型別

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
tinyint	允許從 0 到 255 的所有數字。
smallint	允許從 -32,768 到 32,767 的所有數字。
int	允許從 -2,147,483,648 到 2,147,483,647 的所有數字。
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。
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。
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。
smallmoney	介於 -214,748.3648 和 214,748.3647 之間的貨幣資料。
money	介於 -922,337,203,685,477.5808 和 922,337,203,685,477.5807 之間的貨幣資料。
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。
real	從 -3.40E + 38 到 3.40E + 38 的浮動精度數位資料。

Date 類型：

資料類型	描述

datetime	從 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度為 3.33 毫秒。
datetime2	從 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度為 100 納秒。
smalldatetime	從 1900 年 1 月 1 日 到 2079 年 6 月 6 日，精度為 1 分鐘。
date	僅存儲日期。從 0001 年 1 月 1 日 到 9999 年 12 月 31 日。
time	僅存儲時間。精度為 100 納秒。
datetimeoffset	與 datetime2 相同，外加時區偏移。
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(MariaDB) 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
tinyint	允許從 -128~127 的所有數字。
smallint	允許從 -32,768 到 32,767 的所有數字。
int	允許從 -2,147,483,648 到 2,147,483,647 的所有數字。
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。
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。
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。
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。
real	從 -3.40E + 38 到 3.40E + 38 的浮動精度數位資料。

Date 類型：

資料類型	描述
datetime	1000-01-01 00:00:00 ~ 9999-12-31 23:59:59。

date	僅存儲日期。從 1000 年 1 月 1 日 到 9999 年 12 月 31 日。
time	僅存儲時間。精度為 100 納秒。
timestamp	1970-01-01 00:00:00 ~ 2037-12-31 23:59:59

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
smallint	允許從 -32,768 到 32,767 的所有數字。
int	允許從 -2,147,483,648 到 2,147,483,647 的所有數字。

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。
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。
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。
double precision	15 decimal digits precision。
real	從 -3.40E + 38 到 3.40E + 38 的浮動精度數位資料。

Date 類型：

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

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	說明
布林	**MS SQL / MariaDB**
	tinyint
	PostgreSQL
	boolean
字串	**MS SQL**
	char varchar text
	MariaDB
	CHAR VARCHAR TEXT ENUM SET
	PostgreSQL
	character varying(n) character(n) text
Unicode 字串	**MS SQL**
	nchar nvarchar ntext
	MariaDB / PostgreSQL
	設定資料庫 utf-8
精確數字	**MS SQL**
	bigint numeric smallint decimal

	int money
	MariaDB
	TINYINT SMALLINT MEDIUMINT INT BIGINT
	PostgreSQL
	smallint integer bigint decimal numeric serial
近似數值	**MS SQL / MariaDB / PostgreSQL**
	Float(double) real
日期時間	**MS SQL**
	date datetime2 smalldatetime datetime time
	MariaDB
	YEAR TIME DATE DATETIME TIMESTAMP
	PostgreSQL
	timestamp date time interval
BINARY	**MS SQL**
	binary varbinary

	image
	MariaDB
	BINARY VARBINARY BLOB
	PostgreSQL
	PostgreSQL 並不支援 Blob, Clob 欄位。使用 bytea, oid, text 等欄位來模擬。

第七章

常用的 SQL 語法簡介

結構化查詢語言（Structured Query Language，SQL），是一種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

由於本書並非 SQL 語法的入門書籍，所以無法詳細說明相關的語法，如果您不熟悉 SQL 語法，推薦您可以到下列網站學習：

- 通用 SQL 教學
  - W3Cschool 教學
    https://www.w3schools.com/sql/default.asp
  - sql 常用语法
    https://www.jianshu.com/p/430078421263
- MariaDB(MySQL)
  - MySQL 入門教學
    https://www.itread01.com/study/mysql-tutorial.html
  - 簡明 SQL 資料庫語法入門教學
    https://blog.techbridge.cc/2020/02/09/sql-basic-tutorial/
- PostgreSQL
  - Database Research & Development
    https://www.dbrnd.com/postgresql/
  - PostgreSQL 官方使用手冊
    https://docs.postgresql.tw/
  - PostgreSQL Tutorial
    https://www.postgresqltutorial.com/

接下來，我們就用幾個簡單的範例，實際的來體驗一下常用的 SQL 語法。在第 10 章中，我們會有更多、更完整的範例。

範例一

```
CREATE TABLE [dbo].[Categories] (
    [CategoryID] int NOT NULL IDENTITY(1,1) PRIMARY KEY,
    [CategoryName] nvarchar(15) NOT NULL,
    [Description] ntext,
    [Picture] image
);
```

範例二

```
CREATE TABLE [dbo].[Customers] (
     [CustomerID] nchar(5) NOT NULL PRIMARY KEY,
     [CompanyName] nvarchar(40) NOT NULL,
     [ContactName] nvarchar(30),
     [ContactTitle] nvarchar(30),
     [Address] nvarchar(60),
     [City] nvarchar(15),
     [Region] nvarchar(15),
     [PostalCode] nvarchar(10),
     [Country] nvarchar(15),
     [Phone] nvarchar(24),
     [Fax] nvarchar(24)
);
/* 以下是 Index 的建立語法 */
CREATE NONCLUSTERED INDEX [City]
     ON [dbo].[Customers] ([City])
     WITH (PAD_INDEX=OFF
     ,STATISTICS_NORECOMPUTE=OFF
     ,IGNORE_DUP_KEY=OFF
     ,ALLOW_ROW_LOCKS=ON
     ,ALLOW_PAGE_LOCKS=ON) ON [PRIMARY];
```

範例三

```
/* 含 Foreign key 的建立語法 */
CREATE TABLE [dbo].[Orders] (
     [OrderID] int NOT NULL IDENTITY(1,1) PRIMARY KEY,
     [CustomerID] nchar(5),
     [EmployeeID] int,
     [OrderDate] datetime,
     [RequiredDate] datetime,
     [ShippedDate] datetime,
     [ShipVia] int,
     [Freight] money DEFAULT ((0)),
     [ShipName] nvarchar(40),
     [ShipAddress] nvarchar(60),
     [ShipCity] nvarchar(15),
     [ShipRegion] nvarchar(15),
```

```
    [ShipPostalCode] nvarchar(10),
    [ShipCountry] nvarchar(15),
    FOREIGN KEY ([ShipVia])
        REFERENCES [dbo].[Shippers] ([ShipperID])
        ON UPDATE NO ACTION ON DELETE NO ACTION,
    FOREIGN KEY ([CustomerID])
        REFERENCES [dbo].[Customers] ([CustomerID])
        ON UPDATE NO ACTION ON DELETE NO ACTION,
    FOREIGN KEY ([EmployeeID])
        REFERENCES [dbo].[Employees] ([EmployeeID])
        ON UPDATE NO ACTION ON DELETE NO ACTION
);
```

範例四：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;
```

範例五：新增一個模擬 ms-sql 的 GETDATE() 函式

```
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;
```

- 熟悉了基礎的 SQL 語法後，如果想要善用 SQL 的強大，就要繼續學習 Store Procedure 及 Function 等的寫法，上面的範例四、五就是一個實際的案例。接下來我們提幾個注意事項。
  - 呼叫 Store Procedure 的方法

資料庫	呼叫方法
MS-SQL	EXEC CreateDemoData
MySQL	Call CreateDemoData()
PostgreSQL	Select CreateDemoData(100)
FireBird	EXECUTE PROCEDURE CreateDemoData;

  - Store Procedure(SQL) 語法差異

常用指令	各資料庫語法
字串相加	**MS-SQL**
	'A'+'B'+'C'
	MariaDB(MySQL)
	CONCAT('A', 'B', 'C')
	PostgreSQL
	'A'||'B'||'C'
今天日期	**MS-SQL**
	GETDATE()
	MySQL
	CURRENT_DATE, CURRENT_DATE()
	PostgreSQL
	NOW(), TIMEOFDAY()

千里之行、始於足下，基本上所有的資料庫基礎語法都差不多。剛開始學習時，只要能不斷的用心練習、熟悉這些語法，就可以解決絕大多數的需求。20/80 法則還蠻是用這個狀況，也就是說，開發企業資訊系統時，80% 都是一般難度的 issue，運用基礎的語法大部分都可以解決。但是 20% 是較進階的問題，例如 MRP 展算、月加權成本計算等，除了必須精通商業邏輯 Domain 外，也必須用到一些較進階的語法才能解決(如遞迴)。隨著經驗的持續積累，成為 SQL 大神只是時間的問題而已。

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。關於這段程式的邏輯我就不再詳細說明，純粹就是一段 Demo Code，如果您的 SQL 有一定的經驗，應該能推測出大概的商業邏輯。放在此處只是讓各位理解，所有的商業邏輯只要和資料庫相關，都一定可以寫成 Store Procedure 如此而已。

```
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,RETURNOUTQ
TY)
   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,RE
TURNOUTQTY)
   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
- 動態 SQL 語法執行
- 三種暫存表的寫法
- 配合 DB Func 讓 SQL 更靈活、更強大
- 迴圈式程式寫法的範例

➢ 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。

- 動態 SQL 語法執行

  MS-SQL 提供 exec sp_executesql @sql;

  可以執行動態的 SQL 語法，這增加了極大的彈性，但是有可能會有效能問題，請慎重使用。

  其他的資料庫也有提供類似的語法，在第 10 章的實例轉換會實做。

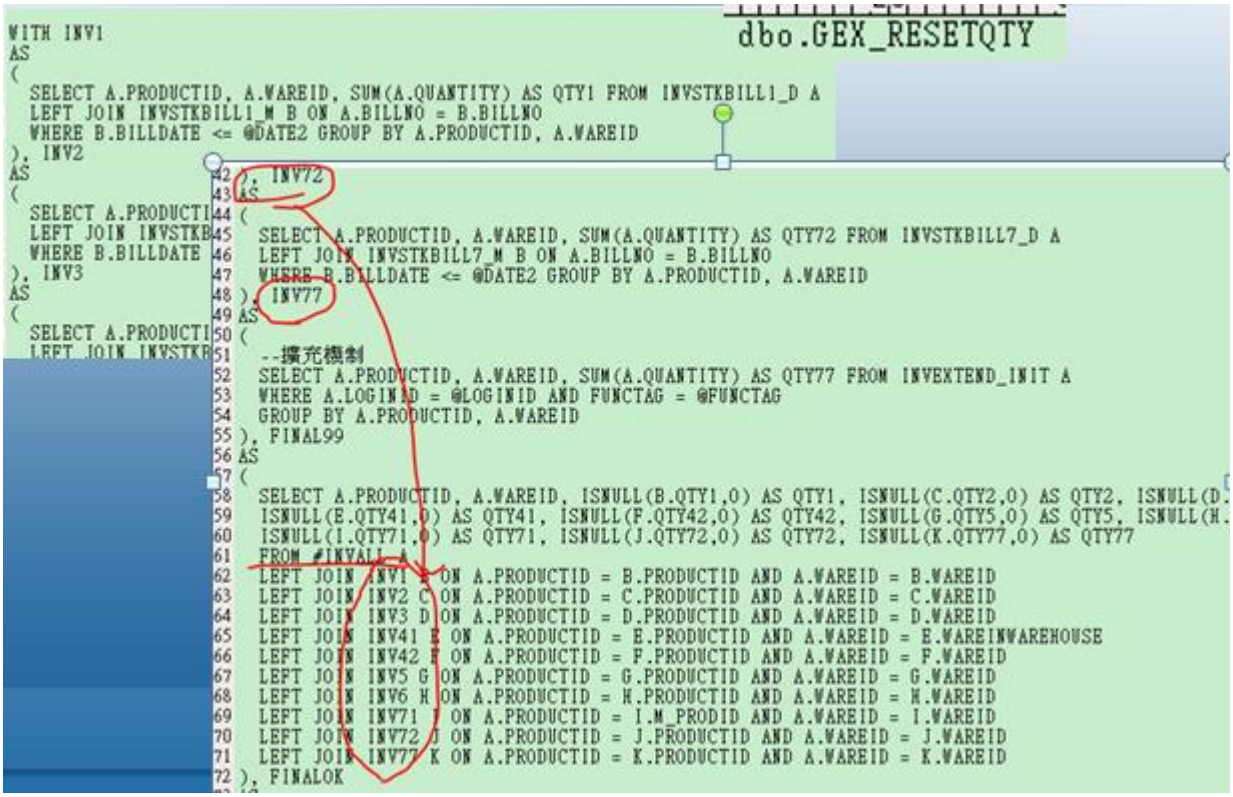
```
declare @start int,@end int, @sql nvarchar(600)
set @sql='select top'+str(@end-@start+1)+'+from T where rid not in(select top'+str(@str-1)+'Rid from T where Rid>-1)'
exec sp_executesql @sql
```

- 四種暫存表的寫法
  - 實體暫存檔(如 ESS_TMP)
  - WITH GEX01 (XX,XX) AS
  - DECLARE @ACT_TMP1 TABLE
  - #TmpDB、##TmpDB

所謂的實體暫存檔，事實上就是一般的資料表，完全一樣。只不過他存的資料內容，通常都是一些暫存的資料，因為要考慮多人同時使用的問題，一般都會有一個 LoginID 及 FuncTag 的欄位來區別呼叫的人員及程式代號。

至於 WITH 及 DECLARE @Table 下面我貼一些程式的片段供各位參考。這幾種暫存表，其他大部分的資料庫都有支援，除了 DECLARE @ACT_TMP1 TABLE 這種語法是 MS-SQL 專用，如果考慮日後有變更資料庫的可能，就盡量少使用即可。

- WITH



- DECLARE @Table

```
47
48 DECLARE @MS_DEPARTMENTID CHAR(1)
49 SET @MS_DEPARTMENTID = 'N'
50
51 SELECT @CNT=COUNT(*) FROM GEXBAS_TMP WHERE LOGINID = @LOGINID AND FUNCTIONTAG = @FUNCTAG ANI
52 IF @CNT > 0 BEGIN SET @MS_DEPARTMENTID = 'Y' END
53
54 EXEC GEX_ACTRETURN SUBJECT '4','zz',@TYPE,@LOGINID
55
56 DECLARE @ACT_TMP1 TABLE (
57                         CLASS1      VARCHAR(2),
58                         NAME1       NVARCHAR(30),
59                         CLASS2      VARCHAR(2),
60                         NAME2       NVARCHAR(30),
61                         TITLEID     VARCHAR(10),
62                         TITLECDESC  NVARCHAR(100),
63                         TITLEEDESC  VARCHAR(100),
64                         FINALAMT    DECIMAL(18,6),
65                         PERCENTRATE DECIMAL(10,2),
66                         TOTALAMT    DECIMAL(18,6),  --For 特殊類別科目合計用
67                         SEQNO       Int IDENTITY(1,1)
68                        );
69
70 WITH ACT4
71 AS
72 (
73   SELECT
74     TITLEID,SUM(CAMOUNT)-SUM(DAMOUNT) AS FINALAMT
75   FROM FIVACTFINAL A
```

- #TmpDB

```
3 SELECT A.AREAID,X.AREACNAME,
4   SUM(CASE WHEN OMONTH = 1 THEN SUMQTY ELSE 0 END) AS QTY1,
5   SUM(CASE WHEN OMONTH = 1 THEN SUMAMOUNT ELSE 0 END) AS SUM1,
6   SUM(CASE WHEN OMONTH = 2 THEN SUMQTY ELSE 0 END) AS QTY2,
7   SUM(CASE WHEN OMONTH = 2 THEN SUMAMOUNT ELSE 0 END) AS SUM2,
8   SUM(CASE WHEN OMONTH = 3 THEN SUMQTY ELSE 0 END) AS QTY3,
9   SUM(CASE WHEN OMONTH = 3 THEN SUMAMOUNT ELSE 0 END) AS SUM3,
0   SUM(CASE WHEN OMONTH = 4 THEN SUMQTY ELSE 0 END) AS QTY4,
1   SUM(CASE WHEN OMONTH = 4 THEN SUMAMOUNT ELSE 0 END) AS SUM4,
2   SUM(CASE WHEN OMONTH = 5 THEN SUMQTY ELSE 0 END) AS QTY5,
3   SUM(CASE WHEN OMONTH = 5 THEN SUMAMOUNT ELSE 0 END) AS SUM5,
4   SUM(CASE WHEN OMONTH = 6 THEN SUMQTY ELSE 0 END) AS QTY6,
5   SUM(CASE WHEN OMONTH = 6 THEN SUMAMOUNT ELSE 0 END) AS SUM6,
6   SUM(CASE WHEN OMONTH = 7 THEN SUMQTY ELSE 0 END) AS QTY7,
7   SUM(CASE WHEN OMONTH = 7 THEN SUMAMOUNT ELSE 0 END) AS SUM7,
8   SUM(CASE WHEN OMONTH = 8 THEN SUMQTY ELSE 0 END) AS QTY8,
9   SUM(CASE WHEN OMONTH = 8 THEN SUMAMOUNT ELSE 0 END) AS SUM8,
0   SUM(CASE WHEN OMONTH = 9 THEN SUMQTY ELSE 0 END) AS QTY9,
1   SUM(CASE WHEN OMONTH = 9 THEN SUMAMOUNT ELSE 0 END) AS SUM9,
2   SUM(CASE WHEN OMONTH = 10 THEN SUMQTY ELSE 0 END) AS QTY10,
3   SUM(CASE WHEN OMONTH = 10 THEN SUMAMOUNT ELSE 0 END) AS SUM10,
4   SUM(CASE WHEN OMONTH = 11 THEN SUMQTY ELSE 0 END) AS QTY11,
5   SUM(CASE WHEN OMONTH = 11 THEN SUMAMOUNT ELSE 0 END) AS SUM11,
6   SUM(CASE WHEN OMONTH = 12 THEN SUMQTY ELSE 0 END) AS QTY12,
7   SUM(CASE WHEN OMONTH = 12 THEN SUMAMOUNT ELSE 0 END) AS SUM12
8 FROM #INVO A
9 LEFT JOIN BASAREA X ON A.AREAID = X.AREAID
0 GROUP BY A.AREAID,X.AREACNAME
1 ORDER BY 1
2
3 DROP TABLE #INVO
4
```

- 配合 DB 的 user define function 讓 SQL 更靈活、更強大
  除了資料庫本身提供的自定義函數，也可以自己利用 SQL 語法撰寫 udf 自行擴充。寫成 udf 的最大優點就是可以在 SQL 語法中直接使用。例如，如果我們寫了一個 getCost(prodid,costmethod) 的 function，傳入產品編號,成本方式後，會回傳該產品的現有成本。我們可以寫成下面的程式

```
select prodid,dbo.getCost(prodid,1) from zen_product;
```

非常的直覺、簡潔有力。

- 迴圈式程式寫法的範例

```
--取得要跑回圈的資料，並寫入暫存檔中
select *,row_number() over(order by 1,2) as rowid into #tmp_curr from xxx order by 1,2

declare @maxcount int
declare @currid int
declare @nextrowid int

select @maxcount=count(*) from #tmp_curr
select @currid=min(rowid) from #tmp_curr

while (@currid <= @maxcount)
begin
   --取得本筆資料內容
   select @xx1=xx1,@xx2=xx2,@xx2=xx2 from #tmp_curr where rowid > @currid

   --執行真正要處理的程式內容
   xxxxxx

   -- 取得下一筆資料
   select @nextrowid = min(rowid) from #tmp_curr where rowid > @currid;

   set @currid=@nextrowid
end

drop table #tmp_curr
```

以下簡單的整理了不同資料庫間 SQL 語法的差異，方便您快速掌握，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 .轉為大寫 .轉為小寫 .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	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)	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)

.比較字串 s1 和 s2 .固定字串長度(左或右)		strcmp(s1,s2) lpad or rpad	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	MariaDB(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]
			case
			  when cond_1 then r_1
			  when cond_2 then r_2
			  [when ...]
			  [else result_n]

		end case	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)	declare r foo%rowtype; begin

		default ''; 	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	Drop temporary table if exists tmp_xxx;	If exists (select relname from pg_class where relname='tmp_xxx')

	2.create #tmp1 Create table #tmp1(f1 varchar(10), f2 int) 3.declare table declare @act_tmp1 table (titleid varchar(10),prevalue float)	Create temporary table if not exists tmp_xxx (fld1 int,fld2 varchar(20),fld3 numeric(8,2)) Create temporary table tmp_xxx as select * from basarea where filters	Then Truncate table tmp_xxx; Else Create temp table tmp_xxx (like basperson_xxx)

udf(user define function)的語法差異對照表

	MS-SQL	MariaDB(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

第九章

範本資料庫 Northwind 內容說明

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。因為大量且高強度的重度使用資料庫的核心功能，當然就必須深入的了解資料庫的功能及優缺點。這種開發方式優點是非常明顯的，但是缺點也不少，其中最為人所詬病的，最主要是

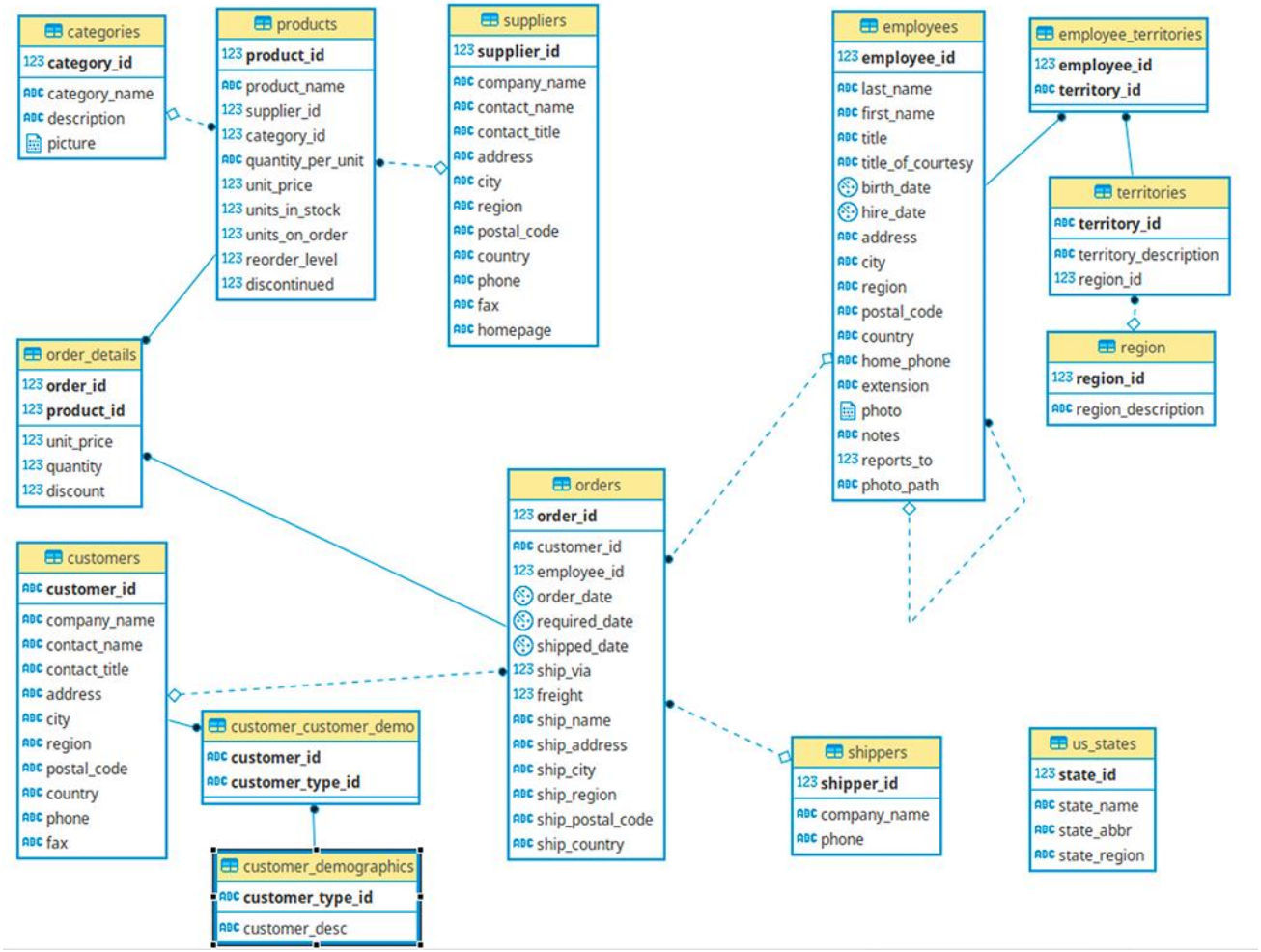
- Store Procedure 的除錯難度很高且非常不便
- 深度綁定資料庫後的跨資料庫問題。

雖然在實際的工作環境中，一旦選定資料庫，更換資料庫的機率並不高(因為風險及成本因素)。但是如果不想被資料庫所限制，最好的方式當然是能對其他的資料庫也有一定程度的了解。因為大部分的 SQL 資料庫都會遵循 ANSI 92 甚至更新的標準，所以讓轉換資料庫的難度大幅降低。

本章的重點，是要以 Microsoft 提供的 Northwind 北風資料庫為基礎並稍加擴充後，透過超過 100 個由淺入深的實際案例，直接在參照的 MariaDB 及 PostgreSQL 資料庫上參照實作，如此一來，各資料庫的功能、語法及效能，可以非常直觀的被展示出來。當然，如果這樣還不夠，或是希望再納入其他不同的資料庫，都可以在現有的基礎上進一步擴充，如此一來對資料庫的轉換或變更，就不在無從下手。

話不多說，在實際進入實作前，我們先簡單的了解一下 Northwind 這個範本資料庫的大致結構。由於本書並非基礎入門書籍，假設讀者至少有基本的資料庫概念，並至少了解基本的 SQL 語法，如果您是初學者，還請您先參考其他入門書籍，謝謝。

下面這張圖，是 Northwind 的實體關聯圖（Entity Relationship Diagram，簡稱 ERD）



下面是資料內容的一些基本的說明

Table(表) / 欄位(Field)	欄位說明
Categories	**商品類別檔**
CategoryID	類別 Id
CategoryName	類別名稱
CustomerCustomerDemo	**客戶類別表 1**
CustomerID	客戶 ID
CustomerTypeID	客戶類別 ID
CustomerDemographics	**客戶類別表 2**
CustomerTypeID	客戶類別 ID
CustomerDesc	客戶描述
Customers	**客戶資料表**
CustomerID	客戶 ID
CompanyName	公司名稱
ContactName	客戶姓名
ContactTitle	客戶頭銜

Address	聯絡地址
City	所在城市
Region	所在地區
Phone	電話
Fax	傳真
Employees	**員工資料表**
EmployeeID	員工代號
LastName + FirstName	員工姓名
BirthDate	生日
HireDate	僱用日期
Address	地址
EmployeeTerritories	**員工區域資料表**
EmployeeID	員工編號
TerritoryID	區域代號
Order_Details	**訂單明細資料**
OrderID	訂單編號
ProductID	產品編號
UnitPrice	單價
Quantity	訂購數量
Discount	折扣
Orders	**訂單主表**
OrderID	訂單編號
CustomerID	客戶編號
EmployeeID	員工編號
OrderDate	訂購日期
RequiredDate	預計到達日
ShippedDate	發貨日期
ShipVia	運貨商
Freight	運費
ShipName	貨主姓名
ShipAddress	貨主地址
Products	**產品資料表**
ProductID	產品 ID
ProductName	產品名稱
SupplierID	供應商 ID
CategoryID	型別 ID
UnitPrice	單價

UnitsInStock	庫存數量
Discontinued	中止
Region	**地區資料表(大區)**
RegionID	地區 ID
RegionDescription	地區描述
Shippers	**運貨商資料表**
ShipperID	運貨商 ID
CompanyName	公司名稱
Phone	聯絡電話
Suppliers	**供應商資料表**
ShipperID	供應商 ID
CompanyName	公司名稱
Phone	聯絡電話
Territories	**區域資料表(小區)**
TerritoryID	區域編號
TerritoryDescription	區域描述
RegionID	地區編號

看過上面的內容後，相信您大概心裡就有點底了，這個資料庫是一個具體而微的小型訂單系統，實際運行的系統 Table 當然更多，欄位也更精細，但是拿來當作範例是足夠的。在我找到的 Northwind 版本中，還有 10 多個 View 及 5,6 個 Store Procedure，用的 SQL 指令並不複雜，剛好可以拿來練手。

另外，為了範例題目的需要，又擴充了幾個 Table 及欄位，一併說明如下

Table(表) / 欄位(Field)	欄位說明
Department	**部門檔**
DepartmentID	部門 Id
DepartmentName	部門名稱
ManagerID	部門主管
Employees	**員工資料表(新增欄位)**
DepartmentID	員工所屬部門 Id
Salary	薪資
EmployeeHistory	**員工薪資異動檔(Trigger)**
ID	PK
EmployeeID	員工代號
ChangeDate	異動日期
org_salary	原薪資

new_salary	新的薪資

下面這些 Table 沒有在 Script 中，因為要配合相關 SQL 語法的操作，在練習題目時會用語法加入資料庫中，簡單說明如下

Table(表) / 欄位(Field)	欄位說明
FinalPay	**客戶帳款餘額**
CustomerID	客戶編號
Amount	客戶帳款餘額
FinalPay2	**廠商帳款餘額**
ShipperID	供應商 ID
Amount	供應商帳款餘額
Orders_history	**Orders 的 History 檔(Trigger)**
LogType	異動類別
其他欄位同 Orders	
FinalInventory	**庫存餘額檔(練習 View 用)**
ProductID	產品 ID
Quantity	現有數量

基本上 Northwind 這個資料庫是 Microsoft 為了 SQL Server 測試所提供的一個範本資料庫，其他資料庫當然不會有相關的 Script，但因本書的需求，所以我在 gitHub 上有找到一些其他資料庫的類似範本。但是實際執行時有發現一些不一致的地方，所以我有修改部分 Script(三個資料庫的 Script 都有修改)，另外資料內容本身也有差異，我就沒有去改動內容了。

我將各資料庫已修正過的 SQL Script 上傳到 GitHub，網址如下

https://github.com/gexMichael/Play-Learn-SQL

- MS-SQL：含 Table Schema、Data 及 Store Procedure 及 View
  northwind-mssql-withdata.txt
- MariaDB：含 Table Schema、Data
  Northwind-Script-mysql.sql
  sp&view_mysql.txt (Store Procedure 及 View)
- PostgreSQL：含 Table Schema、Data
  Northwind-Script-postgre.sql
  sp&view_postgre.txt(Store Procedure 及 View)

在之前各資料庫的安裝教學章節中，已將 Northwind 資料庫都已安裝完成，此處就不再贅述。關於 Northwind 我就簡單的介紹到這邊，如果還有其他的測試的需求，就請自行擴充，我想有一個通用的基礎資料庫對於實作是有很大的幫助的，希望對您學習 SQL 有所助益。

第 10 章

SQL 語法轉換記實

本章是本書的核心章節，我準備以前面已經安裝好的 Northwind 的資料庫為基礎，透過由淺入深的數十個實際案例的實作，以便有系統的學習多個資料庫的 SQL 解決方案。這個架構可以依照您的實際需求擴充，只要不斷的收集更多的案例，並對更多資料庫用相同的方式，就可以自由擴充。

學習任何一種語言，多看範例並不斷實作是學習的不二法門。因為筆者個人對 MS-SQL 有超過 10 年的實際開發經驗，程度還算可以，所以本書的主要視角是以一個對 MS-SQL 已有一定程度理解的研發人員，如何在此基礎上透過實例，快速地學習其他資料庫的 SQL 語法，在寫這本書之前，我對 MariaDB 和 PostgreSQL 的認知，只有遵循 ANSI92 基礎的標準 SQL 語法，但是在撰寫這本書的過程中，透過不斷的實際改寫、測試，對這 2 個資料庫，雖然還稱不上高手，但是一般的問題解決起來問題應該不大，再加上還有谷歌大神的幫忙，解決工作上的問題基本不是難事。

原本是想寫一個資料庫轉換程式，這個轉換程式不僅是轉 Table Schema 及資料，而是希望連 Store Procedure、user define function 及 Trigger 等資料庫程式都能完成轉換，但是實際開工後才發現這不是件容易的事，Table Schema 和資料的轉換基本上還有跡可循，但是 SQL 語法的轉換就是另一個層次的問題了，所以就先放下這個計畫，轉而先完成本書，畢竟對其他資料庫不熟，怎麼寫轉換程式? 所以就有了這本書。接下來，我們就依序由初級、中級、高級，透過實際程式的撰寫來學習 SQL，祝您有個愉快的學習之旅。在那之前，以下是關於 MS-SQL 語法的幾個提醒：

- 變數定義 Declare 可以在 sp 的任意地方，但建議和其他資料庫一樣統一放在程式開頭。
- 句子的結尾可以無特殊符號，但建議用分號 ; 和其他資料庫一致。
- 取得資料為變數的寫法和其他資料庫不同，只能特別注意

  select @custId=CustomerID,@cname=Companyname from Customers;

  vs

  select CustomerID,Companyname into _custId,_Cname from Customers;

alpha (初級)
001 建立資料庫
002 刪除資料庫
003 新建資料表
004 取得所有的 Table 及 View
005 列出表裡的所有的列名
006 增加一個欄位
007 刪除一個欄位
008 添加主鍵
009 刪除主鍵
010 創建索引
011 刪除索引
012 刪除新表
013 創建視圖
014 一般資料取得
015 新增一筆資料
016 更新一筆資料
017 刪除一筆資料
018 資料查詢(過濾)
019 資料排序
020 計算筆數
021 計算合計
022 計算平均值
023 計算最大值
024 計算最小值
025 資料聯集(UNION)
026 從其他表寫入資料
027 從其他表複製並創建新表
028 子查詢
029 顯示客戶、產品和最後訂貨日(子查詢)
030 動態資料查詢(子查詢)
031 between 的用法
032 in 的用法
033 四表聯查問題(left,right join)
034 差異天數的計算
035 前 10 條記錄

036 隨機取出 **10** 條數據
037 隨機選擇記錄
038 初始化表**(table)**
039 選擇從 **10** 到 **15** 筆的資料
040 Get the name of Current Database
041 Encrypt Password(加密**)**
042 查詢分公司最低及平均薪水，且平均薪水**> 1.4** 倍最低薪水的資料
043 工資 **5** 萬以上的員工降薪 **10%** 工資低於 **3** 萬的員工加薪 **20%**
044 找出有銷售記錄的商品**(**使用 **DISTINCT)**
045 找出有銷售記錄的商品**(**使用 **EXISTS)**
046 找出所有單價低於 **20** 的供應商**(**使用 **EXISTS)**
047 靈活使用 **HAVING** 子句
048 針對平均定價少於**$20** 的 **Meat/Poultry, Seafood,**列出這些產品及平均定價
049 一般的 **case when** 語法
050 欄位型態的轉換**(**文數字**)**
051 欄位型態的轉換**(**日期**)**
052 資料庫自增值的做法
053 找出各部門第二高薪人員
054 刪除視圖**(View)**
055 常用的日期函式

等級：alpha (初級)

001 建立資料庫

MS-SQL / MariaDB / PostgreSQL

```
create database Northwind_test;
```

002 刪除資料庫

MS-SQL / MariaDB / PostgreSQL

```
drop database Northwind_test;
```

003 新建資料表

MS-SQL

```
-- 現有庫存
CREATE TABLE FinalInventory (
   [ProductID] [int] NOT NULL,
   [Quantity] [int] NOT NULL,
   PRIMARY KEY (ProductID)
);
-- 客戶帳款餘額(練習 Table 操作用)
CREATE TABLE FinalPay (
   CustomerID varchar(10) NOT NULL,
   Amount int NOT NULL
);
-- 測試重複資料
CREATE TABLE DulpData (
   [DulpID] int identity(1,1) NOT NULL,
   [DulpValue] varchar(20) NOT NULL,
   [DulpDate] datetime
);
```

MariaDB

```
-- 現有庫存
CREATE TABLE FinalInventory (
      ProductID int NOT NULL,
      Quantity int NOT NULL,
      PRIMARY KEY (ProductID)
);
```

```
-- 客戶帳款餘額(練習 Table 操作用)
CREATE TABLE FinalPay (
        CustomerID varchar(10) NOT NULL,
        Amount int NOT NULL
);
/* 測試重複資料 */
CREATE TABLE DulpData (
        DulpID SERIAL,
        DulpValue varchar(20) NOT NULL,
        DulpDate timestamp
);
```

PostgreSQL

```
/* 現有庫存 */
CREATE TABLE FinalInventory (
    ProductID int NOT NULL,
    Quantity int NOT NULL,
    PRIMARY KEY (ProductID)
);
-- 客戶帳款餘額(練習 Table 操作用)
CREATE TABLE FinalPay (
    CustomerID varchar(10) NOT NULL,
    Amount int NOT NULL
);
/* 測試重複資料 */
CREATE TABLE DulpData (
    DulpID int UNSIGNED AUTO_INCREMENT,
    DulpValue varchar(20) NOT NULL,
    DulpDate timestamp,
    PRIMARY KEY (DulpID)
);
```

004 取得所有的 Table 及 View

MS-SQL

```
SELECT * FROM INFORMATION_SCHEMA.Tables
where table_name like '%language%'


select name from sysobjects where type='U';
```

MariaDB

```
SELECT * FROM pg_catalog.pg_tables;
```

PostgreSQL

```
SHOW TABLES FROM northwind LIKE 'order%';


SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public'
ORDER BY table_name;
```

005 列出表裡的所有的列名

MS-SQL

```
select name from syscolumns where id=object_id('Orders');
```

MariaDB

```
SHOW COLUMNS FROM customers;
```

PostgreSQL

```
SELECT D.column, D.type, D.characterMaximumLength, B.description
FROM (
	SELECT
		c.relname AS name,
		CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN
'materialized view' WHEN 'i' THEN 'index' WHEN 'S' THEN 'sequence' WHEN 's' THEN 'special'
WHEN 'f' THEN 'foreign table' END as type,
		pg_catalog.pg_get_userbyid(c.relowner) as owner,
		relfilenode
	FROM pg_catalog.pg_class c
	WHERE
		c.relkind IN ('r','s','') AND c.relname = 'customers'
	ORDER BY 1,2
) A
LEFT JOIN (
	SELECT
		column_name AS column, ordinal_position, data_type AS type, table_name,
character_maximum_length characterMaximumLength
	FROM information_schema.columns
) D ON A.name = D.table_name
LEFT JOIN pg_catalog.pg_description B ON A.relfilenode = B.objoid AND D.ordinal_position =
B.objsubid;
```

006 增加一個欄位

MS-SQL / MariaDB / PostgreSQL

```
ALTER TABLE FinalPay ADD InitAmt int;
ALTER TABLE FinalPay ADD FinalAmt int;
```

007 刪除一個欄位

MS-SQL / MariaDB / PostgreSQL

```
ALTER TABLE FinalPay DROP COLUMN FinalAmt;
```

008 添加主鍵

MS-SQL / MariaDB

```
ALTER TABLE FinalPay ADD CONSTRAINT pk_FinalPay PRIMARY KEY(CustomerID);
```

009 刪除主鍵

MS-SQL / MariaDB / PostgreSQL

```
ALTER TABLE FinalPay DROP CONSTRAINT pk_FinalPay;
```

010 創建索引

MS-SQL / MariaDB / PostgreSQL

```
CREATE INDEX idx_ID_City ON Customers (CustomerID, City);
```

011 刪除索引

MS-SQL / MariaDB / PostgreSQL

```
DROP INDEX Customers.idx_ID_City;
```

PostgreSQL

```
DROP INDEX idx_id_city;
```

012 刪除新表

MS-SQL / MariaDB / PostgreSQL

```
DROP Table FinalPay;
```

013 創建視圖

MS-SQL / MariaDB / PostgreSQL

```
create View vFinalInventory
as
select fi.ProductID,P.ProductName,fi.Quantity from FinalInventory fi
```

```
left join Products P ON fi.ProductID = P.ProductID;
```

014 一般資料取得

MS-SQL / MariaDB / PostgreSQL

```
select * from Categories;
select * from vFinalInventory where ProductId >= 1 and ProductID <= 10;
```

015 新增一筆資料

MS-SQL / MariaDB / PostgreSQL

```
CREATE INDEX idx_ID_City ON Customers (CustomerID, City);
```

016 更新一筆資料

MS-SQL / MariaDB / PostgreSQL

```
update FinalInventory set Quantity=8888 where ProductID = 9999;
```

017 刪除一筆資料

MS-SQL / MariaDB / PostgreSQL

```
delete from FinalInventory where ProductID = 9999;
```

018 資料查詢(過濾)

MS-SQL / MariaDB / PostgreSQL

```
select * from Orders where CustomerID like '%FRANK%'
```

019 資料排序

MS-SQL / MariaDB / PostgreSQL

```
select * from Orders order by CustomerID,OrderDate ASC;
select * from Orders order by CustomerID,OrderDate DESC;
```

020 計算筆數

MS-SQL / MariaDB / PostgreSQL

```
select count(*) as totalcount from FinalInventory;
```

021 計算合計

MS-SQL / MariaDB / PostgreSQL

```
select OD.ProductID,P.ProductName,sum(OD.Quantity) as Quantity
from Order_Details OD
```

```
left join Products P ON OD.ProductID = P.ProductID
group by OD.ProductID,P.ProductName;
```

022 計算平均值

MS-SQL / MariaDB / PostgreSQL

```
select productid,avg(unitprice) as avgvalue from Order_Details group by productid;
```

023 計算最大值

MS-SQL / MariaDB / PostgreSQL

```
select productid,max(unitprice) as max_value from Order_Details group by productid;
```

024 計算最小值

MS-SQL / MariaDB / PostgreSQL

```
select productid,min(unitprice) as minvalue from Order_Details group by productid;
```

025 資料聯集(UNION)

MS-SQL

```
select * From Orders where OrderDate = '1998-05-01'
union all
select * From Orders where OrderDate = '1998-05-06'
```

MariaDB / PostgreSQL

```
select * From Orders where OrderDate >= '2006-07-01' and OrderDate <= '2006-07-10'
union all
select * From Orders where OrderDate = '2006-12-06'
```

026 從其他表寫入資料

MS-SQL / MariaDB / PostgreSQL

```
Insert Into FinalInventory
select OD.ProductID,sum(OD.Quantity) as Quantity from Order_Details OD
group by OD.ProductID;
```

027 從其他表複製並創建新表

MS-SQL

```
select OD.ProductID,sum(OD.Quantity) as Quantity Into #tmpOD
From Order_Details OD group by OD.ProductID;

select * from #tmpOD;
```

```
Drop Table #tmpOD;
```

MariaDB

```
CREATE TEMPORARY TABLE tmpOD
as select OD.ProductID,sum(OD.Quantity) as Quantity
From Order_Details OD group by OD.ProductID;

select * from tmpOD;
Drop Table tmpOD;
```

PostgreSQL

```
select OD.ProductID,sum(OD.Quantity) as Quantity Into temp table tmpOD
From Order_Details OD group by OD.ProductID;

select * from tmpOD;
Drop Table tmpOD;
```

028 子查詢

MS-SQL / MariaDB / PostgreSQL

```
select * from vFinalInventory where ProductID IN (select ProductID from Products where
ProductName like '%ch%');
```

029 顯示客戶、產品和最後訂貨日(子查詢)

MS-SQL / MariaDB / PostgreSQL

```
select X.customerID,C.CompanyName,P.ProductID,P.ProductName,X.oDate From
(
Select O.CustomerID,O2.ProductID,max(OrderDate) oDate From Orders O
left join Order_Details O2 ON O.OrderID = O2.OrderID
where O.CustomerID = 'BERGS'
Group by O.CustomerID,O2.ProductID
) X
left join Customers C ON X.CustomerID = C.CustomerID
left join Products P ON P.ProductID = X.ProductID
```

030 動態資料查詢(子查詢)

MS-SQL / MariaDB / PostgreSQL

```
select * from (select * from Products where ProductName like '%ch%') T where t.ProductID > 30;
```

031 between 的用法

MS-SQL

```
select * from Orders where CustomerID between 'b' and 'd';
select * from Orders where EmployeeID not between 3 and 8;
```

MariaDB / PostgreSQL

```
select * from Orders where CustomerID between 3 and 8;
select * from Orders where EmployeeID not between 3 and 8;
```

032 in 的用法

MS-SQL

```
select * from Orders where CustomerID in ('VINET','TOMSP','HANAR','SUPRD');
```

MariaDB / PostgreSQL

```
select * from Orders where CustomerID in (3,8);
```

033 四表聯查問題(left,right join)

MS-SQL

```
select a.OrderID,E.LastName,a.OrderDate,P.ProductName,b.Quantity,b.UnitPrice
from Orders a
left join Order_Details b on a.OrderID=b.OrderID
right join Products P on b.ProductID = P.ProductID
inner join Employees E on a.EmployeeID=E.EmployeeID
where orderDate = '1996-07-10'
```

MariaDB / PostgreSQL

```
select a.OrderID,E.LastName,a.OrderDate,P.ProductName,b.Quantity,b.UnitPrice
from Orders a
left join Order_Details b on a.OrderID=b.OrderID
right join Products P on b.ProductID = P.ProductID
inner join Employees E on a.EmployeeID=E.EmployeeID
where orderDate = '2006-07-10'
```

034 差異天數的計算

MS-SQL

```
select * from Orders where datediff(day,OrderDate,'1998-05-06') <= 3;
```

MariaDB

```
select * from Orders where EXTRACT(DAY FROM '2008-05-06' - OrderDate) <= 3;
```

PostgreSQL

```
select * from Orders where datediff('2008-05-06',OrderDate) <= 3;
```

035 前 10 條記錄

MS-SQL

```
select top 10 * from FinalInventory where ProductID >= 10;
```

MariaDB / PostgreSQL

```
select * from FinalInventory where ProductID >= 10 limit 10;
```

036 隨機取出 10 條數據

MS-SQL

```
select top 10 * from FinalInventory order by newid();
```

MariaDB

```
select * from FinalInventory order by rand() limit 10;
```

PostgreSQL

```
select * from FinalInventory order by uuid_generate_v4() limit 10;
```

037 隨機選擇記錄

MS-SQL

```
select newid();
```

MariaDB

```
select rand();
select uuid();
```

PostgreSQL

```
CREATE EXTENSION "uuid-ossp";
SELECT uuid_generate_v4();
```

038 初始化表(table)

MS-SQL / MariaDB / PostgreSQL

```
TRUNCATE TABLE FinalInventory;
```

039 選擇從 10 到 15 筆的資料

MS-SQL

```
select top 5 * from (select top 15 * from FinalInventory order by ProductID asc) fi
order by ProductID desc;
```

MariaDB / PostgreSQL

```
select * from (select * from FinalInventory order by ProductID asc limit 15) fi
order by ProductID desc limit 5;
```

040 Get the name of Current Database

MS-SQL

```
SELECT db_name();
```

MariaDB

```
SELECT DATABASE();
```

PostgreSQL

```
SELECT current_database();
```

041 Encrypt Password(加密)

MS-SQL

```
SELECT HASHBYTES('MD2', 'dbrnd.com') AS MD2
SELECT HASHBYTES('MD4', 'dbrnd.com') AS MD4
SELECT HASHBYTES('MD5', 'dbrnd.com') AS MD5
SELECT HASHBYTES('SHA1', 'dbrnd.com') AS SHA1
SELECT HASHBYTES('SHA2_256', 'dbrnd.com') AS SHA2_256
SELECT HASHBYTES('SHA2_512', 'dbrnd.com') AS SHA2_512
```

MariaDB / PostgreSQL

```
select MD5('a');
```

042 查詢分公司最低及平均薪水，且平均薪水> 1.4 倍最低薪水的資料

MS-SQL / MariaDB / PostgreSQL

```
select E.DEPARTMENTID,min(salary) mins,    AVG(salary) avgs
from EMPLOYEES E
left join Department D on D.DepartmentId = E.DepartmentId
where D.DepartmentName like '%District%'
group by E.DEPARTMENTID
having avg(salary) > min(salary) * 1.4
```

043 工資 5 萬以上的員工降薪 10% 工資低於 3 萬的員工加薪 20%

MS-SQL / MariaDB / PostgreSQL

```
UPDATE Employees
SET salary = CASE WHEN salary >= 50000 THEN salary * 0.9
WHEN salary < 30000 THEN salary * 1.2
ELSE salary END;
```

044 找出有銷售記錄的商品(使用 DISTINCT)

MS-SQL / MariaDB / PostgreSQL

```
SELECT DISTINCT P.ProductID,P.ProductName
FROM Products P
INNER JOIN Order_Details OD
ON P.ProductID = OD.ProductID;
```

045 找出有銷售記錄的商品(使用 EXISTS)

MS-SQL / MariaDB / PostgreSQL

```
SELECT ProductID,ProductName FROM Products P
WHERE EXISTS
(
   SELECT * FROM Order_Details OD WHERE P.ProductID = OD.ProductID
);
```

046 找出所有單價低於 20 的供應商(使用 EXISTS)

MS-SQL / MariaDB / PostgreSQL

```
SELECT CompanyName FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID =
Suppliers.supplierID AND UnitPrice < 20);
```

047 靈活使用 HAVING 子句

MS-SQL / MariaDB / PostgreSQL

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

048 針對平均定價少於$20 的 Meat/Poultry, Seafood,列出這些產品及平均定價

MS-SQL / MariaDB / PostgreSQL

```
SELECT ProductName,AVG(UnitPrice) avgPrice
FROM Products
WHERE CategoryID IN (6,8)    -- Meat/Poultry, Seafood
GROUP BY ProductName
HAVING AVG(UnitPrice) < 20;
```

049 一般的 case when 語法

MS-SQL / MariaDB / PostgreSQL

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM Order_Details;
```

050 欄位型態的轉換(文數字)

MS-SQL

```
select CAST('123.4' as decimal(9,2));    -- 123.40
select CONVERT(decimal(9,2), '123.4'); -- 123.40

declare @Num money;
set @Num = 1234.56;
select CONVERT(varchar(20), @Num, 0);    -- 1234.56
select CONVERT(varchar(20), @Num, 1);    -- 1,234.56
select CONVERT(varchar(20), @Num, 2);    -- 1234.5600
```

MariaDB

```
select CAST('123.4' as decimal(9,2));    -- 123.40
~~select CONVERT(decimal(9,2), '123.4'); -- 123.40~~

select CONVERT(1234.56, SIGNED);    -- 1234.56
select CONVERT(1234.56, char);    -- 1234.56
select CONVERT(1234.56, DECIMAL(10,4));    -- 1234.5600
```

PostgreSQL

```
select CAST('123.4' as decimal(9,2));    -- 123.40
~~select CONVERT(decimal(9,2), '123.4');    123.40~~

select round(1::numeric/4::numeric,2);
SELECT substr(CAST (1234 AS text), 3,1);
select to_char(125.8::real, '999');
select to_char(125.8::real, '999D99');
```

051 欄位型態的轉換(日期)

MS-SQL

```
--日期轉字串
```

```
SELECT convert(varchar, getdate(), 111); -- yyyy/mm/dd
SELECT convert(varchar, getdate(), 112); -- yyyymmdd
SELECT convert(varchar, getdate(), 120); -- yyyy-mm-dd hh:mm:ss(24h)
--字串轉日期
SELECT convert(datetime, '2016/10/23', 111); -- yyyy/mm/dd
SELECT convert(datetime, '20161023', 112); -- ISO yyyymmdd
SELECT convert(datetime, '2016-10-23 20:44:11', 120); -- yyyy-mm-dd hh:mm:ss(24h)
```

MariaDB

```
--日期轉字串
select date_format(now(), '%Y-%m-%d');
select date_format(now(), '%Y%m%d');
select date_format(now(), '%Y-%m-%d %H:%i:%s');
--字串轉日期
select str_to_date('2014-08-20', '%Y-%m-%d');
select str_to_date('20140820', '%Y%m%d');
select str_to_date('2014-08-20 01:02:03', '%Y-%m-%d %H:%i:%s');
```

PostgreSQL

```
--日期轉字串
select to_char(current_timestamp, 'YYYY-MM-DD');
select to_char(current_timestamp, 'YYYYMMdd');
select to_char(current_timestamp, 'YYYY-MM-DD HH24:MI:SS');
--字串轉日期
select to_date('2016-11-26','yyyy-MM-dd');
select to_date('20161126','yyyyMMdd');
select to_timestamp('2016-11-26 13:30:32','yyyy-MM-dd hh24:mi:ss');
```

052 資料庫自增值的做法

MS-SQL

```
create table #testAutoNo (id int identity(1,1), AutoValue varchar(100));
-- test data
insert into #testAutoNo (AutoValue) values ('test-auto');
select * from #testAutoNo;
drop table #testAutoNo;
```

MariaDB

```
DROP TABLE IF EXISTS _testAutoNo;
create table _testAutoNo (id int UNSIGNED AUTO_INCREMENT, AutoValue varchar(100), PRIMARY
KEY (id));
```

```
-- test data
insert into _testAutoNo (AutoValue) values ('test-1');
insert into _testAutoNo (AutoValue) values ('test-2');
select * from _testAutoNo;
```

PostgreSQL

```
create table _testAutoNo (id SERIAL, AutoValue varchar(100));
-- test data
insert into _testAutoNo (AutoValue) values ('test-1');
insert into _testAutoNo (AutoValue) values ('test-2');
select * from _testAutoNo;
drop table _testAutoNo;
```

053 找出各部門第二高薪人員

MS-SQL / MariaDB / PostgreSQL

```
SELECT T.EmployeeID , T.EmployeeName, T.DepartmentID, T.Salary
FROM
   (
      SELECT EmployeeID , LastName+ ' ' + FirstName as EmployeeName, DepartmentID, Salary ,
                  RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) AS RANKNO
        FROM Employees
   ) AS T
WHERE RANKNO = 2
```

054 刪除視圖(View)

MS-SQL / MariaDB / PostgreSQL

```
drop view vFinalInventory;
```

055 常用的日期函式

MS-SQL

```
--日期的年月日
select year(getdate()) as YY, month(getdate()) as MM, Day(getdate()) as DD;
--今天星期幾
SELECT DatePart(WeekDay,getdate());
--3 天後, 2 天前的日期
select dateadd(day,3,getdate()),dateadd(day,-2,getdate());
--如何取得本月的天數
select day(dateadd(month,1,getdate()) - day(getdate()));
```

```
--兩個指定日期的差異天數
select DateDiff(Day,'2000-1-1',getdate());
```

MariaDB

```
--  日期的年月日
select year(now()) as YY, month(now()) as MM, Day(now()) as DD;
--  今天星期幾
SELECT WEEKDAY(now())+1 as weekdate;
-- 3 天後, 2 天前的日期
select date_add(now(), interval 3 day),date_add(now(), interval -2 day);
--  如何取得本月的天數
select day(last_day(curdate()));
--  兩個指定日期的差異天數
select datediff(curdate(), '2021-01-02');
```

PostgreSQL

```
--日期的年月日
select EXTRACT(YEAR FROM current_timestamp) as YY, EXTRACT(MONTH FROM
current_timestamp) as MM, EXTRACT(DAY FROM current_timestamp) as DD;
--今天星期幾
SELECT EXTRACT(WEEK FROM current_timestamp);
--3 天後, 2 天前的日期
select current_date+3,current_date-2;
--如何取得本月的天數
SELECT DATE_PART('days', DATE_TRUNC('month', NOW()) + '1 MONTH'::INTERVAL - '1
DAY'::INTERVAL );
--兩個指定日期的差異天數
select ('2009-12-31'::date - '2009-01-01'::date) as daydiff;
```

beta (中級)
001 where 或 order by 子句中的 case-when
002 暫存表
003 Declare Table 的寫法
004 WITH 的寫法
005 store procedure 的架構
006 create procedure 寫法(分頁取回資料)
007 呼叫 sp 的方法
008 create udf 的寫法
009 計算工作年資(udf)
010 呼叫 udf 的方法
011 顯示所有的 sp 及 udf
012 Trigger 的寫法
013 查找所有表的行數
014 兩張關聯表，刪除主表中已經在副表中沒有的資訊
015 一條 sql 語句搞定資料庫分頁
016 選擇在每一組 b 值相同的資料中對應的 a 最大的記錄的所有資訊
017 包括所有在 TableA 中但不在 TableB(TableC) 中並刪除所有重複行的表
018 刪除重複記錄
019 remove New Line Character from a string
020 利用 Trigger 寫入異動紀錄檔
021 Wildcard samples
022 SQL Any ALL
023 完整列出所有 Table 的 Schema 及 Indexs
024 交易 Transaction
025 Maths of ROW_NUMBER, RANK, DENSE_RANK
026 if else 語法和 try-catch 的使用
027 split 字串切割函式
028 View 和 Store Procedure 配合使用(訂單合計金額)
029 年度商品類別銷售金額
030 業務人員國家別銷售金額
031 計算工作天數
032 人員組織圖
033 計算各經理和其部屬薪資
034 顯示資料庫中所有 Table 得筆數
035 計算去掉最高收入及最低收入後，公司員工平均收入是多少

036 某年不同月份的客戶下單次數
037 for-loop 及 **while-loop** 的應用
038 位元運算子**(AND OR** 語法**)**
039 Get the data difference between two Tables
040 考勤資料異常分析
041 如何跨資料庫取得資料

等級：beta (中級)

001 where 或 order by 子句中的 case-when

MS-SQL / MariaDB / PostgreSQL

```
SELECT CompanyName, City, Country
FROM Customers
ORDER BY
(CASE
    WHEN City IS NULL THEN Country
    ELSE City
END);
```

002 暫存表

MS-SQL

```
create table #tmp1 (id int primary key,note nvarchar(max));
insert into #tmp1 values (1,'test');
insert into #tmp1 values (2,'test');
select * from #tmp1;
drop table #tmp1;
```

MariaDB

```
DROP TEMPORARY TABLE IF EXISTS _tmp1;
create TEMPORARY table _tmp1 (id int primary key,note text);
insert into _tmp1 values (1,'test');
insert into _tmp1 values (2,'test');
select * from _tmp1;
```

PostgreSQL

```
DROP TABLE IF EXISTS _tmp1;
create TEMPORARY table _tmp1 (id int primary key,note text);
insert into _tmp1 values (1,'test');
insert into _tmp1 values (2,'test');
select * from _tmp1;
```

003 Declare Table 的寫法

MS-SQL

```
DECLARE @ProductTotals TABLE
(
```

```
    ProductID int,
    Revenue decimal(8,2)
);
```

MariaDB / PostgreSQL

```
/*  只能用  temp table  代替，不支援  Declare Table */
CREATE TEMP TABLE product_totals (
      product_id int
  , revenue money
);
```

004 WITH 的寫法

MS-SQL / MariaDB / PostgreSQL

```
with PSum
AS
(
    select ProductId,sum(quantity) sumQty from Order_Details group by ProductId
)
select PSum.ProductID,P.ProductName,PSum.SumQty from PSum
left join Products P ON PSum.ProductId = P.ProductId;
```

005 store procedure 的架構

MS-SQL

```
create procedure procedure_name
(
    @parameter1 int,
    @parameter2 varchar(100)
)
as
begin
    -- declare variant
    declare @var1 int;
    declare @var2 nvarchar(max);

    -- stored procedure body
end;
```

MariaDB

```
DELIMITER //
```

```
create or replace procedure procedure_name (parameter_list)
begin
   -- variable declaration
   -- stored procedure body
end
//
DELIMITER ;
```

PostgreSQL

```
create [or replace] procedure procedure_name(parameter_list)
language plpgsql
as $$
declare
-- variable declaration
begin
-- stored procedure body
end; $$
```

006 create procedure 寫法(分頁取回資料)

MS-SQL

```
create procedure usp_Services_Paging (
   @tablename varchar(100),
   @pkey varchar(100),
   @pageno int,
   @recperpage int
)
as
/*
   Function:  通用的後端分頁取資料  Services
   Author:      Michael Chen
   Called:      exec usp_Services_Paging 'Orders','OrderID',1,10
   History:
      2020-12-4 build.
*/
begin
   declare @sql nvarchar(max);
   declare @recno int;

   set @recno = (@pageno-1) * @recperpage;
```

```
   set @sql = 'SELECT * FROM ' +@tablename+ ' ORDER BY ' + @pkey+
                                        ' OFFSET ' +cast(@recno as varchar)+ ' ROWS FETCH NEXT '
+cast(@recperpage as varchar)+ ' ROWS ONLY';
   exec sp_executesql @sql;
end
```

MariaDB

```
DELIMITER //
CREATE or replace PROCEDURE usp_Services_Paging(
  in _pagecurrent int,
  in _pagesize int,
  in _ifelse varchar(1000),
  in _where varchar(1000),
  in _order varchar(1000)
)
/*
   call usp_Services_Paging(1,10,'*','customers','order by customerid desc');
*/
BEGIN
   if _pagesize<=1 then
      set _pagesize=20;
   end if;
   if _pagecurrent < 1 then
      set _pagecurrent = 1;
   end if;

   set @strsql = concat('select ',_ifelse,' from ',_where,' ',_order,' limit
',_pagecurrent*_pagesize-_pagesize,',',_pagesize);
   prepare stmtsql from @strsql;
   execute stmtsql;
   deallocate prepare stmtsql;

   set @strsqlcount=concat('select count(1) as count from ',_where);
   prepare stmtsqlcount from @strsqlcount;
   set SQL_SAFE_UPDATES = 0;
   execute stmtsqlcount;
   deallocate prepare stmtsqlcount;
END//
```

```
DELIMITER ;//
```

PostgreSQL

```
CREATE OR REPLACE FUNCTION fn_GetEmployeeData
(
    Paging_PageSize INTEGER = NULL,
    Paging_PageNumber INTEGER = NULL
)
RETURNS TABLE
(
    o_employeeid INTEGER,
    o_lastname CHARACTER VARYING,
    o_phone CHARACTER VARYING
) AS
$BODY$
DECLARE
    PageNumber BIGINT;
BEGIN
  /*    Construct Custom paging parameter... *** */
  IF (paging_pagesize IS NOT NULL AND paging_pagenumber IS NOT NULL) THEN
    PageNumber := (Paging_PageSize * (Paging_PageNumber-1));
  END IF;

  /*    Custome paging SQL Query construction.....**** */
  RETURN QUERY
  SELECT
     employeeid,    lastname,    phone
  FROM public.Employees
  ORDER BY employeeid
  LIMIT Paging_PageSize
  OFFSET PageNumber;

  EXCEPTION WHEN OTHERS THEN
  RAISE;
END;
$BODY$
LANGUAGE 'plpgsql';
```

007 呼叫 sp 的方法

MS-SQL

```
exec usp_Services_Paging 'Orders','OrderID',1,10;
```

MariaDB

```
call usp_Services_Paging('orders','orderid',1,10);
```

PostgreSQL

```
call usp_Services_Paging(1,10,'*','customers','order by customerid desc');
```

008 create udf 的寫法

MS-SQL

```
-- 取得指定的 Employeer 所屬的 RegionName
Create function udf_GetRegionName
(
   @EmpID int
)
/*
   Function: 傳回 Employee 所屬的地區名稱
   called:     select dbo.udf_GetRegionName(1);
*/
returns nvarchar(100)
as
begin
   declare @RegionName varchar(100);

   select Distinct @RegionName=R.RegionDescription from Employees E
   left join EmployeeTerritories ET ON E.EmployeeID = ET.EmployeeID
   left join Territories T ON T.TerritoryID = ET.TerritoryID
   left join Region R ON T.RegionID = R.RegionID
   where E.EmployeeID = @EmpID;

   return @RegionName;
end
```

MariaDB

```
DELIMITER //
Create or replace function udf_GetRegionName
(
   _EmpID int
)
/*
```

```
   Function:  傳回  Employee  所屬的地區名稱
   called:      select udf_GetRegionName(1);
*/
returns varchar(100)
begin
   declare _RegionName varchar(100);

   select Distinct R.RegionDescription into _RegionName from Employees E
   left join EmployeeTerritories ET ON E.EmployeeID = ET.EmployeeID
   left join Territories T ON T.TerritoryID = ET.TerritoryID
   left join Region R ON T.RegionID = R.RegionID
   where E.EmployeeID = _EmpID;

   return _RegionName;
END; //
DELIMITER ;
```

PostgreSQL

```
Create or replace function udf_GetRegionName
(
   _EmpID int
)
/*
   Function:  傳回  Employee  所屬的地區名稱
   called:      select udf_GetRegionName(1);
*/
RETURNS varchar(100)
AS
$BODY$
DECLARE
   _RegionName varchar(100);
begin
   select Distinct R.RegionDescription into _RegionName from Employees E
   left join EmployeeTerritories ET ON E.EmployeeID = ET.EmployeeID
   left join Territories T ON T.TerritoryID = ET.TerritoryID
   left join Region R ON T.RegionID = R.RegionID
   where E.EmployeeID = _EmpID;

   return _RegionName;
```

```
END;
$BODY$
LANGUAGE 'plpgsql';
```

009 計算工作年資(udf)

MS-SQL

```
-- 取得指定的 Employeer 所屬的 RegionName
Create function udf_GetWorkAge
(
   @EmpID int
)
/*
   Function: 傳回 Employee 的工作年資
   called:     select dbo.udf_GetWorkAge(1);
*/
returns smallint
as
begin
   declare @workage smallint;
   select
      @workage=FLOOR(DATEDIFF(DY,Hiredate,GETDATE())/365.25) WorkYears
   from Employees
   where EmployeeID = @EmpID;

   return @RegionName;
end
```

MariaDB

```
DELIMITER //
Create or replace function udf_GetWorkAge
(
   _EmpID int
)
/*
   Function: 傳回 Employee 的工作年資
   called:     select EmployeeId,Lastname,firstname,udf_GetWorkAge(EmployeeId) as WorkAge
from Employees;
*/
RETURNS smallint
```

```
BEGIN
   declare _workage smallint;

   select
      FLOOR(datediff(now(),Hiredate)/365.25) into _workage
   from Employees
   where EmployeeID = _EmpID;

   return _workage;
END; //
DELIMITER ;
```

PostgreSQL

```
Create or replace function udf_GetWorkAge
(
   _EmpID int
)
/*
   Function:  傳回  Employee  的工作年資
   called:      select EmployeeId,Lastname,firstname,udf_GetWorkAge(EmployeeId) as WorkAge
from Employees;
*/
RETURNS smallint
AS
$BODY$
DECLARE
   _workage smallint;
BEGIN
   select
      FLOOR(EXTRACT(DAY FROM NOW() - Hiredate)/365.25) into _workage
   from Employees
   where EmployeeID = _EmpID;

   return _workage;
end;
$BODY$
LANGUAGE 'plpgsql';
```

010 呼叫 udf 的方法

MS-SQL

```
-- udf 可以被 select 語法中使用
select dbo.udf_GetWorkAge(1);
-- 而 procedure 只能被 execute (或 call)
exec usp_Services_Paging 'Orders','OrderID',1,10
```

MariaDB / PostgreSQL

```
select EmployeeId,Lastname,firstname,udf_GetWorkAge(EmployeeId) as WorkAge
from Employees;
```

011 顯示所有的 sp 及 udf

MS-SQL

```
/*
   取得所有的 Table 及 View
*/
SELECT * FROM INFORMATION_SCHEMA.Tables
where table_name like '%language%'

/*
   取得所有的 SP, Trigger, udf 及 View
*/
SELECT distinct o.name ,o.xtype , c.text FROM SYS.SYSOBJECTS    o
inner join SYS.SYSCOMMENTS c on o.id =c.id
WHERE xtype<>'D' and xtype<>'C' and xtype<>'U' -- D 是顯示預設值所以不看
order by xtype,name
```

MariaDB

```
SHOW PROCEDURE STATUS;
```

PostgreSQL

```
select n.nspname as schema_name,
            p.proname as specific_name,
            l.lanname as language,
            case when l.lanname = 'internal' then p.prosrc
                  else pg_get_functiondef(p.oid)
                  end as definition,
            pg_get_function_arguments(p.oid) as arguments
from pg_proc p
left join pg_namespace n on p.pronamespace = n.oid
left join pg_language l on p.prolang = l.oid
left join pg_type t on t.oid = p.prorettype
```

```
where n.nspname not in ('pg_catalog', 'information_schema')
        -- and p.prokind = 'p'
order by schema_name,
            specific_name;
```

012 Trigger 的寫法

MS-SQL

```
-- 當 Orders 新增/刪除時，記錄到 Orders_History
CREATE TRIGGER DeleteOrdersLog on Orders
AFTER INSERT,DELETE
AS

if (Select Count(*) From inserted) > 0 and (Select Count(*) From deleted) = 0
begin
   Insert Into Orders_History
   Select 'I',* From inserted;
end

if (Select Count(*) From inserted) = 0 and (Select Count(*) From deleted) > 0
begin
   Insert Into Orders_History
   Select 'D',* From deleted;
end
```

MariaDB

```
DELIMITER //
CREATE or replace TRIGGER InsertOrdersLog
AFTER INSERT ON Orders
FOR EACH ROW
BEGIN
   Insert Into Orders_History
   Select 'I',NEW.* From NEW;
END; //
DELIMITER ;

DELIMITER //
CREATE or replace TRIGGER DeleteOrdersLog
AFTER DELETE ON Orders
FOR EACH ROW
```

```
BEGIN
   Insert Into Orders_History
   Select 'D',OLD.* From OLD;
END; //
DELIMITER ;
```

PostgreSQL

```
/* http://twpug.net/docs/postgresql-doc-8.0-zh_TW/plpgsql-trigger.html */
CREATE TRIGGER InsertDeleteOrdersLog
AFTER INSERT OR DELETE
   ON Orders
   FOR EACH ROW
   EXECUTE PROCEDURE Orders_changed();

CREATE OR REPLACE FUNCTION Orders_changed()
      RETURNS trigger AS
$$
BEGIN
   IF (TG_OP = 'INSERT') THEN
      Insert Into Orders_History
      Select 'I',NEW.* From NEW;
   else
      Insert Into Orders_History
      Select 'D',OLD.* From OLD;
   end if;

   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

013 查找所有表的行數

MS-SQL

```
select o.name, i.rows
from sysobjects o inner join sysindexes i on o.id = i.id
where i.indid = 1
order by i.rows desc
```

MariaDB

```
SHOW TABLE STATUS;
```

```
SELECT
      TABLE_NAME, ENGINE, VERSION, ROW_FORMAT, TABLE_ROWS, AVG_ROW_LENGTH,
      DATA_LENGTH, MAX_DATA_LENGTH, INDEX_LENGTH, DATA_FREE, AUTO_INCREMENT,
      CREATE_TIME, UPDATE_TIME, CHECK_TIME, TABLE_COLLATION, CHECKSUM,
      CREATE_OPTIONS, TABLE_COMMENT
   FROM INFORMATION_SCHEMA.TABLES
   WHERE table_schema = 'Northwind';

-- 察看空間
SELECT table_name, table_rows, row_format,
            data_length / 1024 AS 'data(K)',
            index_length / 1024 AS 'index(K)'
FROM INFORMATION_SCHEMA.TABLES
where row_format = 'Dynamic';
```

PostgreSQL

```
SELECT schemaname,relname,n_live_tup
   FROM pg_stat_user_tables
   ORDER BY n_live_tup DESC;

SELECT
   nspname AS schemaname,relname,reltuples
FROM pg_class C
LEFT JOIN pg_namespace N ON (N.oid = C.relnamespace)
WHERE
   nspname NOT IN ('pg_catalog', 'information_schema') AND
   relkind='r'
ORDER BY reltuples DESC;
```

014 兩張關聯表，刪除主表中已經在副表中沒有的資訊

MS-SQL / MariaDB / PostgreSQL

```
-- 先創建一筆異常資料
insert into FinalInventory(ProductID,Quantity) values(9999,9999);
-- 利用差異比對並刪除錯誤資料
delete from FinalInventory where not exists ( select * from Products where
Products.ProductID=FinalInventory.ProductID);
```

015 一條 sql 語句搞定資料庫分頁

MS-SQL

```
-- SQL2012 後
SELECT * FROM Customers
ORDER BY CustomerID
OFFSET 10 ROWS
FETCH NEXT 10 ROWS ONLY

-- SQL2008 前
SELECT *
FROM (
   SELECT ROW_NUMBER() OVER(ORDER BY CustomerID) rowNumber,*
   FROM Customers
) myTable
WHERE rowNumber>10 AND rowNumber<=20
```

MariaDB / PostgreSQL

```
SELECT * FROM Customers
ORDER BY CustomerID
LIMIT 10 OFFSET 10
```

016 選擇在每一組 b 值相同的資料中對應的 a 最大的記錄的所有資訊

MS-SQL / MariaDB / PostgreSQL

```
select * from Orders o1 where orderdate=(select max(orderdate) from Orders o2 where
o1.EmployeeID=o2.EmployeeID) order by 2
```

017 包括所有在 TableA 中但不在 TableB(TableC) 中並刪除所有重複行的表

MS-SQL / MariaDB / PostgreSQL

```
-- Insert 一筆錯誤資料
insert into FinalInventory(ProductID,Quantity) values(9999,9999);
-- 測試執行
(select ProductID from FinalInventory ) except (select ProductID from Products);
```

018 刪除重複記錄

MS-SQL

```
-- Insert Dulp Test Data
insert into DulpData Values ('test');
insert into DulpData Values ('test');
insert into DulpData Values ('test');
-- Delete Dulp Data
```

```
delete from DulpData where [DulpID] not in (select max(DulpID) from DulpData group by
DulpValue);

-- Method 2, use temp table
select distinct DulpValue into #temp from DulpData;
delete from DulpData;
insert into DulpData select * from #temp;
Drop Table #temp;
```

MariaDB / PostgreSQL

```
-- Insert Dulp Test Data
insert into DulpData Values (1,'test');
insert into DulpData Values (2,'test');
insert into DulpData Values (3,'test');

select * from DulpData;

DELETE FROM DulpData
WHERE DulpID IN
(SELECT DulpID
   FROM (SELECT DulpID,
         ROW_NUMBER() OVER (partition BY DulpValue ORDER BY DulpID) AS RowNumber
        FROM DulpData) AS T
WHERE T.RowNumber > 1);
```

019 remove New Line Character from a string

MS-SQL

```
declare @str varchar(100);
set @str = 'abc'+char(10)+'def';

select @str
union all
SELECT REPLACE(REPLACE(@str, CHAR(13), ''), CHAR(10), '')
```

MariaDB

```
select concat('aaa','\n','bbb','ccc') test
union all
select REPLACE(concat('aaa','\n','bbb','ccc'),'\n','');
```

PostgreSQL

```
select 'aaa'||chr(10)||'bbb'||'ccc'
```

```
union all
select REPLACE('aaa' || chr(10) || 'bbb' || 'ccc',chr(10),'');
```

020 利用 Trigger 寫入異動紀錄檔

MS-SQL

```
-- 當 Salary 變動時，記錄到 EmployeeHistory
CREATE TRIGGER UpdateSalaryLog on Employees
AFTER UPDATE
AS
declare @org_salary int;
declare @new_salary int;

if (Select Count(*) From inserted) > 0 and (Select Count(*) From deleted) > 0
begin
   select @new_salary=salary from inserted;
   select @org_salary=salary from deleted;

   Insert Into EmployeeHistory
   Select EmployeeId,getdate(),@org_salary,@new_salary From inserted;
end;
```

MariaDB

```
DELIMITER //
CREATE TRIGGER UpdateSalaryLog
AFTER UPDATE ON Employees
FOR EACH ROW
BEGIN
  IF NEW.salary <> OLD.salary THEN
    INSERT INTO EmployeeHistory (
              EmployeeID,
              ChangeDate,
              org_salary,
              new_salary)
          VALUES
               (OLD.EmployeeID,
                CURDATE(),
                OLD.salary,
                NEW.salary);
      END IF;
```

```
END; //
DELIMITER ;
```

PostgreSQL

```
CREATE TRIGGER UpdateSalaryLog
   AFTER UPDATE
   ON Employees
   FOR EACH ROW
   EXECUTE PROCEDURE Employees_salarychanged();

CREATE OR REPLACE FUNCTION Employees_salarychanged()
      RETURNS trigger AS
$$
BEGIN
      IF NEW.salary <> OLD.salary THEN
         INSERT INTO EmployeeHistory (
            EmployeeID,
            ChangeDate,
            org_salary,
            new_salary)
         VALUES
               (OLD.EmployeeID,
                 now(),
                 OLD.salary,
                 NEW.salary);
      END IF;
      RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

021 Wildcard samples

MS-SQL / MariaDB / PostgreSQL

```
SELECT * FROM Customers
WHERE City LIKE 'L_n_on';

SELECT * FROM Customers
WHERE City LIKE '[a-c]%';

SELECT * FROM Customers
```

```
WHERE City LIKE '%es%';
```

022 SQL Any ALL

MS-SQL / MariaDB / PostgreSQL

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY (SELECT ProductID FROM Order_Details WHERE Quantity > 99);

SELECT ProductName
FROM Products
WHERE ProductID = ALL (SELECT ProductID FROM Order_Details WHERE Quantity = 77);

SELECT ProductName
FROM Products
WHERE ProductID IN (SELECT ProductID FROM Order_Details WHERE Quantity = 77);
```

023 完整列出所有 Table 的 Schema 及 Indexs

MS-SQL

```
SELECT ORDINAL_POSITION, COLUMN_NAME, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH
        , IS_NULLABLE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'feature'

SELECT CONSTRAINT_NAME
FROM INFORMATION_SCHEMA.CONSTRAINT_TABLE_USAGE
WHERE TABLE_NAME = 'feature'

SELECT name, type_desc, is_unique, is_primary_key
FROM sys.indexes
WHERE [object_id] = OBJECT_ID('dbo.feature')
```

MariaDB

```
SELECT table_name,column_name,column_type, ordinal_position
FROM information_schema.columns where table_schema = 'northwind'
order by 1,3
```

PostgreSQL

```
SELECT
    table_name, column_name, ordinal_position, data_type, character_maximum_length
```

```
 FROM (
        SELECT n.nspname as schema,
               c.relname as name,
               CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN
'materialized view' WHEN 'i' THEN 'index' WHEN 'S' THEN 'sequence' WHEN 's' THEN 'special'
WHEN 'f' THEN 'foreign table' END as type,
               pg_catalog.pg_get_userbyid(c.relowner) as owner,
               relfilenode,
               column_name, ordinal_position, data_type, table_name, character_maximum_length
        FROM pg_catalog.pg_class c
        LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
        LEFT JOIN (
            SELECT
                  column_name, ordinal_position, data_type, table_name,
character_maximum_length
               FROM information_schema.columns
               ) D ON c.relname = D.table_name
        WHERE
               c.relkind IN ('r','s','') AND n.nspname !~ '^pg_toast' AND n.nspname ~ '^(public)$'
        ORDER BY 1,2
 ) A LEFT JOIN pg_catalog.pg_description B ON A.relfilenode = B.objoid AND B.objsubid = 0;
```

024 交易 Transaction

MS-SQL

```
--定義變數
DECLARE @M_err int,@D_err int;
declare @maxId int;

--開啟交易
BEGIN TRANSACTION;

--新增訂單表頭
INSERT INTO Orders (CustomerID,EmployeeID,OrderDate,RequiredDate,ShippedDate)
VALUES ('VINET',5,'2020-12-15','2021-01-05','2020-12-26');

--擷取錯誤代碼
select @maxId=max(OrderID) from Orders;
SELECT @M_err = @@ERROR;
```

```
--新增進貨明細
INSERT INTO order_Details (OrderID,ProductID,UnitPrice,Quantity,Discount)
VALUES (@maxId,14,18.6,5,0);

--擷取錯誤代碼
SELECT @D_err = @@ERROR;

--是否有錯?
IF @M_err = 0 AND @D_err = 0
   --ok
   COMMIT TRANSACTION;
ELSE
   --error
   ROLLBACK TRANSACTION;
```

MariaDB

https://www.itread01.com/p/1410644.html
https://www.yiibai.com/mysql/error-handling-in-stored-procedures.html

```
DELIMITER //
CREATE or replace PROCEDURE test_trans ()
BEGIN
   START TRANSACTION;
   tblock: BEGIN

   /* catch any exceptions, then rollback */
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      GET DIAGNOSTICS CONDITION 1
            @state = RETURNED_SQLSTATE,
            @rtc       = MYSQL_ERRNO,
            @rmg       = MESSAGE_TEXT;
      ROLLBACK;
   END;

   /* table transactions here */
   Insert Into department (departmentid,DepartmentName,ManagerID) Values ('T2','testOK',2);
   Insert Into department (departmentid,DepartmentName,ManagerID) Values ('A1','testErr',2);
```

```
    COMMIT;
    END tblock;

    SELECT    @rtc AS retcode, @rmg AS retmsg, 'some ret value' AS retval;

END//
DELIMITER ;
```

PostgreSQL

經案例 26 實際測試，即使沒有下 rollback，同一個 store procedure 的程式當有錯誤發生時會自動 RollBack。

025 Maths of ROW_NUMBER, RANK, DENSE_RANK

MS-SQL / MariaDB / PostgreSQL

```
CREATE TABLE tbl_numbers (name varchar(10));

INSERT INTO tbl_numbers
VALUES ('ABC'),('ABC'),('ABC'),('XYZ'),('XYZ'),('LMN'),('OPQ'),('OPQ');

SELECT
    name
    ,ROW_NUMBER() OVER(ORDER BY name) AS RowNumber
    ,RANK() OVER(ORDER BY name) AS RankNumber
    ,DENSE_RANK() OVER(ORDER BY name) AS DENSE_RankNumber
FROM tbl_numbers;

drop table tbl_numbers;
```

026 if else 語法和 try-catch 的使用

MS-SQL

```
Create PROCEDURE usp_errHandle ()
AS
/*
    exec usp_errHandle;
*/
BEGIN
    /*  測試  try catch */
    BEGIN TRY
        /* table transactions here */
```

```
    Insert Into department (departmentid,DepartmentName,ManagerID) Values ('T2','testOK',2);
    Insert Into department (departmentid,DepartmentName,ManagerID) Values ('A1','testErr',2);

  END TRY
  BEGIN CATCH
    --系統拋回訊息用
    DECLARE @ErrorMessage As VARCHAR(1000) =
      CHAR(10)+'錯誤代碼：' +CAST(ERROR_NUMBER() AS VARCHAR)
      +CHAR(10)+'錯誤訊息：'+ ERROR_MESSAGE()
      +CHAR(10)+'錯誤行號：'+ CAST(ERROR_LINE() AS VARCHAR)
      +CHAR(10)+'錯誤程序名稱：'+ ISNULL(ERROR_PROCEDURE(),'')

    DECLARE @ErrorSeverity As Numeric = ERROR_SEVERITY();
    DECLARE @ErrorState As Numeric = ERROR_STATE();
    RAISERROR( @ErrorMessage, @ErrorSeverity, @ErrorState);
  END CATCH
END
```

MariaDB

```
DELIMITER //
create procedure usp_errHandle ()
begin
  DECLARE t_error INTEGER DEFAULT 0;
  declare _maxId INTEGER;
  -- 異常時設為 1
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET t_error=1;

  START TRANSACTION;
  -- 新增訂單表頭
  INSERT INTO Orders (CustomerID,EmployeeID,OrderDate,RequiredDate,ShippedDate,ShipperID)
  VALUES (85,5,'2020-12-15','2021-01-05','2020-12-26',3);

  -- 新增進貨明細
  INSERT INTO order_Details (OrderID,ProductID,UnitPrice,Quantity,Discount)
  VALUES (_maxId,14,18.6,5,0);

  IF t_error = 1 THEN
    ROLLBACK;
  ELSE
```

```
        COMMIT;
    END IF;
end;//
DELIMITER ;
```

PostgreSQL

https://stackoverflow.com/questions/40802589/postgresql-9-5-exception-handling

```
create or replace function usp_errHandle ()
returns int as
$body$
declare
    err_context text;
begin
    /* table transactions here */
    Insert Into department (departmentid,DepartmentName,ManagerID) Values ('T2','testOK',2);
    Insert Into department (departmentid,DepartmentName,ManagerID) Values ('A1','testErr',2);

    return 0;

    exception
    when others then
        GET STACKED DIAGNOSTICS err_context = PG_EXCEPTION_CONTEXT;
            RAISE INFO 'Error Name:%',SQLERRM;
            RAISE INFO 'Error State:%', SQLSTATE;
            RAISE INFO 'Error Context:%', err_context;
        return -1;

end;
$body$
language plpgsql;
```

027 split 字串切割函式

MS-SQL

```
create function dbo.udf_split (
    @stringtosplit varchar(max) ,
    @sepchar char(1)
)
returns
    @returnlist table ([strdata] [nvarchar] (500))
```

```
as
begin
   -- define varant
   declare @strdata nvarchar(500);
   declare @pos int;

   if @sepchar = '' set @sepchar = ',';
   while charindex(@sepchar, @stringtosplit) > 0
   begin
      select @pos    = charindex(@sepchar, @stringtosplit)
      select @strdata = substring(@stringtosplit, 1, @pos-1)

      insert into @returnlist
      select @strdata

      select @stringtosplit = substring(@stringtosplit, @pos+1, len(@stringtosplit)-@pos)
   end

   insert into @returnlist
   select @stringtosplit

   return
end
```

MariaDB

```
set @str = '1,2,3,4,5,6,7,8,9';
drop table if exists t;
create table t( txt text );
insert into t values(@str);

drop temporary table if exists temp;
create temporary table temp( val char(255) );
set @sql = concat("insert into temp (val) values ('", replace(( select group_concat(distinct txt) as
data from t), ",", "'),('"),"');");
prepare stmt1 from @sql;
execute stmt1;
select val from temp;
```

PostgreSQL

```
SELECT unnest(string_to_array('1,2,3,4,5,6', ',')) as val;
```

028 View 和 Store Procedure 配合使用(訂單合計金額)

MS-SQL / MariaDB / PostgreSQL

```
create view Order_Subtotals
AS
   SELECT OD.OrderID, Round(Sum((OD.UnitPrice*Quantity*(1-Discount)/100)*100),0) AS Subtotal
   FROM Order_Details OD
   GROUP BY OD.OrderID
```

029 年度商品類別銷售金額

MS-SQL

```
CREATE PROCEDURE [dbo].[SalesByCategory]
      @CategoryName nvarchar(15), @OrdYear nvarchar(4) = '1998'
AS
IF @OrdYear != '1996' AND @OrdYear != '1997' AND @OrdYear != '1998'
BEGIN
      SELECT @OrdYear = '1998'
END

SELECT ProductName,
      TotalPurchase=ROUND(SUM(CONVERT(decimal(14,2), OD.Quantity * (1-OD.Discount) *
OD.UnitPrice)), 0)
FROM [Order_Details] OD, Orders O, Products P, Categories C
WHERE OD.OrderID = O.OrderID
      AND OD.ProductID = P.ProductID
      AND P.CategoryID = C.CategoryID
      AND C.CategoryName = @CategoryName
      AND SUBSTRING(CONVERT(nvarchar(22), O.OrderDate, 111), 1, 4) = @OrdYear
GROUP BY ProductName
ORDER BY ProductName
```

MariaDB

```
DELIMITER //
CREATE or replace PROCEDURE SalesByCategory
(
   _CategoryName varchar(20),
   _OrdYear int
)
/*
```

```
   call SalesByCategory('Beverages',2007);
*/
begin
   if (_OrdYear != 2006 AND _OrdYear != 2007 AND _OrdYear != 2008) then
      set _OrdYear = 2006;
   end if;

   SELECT ProductName,
      ROUND(SUM(OD.Quantity * (1-OD.Discount) * OD.UnitPrice), 0) TotalPurchase
   FROM Order_Details OD, Orders O, Products P, Categories C
   WHERE OD.OrderID = O.OrderID
      AND OD.ProductID = P.ProductID
      AND P.CategoryID = C.CategoryID
      AND C.CategoryName = _CategoryName
      AND DATE_FORMAT(O.OrderDate, '%Y') = _OrdYear
   GROUP BY ProductName
   ORDER BY ProductName;
end;
//
DELIMITER ;
```

PostgreSQL

```
CREATE or replace function SalesByCategory
(
   _CategoryName varchar(15),
   _OrdYear int
)
RETURNS TABLE
(
   _ProductName CHARACTER VARYING,
   _TotalPurchase decimal
)
language plpgsql
as $$
/*
   select * from SalesByCategory('Beverages',2006);
*/
declare
-- variable declaration
```

```
begin
    if (_OrdYear != 2006 AND _OrdYear != 2007 AND _OrdYear != 2008) then
        SET _OrdYear = 2006;
    end if;

    RETURN QUERY
    SELECT ProductName,
        ROUND(SUM(OD.Quantity * (1-OD.Discount) * OD.UnitPrice), 0) AS TotalPurchase
    FROM Order_Details OD, Orders O, Products P, Categories C
    WHERE OD.OrderID = O.OrderID
        AND OD.ProductID = P.ProductID
        AND P.CategoryID = C.CategoryID
        AND C.CategoryName = _CategoryName
        AND date_part('year', O.OrderDate) = _OrdYear
    GROUP BY ProductName
    ORDER BY ProductName;
end; $$
```

030 業務人員國家別銷售金額

MS-SQL

```
create procedure [dbo].[Employee_Sales_by_Country] (
    @Beginning_Date DateTime,
    @Ending_Date DateTime
)
/*
    exec Employee_Sales_by_Country '1998-01-01','1998-12-31';
*/
AS
    SELECT Employees.Country, Employees.LastName, Employees.FirstName, Orders.ShippedDate,
    Orders.OrderID, "Order_Subtotals".Subtotal AS SaleAmount
    FROM Employees INNER JOIN
        (Orders INNER JOIN "Order_Subtotals" ON Orders.OrderID = "Order_Subtotals".OrderID)
        ON Employees.EmployeeID = Orders.EmployeeID
WHERE Orders.ShippedDate Between @Beginning_Date And @Ending_Date
```

MariaDB

```
DELIMITER //
create or replace procedure Employee_Sales_by_Country
(
```

```
    _Beginning_Date Date,
    _Ending_Date Date
)
/*
    call Employee_Sales_by_Country('2006-01-01','2006-12-31');
*/
begin
    SELECT Employees.Country, Employees.LastName, Employees.FirstName, Orders.ShippedDate,
Orders.OrderID, Order_Subtotals.Subtotal AS SaleAmount
    FROM Employees INNER JOIN
        (Orders INNER JOIN Order_Subtotals ON Orders.OrderID = Order_Subtotals.OrderID)
        ON Employees.EmployeeID = Orders.EmployeeID
    WHERE Orders.ShippedDate Between _Beginning_Date And _Ending_Date;
end;
//
DELIMITER ;
```

PostgreSQL

```
create or replace function Employee_Sales_by_Country
(
    _Beginning_Date TIMESTAMP,
    _Ending_Date TIMESTAMP
)
returns table
(
    _Country CHARACTER VARYING,
    _LastName CHARACTER VARYING,
    _FirstName CHARACTER VARYING,
    _ShippedDate TIMESTAMP,
    _OrderID int,
    _SaleAmount decimal
)
language plpgsql
as $$
/*
    select * from Employee_Sales_by_Country('2006-01-01','2006-12-31');
*/
declare
-- variable declaration
```

```
begin
   RETURN QUERY
   SELECT Employees.Country, Employees.LastName, Employees.FirstName, Orders.ShippedDate,
Orders.OrderID, Order_Subtotals.Subtotal AS SaleAmount
   FROM Employees INNER JOIN
      (Orders INNER JOIN Order_Subtotals ON Orders.OrderID = Order_Subtotals.OrderID)
      ON Employees.EmployeeID = Orders.EmployeeID
   WHERE Orders.ShippedDate Between _Beginning_Date And _Ending_Date;
end; $$
```

031 計算工作天數

MS-SQL

```
CREATE TABLE WorkCalendar
(
   dt DATE PRIMARY KEY,
   IsWorkDay smallint
);

DECLARE @s DATE, @e DATE, @new DATE;
set @s = '2000-01-01';
set @e = '2049-12-31';

set @new = @s;
while @new <= @e begin
   insert into WorkCalendar values (@new, 1);
   set @new = DATEADD(DAY, 1, @new);
end;

-- weekends
UPDATE WorkCalendar SET IsWorkDay = 0
   WHERE DATEPART(WEEKDAY, dt) IN (1,7);

-- 計算天數
SELECT COUNT(*) FROM WorkCalendar
WHERE dt >= '2013-01-10'
   AND dt < '2013-01-25'
   AND IsWorkDay = 1;
```

MariaDB

```
--建立工作日曆檔
CREATE TABLE WorkCalendar
(
   dt DATE PRIMARY KEY,
   IsWorkDay smallint
);

--填入工作日曆
DELIMITER //
create or replace procedure usp_WorkCalendar ()
/*
   call usp_WorkCalendar();
*/
begin
   set @s = str_to_date('20000101', '%Y%m%d');
   set @e = str_to_date('20491231', '%Y%m%d');

   set @new = @s;
   while @new <= @e do
      insert into WorkCalendar values (@new, 1);
      set @new = DATE_ADD(@new, INTERVAL 1 DAY);
   end while;

end;
//
DELIMITER ;

-- weekends
UPDATE WorkCalendar SET IsWorkDay = 0
   WHERE DAYOFWEEK(dt) IN (1,7);

-- 計算天數
SELECT COUNT(*) FROM WorkCalendar
WHERE dt >= '2013-01-10'
   AND dt < '2013-01-25'
   AND IsWorkDay = 1;
```

PostgreSQL

```
/* step 1: create table */
```

```
CREATE TABLE WorkCalendar
(
   dt DATE PRIMARY KEY,
   IsWorkDay smallint
);

/* step 2: prepare work calendar , sun& sat not write*/
do $$
DECLARE
   _s DATE;
   _e DATE;
   _new DATE;
begin
   select '2000-01-01' into _s;
   select '2049-12-31' into _e;

   select _s into _new;
   while _new <= _e loop
      insert into WorkCalendar values (_new, 1);
      select _new + INTERVAL '1 day' into _new;
   end loop;

   -- weekends
   UPDATE WorkCalendar SET IsWorkDay = 0
      WHERE EXTRACT(DOW FROM dt) IN (6,0);

end; $$

/* step 3: 計算天數*/
SELECT COUNT(*) FROM WorkCalendar
WHERE dt >= '2013-01-10'
   AND dt < '2013-01-25'
   AND IsWorkDay = 1;
```

032 人員組織圖

MS-SQL

```
WITH OrgSpread AS
(
```

```
    SELECT
        EmployeeID , FirstName , ReportsTo ,
        CAST(NULL AS VARCHAR(100)) AS Manager ,
        0 AS lvl ,
        CAST(FirstName AS VARCHAR(100)) AS OrgTree ,
        CAST(FirstName AS VARCHAR(100)) AS OrgTreePath
    FROM Employees
    WHERE ReportsTO IS NULL
    UNION ALL
    SELECT
        E.EmployeeID , E.FirstName , E.ReportsTO ,
        CAST(T.FirstName AS VARCHAR(100)),
        lvl + 1 ,
        CAST(REPLICATE(SPACE(4),lvl + 1) + E.FirstName AS VARCHAR(100)) ,
        CAST(T.OrgTreePath + '_' + E.FirstName AS VARCHAR(100))
    FROM OrgSpread AS T
    inner JOIN Employees AS E ON T.EmployeeID = E.ReportsTo
)
SELECT * FROM OrgSpread ORDER BY OrgTreePath;
```

MariaDB

```
WITH RECURSIVE OrgSpread AS
(
    SELECT
        EmployeeID , FirstName , MgrId ,
        CAST(NULL AS CHAR) AS Manager ,
        0 AS lvl ,
        CAST(FirstName AS CHAR) AS OrgTree ,
        CAST(FirstName AS CHAR) AS OrgTreePath
    FROM Employees
    WHERE MgrId IS NULL
    UNION ALL
    SELECT
        E.EmployeeID , E.FirstName , E.MgrId ,
        CAST(T.FirstName AS CHAR),
        lvl + 1 ,
        CAST(REPEAT(SPACE(4),lvl + 1) + E.FirstName AS CHAR) ,
        CAST(T.OrgTreePath + '_' + E.FirstName AS CHAR)
    FROM OrgSpread AS T
```

```
    left JOIN Employees AS E ON T.EmployeeID = E.MgrId
)
SELECT * FROM OrgSpread ORDER BY OrgTreePath;
```

PostgreSQL

```
WITH RECURSIVE OrgSpread AS
(
    SELECT
        EmployeeID , FirstName , MgrId ,
        CAST(NULL AS text) AS Manager ,
        0 AS lvl ,
        CAST(FirstName AS text) AS OrgTree ,
        CAST(FirstName AS text) AS OrgTreePath
    FROM Employees
    WHERE MgrId IS NULL
    UNION ALL
    SELECT
        E.EmployeeID , E.FirstName , E.MgrId ,
        CAST(T.FirstName AS text),
        lvl + 1 ,
        CAST(repeat('        ',lvl + 1) || E.FirstName AS CHAR) ,
        CAST(T.OrgTreePath || '_' || E.FirstName AS CHAR)
    FROM OrgSpread AS T
    inner JOIN Employees AS E ON T.EmployeeID = E.MgrId
)
SELECT * FROM OrgSpread ORDER BY OrgTreePath;
```

034 顯示資料庫中所有 Table 得筆數

MS-SQL

```
--方法一
select o.name, i.rows
from sysobjects o inner join sysindexes i on o.id = i.id
where i.indid = 1
order by i.rows desc;

--方法二
SELECT O.NAME '資料表名稱', P.ROWS '列總數'
FROM SYS.OBJECTS O INNER JOIN SYS.SCHEMAS S
ON O.SCHEMA_ID = S.SCHEMA_ID
```

```
INNER JOIN SYS.PARTITIONS P
ON O.OBJECT_ID = P.OBJECT_ID
WHERE (O.TYPE = 'U') AND
(P.INDEX_ID IN (0,1))
ORDER BY S.NAME, O.NAME ASC;
```

MariaDB

```
Select
   -- Sort the tables by count
   concat(
      'select * from (',
      -- Aggregate rows into a single string connected by unions
      group_concat(
         -- Build a "select count(1) from db.tablename" per table
         concat('select ',
            quote(db), ' db, ',
            quote(tablename), ' tablename, '
            'count(1) "rowcount" ',
            'from ', db, '.', tablename)
         separator ' union ')
      , ') t order by 3 desc')
into @sql
from (
   select
      table_schema db,
      table_name tablename
   from information_schema.tables
   where table_schema not in
      ('performance_schema', 'mysql', 'information_schema')
) t;

-- Execute @sql
prepare s from @sql; execute s; deallocate prepare s;
```

PostgreSQL

```
SELECT relname, reltuples
FROM pg_class r JOIN pg_namespace n
ON (relnamespace = n.oid)
WHERE relkind = 'r' AND n.nspname = 'public';
```

035 計算去掉最高收入及最低收入後，公司員工平均收入是多少

MS-SQL / MariaDB / PostgreSQL

```
select avg(salary) from employees t
where
salary not in    (
   select max(salary) from employees
union
   select min(salary) from employees
)
```

036 某年不同月份的客戶下單次數

MS-SQL / MariaDB / PostgreSQL

```
select
   month(orderdate) as dtmonth,
   count(distinct customerid) as count_users
from orders
where year(orderdate) = 1998
group by month(orderdate)
```

037 for-loop 及 while-loop 的應用

MS-SQL

無 loop 類的語法，僅能使用 while loop

MariaDB

```
delimiter //
CREATE procedure repeat_loop_example()
/*
   call repeat_loop_example();
*/
BEGIN
   DECLARE x INT;
   DECLARE str VARCHAR(255);
   SET x = 5;
   SET str = '';

   REPEAT
      SET str = CONCAT(str,x,',');
      SET x = x - 1;
```

```
    UNTIL x <= 0
  END REPEAT;

  SELECT str;
END//
```

PostgreSQL

```
do $$
declare
    n integer:= 10;
    counter integer := 0 ;
    str text;
begin
     str := '';
     loop
          exit when counter = n ;
          counter := counter + 1 ;
          str := str || cast(counter as text) || ',' ;
     end loop;

     raise notice 'message: %',str;
end; $$
```

038 位元運算子(AND OR 語法)

MS-SQL / MariaDB / PostgreSQL

```
/*
位元運算子
15     8        2
1111 1000    0010

*/
select 15 & 2 ==> 2
select 15 & 8 ==> 8
select 8 & 2 ==> 0
```

039 Get the data difference between two Tables

MS-SQL / PostgreSQL

```
CREATE TABLE cmp_A (F1 varchar(10));
CREATE TABLE cmp_B (F2 varchar(10));
```

```
INSERT INTO cmp_A VALUES ('A'),('B'),('C');
INSERT INTO cmp_B VALUES ('A'),('D'),('X');

SELECT
   a.F1 as cmp_B_Null
   ,b.F2 as cmp_A_Null
FROM cmp_A a
FULL OUTER JOIN cmp_B b    ON a.F1 = b.F2
WHERE a.F1 IS NULL OR b.F2 IS NULL;
--PostgreSQL 要分開執行
Drop Table cmp_A;
Drop Table cmp_B;
```

MariaDB

```
--不支援 FULL OUTER JOIN, 改用 left + right UNION
CREATE TABLE cmp_A (F1 varchar(10));
CREATE TABLE cmp_B (F2 varchar(10));
INSERT INTO cmp_A VALUES ('A'),('B'),('C');
INSERT INTO cmp_B VALUES ('A'),('D'),('X');

SELECT
   a.F1 as cmp_B_Null
   ,b.F2 as cmp_A_Null
FROM cmp_A a
LEFT JOIN cmp_B b    ON a.F1 = b.F2
WHERE a.F1 IS NULL OR b.F2 IS NULL
UNION
SELECT
   a.F1 as cmp_B_Null
   ,b.F2 as cmp_A_Null
FROM cmp_A a
RIGHT JOIN cmp_B b    ON a.F1 = b.F2
WHERE a.F1 IS NULL OR b.F2 IS NULL;

Drop Table cmp_A;
Drop Table cmp_B;
```

040 考勤資料異常分析

1.結合工作日曆(請參考中階例 31)

2.找出異常出勤資料，包括
-EmployeeID in (1,2,3) and 2020-11-7 2020-11-15 間的日期
-遲到、早退(09:00-18:00)
-無打卡資料的異常出勤

MS-SQL

```
with wd
as
(
   select * from WorkCalendar where dt >= '2020-11-7'
   and dt <= '2020-11-15' and IsWorkDay = 1
), emp_wd
as
(
   select EmployeeId,dt,'WON' ClockType from Employees,wd
   where EmployeeId in (1,2,3)
   union
   select EmployeeId,dt,'WOFF' ClockType from Employees,wd
   where EmployeeId in (1,2,3)
)

select
   e.EmployeeID,e.dt,e.ClockType,h.ClockDate,
   case
      when h.clockDate is null then 'NO-Work'
      when convert(varchar(5), ClockDate, 114) >= '09:00'
         and e.ClockType = 'WON' then 'ON-delay'
      when convert(varchar(5), ClockDate, 114) <= '18:00'
         and e.ClockType = 'WOFF' then 'OFF-early'
      else 'OK' end ClockStatus
from emp_wd e left join hr_attendance h ON e.EmployeeID = h.EmployeeID
and e.dt = CONVERT(char(10), h.ClockDate, 20) and e.ClockType = h.ClockType
order by 1,2,e.ClockType desc;
```

MariaDB

```
with wd
as
(
   select * from WorkCalendar where dt >= '2020-11-7'
   and dt <= '2020-11-15' and IsWorkDay = 1
```

```
), emp_wd
as
(
   select EmployeeId,dt,'WON' ClockType from Employees,wd
   where EmployeeId in (1,2,3)
   union
   select EmployeeId,dt,'WOFF' ClockType from Employees,wd
   where EmployeeId in (1,2,3)
)

select
   e.EmployeeID,e.dt,e.ClockType,h.ClockDate,
   case
      when h.clockDate is null then 'NO-Work'
      when date_format(ClockDate, '%H:%i') >= '09:00'
         and e.ClockType = 'WON' then 'ON-delay'
      when date_format(ClockDate, '%H:%i') <= '18:00'
         and e.ClockType = 'WOFF' then 'OFF-early'
      else 'OK' end ClockStatus
from emp_wd e left join hr_attendance h ON e.EmployeeID = h.EmployeeID
and e.dt = date_format(ClockDate, '%Y-%m-%d') and e.ClockType = h.ClockType
order by 1,2,e.ClockType desc;
```

PostgreSQL

```
with wd
as
(
   select * from WorkCalendar where dt >= '2020-11-7'
   and dt <= '2020-11-15' and IsWorkDay = 1
), emp_wd
as
(
   select EmployeeId,dt,'WON' ClockType from Employees,wd
   where EmployeeId in (1,2,3)
   union
   select EmployeeId,dt,'WOFF' ClockType from Employees,wd
   where EmployeeId in (1,2,3)
)
```

```
select
   e.EmployeeID,e.dt,e.ClockType,h.ClockDate,
   case
      when h.clockDate is null then 'NO-Work'
      when to_char(ClockDate, 'HH24:MI') >= '09:00'
         and e.ClockType = 'WON' then 'ON-delay'
      when to_char(ClockDate, 'HH24:MI') <= '18:00'
         and e.ClockType = 'WOFF' then 'OFF-early'
      else 'OK' end ClockStatus
from emp_wd e left join hr_attendance h ON e.EmployeeID = h.EmployeeID
and to_char(e.dt, 'yyyy-MM-dd') = to_char(ClockDate, 'yyyy-MM-dd') and e.ClockType =
h.ClockType
order by 1,2,e.ClockType desc;
```

041 如何跨資料庫取得資料

```
create database testDB;
use testDB;
create table test_Invoice (
   InvoiceDate date,
   InvoiceNo varchar(12),
   SupplierID int,
   InvoiceAmount int,
   TaxAmount int,
   primary key(InvoiceNo)
);

Insert Into test_Invoice (InvoiceDate,InvoiceNo,SupplierID,InvoiceAmount,TaxAmount)
Values ('2020-11-11','XX12345678',1,100,5),
('2020-11-15','XX12345666',3,50,3),
('2020-11-21','XX64345678',3,500,25),
('2020-11-30','XX12337678',5,100,5);
```

MS-SQL

```
use northwind;
select S.SupplierID,S.CompanyName,isnull(X.Tax,0) TaxAmt
from Suppliers S
left Join (
   Select SupplierID,Sum(TaxAmount) Tax From testdb.dbo.test_Invoice Group By SupplierID
) X ON S.SupplierID = X.SupplierID
```

MariaDB

```
use northwind;
select S.SupplierID,S.CompanyName,ifnull(X.Tax,0) TaxAmt
from Suppliers S
left Join (
   Select SupplierID,Sum(TaxAmount) Tax From testdb.test_Invoice Group By SupplierID
) X ON S.SupplierID = X.SupplierID
```

PostgreSQL

```
--安裝
CREATE EXTENSION dblink

Select * Into temp table tmpOD
From dblink('hostaddr=127.0.0.1 port=5432 dbname=testdb user=postgres password=zaqwedcxs',
'Select SupplierID,Sum(TaxAmount) Tax From test_Invoice Group By SupplierID;')
AS T1(SupplierID integer, Tax integer);

select
   S.SupplierID,S.CompanyName,COALESCE(X.Tax,0) TaxAmt
From Suppliers S
left Join tmpOD X ON S.SupplierID = X.SupplierID;

Drop Table tmpOD;
```

gamma (高級)
001 取代 **cursor** 的 **while-loop** 寫法 **and Add if-else Code**
002 累計加總**(Running Total)**
003 指定的部門其員工薪資的中位數
004 練習多 **SubQuery** 及 **case when** 語法
005 Schema Compare between two Table
006 使用 **PIVOT** 扭轉資料，由直列轉為橫向資料
007 資料橫轉直
008 新增腳色**(Role)**
009 DB Backup
010 DB Restore
011 操作 **xml**
012 操作 **json**
013 BOM 的正展開
014 Replace String data in all the Columns of a Table
015 將包含關鍵字內容的所有資料表及所有欄位，全部搜尋出來

等級：gamma (高級)

001 取代 cursor 的 while-loop 寫法 and Add if-else Code

MS-SQL

```
-- 逐筆計算訂單數量, 顯示每天的累計產品訂單數量
declare @maxcount int;
declare @currid int;
declare @nextrowid int;
declare @finalqty int;
declare @nowqty int;
declare @pre_prod int;
declare @now_prod int;

-- 取得訂單明細的產品日期合計
select
   OD.ProductID,O.OrderDate,sum(OD.Quantity) SumQty,0 FinalQty,
   ROW_NUMBER() OVER(ORDER BY ProductID,OrderDate) AS ROWID into #tmp1
from Order_Details OD
left join Orders O on O.OrderID = OD.OrderID
group by OD.ProductID,O.OrderDate;

select @maxcount=count(*) from #tmp1;
select @currid=min(rowid) from #tmp1;

--init
set @finalqty = 0;
set @pre_prod = '';
while (@currid <= @maxcount)
begin
   select @now_prod=ProductID,@nowqty=sumqty from #tmp1 where rowid = @currid
   --計算餘額(by productID)
   if @now_prod <> @pre_prod
      set @finalqty = @nowqty;
   else
      set @finalqty = @finalqty + @nowqty;

   --更新
```

```
    update #tmp1 set finalqty=@finalqty where rowid = @currid;

    set @pre_prod = @now_prod;
    -- 取得下一筆資料
    select @nextrowid = min(rowid) from #tmp1 where rowid > @currid;
    set @currid=@nextrowid;
end

select * from #tmp1;
drop table #tmp1;
```

MariaDB

```
DELIMITER //
create or replace procedure usp_calc_dailyqty (
    _startdate date,
    _enddate date
)
/*
select * from usp_calc_dailyqty('2006-07-01','2006-07-31');
*/
begin
    declare _maxcount int;
    declare _currid int;
    declare _nextrowid int;
    declare _finalqty int;
    declare _nowqty int;
    declare _pre_prod int;
    declare _now_prod int;

    DROP TEMPORARY TABLE IF EXISTS _tmp1;
    create TEMPORARY table _tmp1 (ProductID integer, OrderDate Date, SumQty integer, FinalQty
integer, ROWID integer);

    -- 取得訂單明細的產品日期合計
    insert into _tmp1
    select
        OD.ProductID,O.OrderDate,sum(OD.Quantity) SumQty,0 FinalQty,
        ROW_NUMBER() OVER(ORDER BY ProductID,OrderDate) AS ROWID
    from Order_Details OD
```

```
    left join Orders O on O.OrderID = OD.OrderID
    where orderdate >= _startdate and orderdate <= _enddate
    group by OD.ProductID,O.OrderDate;

    select count(*) into _maxcount from _tmp1;
    select min(rowid) into _currid from _tmp1;

    -- init
    select 0 into _finalqty;
    select 0 into _pre_prod;
    while (_currid <= _maxcount)
    do
        select ProductID,sumqty into _now_prod,_nowqty from _tmp1 where rowid = _currid;
        -- 計算餘額(by productID)
        if _now_prod <> _pre_prod then
            set _finalqty = _nowqty;
        else
            set _finalqty = _finalqty + _nowqty;
        end if;

        -- 更新
        update _tmp1 set finalqty=_finalqty where rowid = _currid;

        select _now_prod into _pre_prod;
        -- 取得下一筆資料
        select min(rowid) into _nextrowid from _tmp1 where rowid > _currid;
        select _nextrowid into _currid;
    end while;

    select * from _tmp1;
    drop table _tmp1;

end
//
DELIMITER ;
```

PostgreSQL

```
-- 逐筆計算訂單數量, 顯示每天的累計產品訂單數量(執行會報錯)
CREATE OR REPLACE function usp_calc_dailyqty (
```

```
    _startdate date,
    _enddate date
)
/*
select * from usp_calc_dailyqty('2006-07-01','2006-07-31');
*/
RETURNS TABLE
(
    _ProductID int,
    _OrderDate date,
    _SumQty int,
    _FinalQty int,
    _ROWID int
)
language plpgsql
as $$
DECLARE
    _maxcount integer;
    _currid integer;
    _nextrowid int;
    _finalqty int := 0;
    _nowqty int;
    _pre_prod int := 0;
    _now_prod int;
BEGIN
    -- create Template
    DROP TABLE IF EXISTS _tmp1;
    create TEMPORARY table _tmp1 (ProductID int, OrderDate date, SumQty int, FinalQty int,
ROWID int);

    -- 取得訂單明細的產品日期合計
    insert into _tmp1
    select
        OD.ProductID,O.OrderDate,sum(OD.Quantity) SumQty,0 FinalQty,
        ROW_NUMBER() OVER(ORDER BY ProductID,OrderDate) AS rowid
    from Order_Details OD
    left join Orders O on O.OrderID = OD.OrderID
    where orderdate >= _startdate and orderdate <= _enddate
```

```
    group by OD.ProductID,O.OrderDate;

    select count(*) into _maxcount from _tmp1;    -- 4212
    select min(rowid) into _currid from _tmp1;

    -- init
    while (_currid <= _maxcount) loop
        select ProductID,sumqty into _now_prod,_nowqty from _tmp1 where rowid = _currid;
        --計算餘額(by productID)
        if (_now_prod <> _pre_prod) then
            _finalqty := _nowqty;
        else
            _finalqty := _finalqty + _nowqty;
        end if;

        --更新
        update _tmp1 set finalqty=_finalqty where rowid = _currid;

        select _now_prod into _pre_prod;
        --  取得下一筆資料
        select min(rowid) into _nextrowid from _tmp1 where rowid > _currid;
        select _nextrowid into _currid;
    end loop;

    RETURN QUERY
    select * from _tmp1;

END;
$$
```

002 累計加總(Running Total)

MS-SQL / MariaDB / PostgreSQL

```
SELECT
    O.OrderDate,OD.ProductID,OD.Quantity
  ,SUM(Quantity) OVER(PARTITION BY ProductId ORDER BY ProductId,O.OrderDate)    Acc_Qty
  ,SUM(Quantity) OVER(PARTITION BY ProductId) Total_Qty
FROM Order_Details OD
left join Orders O ON O.OrderID = OD.OrderID
```

```
order by OD.ProductID,O.OrderDate;
```

003 指定的部門其員工薪資的中位數

MS-SQL

```
select
   avg(salary)
from (
   select salary,
      count(*) over() total,
      cast(count(*) over() as decimal)/2 mid,
      ceiling(cast(count(*)over() as decimal)/2) next,
      row_number() over(order by salary) rn
   from Employees
   where DepartmentID = 'S1'
) x
where (
   total%2 = 0 and rn in ( mid, mid+1 )
         )
   or ( total%2 = 1 and rn = next
         )
```

MariaDB / PostgreSQL

```
select
   avg(x.salary)
from (
   select e.salary
     from employees e, employees d
    where e.departmentid = d.departmentid
      and e.departmentid = 'A1'
    group by e.salary
   having sum(case when e.salary = d.salary then 1 else 0 end)
                  >= abs(sum(sign(e.salary - d.salary)))
      ) x;
```

004 練習多 SubQuery 及 case when 語法

MS-SQL / MariaDB / PostgreSQL

```
-- Region 業績合計(不管邏輯是否完美, 僅測試 SQL 指令)
select
   case RegionID when 1 then 'Eastern'
```

```
        when 2 then 'Western'
            when 2 then 'Northern'
            when 2 then 'Southern'
    else 'Others'
    end RegionName,sum(Amount) RegionAmt
from
(
select Distinct O.OrderID,OD.Amount,T.RegionID from Orders O
left join
(
    select OrderID,sum(UnitPrice*Quantity) Amount From Order_Details
    Group by OrderID
) OD on O.OrderID = OD.OrderID
left join Employees E ON O.EmployeeID = E.EmployeeID
left join EmployeeTerritories ET ON E.EmployeeID = ET.EmployeeID
left join Territories T ON T.TerritoryID = ET.TerritoryID
) X
Group by RegionID
```

005 Schema Compare between two Table

MS-SQL

```
SELECT
        tbl_A.name as tbl_A_ColumnName
        ,tbl_B.name as tbl_B_ColumnName
        ,tbl_A.system_type_name as tbl_A_Datatype
        ,tbl_B.system_type_name as tbl_B_Datatype
        ,tbl_A.is_identity_column as tbl_A_is_identity
        ,tbl_B.is_identity_column as tbl_B_is_identity
        ,tbl_A.is_nullable as tbl_A_is_nullable
        ,tbl_B.is_nullable as tbl_B_is_nullable
FROM sys.dm_exec_describe_first_result_set (N'SELECT * FROM Customers', NULL, 0) tbl_A
FULL OUTER JOIN    sys.dm_exec_describe_first_result_set (N'SELECT * FROM Suppliers', NULL, 0)
tbl_B
ON tbl_A.name = tbl_B.name;
```

MariaDB

```
SELECT column_name,ordinal_position,data_type,column_type FROM
(
        SELECT
```

```
            column_name,ordinal_position,
            data_type,column_type,COUNT(1) rowcount
        FROM information_schema.columns
        WHERE table_schema=DATABASE()
        AND table_name IN ('Customers','Suppliers')
        GROUP BY
            column_name,ordinal_position,
            data_type,column_type
        HAVING COUNT(1)=1
) A;
```

PostgreSQL

```
select COALESCE(c1.table_name, c2.table_name) as table_name,
            COALESCE(c1.column_name, c2.column_name) as table_column,
            c1.column_name as schema1,
            c2.column_name as schema2
from
        (select table_name,
                    column_name
          from information_schema.columns c
          where c.table_name = 'customers') c1
full join
                (select table_name,
                            column_name
                  from information_schema.columns c
                  where c.table_name = 'suppliers') c2
on c1.column_name = c2.column_name
where c1.column_name is null
          or c2.column_name is null
order by table_name, table_column;
```

006 使用 PIVOT 扭轉資料，由直列轉為橫向資料

MS-SQL

```
-- create table
CREATE TABLE Lang (
   CodeId int NOT NULL,
   LangType VARCHAR(20) NOT NULL,
   ShowText NVARCHAR(400) NULL,
   Primary Key (CodeId,LangType)
```

```
);

INSERT Lang (CodeId, [LangType], [ShowText]) VALUES (1, N'en-US', N'Link to company type');
INSERT Lang (CodeId, [LangType], [ShowText]) VALUES (1, N'zh-CN', N'连结厂商型态');
INSERT Lang (CodeId, [LangType], [ShowText]) VALUES (1, N'zh-TW', N'連結廠商型態');
INSERT Lang (CodeId, [LangType], [ShowText]) VALUES (2, N'en-US', N'Export');
INSERT Lang (CodeId, [LangType], [ShowText]) VALUES (2, N'zh-CN', N'汇出');
INSERT Lang (CodeId, [LangType], [ShowText]) VALUES (2, N'zh-TW', N'匯出');
INSERT Lang (CodeId, [LangType], [ShowText]) VALUES (3, N'en-US', N'Product Name');
INSERT Lang (CodeId, [LangType], [ShowText]) VALUES (3, N'zh-CN', N'产品名称');
INSERT Lang (CodeId, [LangType], [ShowText]) VALUES (3, N'zh-TW', N'產品名稱');

SELECT *
FROM (
   SELECT l.CodeId, l.LangType, l.ShowText
   FROM Lang l
) t
PIVOT (
   -- 設定彙總欄位及方式
   MAX(ShowText)
   -- 設定轉置欄位，並指定轉置欄位中需彙總的條件值作為新欄位
   FOR LangType IN ([zh-TW], [zh-CN], [en-US])
) p;
```

MariaDB

```
-- create table
CREATE TABLE Lang (
   CodeId int NOT NULL,
   LangType VARCHAR(20) NOT NULL,
   ShowText VARCHAR(400) NULL,
   Primary Key (CodeId,LangType)
);

INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (1, 'en-US', 'Link to company type');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (1, 'zh-C', '连结厂商型态');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (1, 'zh-TW', '連結廠商型態');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (2, 'en-US', 'Export');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (2, 'zh-C', '汇出');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (2, 'zh-TW', '匯出');
```

```
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (3, 'en-US', 'Product Name');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (3, 'zh-C', '产品名称');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (3, 'zh-TW', '產品名稱');

SELECT
   CodeId,
   MAX(CASE WHEN LangType = "en-US" THEN ShowText END) "US",
   MAX(CASE WHEN LangType = "zh-CN" THEN ShowText END) "CN",
   MAX(CASE WHEN LangType = "zh-TW" THEN ShowText END) "TW"
FROM Lang
GROUP BY CodeId
ORDER BY CodeId ASC;
```

PostgreSQL

```
-- create table
CREATE TABLE Lang (
   CodeId int NOT NULL,
   LangType text NOT NULL,
   ShowText text NULL,
   Primary Key (CodeId,LangType)
);

INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (1, 'en-US', 'Link to company type');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (1, 'zh-CN', '连结厂商型态');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (1, 'zh-TW', '連結廠商型態');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (2, 'en-US', 'Export');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (2, 'zh-CN', '汇出');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (2, 'zh-TW', '匯出');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (3, 'en-US', 'Product Name');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (3, 'zh-CN', '产品名称');
INSERT INTO Lang (CodeId, LangType, ShowText) VALUES (3, 'zh-TW', '產品名稱');

SELECT *
FROM CROSSTAB
(
   'SELECT CodeId, LangType, ShowText
     FROM Lang
     ORDER BY 1,2'
) AS (CodeId int, US text, CN text, TW text);
```

007 資料橫轉直

MS-SQL

```
CREATE TABLE #UNPIVOT (
   EmployeeID int,
   L01 decimal(4,2), -- 事假
   L02 decimal(4,2), -- 病假
   L03 decimal(4,2), -- 公假
   L04 decimal(4,2), -- 特休
   L05 decimal(4,2) -- 陪產假
);

INSERT INTO #UNPIVOT VALUES(1,15,9,24,2,NULL);
INSERT INTO #UNPIVOT VALUES(2,6,16,0,3,24);



SELECT *
FROM
      (
         Select EmployeeID,L01,L02,L03,L04,L05 FROM #UNPIVOT
      ) AS p
UNPIVOT
      (
         Hours FOR Kind IN (L01,L02,L03,L04,L05)
      ) AS pv

drop table #UNPIVOT;
```

MariaDB

```
DROP TEMPORARY TABLE IF EXISTS _UNPIVOT;
CREATE TEMPORARY TABLE _UNPIVOT (
   EmployeeID int,
   L01 decimal(4,2), -- 事假
   L02 decimal(4,2), -- 病假
   L03 decimal(4,2), -- 公假
   L04 decimal(4,2), -- 特休
   L05 decimal(4,2) -- 陪產假
);
```

```
INSERT INTO _UNPIVOT VALUES(1,15,9,24,2,NULL);
INSERT INTO _UNPIVOT VALUES(2,6,16,0,3,24);

select EmployeeID,'L01' as LeaveType,L01 as LeaveDay From _UNPIVOT
union all
select EmployeeID,'L02',L02 as LeaveDay From _UNPIVOT
union all
select EmployeeID,'L03',L03 as LeaveDay From _UNPIVOT
union all
select EmployeeID,'L04',L04 as LeaveDay From _UNPIVOT
union all
select EmployeeID,'L05',L05 as LeaveDay From _UNPIVOT;
```

PostgreSQL

```
DROP TABLE IF EXISTS _UNPIVOT;
CREATE TEMPORARY TABLE _UNPIVOT (
   EmployeeID int,
   L01 decimal(4,2), -- 事假
   L02 decimal(4,2), -- 病假
   L03 decimal(4,2), -- 公假
   L04 decimal(4,2), -- 特休
   L05 decimal(4,2) -- 陪產假
);

INSERT INTO _UNPIVOT VALUES(1,15,9,24,2,NULL);
INSERT INTO _UNPIVOT VALUES(2,6,16,0,3,24);

select EmployeeID,'L01' as LeaveType,L01 as LeaveDay From _UNPIVOT
union all
select EmployeeID,'L02',L02 as LeaveDay From _UNPIVOT
union all
select EmployeeID,'L03',L03 as LeaveDay From _UNPIVOT
union all
select EmployeeID,'L04',L04 as LeaveDay From _UNPIVOT
union all
select EmployeeID,'L05',L05 as LeaveDay From _UNPIVOT;
```

008 新增腳色(Role)

MS-SQL

--創建帳號
CREATE LOGIN [Account] WITH PASSWORD=N'Password', DEFAULT_DATABASE=[DBName], DEFAULT_LANGUAGE=[Traditional Chinese], CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
GO

--帳號啟用
ALTER LOGIN [Account] EnABLE
GO

--授權使用者登入[DBName]
USE [DBName]
GO
CREATE USER [UserName] FOR LOGIN [Account];
GO

--授權使用者在[DBName]裡的角色
USE [DBName]
GO
ALTER ROLE [db_owner] ADD MEMBER [UserName]
GO

MariaDB

-- 新增使用者，設定密碼
CREATE USER 'my_user'@'localhost' IDENTIFIED BY 'my_password';
-- 設定使用者權限
GRANT ALL PRIVILEGES ON my_db.* TO 'my_user'@'localhost';

PostgreSQL

CREATE ROLE kylin SUPERUSER NOCREATEDB NOCREATEROLE NOINHERIT LOGIN PASSWORD 'kylin';

009 DB Backup

MS-SQL

BACKUP DATABASE [Northwind] TO DISK=N'D:\db_backup\Northwind-20201218.bak' WITH NOFORMAT, INIT, NAME=N'Northwind-Full Database Backup', NOSKIP, REWIND, NOUNLOAD, STATS=10, CHECKSUM, CONTINUE_AFTER_ERROR

MariaDB

mysqldump -h hostname -u root -p database_name > backup.sql

sample:
cd C:\Program Files\MariaDB 10.3\bin

mysqldump -h localhost -u root -p Northwind > Northwind_bp.sql
ps: 檔案會直接產生在 bin 目錄中

PostgreSQL

pg_dump dbname > outfile

sample: (請注意大小寫)
cd C:\Program Files\PostgreSQL\13\bin
pg_dump -U kylin northwind > Northwind-bp-20201218.backup.sql

010 DB Restore

MS-SQL

```
RESTORE DATABASE [vuebms] FROM DISK=N'C:\data\vuebms-20201220.bak' WITH MOVE
N'Northwind' TO N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\vuebms.mdf',    MOVE N'Northwind_log' TO
N'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\vuebms.LDF',
NOUNLOAD, REPLACE, STATS=10
```

MariaDB

mysql -u root -p database_name < backup.sql

sample:
cd C:\Program Files\MariaDB 10.3\bin
mysql -u root -p Northwind < Northwind_bp.sql

PostgreSQL

psql dbname < infile

sample:
cd C:\Program Files\PostgreSQL\13\bin
psql n-U kylin northwind < Northwind-bp-20201218.backup.sql

011 操作 xml

-讀取 xml 檔(僅支援特定格式如下例)，並轉為 Table 格式顯示

MS-SQL

```
declare @x xml;
set @x=
'<xml>
   <book id="1" title="C# Program" author="David" price="21" />
   <book id="2" title="Python" author="Michael" price="100" />
```

```
</xml>';

select x.value(N'@id', N'int') as BookId
   , x.value(N'@title',N'nvarchar(100)') as title
   , x.value(N'@author', N'nvarchar(100)') as author
   , x.value(N'@price', N'int') as price
from @x.nodes(N'/xml/book') t(x);
```

MariaDB

MySQL 和 MariaDB 對於 xml 的處理方式，多是採用直接匯入 Table 的寫法

```
CREATE    TABLE book (
      `id`          INT                  NOT NULL ,
      `title`    VARCHAR(100) NOT NULL ,
      `author` VARCHAR(100) NULL ,
      `price` int ,
PRIMARY KEY (`id`) );

DELIMITER //
create or replace procedure import_book_xml (
   path varchar(255), node varchar(255)
)
/*
   call import_applicants_xml('C:\\test-book.xml', '/xml/book');
*/
BEGIN
      declare xml_content text;
      declare v_row_index int unsigned default 0;
      declare v_row_count int unsigned;
      declare v_xpath_row varchar(255);

      set xml_content = load_file(path);

      -- calculate the number of row elements.
      set v_row_count    = extractValue(xml_content, concat('count(', node, ')'));

      -- loop through all the row elements
      while v_row_index < v_row_count do
            set v_row_index = v_row_index + 1;
            set v_xpath_row = concat(node, '[', v_row_index, ']/@*');
```

```
            insert into book values (
                extractValue(xml_content, concat(v_xpath_row, '[1]')),
                extractValue(xml_content, concat(v_xpath_row, '[2]')),
                extractValue(xml_content, concat(v_xpath_row, '[3]')),
                extractValue(xml_content, concat(v_xpath_row, '[4]'))
            );
        end while;
END
//
DELIMITER ;
```

PostgreSQL

```
with t(x) as (values('<?xml version="1.0" encoding="UTF-8"?>
<xml>
   <book id="1" title="C# Program" author="David" price="21" />
   <book id="2" title="Python" author="Michael" price="100" />
</xml>'::xml))
select
   f1::text::int as id,
   f2::text as title,
   f3::text as author,
   f4::text::int as price
from
   t,
   unnest(
      xpath('/xml/book/@id', x),
      xpath('/xml/book/@title', x),
      xpath('/xml/book/@author', x),
      xpath('/xml/book/@price', x)
      ) as d(f1,f2,f3,f4);
```

012 操作 json

-讀取 json 格式檔(僅支援特定格式如下例)，並轉為 Table 格式顯示

MS-SQL

SQL2016 前的版本，請配合使用 parseJSON 這個 function，請參考 Phil Factor 在 https://www.red-gate.com/simple-talk/sql/t-sql-programming/consuming-json-strings-in-sql-server/ 的說明

```
declare @jsonstr varchar(max);
set @jsonstr = '[
```

```
    {
        "LocationID":    "1",
        "Name":    "Tool Crib",
        "CostRate":    "0.00",
        "Availability":    "0.00",
        "ModifiedDate":    "2002-01-01 00:00:00"
    },
    {
        "LocationID":    "2",
        "Name":    "Sheet Metal Racks",
        "CostRate":    "0.00",
        "Availability":    "0.00",
        "ModifiedDate":    "2002-01-01 00:00:00"
    },
    {
        "LocationID":    "3",
        "Name":    "Paint Shop",
        "CostRate":    "0.00",
        "Availability":    "0.00",
        "ModifiedDate":    "2002-01-01 00:00:00"
    },
    {
        "LocationID":    "4",
        "Name":    "Paint Storage",
        "CostRate":    "0.00",
        "Availability":    "0.00",
        "ModifiedDate":    "2002-01-01 00:00:00"
    },
    {
        "LocationID":    "5",
        "Name":    "Metal Storage",
        "CostRate":    "0.00",
        "Availability":    "0.00",
        "ModifiedDate":    "2002-01-01 00:00:00"
    }
]';

Select parent_ID,
```

```
    max(case when name='LocationID' then convert(int,StringValue) else 0 end) as LocationID,
    max(case when name='Name' then convert(Varchar(50),StringValue) else '' end) as Name,
    max(case when name='CostRate' then convert(SmallMoney,StringValue) else 0 end) as CostRate,
    max(case when name='Availability' then convert(Decimal(8,2),StringValue) else 0    end) as
Availability,
    max(case when name='ModifiedDate' then convert(DateTime,StringValue) else 0 end) as
ModifiedDate
from dbo.parseJSON(@jsonstr)
where ValueType = 'string'
group by parent_Idl;
```

MariaDB

-請參考 http://benjr.tw/103200 的說明
-MySQL 的寫法稍有不同，請查閱相關說明(和 PostgreSQL 較類似)

```
DROP TABLE IF EXISTS _tmp1;
create TEMPORARY table _tmp1 (id INT NOT NULL AUTO_INCREMENT, info JSON NOT NULL,
PRIMARY KEY(id));

insert into _tmp1(info) values
    ('{
        "LocationID":    "1",
        "Name":    "Tool Crib",
        "CostRate":    "0.00",
        "Availability":    "0.00",
        "ModifiedDate":    "2002-01-01 00:00:00"
    }'),
    ('{
        "LocationID":    "2",
        "Name":    "Sheet Metal Racks",
        "CostRate":    "0.00",
        "Availability":    "0.00",
        "ModifiedDate":    "2002-01-01 00:00:00"
    }'),
    ('{
        "LocationID":    "3",
        "Name":    "Paint Shop",
        "CostRate":    "0.00",
        "Availability":    "0.00",
        "ModifiedDate":    "2002-01-01 00:00:00"
```

```
    }'),
    ('{
        "LocationID":    "4",
        "Name":    "Paint Storage",
        "CostRate":    "0.00",
        "Availability":    "0.00",
        "ModifiedDate":    "2002-01-01 00:00:00"
    }'),
    ('{
        "LocationID":    "5",
        "Name":    "Metal Storage",
        "CostRate":    "0.00",
        "Availability":    "0.00",
        "ModifiedDate":    "2002-01-01 00:00:00"
    }');

SELECT
   JSON_VALUE(info, '$.LocationID') as LocationID,
   JSON_VALUE(info, '$.Name') as Name,
   JSON_VALUE(info, '$.CostRate') as CostRate,
   JSON_VALUE(info, '$.Availability') as Availability,
   JSON_VALUE(info, '$.ModifiedDate') as ModifiedDate
FROM _tmp1;
```

PostgreSQL

```
DROP TABLE IF EXISTS _tmp1;
create TEMPORARY table _tmp1 (id SERIAL NOT NULL PRIMARY KEY, info JSON NOT NULL);

insert into _tmp1(info) values
    ('{
        "LocationID":    "1",
        "Name":    "Tool Crib",
        "CostRate":    "0.00",
        "Availability":    "0.00",
        "ModifiedDate":    "2002-01-01 00:00:00"
    }'),
    ('{
        "LocationID":    "2",
        "Name":    "Sheet Metal Racks",
```

```
            "CostRate":    "0.00",
            "Availability":    "0.00",
            "ModifiedDate":    "2002-01-01 00:00:00"
        }'),
        ('{
            "LocationID":    "3",
            "Name":    "Paint Shop",
            "CostRate":    "0.00",
            "Availability":    "0.00",
            "ModifiedDate":    "2002-01-01 00:00:00"
        }'),
        ('{
            "LocationID":    "4",
            "Name":    "Paint Storage",
            "CostRate":    "0.00",
            "Availability":    "0.00",
            "ModifiedDate":    "2002-01-01 00:00:00"
        }'),
        ('{
            "LocationID":    "5",
            "Name":    "Metal Storage",
            "CostRate":    "0.00",
            "Availability":    "0.00",
            "ModifiedDate":    "2002-01-01 00:00:00"
        }');

SELECT
    info ->> 'LocationID' as LocationID,
    info ->> 'Name' as Name,
    info ->> 'CostRate' as CostRate,
    info ->> 'Availability' as Availability,
    info ->> 'ModifiedDate' as ModifiedDate
FROM _tmp1;
```

013 BOM 的正展開

MS-SQL

```
CREATE PROC [dbo].[usp_BOM_Spread]
(
```

```
   @BOM_ID int
)
/*
   bom 表正展開
   exec usp_BOM_Spread 1;
*/
AS
      -------- 正展 BOM ----------
      ;WITH bom(comp_id, m_prod, lvl, ord,bom_03,bom_05,last_qty)
      AS
      (
        --Recursive CTE
        SELECT comp_id, m_prod, 1 ,
              CONVERT(nvarchar(128),cast(comp_id as varchar)+' '+bom_03),
              bom_03,
              bom_05,
              convert(real,bom_05) as unit_qty
        FROM bom_d
        WHERE m_prod=@BOM_ID
        UNION ALL
        --UNION ALL 後稱為 Recursive ,會遞迴反覆執行,直到無任何查詢結果為止
        SELECT p.comp_id,
              p.m_prod,
              b.lvl+1,
              CONVERT(nvarchar(128),b.ord+' - '+cast(p.comp_id as varchar)+' '+p.bom_03),
              p.bom_03,
              p.bom_05,
              convert(real,p.BOM_05 * b.last_qty) as last_qty
        FROM bom_d p, bom b
        WHERE P.m_prod=b.comp_id
      )
      SELECT lvl as 階層,
        (REPLICATE(' ', lvl) + b.comp_id) as '子件',
        b.ord,
        b.bom_03 as 組裝序號,
        b.bom_05 as 單位用量,
        last_qty as 展開用量 ,
        e.productname as 品名
```

```
    FROM bom b
    left join Products e on b.comp_id = e.productid
    ORDER BY ord
    option(maxrecursion 10);
```

MariaDB

```
DELIMITER //
create or replace procedure usp_BOM_Spread (_BOM_ID int)
/*
  call usp_BOM_Spread(1);
*/
begin
  WITH RECURSIVE bom(comp_id, m_prod, lvl, ord,bom_03,bom_05,last_qty)
  AS
  (
    -- Recursive CTE
    SELECT comp_id, m_prod, 1 ,
      CONVERT(CONCAT(cast(comp_id as char),' ',bom_03), char),
      bom_03,
      bom_05,
      convert(bom_05, int) as unit_qty
    FROM bom_d
    WHERE m_prod = _BOM_ID
    UNION ALL
    -- UNION ALL 後稱為 Recursive ,會遞迴反覆執行,直到無任何查詢結果為止
    SELECT p.comp_id,
      p.m_prod,
      b.lvl+1,
      CONVERT(CONCAT(b.ord,' - ',cast(p.comp_id as char),' ',p.bom_03), char),
      p.bom_03,
      p.bom_05,
      convert(p.BOM_05 * b.last_qty, int) as last_qty
    FROM bom_d p, bom b
    WHERE P.m_prod=b.comp_id
  )
  SELECT lvl as 階層,
  CONCAT(repeat(' ', lvl) , b.comp_id) as '子件',
  b.ord,
  b.bom_03 as 組裝序號,
```

```
    b.bom_05 as  單位用量,
    last_qty as  展開用量 ,
    e.productname as  品名
    FROM bom b
    inner join Products e on b.comp_id = e.productid
    ORDER BY ord;
end
//
DELIMITER ;
```

PostgreSQL

```
create or replace function usp_BOM_Spread (_BOM_ID int)
/*
    select * from usp_BOM_Spread(1);
*/
RETURNS TABLE
(
    _階層  integer,
    _子件  CHARACTER VARYING,
    _ord CHARACTER VARYING,
    _組裝序號  CHARACTER VARYING,
    _單位用量  decimal(10,2),
    _展開用量  decimal(10,2),
    _品名  CHARACTER VARYING
)
language plpgsql
as $$
-- declare
-- variable declaration
begin
    DROP TABLE IF EXISTS _tmp;
    create TEMPORARY table _tmp(
        階層  integer,
        子件  CHARACTER VARYING,
        ord CHARACTER VARYING,
        組裝序號  CHARACTER VARYING,
        單位用量  decimal(10,2),
        展開用量  decimal(10,2),
```

```
    品名  CHARACTER VARYING );

WITH RECURSIVE bom(comp_id, m_prod, lvl, ord,bom_03,bom_05,last_qty)
AS
(
    -- Recursive CTE
    SELECT comp_id, m_prod, 1 ,
        cast(comp_id as varchar) || ' ' || bom_03,
        bom_03,
        bom_05,
        cast(bom_05 as decimal(10,2)) as unit_qty
    FROM bom_d
    WHERE m_prod = _BOM_ID
    UNION ALL
    -- UNION ALL 後稱為 Recursive ,會遞迴反覆執行,直到無任何查詢結果為止
    SELECT p.comp_id,
        p.m_prod,
        b.lvl+1,
        b.ord || ' - ' || cast(p.comp_id as varchar) || ' ' || p.bom_03,
        p.bom_03,
        p.bom_05,
        cast(p.BOM_05 * b.last_qty as decimal(10,2)) as last_qty
    FROM bom_d p, bom b
    WHERE P.m_prod=b.comp_id
)

Insert Into _tmp
SELECT lvl as  階層,
(repeat(' ', lvl) || b.comp_id) as  子件,
b.ord,
b.bom_03 as  組裝序號,
b.bom_05 as  單位用量,
last_qty as  展開用量  ,
e.productname as  品名
FROM bom b
inner join Products e on b.comp_id = e.productid
ORDER BY ord;
```

```
    RETURN QUERY
    select * from _tmp;

end; $$
```

014 Replace String data in all the Columns of a Table

```
--測試資料庫及資料
create table tbl_str (t1 varchar(10),t2 varchar(10));

insert into tbl_str values ('AA','BB'),('BB','DD')
,('CC','BB'),('FF','GG');
```

MS-SQL

```
create procedure usp_replace_string_for_all_columns
(
    @tablename varchar(100),
    @oldvalue varchar(100),
    @newvalue varchar(100)
)
/*
    exec usp_replace_string_for_all_columns 'tbl_str', 'BB', 'ZZ';
*/
as
    declare @sql nvarchar(max);
    declare @fld_name varchar(100);
    declare @maxcount int;
    declare @currid int;
    declare @nextrowid int;

    -- 取得所有字串欄位
    SELECT COLUMN_NAME,ROW_NUMBER() OVER(ORDER BY COLUMN_NAME) AS ROWID
    into #tmp_fld FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = @tablename
and DATA_TYPE like '%char%'

    select @maxcount=count(*) from #tmp_fld;
    select @currid=min(rowid) from #tmp_fld;

    --init
    while (@currid <= @maxcount)
```

```
    begin
        select @fld_name=COLUMN_NAME from #tmp_fld where rowid = @currid;

        set @sql = 'Update ' + @tablename + ' set ' + @fld_name + ' = ' + char(39) + @newvalue +
char(39) +
                        ' where ' + @fld_name + ' = ' + char(39) + @oldvalue + char(39);
        exec sp_executesql @sql;

        -- 取得下一筆資料
        select @nextrowid = min(rowid) from #tmp_fld where rowid > @currid;
        set @currid=@nextrowid;
    end

    drop table #tmp_fld;
```

MariaDB

```
DELIMITER //
CREATE or replace PROCEDURE usp_replace_string_for_all_columns(
    in _tablename varchar(100),
    in _oldvalue varchar(100),
    in _newvalue varchar(100)
)
/*
    call usp_replace_string_for_all_columns('tbl_str', 'BB', 'ZZ');
*/
BEGIN
    declare _sql varchar(600);
    declare _fld_name varchar(100);
    declare _maxcount int;
    declare _currid int;
    declare _nextrowid int;

    DROP TEMPORARY TABLE IF EXISTS _tmp_fld;
    create TEMPORARY table _tmp_fld (COLUMN_NAME varchar(100), ROWID integer);

    insert into _tmp_fld
    SELECT column_name,ROW_NUMBER() OVER(ORDER BY COLUMN_NAME) AS ROWID
    FROM information_schema.columns where table_schema = 'northwind'
    and table_name = _tablename;
```

```
   select count(*) into _maxcount from _tmp_fld;
   select min(rowid) into _currid from _tmp_fld;

   -- init
   while (_currid <= _maxcount)
   do
      select COLUMN_NAME into _fld_name from _tmp_fld where rowid = _currid;

       set @strsql = concat('Update ',_tablename,' set ',_fld_name,' = ','"', _newvalue ,
            '"',' where ', _fld_name,' = ','"', _oldvalue,'"');
       select @strsql;
      prepare stmtsql from @strsql;
      execute stmtsql;
      deallocate prepare stmtsql;

      -- 取得下一筆資料
      select min(rowid) into _nextrowid from _tmp_fld where rowid > _currid;
      set _currid=_nextrowid;
   end while;
   drop table _tmp_fld;
END//
DELIMITER ;//
```

PostgreSQL

```
/* https://www.dbrnd.com/2017/12/postgresql-replace-string-data-in-all-the-columns-of-a-table/
*/
create or replace function fn_replace_string_for_all_columns(
   tablename text,oldvalue text,newvalue text
)
/*
   select *from fn_replace_string_for_all_columns('tbl_str', 'BB', 'ZZ');
*/
returns void as
$$
declare
eachrw record;
begin
      for eachrw in
```

```
        select 'UPDATE '||$1||' SET '||C.COLUMN_NAME||' = REPLACE
('||C.COLUMN_NAME||','''||$2||''','''||$3||'''); ' SQLQuery
        FROM (select column_name from information_schema.columns where
table_schema='public' and table_name =$1)c
    loop
        EXECUTE eachrw.SQLQuery;
    end loop;
end;
$$language plpgsql;
```

015 將包含關鍵字內容的所有資料表及所有欄位，全部搜尋出來

MS-SQL

```
/*
https://thesitedoctor.co.uk/blog/search-every-table-and-field-in-a-sql-server-database-updated/
*/
```

MariaDB

```
/*
https://www.tfzx.net/article/69548.html
https://ithelp.ithome.com.tw/articles/10198638
*/
```

PostgreSQL

```
/*
https://stackoverflow.com/questions/5350088/how-to-search-a-specific-value-in-all-tables-postgresql */
```

第 11 章

Store Procedure 的除錯技巧

Store Procedure 並不是萬靈丹，這個我必須說在前頭，就像其他程式語言一樣，他有很多優點，但是缺點也不少。他的缺點主要可以從三個方向來說

- 每一個資料庫都有自己的專屬語法(也就是方言)，這會造成轉換資料庫的困擾
- 非物件導向語言
- 除錯困難，無 IDE 環境可以下中斷點除錯

當然，也許其他人員會說他的語法不嚴謹之類，但是以我的實務經驗看來，這問題不大，所以我就沒有列出來。在上面這三點中，第一點我們已經在前文討論過，實務上，轉換資料庫的情況並不多見，我個人的看法是「慎選資料庫、一門深入」，一旦選定，就買定離手，不要輕易更換，那這個問題就不大。就算萬不得已要換，以我的個人經驗，只要真的對某一個資料庫下過苦功，要在短期內搞懂另一個資料庫並不難。請各位參考前面的幾篇文章應該就可以理解。

至於第二點，非物件導向語言。這就真的是見仁見智了。有些 Developer 認為程式開發語言，都已經被物件導向統一了，但是一接到資料庫的 SQL 程式，就好像回到原始人的世界，滿心的圈圈叉叉。

關於 ORM 我在後面的單元，有專文討論，這邊就簡單說明帶過。

＊＊ＳＱＬ小常識＊＊
所謂的 ORM，即 Object-Relational Mapping(對象關係映射)，它的作用是在關係數據庫和業務實體對象之間作一個映射，這樣，我們在具體的操作業務對象的時候，就不需要再去和複雜的 SQL 語句打交道，只需簡單的操作對象的屬性和方法。

我個人實際的經驗，這的確是有一點不順，但並不嚴重，尤其 RESTFul API 架構興起後，我甚至用 Store Procedure 寫了一支通用的 orm_api，可以幾乎完全取代 ORM 的大部分功能，重點是，完全不需要安裝、學習任何的 ORM 框架。當然，這只是我的個人經驗，後面會再詳細說明。

我也研究過幾套 ORM(Nhibernate、EF)，功能也都還不錯，配合一些工具，甚至 MVC 架

構的 Model 都能自動產生，看起來是很方便。可是對於像我這樣的 SQL 重度使用者來說，用 ORM 真的有隔靴搔癢的嚴重不適感。我想應該有很多重度 SQL 使用者會感同身受，明明一句 SQL 就能簡單搞定的事，用 ORM 就變成天大地大的難事，所以也才有了本系列文的產生。

我認為任何的語言，如果經過幾十年仍能屹立不搖，就一定有其存在的必要性。SQL 語言事實上已經存在很久了，我一直有一個很大的疑惑，為甚麼在程式設計這個領域，永遠都是革命、創新會受到追捧，原有的語言卻只能拋棄? 我個人認為 Borland 的 Delphi 系列，是地球上最佳開發工具(我還在使用 Delphi 7，目前的 XE 太晚推出，很難有大作為)。但是又如何? 我後來也只能改用 Visual Studio 20xx，現在重新學習 Vue + Element UI 中。好了，就只是發發牢騷，別佔太多篇幅。

簡單的結論，我認為非物件導向並不是重大的 issue，但是如果堅持要全面使用物件導向，那就請繞道。不過讓我們面對現實，SQL 短期內還是無法被替代，有些複雜的計算，如果不透過 SQL 而用前端來處理，會付出效能的代價。所以，我會繼續努力地用 SQL，並想辦法讓所有的事情都用 SQL 解決，前端對我而言，就是介面處理。這是最有效率的分工。

再來就是第三點，除錯困難，無 IDE 環境可以下中斷點除錯。這是事實，而且我認為這才是阻礙 SQL 推廣的最大問題，就算我已經算是資深的 SQL Developer，還是會常常陷入除錯的誤區，等到千辛萬苦解決問題後，除了無語問蒼天外，很難有其他的想法。

Store Procedure 的除錯，通常有下列幾個方式

- Print： 就是最常用的 alert
- 將有問題的程式片段 copy 到 Database4 中，mark 部分程式重複執行
- 寫到 log 暫存檔
- 查看錯誤訊息紀錄

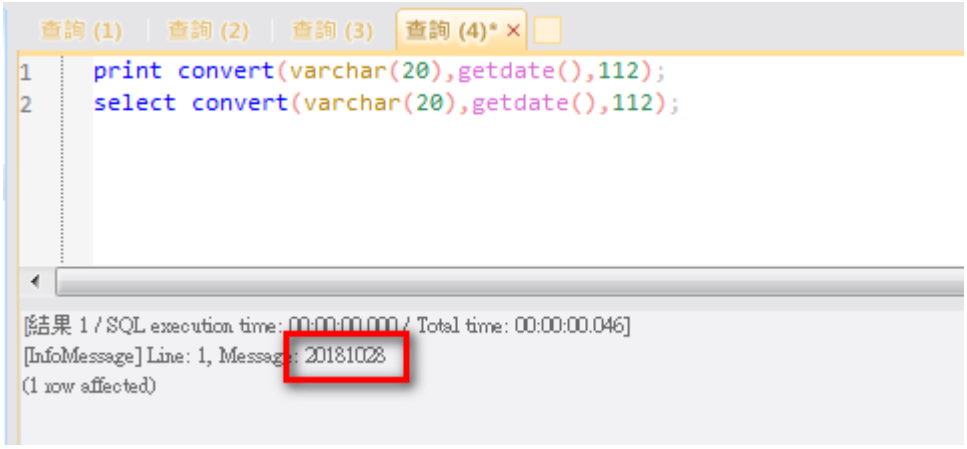
➢ PRINT 指令(或是 select) ，這是最基本的除錯指令

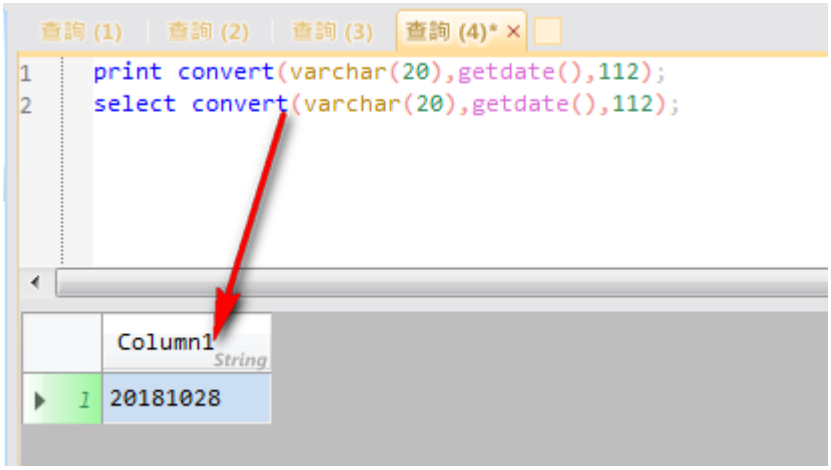
不知道從甚麼時候開始，很多程式入門的書籍的第一個範例程式，都是著名的 Hello World

```
10 PRINT 'Hello World'
20 GOTO 10
```

- 所以，我們今天要介紹的第一個除錯指令，就是在程式中顯示某些變數或是欄位的值
  - MS-SQL

```
print convert(varchar(20),getdate(),112);
select convert(varchar(20),getdate(),112);
```





- MariaDB 沒有 print，而是直接用 select 指令

```
SELECT 'some content'
```

這是最基本的除錯方式，每個 Developer 都會用，所以我就不囉嗦了。

➢ 將有問題的程式片段 copy 到 Database4 中，mark 部分程式重複執行

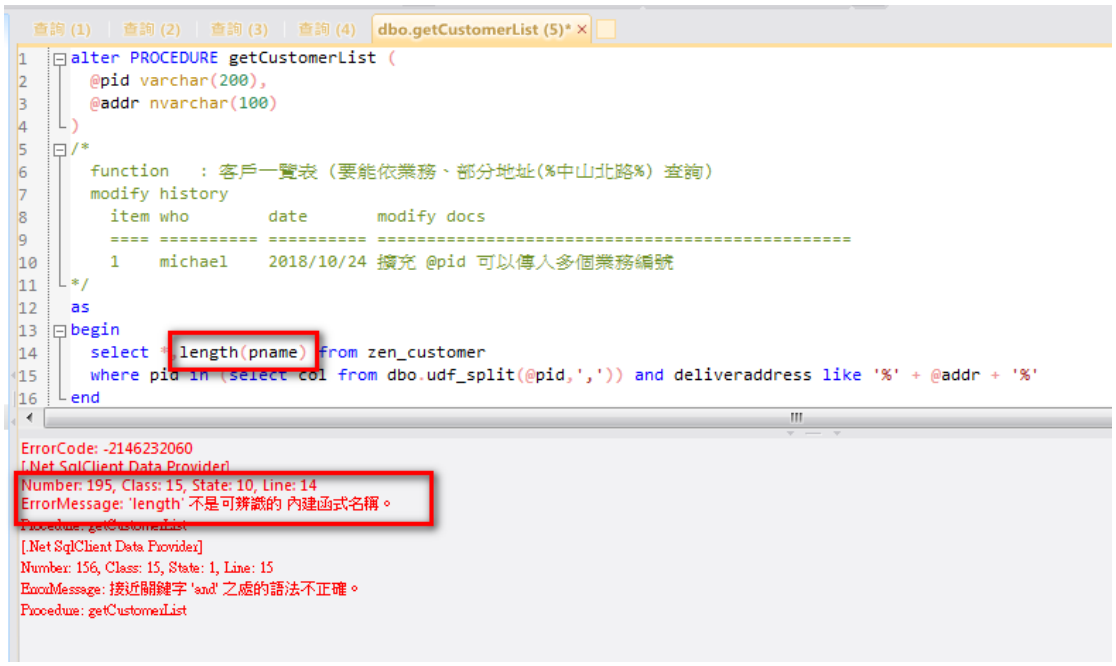
通常寫 Store Procedure 時，發生的錯誤可以分為兩大類

1. 語法錯誤
2. 邏輯錯誤

- **語法錯誤**

  如果是語法錯誤，通常在 create 或 alter 程式時，就會報錯，也會顯示相關的錯誤訊息，這和一般的 Compiler 是很類似的。這一類的問題，錯誤通常發生是對語法的不熟悉，通常不難解決，如果錯誤訊息看不懂，有時我會直接將錯誤訊息問 google 大神，根據經驗，通常都會找到錯誤的原因。下面舉個簡單的範例(程式片段)

```
select *,length(pname) from zen_customer
  where pid in (select col from dbo.udf_split(@pid,','))
  and deliveraddress like '%' + @addr + '%'
```



錯誤的原因說得很清楚，沒有 length() 這個函數，改用 len() 就可以了。

- **邏輯錯誤**

  如上所述，語法錯誤通常不難解決，SQL 的難以 debug 通常是邏輯問題，而非語法問題，也就是取得的資料或是數字結果和預期不符。這個部分真要認真說起來，天就會黑一半。說實話，還真的沒有銀彈可以解。我只能作一些經驗上的分享。

1.對 Table Schema 要完全透徹了解
2.對可能有 null 值欄位的任何判斷或取值、計算等，都要防範錯誤發生
3.對任何 join 語法，要特別注意 on 的條件，要考慮 join 是否會取不到資料，或是資料重複
4.豐富的 debug 經驗會累積解決問題的直覺。

➢ 寫到 log 暫存檔

就像一般的程式語言，有些錯誤 Compiler 是無法發現，而是在執行時才會發生(這通常和資料內容有關，語法和邏輯事實上都沒有錯誤)。但是我們寫程式時，當然不希望看到由 SQL Enginer 所發出的令人看不懂的錯誤訊息，所以我們通常會利用 try catch 將可能會報錯的程式包起來。可是如果沒有將錯誤記錄起來，資料內容就會錯亂，所以通常我們會寫到 log 檔紀錄，而在執行完後，將錯誤的內容顯示出來。

當然，除了錯誤 log，有時我們也會記錄執行過程中的資訊，因為有時錯誤只會發生在某些特定情況下(例如某產品 item 或某客戶)，所以只好將計算過程也一併寫到 log，然後用手算的方式，追蹤可能的錯誤原因。相信我，邏輯性的錯誤，通常是靠這種笨方法查出來的。sa 在查帳時，必須有這一種 log，否則有些問題是查不出來的。

```
begin try
   sql statement
end try
   begin catch
      --if error, write to log
      print ERROR_MESSAGE();
      insert into error_log (LOGINID,UNIQUENO,ONHANDDATE,BILLNO) values
         (@LOGINID,@opouniqueno,@BILLDATE,@BILLNO)
   end catch
```

➢ 查看錯誤訊息紀錄

上面談的是批次處理資料時，要將問題資料記錄到 log，方便追蹤錯誤發生。
而下面的語法，通常是用在 api 回傳 json 時配合使用。這是最後一段錯誤訊息的防錯機制，就算是發生的嚴重的錯誤，也要讓呼叫端不受影響，能繼續執行下去。

```
sql statement ...
begin try
      execute sp_executesql @sql;
      select 0 code,'成功' msg,0 recno;
end try
begin catch
   -- Execute the error retrieval routine.
   -- 或是也可 SELECT ERROR_NUMBER() AS ErrorNumber, ERROR_MESSAGE() AS ErrorMessage;
   print @sql
   execute usp_GetErrorInfo_v2;
end catch
CREATE PROCEDURE usp_GetErrorInfo_v2
/*
   https://technet.microsoft.com/zh-tw/library/ms179296(v=sql.105).aspx
   傳回錯誤訊息, 通常配合 begin catch 中使用
*/
AS
```

```
SELECT
    ERROR_NUMBER() AS code,
    ERROR_MESSAGE() as msg,
    0 as recno;
```

不論學習哪種程式語言，不斷的實作及練習是不二法門，而除錯則是最難掌握的技巧。跟個人經驗有直接的關係，所以很難用簡單的文字說明清楚，只能就大原則說明，其他更重要的，還是經驗的累積所培養出的直覺，當然多請教 Google 大神也是很有幫助的，在附錄中筆者有列出一些很棒的 blog，多磨多練終會成就一代大神。

第 12 章

ORM 解決方案

關於 ORM 的定義及優缺點，網路上已經有一堆的說明，我就不在重複囉嗦了，各為童鞋可以自行 Google 或參考下面幾篇 blog

- ORM 介紹及 ORM 優點、缺點
  http://blog.twbryce.com/what-is-orm/
- ORM VS SQL
  https://dotblogs.com.tw/code6421/2009/02/10/7100
- 好用的微型 ORM：Dapper
  https://www.huanlintalk.com/2014/03/a-micro-orm-dapper.html
- Nodejs 最好的 ORM - TypeORM
  https://cloud.tencent.com/developer/article/1012625

簡單的整理幾個 ORM 重點

1.用物件導向操作資料庫 2.透過 ORM 框架，建立 MVC 架構中的 Model(或反向產生 Schema)

以上這 2 點，事實上都是近年來開發方法論中的最重要的主流，希望讓系統的開發更標準化、更邏輯化。這絕對是正確的方向，但是條條大路通羅馬，要到羅馬當然不只一條路，搭飛機可能更加便利。ORM 最常被應用的實際場景(但不限於)，最常見的是表單基本的 CRUD，也就是資料的

新增(Create) 查詢(Read) 修改(Update) 刪除(Delete)。

這當然是非常重要且基本的功能，大部分的 ORM 也都能很輕鬆地完成上面的功能。但是我實在不喜歡 Model 中長串的 GET/SET 設定，雖然有很多的工具可以協助產生。目前市面上已經有非常多的 ORM 框架，也各有優缺點及擁護者，因為我沒有深入的使用，所以當然沒有資格提出意見。如果想要深入的使用任何一個框架，都必須花很大的力氣才能成為專家，一般基礎的需求都不會造成困壞，很快就能上手，但是有點經驗的

Developer 就會知道，魔鬼隱藏在細節中，碰到較奇怪的 bug 時，在初階學習階段，常常會求救無門，或是框架本身就沒有考慮到某些較特殊的需求。

基於以上的種種原因，我個人一直都沒有真的將 ORM 框架導入到實際的系統開發。但是，ORM 的確會帶來很好的開發體驗，一些例行性的通用功能(如 CRUD)，的確是 ORM 的強項，加上我個人還算精通 SQL，所以很自然地就會想到要用 SQL(Store Procedure) 來實作 ORM 的相關功能。
將 ORM Store Procedure 化(實作) 最大的優點，大概說明如下

1.不必引用任何框架，無學習成本
2.不需要寫任何 Model 的設定程式
3.配合 RESTFul API 架構下，任何前端都能輕易整合(Ajax)
4.使用 Angular 或 Vue 等新漸進式框架的雙向綁定，幾乎完全物件導向化

事實上，將 ORM Store Procedure 化，是我不斷強調的「以資料庫為開發核心」的一個非常重要的實作，透過這個實作，您會發現，前端程式變得非常單純，只是在需要時呼叫 API，而後端程式的開發，是先將通用的 Store Procedure 先共用模組化，無法共用的，再依需求寫成 Store Procedure。在這樣的架構下，所需要學習的開發工具被大幅簡化。

太過複雜的架構及方法，通常表示這個方案只是過渡技術，因為還沒收斂。
人類文明的進步歷程，事實上是善用工具的過程，而究其根本，就是想偷懶(效率)。

如果要學習一堆的技術才能開發一個小系統，這絕對是有問題的。應該是只要學會幾個重要的核心工具，就可以搞定大部分的系統才對。當然，筆者才疏學淺，很多概念仍在摸索、學習，僅此與各位童鞋共勉。但是說來容易，如何實作? 雖然之前也有根據 MS-SQL 及 MariaDB 有說明入門及進階的 SQL 語法，但是寫 ORM 可不是件簡單的活，沒有三兩三，怎敢上梁山。

下面我再分別補充將 ORM Store Procedure 化會用到的更進階 SQL 語法。事實上，最核心的程式就是用來執行 [動態 SQL 指令]。

```
*** MS-SQL ***
execute sp_executesql @sql;

*** MySQL(MariaDB) ***
PREPARE runsql FROM @sql;
EXECUTE runsql;
```

- MS-SQL

```
declare @sql nvarchar(max);
if @action = 'C' or @action = 'Insert'
```

```
    set @sql = 'Insert Into ' + @tablename + ' (' + @fields + ')' + ' Values (' + @datas + ')'
else if @action = 'U' or @action = 'Update'
    set @sql = 'update ' + @tablename + ' set ' + @updatedatas + ' where ' + @pkwhere
else if @action = 'D' or @action = 'Delete'
    set @sql = 'delete from ' + @tablename + ' where ' + @pkwhere;


begin try
    execute sp_executesql @sql;
end try
begin catch
    -- Execute the error retrieval routine.
    --  或是也可  SELECT ERROR_NUMBER() AS ErrorNumber, ERROR_MESSAGE() AS ErrorMessage;
    execute usp_GetErrorInfo;
end catch
```

- MariaDB

```
set _uuid2 = uuid();
if _wheresql = '' then
    set @sql = concat('Insert Into _split (guid, col) select ', char(39), _uuid2, char(39),
                      ',count(*) from ', _tablename);
else
    /*  這裡應該也要考慮  jointables  的寫法(如果  _wheresql  有  ma. _tablename  就要加) */
    if position('ma.' in _wheresql) > 0 then
        set @sql = concat('Insert Into _split (guid, col) select ', char(39), _uuid2, char(39),
                          ',count(*) from ', _tablename, ' ma where ', _wheresql);
    else
        set @sql = concat('Insert Into _split (guid, col) select ', char(39), _uuid2, char(39),
                          ',count(*) from ', _tablename, ' where ', _wheresql);
    end if;
end if;
/* select @sql; */
PREPARE runsql FROM @sql;
set SQL_SAFE_UPDATES = 0;
EXECUTE runsql;
DEALLOCATE PREPARE runsql;
```

MySQL(MariaDB) 的語法有一個坑要小心，請小心繞道。

set SQL_SAFE_UPDATES = 0;

另外關於 Error Handle，MS-SQL 的機制很完整，但是 MySQL(MariaDB 就有點遜)

PS：也有可能是我對 MariaDB 還不熟。

```
/*
   MySQL 的 Error Handel 目前只找到以下的寫法, 還想不到共用 function 的寫法
*/
DECLARE CONTINUE HANDLER FOR 1048
begin
   set _errcode = '1048';
   set _errmsg = '主鍵值不能為空白或 null';
end;
DECLARE CONTINUE HANDLER FOR 1062
begin
   set _errcode = '1062';
   set _errmsg = '主鍵值重複，無法存檔!!';
end;
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
begin
   -- set _errcode = '9999';
   -- set _errmsg = '資料庫操作失敗, 請洽客服工程師協助';
   GET DIAGNOSTICS CONDITION 1
   _errcode = RETURNED_SQLSTATE, _errmsg = MESSAGE_TEXT;
end;
```

要將 ORM Store Procedure 化並不是非常困難的事，對 SQL 語法有一定的理解就不會太難。一旦寫好，您會發現帶來的效益非常巨大，基本的 CRUD 真的在彈指間就完成(當然，前端的程式也要組出對應的內容再呼叫 API)，對開發資訊系統的效率，有非常大的提升。稍後我們會將 ORM 應具備的功能做詳細說明，並會配合幾個實際的前端呼叫的程式片段來說明。

orm_api 是一支特殊的 Store Procedure，配合 server 端同一支通用 api，只要傳入適當的參數，就可以處理包括新增、修改、刪除、分頁查詢、單筆查詢等常用的資料庫存取動作。

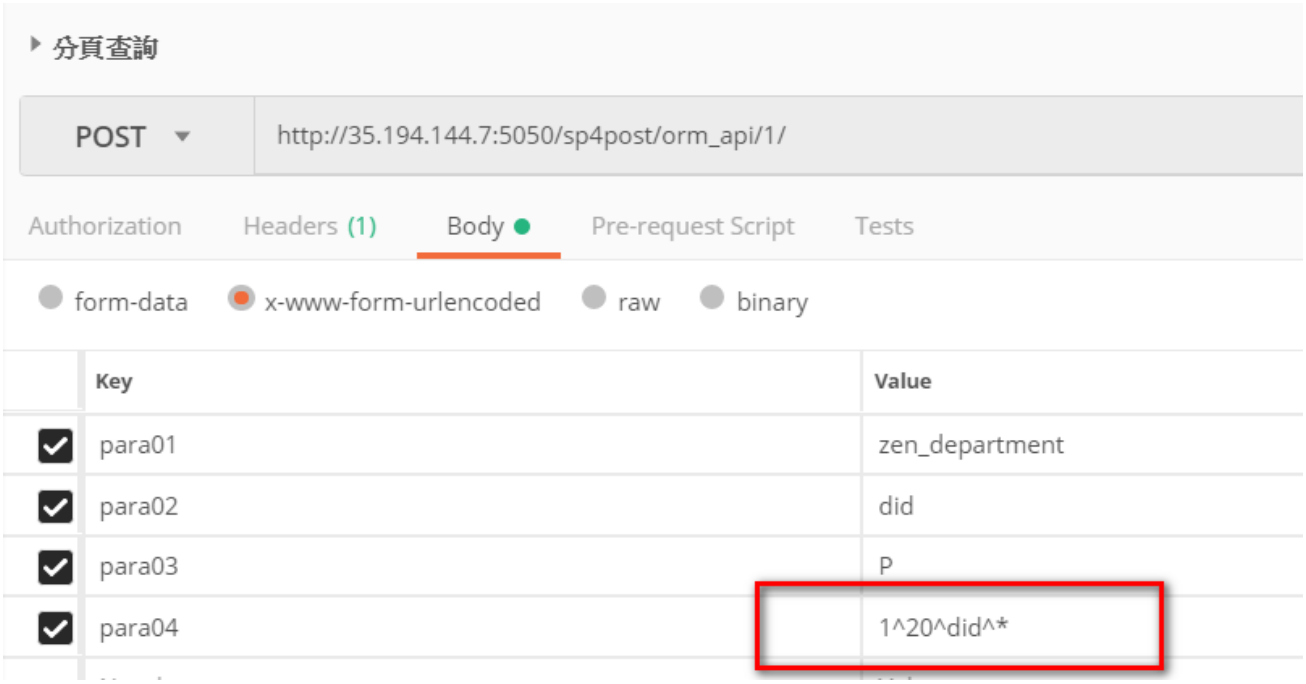
下面是 orm_api 的註解說明，請參考

```
create proc orm_api
(
   @token varchar(100),          --加入 token 機制
   @tablename varchar(100),
   @pk varchar(100),              --primary key(for update delete)
   @action varchar(20),          --CRUDP(如果是 Read, @paras 參數內容有特殊寫法)
   @paras nvarchar(max)          --前端組好的資料內容(應該是 json 格式的內容)
)
```

```
/*
  Function    : 寫一支通用 orm-api 依傳入的參數及設定, 智慧的組出相應的 sql
  Description: 本程式是想模仿 orm 的設定,然後讓前端程式只要傳入參數,就能執行 orm 的增刪查改的基本功能
       先寫 ms-sql 版, 其他 db 稍後再想辦法。
       R(Read) 是讀取單筆資料, @paras 的參數內容同 CU
       P(Page) 是以頁為單位的讀取模式, 一般是 cowork with DataGrid

       呼叫 api 範例如下
       --新增資料, @paras 內容是由前端傳入的 json 資料內容
       exec orm_api 'basarea','areaid','C',''
       exec orm_api 'basarea','areaid','U',''   --修改資料
       exec orm_api 'basarea','areaid','D',''   --刪除資料

       declare @para varchar(100);
       --單筆查詢
       set @para = 'areaid='+char(39)+'A'+char(39)+'^areaid,areacname^#menuid';
       exec orm_api 'basarea','areaid','R',@para;
       --分頁查詢
       exec orm_api 'basarea','areaid','P','1^20^areaid^areaid,areacname^#menuid';

       -- 新增 Query 後要分頁顯示(擴充 P 的功能, 多傳 wheresql)
       set @para = 'areaid like '+char(39)+'%A%'+char(39);
       exec orm_api 'basarea','areaid','P','1^20^areaid^areaid,areacname^'+@para;

  Modify History
    Item Who          Date          Modify Docs
    ==== ========== ========== ============================================
    1    Michael    2017/12/20 Init Thinking
*/
```

利用 Postman 測試 API 範例如下

呼叫範例： {url}/orm_api/2/



POST 參數	參數說明
para01	TableName，如 zen_department
para02	Primary Key，如 did (如果是複合主鍵，用逗號區隔即可)
para03	C 代表新增、U 代表修改、D 代表刪除、P 代表分頁查詢、R 代表單筆資料查詢
para04	args，api 所需的實際參數，主要分為 2 大類，CUD 為一類，內容為 json，是將欄位的內容組成 json，另外是分頁查詢，最多有 6 個參數，利用 ^ 為字串的分隔符號。下面會配合 para03 的狀態，逐項說明。

下面是前端使用 Vue.js 呼叫的片段程式碼，請參考

```
//雙向綁定的 data 區寫法, 實際上是 OO，所以不再需要 Model
data() {
   return {
      didDisable: false,
      action: 'new',
      form: {
            did: '',
            dname: '',
            dboss: '',
            dmemo: ''
      },
      fulltextsearch: '',
```

```
//後端分頁取得列表資料
getData() {
    const params = {
        para01: 'zen_department',
        para02: 'did',
        para03: 'P',
        para04: `${this.cur_page}^${this.cur_size}^did^*^dname like "%${this.fulltextsearch}%"^basm01`
    }
    this.$axios.post('/api/orm_api/1/', qs.stringify(params)).then((res) => {
        this.loading = false;
        if(res.length && res[0]){
            this.tableData = res[0];
            if(typeof res[0][0] === 'object'){
                this.total = res[0][0]._TotalRec;
            }
        }
    })
},

// 保存數據
saveEdit(formName) {
    this.$refs[formName].validate((valid) => {
        if (valid) {
            const type = this.action==='new'?'C':'U';
            const params = {
                para01: 'zen_department',
                para02: 'did',
                para03: type,
                para04: JSON.stringify(this.form)
            }
            this.$axios.post('/api/orm_api/1/', qs.stringify(params)).then(res=>{
                if(res.length === 0){
                    this.action==='new'?this.tableData.push(this.form):this.$set(this.tableData, this.idx, this.form);
                    this.editVisible = false;
                    this.$message.success('存檔成功');
                }
            })
        }
```

```
        })
    },
}
```

經由上面的程式範例，您應該能體會 orm_api 的優勢，如果配合適當的 Wizard 或是 Code Generator Tools，前端的程式可以善用這樣的機制被快速的產生接近商用版本的雛型程式。這又回到本系列文的核心，我們的目的是想要建構一個好用的、能快速上手的開發框架，而且必須架構在幾穩定的核心開發工具。

關於「以資料庫為開發核心」這個開發方法論，筆者另有一本「ERP 開發勝經」專門討論，限於個人的經驗，可能還有許多的不足，請各位先進多指教。

第 13 章

其他可以應用的解決方案

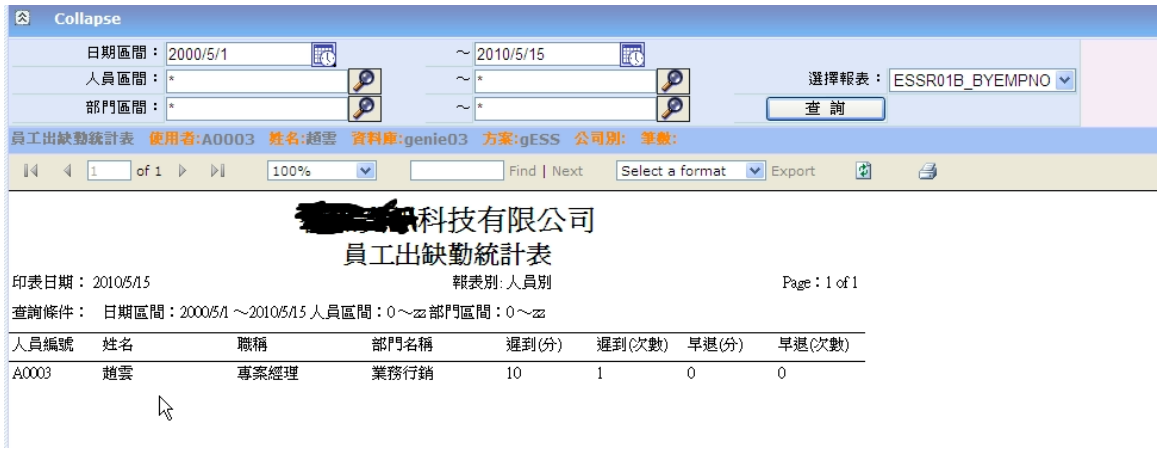
企業管理系統，如 ERP、CRM 或是 POS 門市銷售系統，不外乎就是資料輸入、分析處理後，產出相關的分析報表(實際當然沒這麼簡單，這是簡化的說法)。有沒有可能將所有的商業邏輯，都寫在資料庫的 Store Procedure 及 user define function? 然後前端透過各式的程式開發工具存取資料庫呢? 能否舉更多的實際案例供參考嗎?

前一章的 ORM 就是一個核心的共用機制，透過這個機制，只要配合前端的 Wizard 開發，就可以快速的產生具備基礎功能的前端程式。我個人實際的開發經驗是超過 90%以上的中小型 ERP 系統功能，都能透過這種機制來完成，當然，不可能只靠 SQL 就全部搞定整套系統，但是存取資料庫內容的後端程式是絕對沒有問題的。

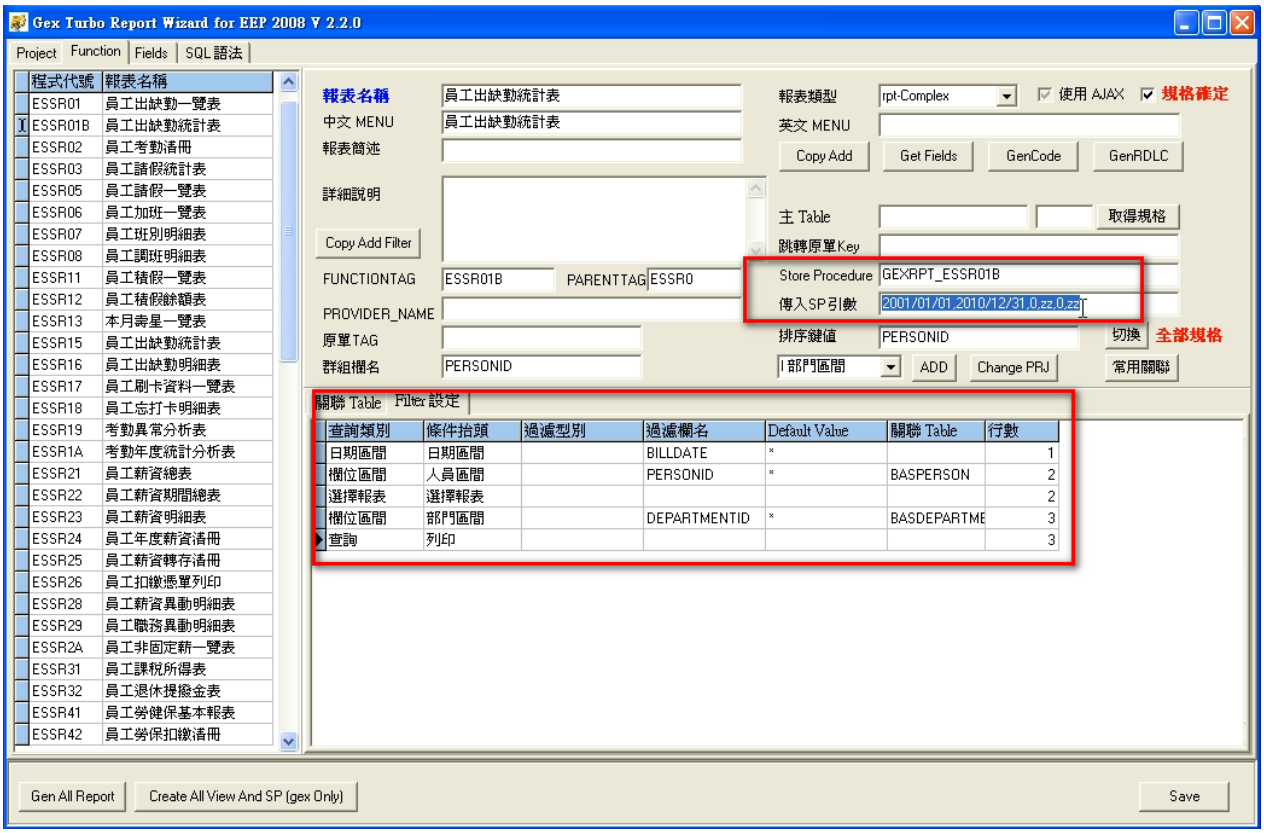
除了 ORM 這個核心大殺器外，接來來我就簡單的利用 3 個實際的應用案例，來說明更多實際的使用情境，希望能拋磚引玉，讓開發企業資訊系統能更大程度的透過 SQL 的標準內建機制來完成。

- 所有的分析報表
- 電子簽核核心引擎
- 薪資計算公式

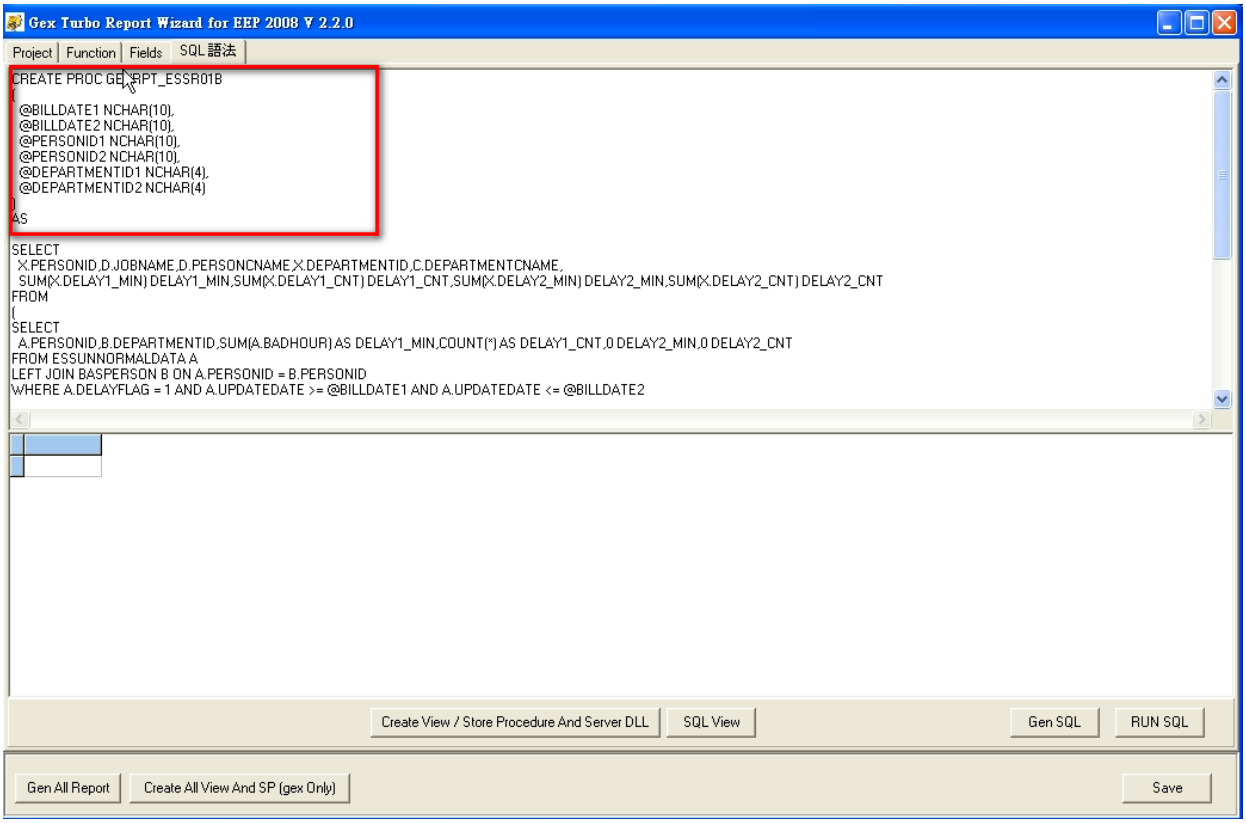
- 所有的分析報表
  - 報表是所有企業資訊系統最終呈現的結果。隨著市場的瞬息萬變，資訊系統本就應該提供各式的報表供決策者參考，但是對程式開發團隊而言，使用者層出不窮、永無止境且充滿想像力的報表需求，是一大挑戰(甚至是惡夢)，為了拯救程式師脫離永無止境的報表黑洞，所以我們就將報表開發全 Wizard 化，當然，取資料的部分要配合寫成 Store Procedure。
  - 一般的報表通常會包含二大部份，一是各式的查詢條件，二則是根據條件過濾出資料即顯示格式，請參考下圖。除了顯示樣式有所差別，基本上大多是長這個樣子。



  - 下面是利用自行開發的 Wizard，配合寫好的 Store Procedure(GEXRPT_ESSR01B)，既可產生完整的報表程式。

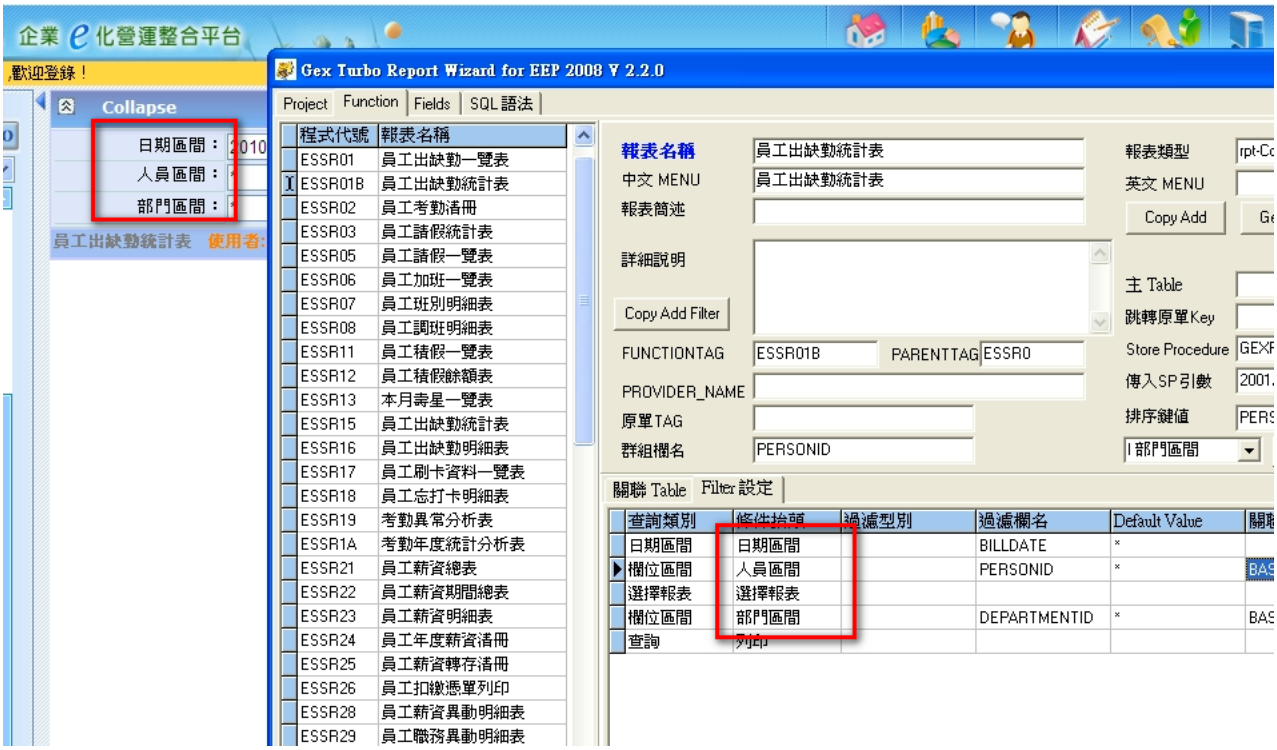


- Store Procedure 大概的內容



- 報表的顯示格式設定，最多可支援 10 個不同格式。

- Wizard 的設定畫面及報表程式執行的對照



現有庫存一覽表

印表日期: 91/10/03　　成本方式: 移動加權平均　　Page : 1

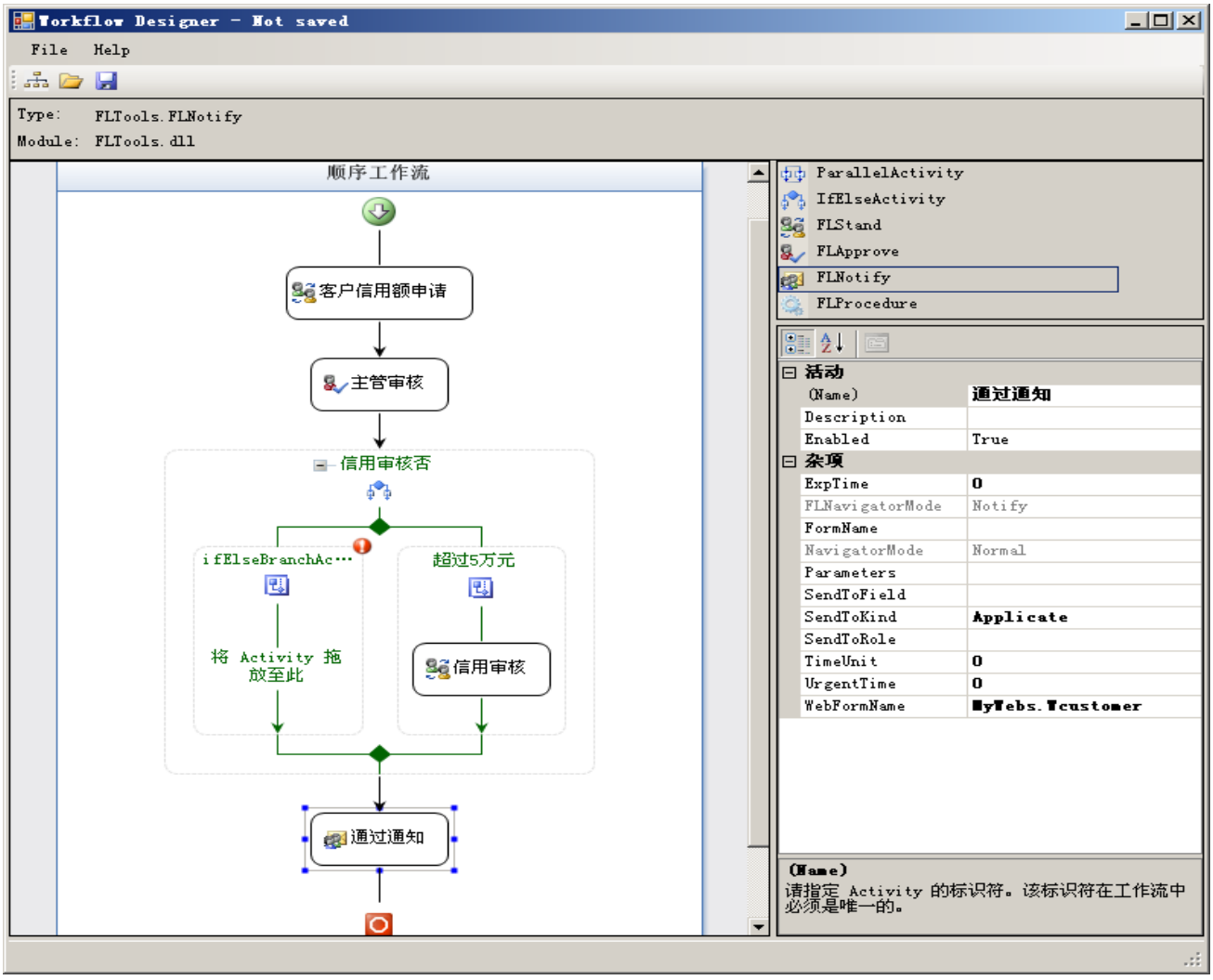
倉庫區間: A ☐ D　產品區間: A001 ☐ A003　分倉　零值不顯示　不顯示小計

產品編號	品名	倉庫	實際在庫量	成本	借出未還	借入未還	理論庫存量	安全庫存
A001	高級鋼筆組	A	4,997	2,500.0	3	0	5,000	500
		B	55	4,500.0	0	5	50	200
		C	100	5,005.0	0	0	100	100
		D	150	6,000.0	0	0	150	50
A002	小叮噹釘書機	A	8,997	1,750.0	3	0	9,000	1,000
		B	55	4,000.0	0	5	50	10
		C	100	4,540.0	0	0	100	10
		D	150	5,000.0	0	0	150	10
A003	寶寶奶嘴製造機	A	91,132	27,208.8	8	0	91,140	500
		B	50	45,456.0	0	0	50	20
		C	102	45,665.0	0	2	100	20
		D	151	15,566.0	0	1	150	20
合計:			106,039		14	13	106,040	2,440
庫存總值:								1,501,655,040

透過這種機制，全系統的 99%以上的報表都可以在 Store Procedure 寫好並測試後，幾乎就可以快速(同步)的開發完成。如此一來就不必擔心爆肝都寫不完的報表了。

- 電子簽核(WorkFlow) 核心引擎
  - 工作流管理系統（Workflow Management System, WfMS）的主要功能是通過電腦技術的支持去定義、執行和管理工作流，協調工作流執行過程中工作之間以及群體成員之間的訊息交換。工作流需要依靠工作流管理系統來實現。
  - 工作流要解決的主要問題是：為實現某個業務目標，在多個參與者之間，利用電腦，按某種預定規則自動傳遞文檔、訊息或任務。
  - WorkFlow Engine 主要包括下列功能
    - 組織/群組/角色設定
    - 作業流程設計器
    - 流程引擎
    - 使用者工作桌面

    下圖是流程設計器的參考介面



  - WorkFlow 的主要作業流程如下

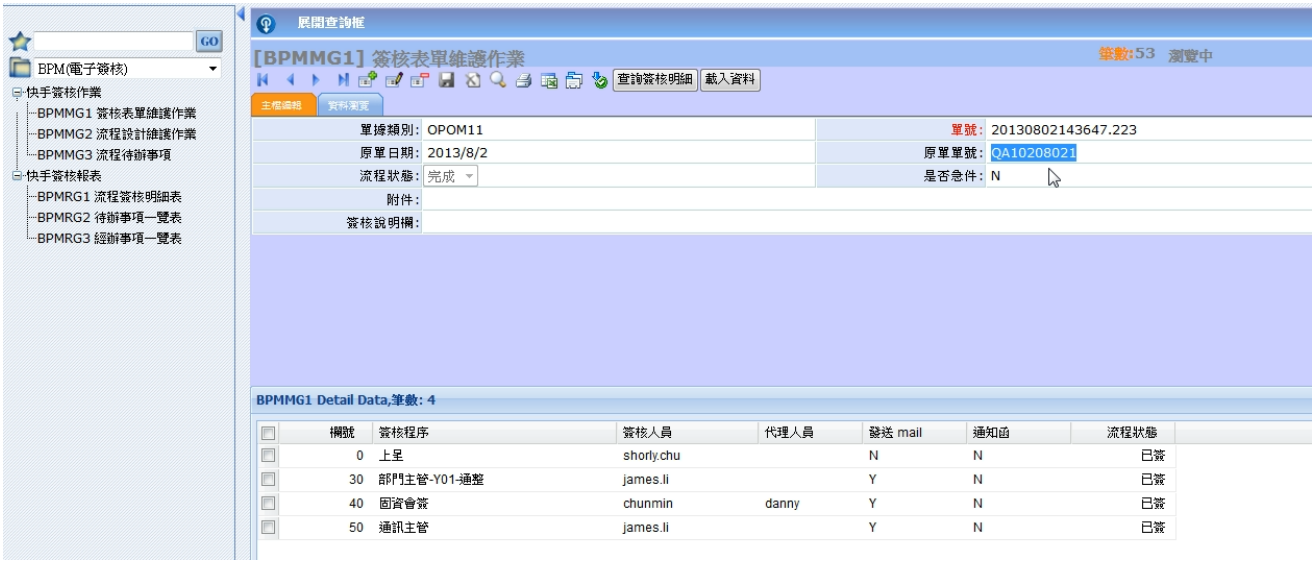
    上呈 -> 核可(退簽) -> 簽結(作廢)

    MANAGER 、 AGENT 、FIELDDEF

- 所有的表單都可以改為簽核表單(FLOWFLAG)
  BPMMG2 簽核設定作業
  BPMMG1 簽核表單維護(管理及控管)
  BPMMG3 流程待辦
- BPM store procedure 詳細說明
  - GEXBPM_GENFLOW1 主 menu 全部上呈
  - GEXBPM_GENFLOW2 單據上呈
  - GEXBPM_GENFLOW3 重新上呈(未開始)
  - GEXBPM_GENFLOW4 重新上呈(已開始)
  - GEXBPM_SIGNOK 核可
  - GEXBPM_SIGNBACK 退簽
  - GEXBPM_SIGNUNDO 作廢
  - GEX_SEND_FLOWMAIL 發送 Mail

- 簽核流程設定

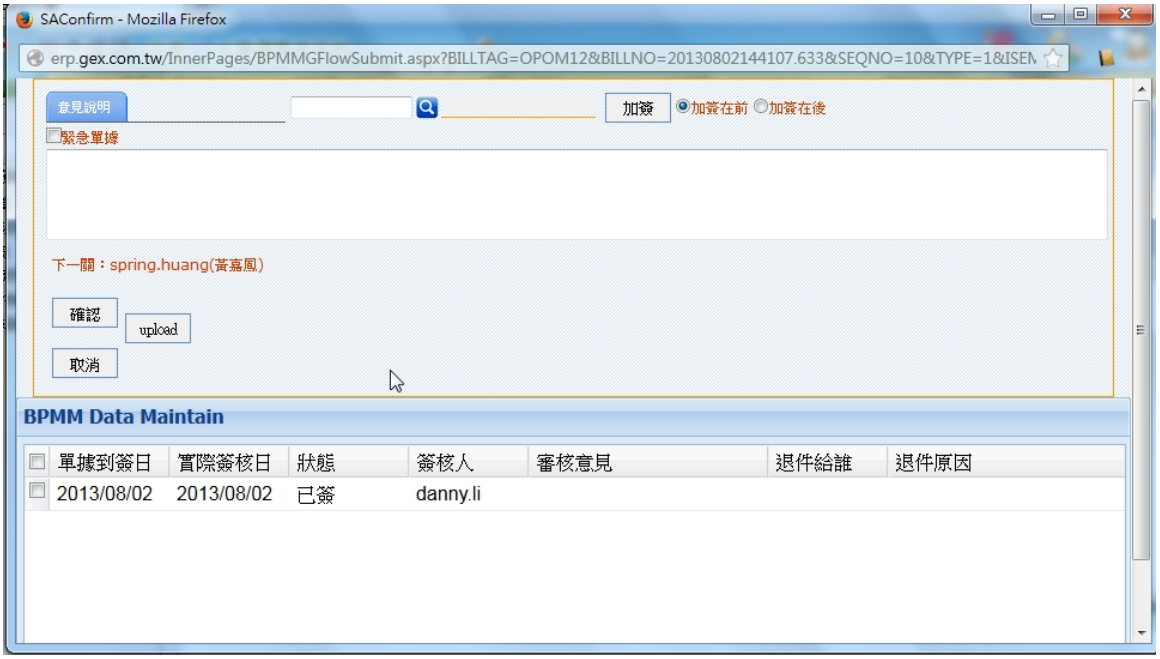
- 簽核表單維護



- 原單據上加入相關 Button

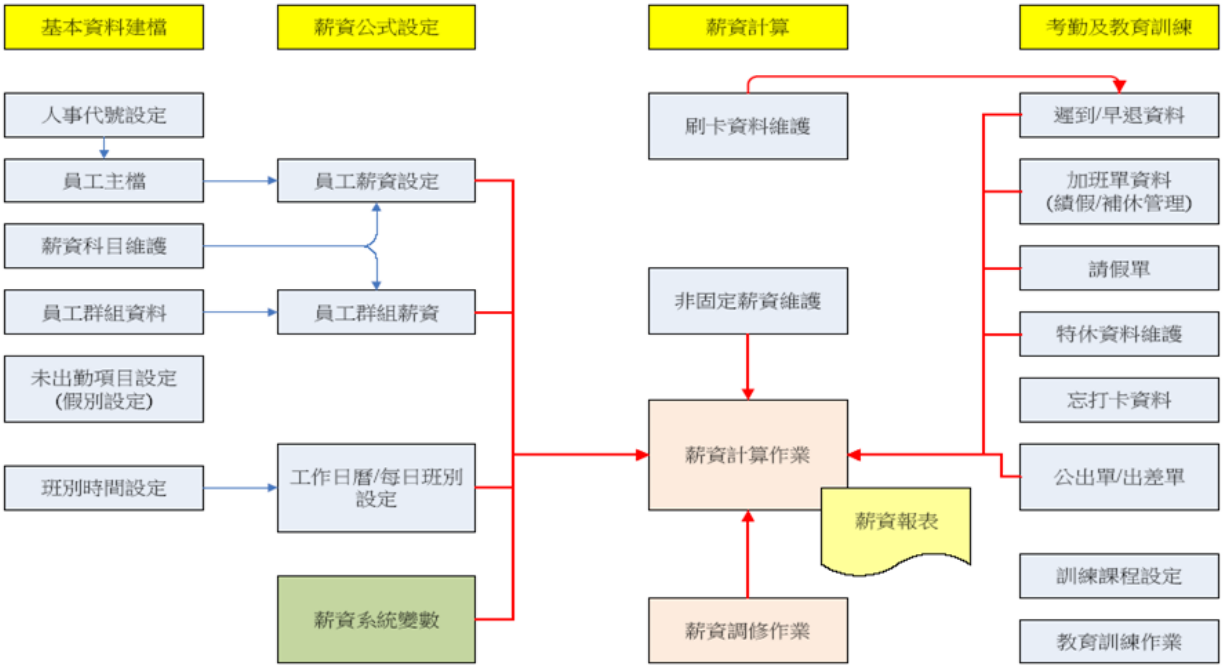
- 流程代辦事項

- 實際簽核介面



由於流程簽核引擎本身就是一個很大的議題，此處我們只做簡單的介紹及說明。但是將其中控制流程走向的引擎核心用 Store Procedure 實作有很多的優點，效能也很好，這也是筆者之前曾經重點實作的 SQL 核心實作模組之一，供您參考。

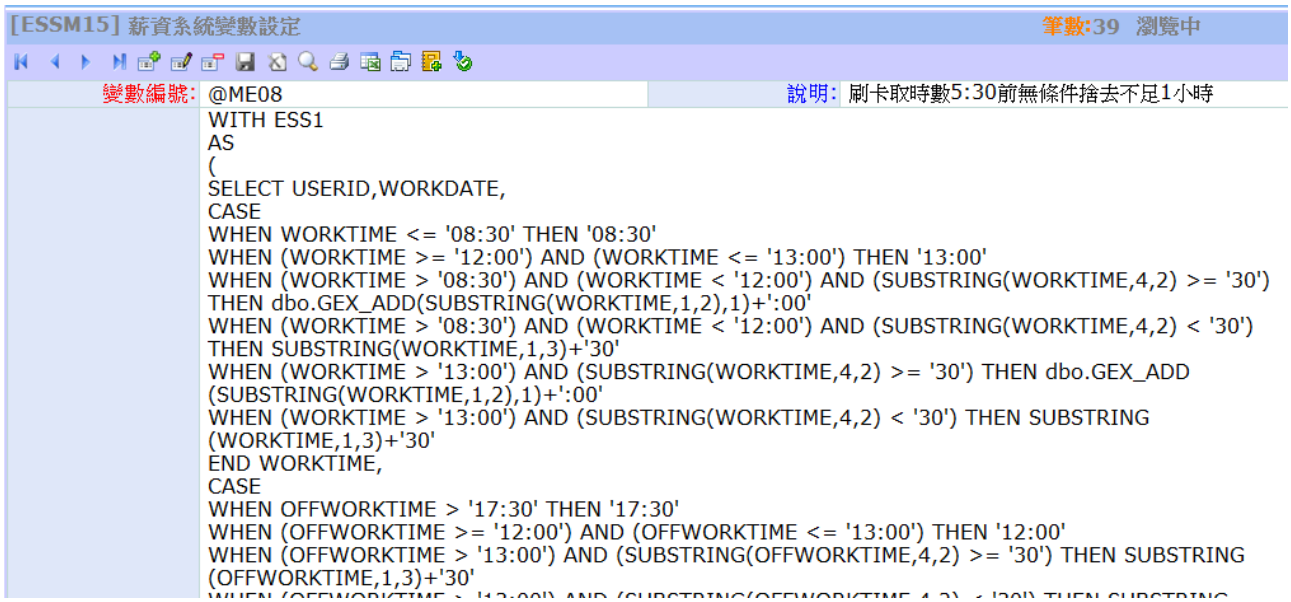
- 薪資計算公式
  - 最後我們在來討論在人事薪資系統(Human Resource：HR) 中，扮演非常吃重腳色的薪資計算模組。簡單的說，HR 系統主要功能如下圖

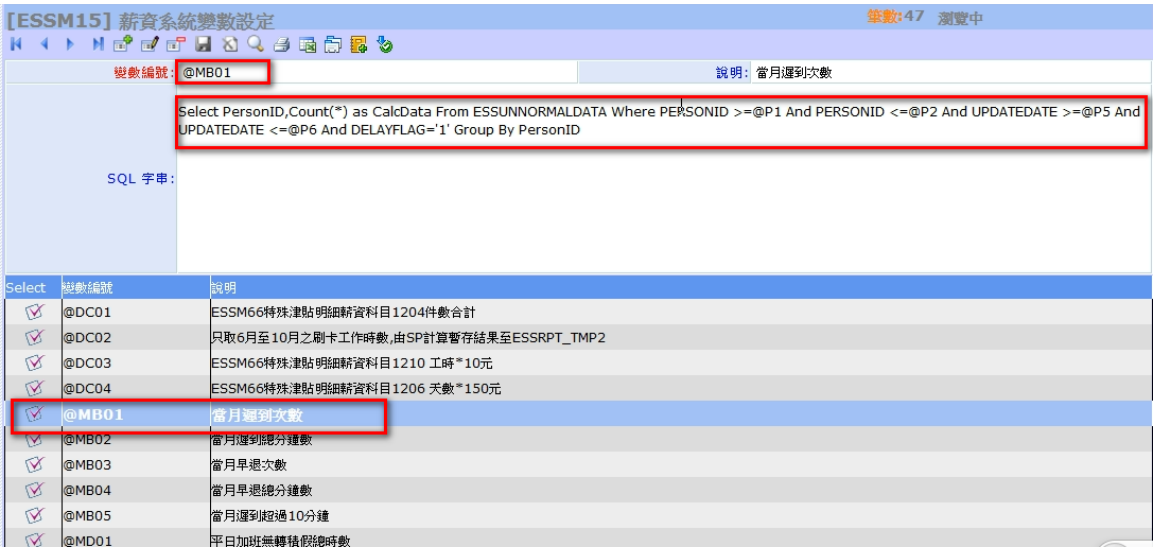


  - 其中因為每家企業的薪資計算有各種花樣(餐飲業的排班和工廠的三班制等)，再加上各種季節獎金、年終獎金，一套系統要搞定這麼多問題可不是件容易的事。如果企業內部又常會變動計算公式(公告)，那開發工程師又要快樂的不要不要的。如何解決這個問題? 如何提高系統彈性，能靈活地根據企業需求調整，而不必不斷的更新程式? 下面是我們的一個解決思路，一樣符合本書主題，用 SQL 來搞定。
    - 薪資期別、薪資科目設定
    - 薪資變數(重中之重)
    - 個人及群組的固定金額設定
    - 個人及群組的公式設定

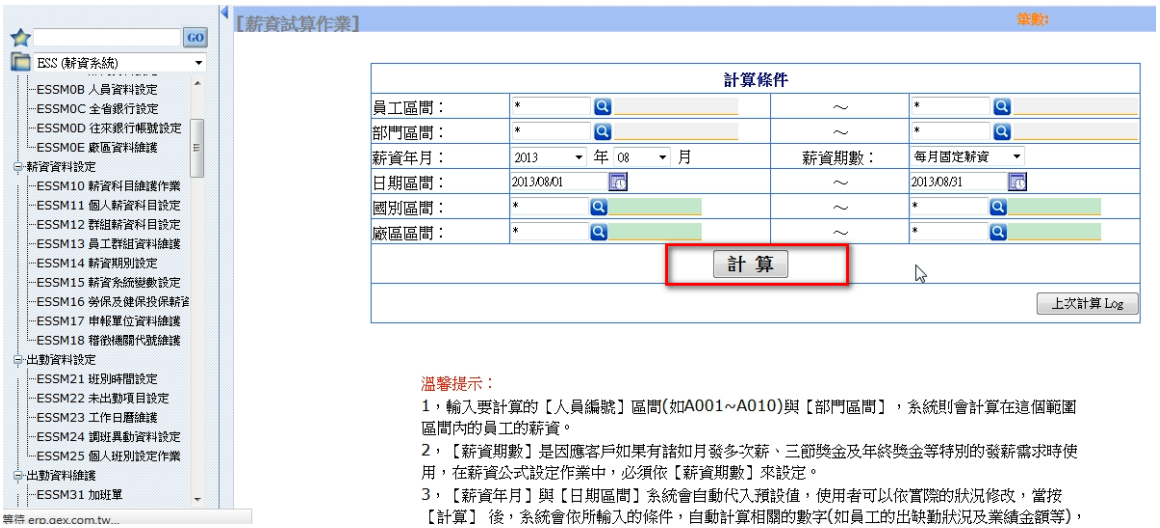
	序號	類別	公式	若真	若假	計算結果
1	0010	DC	底薪增加不足月判斷			0
2	0020	IF	@MY01 >0	0030	0050	0
3	0030	CA	((^1101/@MY02) *@MY01)*1.6			0
4	0040	GO	#0060			0
5	0050	CA	(^1101*1.6)			32000
6	0060	SM	#0030+#0050			32000


[ESSM10] 薪資科目維護作業
加項科目
扣項科目
刷新
新增
刪除
保存
薪資科目類別
薪資代號
中文名稱
備註
IS_ADD
1 01 1101 基本底薪
2 01 1102 職務加給 Y
3 01 1103 應稅加班費 Y
4 01 1104 免稅加班費 Y
5 01 1105 出差津貼 Y
6 01 1106 交通津貼 Y
7 01 1201 全勤獎金 Y
8 01 1202 交際津貼 Y
9 01 1203 伙食津貼 Y
10 01 1204 專業津貼 Y
11 01 1205 技術津貼 Y
12 01 1206 夜班津貼 Y
Page 1 of 1
顯示從1條數據到23條數據，共23條數據


[ESSM15] 薪資系統變數設定
筆數:39 瀏覽中
變數編號: @ME08
說明: 刷卡取時數5:30前無條件捨去不足1小時
WITH ESS1
AS
(
SELECT USERID,WORKDATE,
CASE
WHEN WORKTIME <= '08:30' THEN '08:30'
WHEN (WORKTIME >= '12:00') AND (WORKTIME <= '13:00') THEN '13:00'
WHEN (WORKTIME > '08:30') AND (WORKTIME < '12:00') AND (SUBSTRING(WORKTIME,4,2) >= '30')
THEN dbo.GEX_ADD(SUBSTRING(WORKTIME,1,2),1)+':00'
WHEN (WORKTIME > '08:30') AND (WORKTIME < '12:00') AND (SUBSTRING(WORKTIME,4,2) < '30')
THEN SUBSTRING(WORKTIME,1,3)+'30'
WHEN (WORKTIME > '13:00') AND (SUBSTRING(WORKTIME,4,2) >= '30') THEN dbo.GEX_ADD
(SUBSTRING(WORKTIME,1,2),1)+':00'
WHEN (WORKTIME > '13:00') AND (SUBSTRING(WORKTIME,4,2) < '30') THEN SUBSTRING
(WORKTIME,1,3)+'30'
END WORKTIME,
CASE
WHEN OFFWORKTIME > '17:30' THEN '17:30'
WHEN (OFFWORKTIME >= '12:00') AND (OFFWORKTIME <= '13:00') THEN '12:00'
WHEN (OFFWORKTIME > '13:00') AND (SUBSTRING(OFFWORKTIME,4,2) >= '30') THEN SUBSTRING
(OFFWORKTIME,1,3)+'30'



- 薪資計算的 sp 計算流程說明
  - GEX_CALCSALARY
    刪除當月當期資料
    GEX_CALCSALARY_STEP1
    GEX_CALCSALARY_STEP2
    計算後，整理 LOG 檔的內容，寫回年檔
  - GEX_CALCSALARY_STEP1
    將薪資變數，執行後填入暫存檔待用 ESS_TMPC1
    LOGINID,VARID,PERSONID,RESULT_VALUE
  - GEX_CALCSALARY_STEP2
    呼叫 GEX_SALFORMULA_V3 展開人員薪資科目的所有項目
    展開每一個 user 每個科目的計算公式(loop)



以上就配合一些畫面及實際的使用案例，說明善用 Store Procedure 可以大幅減低前端的程式開發工作，除了上面的案例，其他諸如

- 傳輸傳票機制
- 單價設定取用機制
- 月加權成本計算
- 移動加權成本計算
- 庫存現有數量
- MRP 展算

等等

都可以用 SQL 輕鬆解決，而且，當開發平台轉換，如 PHP 後台網站改為 APP 手機，只要寫一個新的接口，如利用 Node.js 或 Python 等，就立刻可以將 Store Procedure 華麗轉身為 RESTful API，不用重新在開發一次相同的商業邏輯。

「以資料庫為開發核心」，這並不是當前的開發主流，會有這個想法實在是被逼出來的。如果有經歷過 Client/Server 時代的開發人員，要轉型為 Web 開發人員，在那個幾乎每天都有新框架、新平台、新 XX 的草莽時代，不知道逼死了多少英雄好漢。凡是存在必有其道理，本書就是希望提供一個另類的開發觀點，盡可能的善用已存在的良好工具，讓開發人員不必一味的苦苦追趕層出不窮的最新技術，希望和您能彼此共勉。

感謝您努力地看到最後，如果您有關於本書的任何問題，歡迎您寫 email 給作者，genie.michael@gmail.com。

附錄

參考網路相關資料來源一覽表

- 以資料庫為開發核心，利用通用 API 玩轉後端資料存取的概念與實作
  https://ithelp.ithome.com.tw/users/20111421/ironman/1615
- W3Schools
  https://www.w3schools.com/sql/
- DB-Engines Knowledge Base of Relational and NoSQL Database Management Systems
  https://db-engines.com/en/ranking
  https://iter01.com/443735.html
- 楓花雪岳
  http://jengting.blogspot.com/2012/07/beyond-relational-tsql-challenge.html
- T-SQL Challenge(Beyond Relational TSQL Challenge)
  https://www.sqlservercentral.com/search/T-SQL+Challenge
- Sql 資料挑戰賽&網路銷售案例分析
  https://zhuanlan.zhihu.com/p/66761416
- Database Research & Development
  https://www.dbrnd.com/postgresql/
- STACK OVERFLOW
  https://stackoverflow.com/
- 資料庫優缺點比較說明
  https://kknews.cc/zh-tw/news/x8xpy8q.html
- MariaDB 官網(Function)
  https://mariadb.com/kb/en/built-in-functions/
- PostgreSQL 官網
  http://postgresql.wisdomfish.org/postgresql/stored-proc-vs-func
- DataEDO
  https://dataedo.com/kb/query
- 易百教程
  https://www.yiibai.com/html/database/
- 極客教程
  https://geek-docs.com/sql/sql-examples/sql-retrieves-all-rows-and-columns.html

One More Things

ERP 開發勝經

- 完整介紹開發企業資訊系統，如 ERP、CRM 等現代企業不可或缺的資訊系統，所需瞭解的各層面技術及管理相關知識。
- 深入探討【以資料庫為開發核心】的資訊系統開發框架概念。
- 不空談理論，以實際完成的 WebERP 系統為討論核心，說明實作的技巧，並可提供顧問輔導，協助企業快速開發所需的系統。
- 透過以資料庫為開發核心的理念，透過開發四化的實作框架，實現資訊系統從無到有的完整開發流程。
- 後續會提供一套麻雀雖小但五臟俱全的 Windows 版本的 ERP 系統(買賣業) 供學習研究。

萬事如易官網的介紹

- 萬事如易是由捷克仕移動所開發的一套最酷炫的占卜及八字的預測平台。 目前系統仍在持續的開發和完善中，歡迎各位愛用者提供您的使用心得及各種改善的建議。
- 這個網站緣起於在多年以前，一次和客戶的業務拜訪時，客戶提出了一個需求，如何有效掌握下個月、下一季甚至下一年度的產品生產需求預估。 當下雖然以要準確預估某時間的產品需求量，和氣象局的天氣預估一樣，越久越不準確，僅具備基本的參考價值。但是，我的心中 有一個聲音響起，能不能用古老的易經占卜來解決這個看似不可解的問題，這個念頭一直縈繞在我的心中，這就是您現在手上的 [萬事如易] 的源起。
- 古老的易經歷經千年歲月，仍不減其魅力，但是信者恆信，不信者嗤之以鼻。我們希望藉由本 App 能夠讓蒙塵的明珠，風華再現。 借助現代科學的統計，驗證這歷久彌新的古老智慧。然此工程浩大，非數年無法盡其功，僅拋磚引玉，希望有朝一日能見其功。末學以此自勉。
- [萬事如易] 和其他占卜 APP 最大的不同，在於我們可以讓許多人(成千上萬人) 針對同一個問題占卜，這在以往是根本無法想像的，但是現在我們可以透過如易 APP 做到。為什麼我們要這麼做? 因為占卜的準確率，很難達到 100%，這幾乎是不可能的任務。如易 APP 還只是個小學生，當然也不例外。所以我們想透過統計及大數據來解這個看似無解的難題。我們不求 100% 的準確率，我們所求的是希望能得到一個值得參考的趨勢預測即可。
- 每個人在日常生活中，總會有一些困惑，不管是工作事業、開店是否賺錢、感情迷惑、兒女課業或是長輩身體健康等，都希望身邊能有一個可以提供未來預測參考的工具，[萬事如易] 能讓您可以自行增加新的議題，甚至可以自己設定群組，然後針對不同的議題，可以設定某一個群組內的人，才能占問，如此就能提供非常大的討論彈性及隱密性。
- 整個平台都是採用 [玩轉 SQL] 及 [ERP 開發勝經] 這 2 本書中所提的開發方式，所以它是這 2 本書的實作案例。

玩轉 SQL

超過一百個由淺入深的實際案例，帶您快速遨遊 MS-SQL、MariaDB 及 PostgreSQL 資料庫的 SQL 語法天地

作　　者：陳威均
發 行 人：陳威均
電子郵件：genie.michael@gmail.com
網　　址：www.gex.com.tw
萬事如易：fate.gex.com.tw
出版年月：2021 年 02 月初版
定 價：199 元
ISBN：978-957-438-626-0

展售處（銷售服務）：
Pubu 飽讀電子書平台
網 址：https://www.pubu.com.tw/

Readmoo 讀墨電子書平台
網 址：https://readmoo.com/