Praise for Effective SQL

“鑑於作者的聲譽，我認為這本書會不錯。但不錯還不夠，在這個時刻我感覺我爆了！大多數 SQL 書籍告訴你‘如何’，這本書告訴你‘為什麼’。大多數 SQL 書籍將資料庫設計與實作分離，這本書將設計考量與 SQL 使用的各個方面做整合。大多數 SQL 書籍都躺在我的書架上，這本書會進駐我的桌上”。

—Roger Carlson, Microsoft Access MVP (2006–2015)

“SQL 很容易學，但建立精確有效的 SQL 很難，特別是對於有複雜需求的關鍵系統更難。現在你可以透過這本書快速上手並更快的撰寫出高效率 SQL，無論是使用什麼 DBMS”。

—Craig S. Mullins, Mullins Consulting, Inc., DB2 Gold Consultant and IBM Champion for Analytics

“這是一本好書。它以新手可懂的語言寫出，同時又讓老手受惠。它會吸引各種程度的讀者並應該人手一本”。

—Graham Mandeno, database consultant and Microsoft MVP (1996–2015)

“這本書是資料庫設計者與關聯式 SQL 資料庫開發者的優良資源—易讀又有結合理論與實務的範例。內容包括 Oracle、DB2、SQL Server、MySQL，與 PostgreSQL 的範例。這本書帶領讀者討論處理階層資料與對應表的複雜技術，加上使用 `GROUP BY`、`EXISTS`、`IN`、關聯與非關聯子查詢、窗口函式，與聯集操作的 SQL。別的地方找不到這些技巧，且有趣的範例更讓這本書鶴立雞群”。

—Tim Quinlan, database architect and Oracle Certified DBA

“這本書適合支援多種 SQL 方言的人，它的內容能讓你立即掌握與運用。我從 1992 年就開始玩過各種 SQL，但還是受益良多”。

—Tom Moreau, Ph.D., SQL Server MVP (2001–2012)

“這本書有力、精簡、且容易理解—它展示真實世界的 SQL 應用，教你如何寫查詢並解釋‘資料如何儲存’與‘資料如何查詢’以讓你成功有效率的取得結果”。

—Kenneth D. Snell, Ph.D., database consultant and former Microsoft Access MVP

“這本書出現前有許多人找不到從資料庫新手升級的好書。這本書是地圖、指南、Rosetta Stone，與 Structured Query Language（SQL）新手升級的教練。與其從新發明輪子或找尋使用資料庫的正確方法，不如就買這本書。你不只會看到資料庫顧問的秘訣，還會學到不同資料庫廠商為何有不同的做法。買這本書讓你事半功倍又省事”。

—Dave Stokes, MySQL Community Manager, Oracle Corporation

“這是資料庫開發者‘必讀’的書。它逐步展示如何以 SQL 解決真實問題。作者用容易理解的語言指出各種方案的優缺點。我們都知道在 SQL 中解決問題的方式有好幾種，但作者解釋哪一種更有效率。我最喜歡的部分是每一節後面的重點摘要。強烈推薦這一本書給所有資料庫開發者”。

—Leo (theDBguy ™), UtterAccess Moderator and Microsoft Access MVP

“我認為這本書不只與開發者有關，還有 DBA，因為它討論撰寫有效率的 SQL 與達成結果的各種方式。我認為這是一本必讀的書。買這本書的另一個理由是它討論常見的 RDBMS，如果有人需要從一個 RDBMS 轉換到另一個，就要看這本書。了不起，不得了，恭喜作者”。

—Vivek Sharma, technologist, Hybrid Cloud Solutions, Core Technology and Cloud, Oracle Asia Pacific

序

從 SQL 資料庫語言開始成為國際標準的三十年間，SQL 語言被實作成多種資料庫產品。今天 SQL 無所不在。它是高效能的交易處理系統，存在於智慧手機應用程式與網頁介面之後。甚至有一種資料庫稱為 NoSQL，其共通特徵是（或曾經是）不使用 SQL。隨著 NoSQL 資料庫加上 SQL 介面，“No”現在解釋成“Not Only”SQL。

由於 SQL 的流行，你很可能在多種產品與環境中遇過 SQL。對 SQL 的（合理）批評中有一條是雖然很相似，但產品間有微妙的差異。這些差異來自對標準的不同解釋、不同的開發風格，或不同的底層架構。為了解這些差異，以範例來比較不同 SQL 方言間的微妙差異會很有幫助。本書提供 SQL 查詢的羅塞塔石碑，展示查詢在不同方言間如何撰寫並解釋其間的差異。

我通常會說學習的最好方式是犯錯。這是說知道最多的人曾經犯下最多的錯並從他人的錯誤中學習。此書範例包括不完整與不正確的 SQL 查詢以及為何是不完整與錯誤的說明，如此可讓你從他人的錯誤中學習。

SQL 是有力並複雜的資料庫語言。身為資料庫顧問與 SQL 標準委員會的一員，我看過很多沒有善用 SQL 功能的查詢。完整學習過 SQL 的威力與複雜性的開發者不只可以完全的利用 SQL 的應用程式功能，還可以有效率的建構應用程式。本書的 61 個範例能夠幫助學習。

—Keith W. Hare
資深顧問，JCC Consulting, Inc.；
副主席，INCITS DM32.2—U.S. SQL Standards Committee；
召集人，ISO/IEC JTC1 SC32 WG3—
International SQL Standards Committee

前言

結構化查詢語言，又稱為 SQL，是與大部分資料庫系統溝通的標準語言。我們認為你看這本書表示你需要使用 SQL 的資料庫系統的資訊。

本書的目標是工作有一部分與 SQL 有關的應用程式開發者與初階資料庫管理員（DBA）。我們認為你已經熟悉 SQL 的基本語法並專注於提供有用的建議以讓你善用 SQL 語言。我們發現，當我們從基於程序的方法轉向基於集合的方法來解決問題時，所需的思維方式與用於電腦程式設計的思維方式截然不同。

關聯式資料庫管理系統（RDBMS）是你用來建構、維護、修改，與操作關聯式資料庫的應用程式，許多 RDBMS 程式還提供建構與資料庫中的資料互動的終端使用者應用程式的工具。RDBMS 程式從問世開始就不斷的演進，它們與硬體技術與操作環境一樣越來越全功能與完整。

SQL 簡史

IBM 的研究員 Edgar F. Codd 博士（1923-2003）首先於 1969 年提出關聯式資料庫模型。他於 1960 年代後期正在尋找處理大量資料的新方式並開始思考如何套用數學原理以解決各種問題。

在 1970 年 Codd 博士發表關聯式資料庫模型後，大學與研究實驗室等單位開始開發可作為關聯式資料庫系統的基礎的語言。初始工作在 1970 年代中期產生了不同的語言，其中一個發生於 IBM 位於加州的 Santa Teresa Research Laboratory。

IBM 在 1970 年代初開始進行稱為 System/R 的研究計劃，目標是證明關聯式模型的可行性並獲取設計與實作關聯式資料庫的經驗。他們在 1974 年和 1975 年之間的初步嘗試證明是成功的，並製作出一個關聯式資料庫的原型。

與開發關聯式資料庫同時間，研究員還開始定義資料庫語言。Donald Chamberlin 博士與他的同事在 1974 年開發出 Structured English Query Language（SEQUEL），能讓使用者以清楚定義的英文句子查詢關聯式資料庫。SEQUEL-XRM 這個原型資料庫的初步成功鼓勵了 Chamberlin 博士與他的同事繼續進行研究。他們在 1976 至 1977 年間將 SEQUEL 改版為 SEQUEL/2，但名稱因法律問題而從 SEQUEL 換成 SQL（Structured Query Language 或 SQL Query Language）—已經有人用了 SEQUEL 這個縮寫字。此時還是有很多人將 SQL 念做“sequel”，但很多人認為“標準”的念法應該是“ess-cue-el”。

雖然 IBM 的 System/R 與 SQL 證明關聯式資料庫可行，但當時的硬體技術並不足以讓此產品提供商業上夠吸引人的效能。

1977 年一群來自加州的工程師成立 Relational Software 公司以打造稱為 Oracle 的基於 SQL 的資料庫產品。Relational Software 於 1979 年開始交付它的產品，成為第一個可購買的 RDBMS。Oracle 的優勢之一是在 Digital 的 VAX 迷你電腦而非昂貴的 IBM 主機上執行。Relational Software 後來改名 Oracle Corporation 並成為 RDBMS 軟體的領導廠商之一。

大約在同一時期，Michael Stonebraker、Eugene Wong，與其他 University of California 的 Berkeley 電腦實驗室的專家也在研究關聯式資料庫技術。它們開發了命名為 Ingres 的關聯式資料庫原型。Ingres 包括稱為 Query Language（QUEL）的資料庫語言，較 SQL 更為結構化，但句子比較不像英文。然而，SQL 明顯的會成為標準資料庫語言，因此 Ingres 最終轉換成基於 SQL 的 RDBMS。好幾個專家在 1980 年離開 Berkeley 並成立 Relational Technology 公司，他們於 1981 年發佈第一個 Ingres 的商業版本。Relational Technology 經過好幾次轉型，之前被 Computer Associates International 收購，現在是 Actian 的子公司，Ingres 現在還是資料庫業界的領導產品之一。

同時間，IBM 於 1981 年發佈稱為 SQL/Data System（SQL/DS）的 RDBMS 並於 1982 年開始交付。該公司於 1983 年發佈稱為 Database 2（DB2）的新 RDBMS 產品，可用於使用 IBM 的主流 MVS 作業系統的 IBM 主機。第一版於 1985 年交付，DB2 成為 IBM 的明星 RDBMS，其技術被整合進 IBM 的產品線。

隨著圍繞資料庫語言發展的一系列活動，資料庫社群開始提出標準化的想法。但誰來主導標準或基於哪一個版本則一直沒有產生共識，因此各家廠商繼續開發與改進自己的資料庫產品並期待一它的 SQL 版本一成為業界標準。

客戶的意見與要求使得許多廠商在它們的 SQL 中加入特定元素並出現了非官方的標準。以現在的標準來看它是較小的規格，因為它只包含各種 SQL 方言間類似的元素。但此規格（例如以前）確實提供給資料庫客戶一組核心條件來評估市場上的各種資料庫程式，且它還讓使用者知道他們可以從一個資料庫程式換到另一個資料庫。

American National Standards Institute（ANSI）於 1982 年對關聯式資料庫語言的官方標準的需求以委託 X3 資料庫技術委員會（X3H2）開發標準提案做出了回應。經過一番折騰（包括許多對 SQL 的改進），委員會發現它的新標準與現有的主要 SQL 方言不相容，而對 SQL 的改善並不足以抵消不相容的問題。因此他們倒退回到資料庫廠商必須符合的“最小公因數”。

ANSI 在 1986 年批准了“ANSI X3.135-1986 Database Language SQL”這個一般稱為 SQL/86 的標準。基本上，它賦予各種 SQL 方言間類似與廠商已經實作的元素的官方認可。雖然委員會知道這樣做的缺點，但至少新標準提供可以進一步發展語言與其實作的規格基礎。

International Organization for Standardization（ISO）於 1987 年通過了它自己的文件（完全對應 ANSI SQL/86）作為國際標準並以“ISO 9075:1987 Database Language SQL”名義公佈（兩者通常都稱為 SQL/86），現在國際資料庫廠商可與美國的廠商以相同的標準為依據。雖然 SQL 有了官方標準，但此語言還不算完整。

SQL/86 很快就受到政府與 C. J. Date 等產業大咖對於 SQL 語法的冗贅（定義相同的查詢有多種方式）、缺少特定關聯式運算，與缺少參考完整性等問題的批評。

ISO 與 ANSI 都嘗試修訂標準以解決受批評的問題，特別是參考完整性的部分。ISO 於 1989 年中公佈“ISO 9075: 1989 Database Language SQL with Integrity Enhancement”，而 ANSI 於同年稍後採用被稱為 SQL/89 的“X3.135-1989 Database Language SQL with Integrity Enhancement”。

一般認為 SQL/86 與 SQL/89 都缺少一些成功的資料庫系統所需的基本功能。舉例來說，兩者都沒有指定在資料庫結構定義完成後如何修改。它無法修改或刪除任何結構元件或改變資料庫的安全設定，然而所有廠商都在他們的商業產品中提供了執行方式（舉例來說，你可以 CREATE 資料庫物件但卻沒有定義 ALTER 或 DROP 語法）。

ANSI 與 ISO 都不想提供另一個“最小公因數”標準，兩者持續進行 SQL 的大改版以讓它完整與紮實。新版（SQL/92）包含了主要資料庫廠商已經廣泛採用的功能，但也包括未普遍採用的功能以及目前都還沒有實作的新功能。

ANSI 與 ISO 於 1992 年 10 月分別公佈新的 SQL 標準—“X3.135-1992 Database Language SQL”與“ISO/IEC 9075:1992 Database Language SQL”。SQL/92 的文件較 SQL/89 大，涵蓋面也較廣。舉例來說，它提供定義完成後修改資料庫結構的方法、支援額外的文字與日期時間操作，以及定義其他安全功能。SQL/92 較之前的版本邁出一大步。

資料庫廠商在實作 SQL/92 的功能同時也在開發與實作自有的功能，對 SQL 標準加上“擴充”。雖然擴充（像是提供比 SQL/92 所定義的六個更多的資料型別）提供更多的功能並讓廠商做出與其他廠商不同的差異，但這也是有缺點的。擴充的主要問題是它導致個別廠商的 SQL 方言與原始標準的差別越來越大，妨礙資料庫開發者建構在各種 SQL 資料庫上都能執行的可攜應用程式。

1997 年，ANSI 的 X3 組織更名為 National Committee for Information Technology Standards（NCITS），負責 SQL 標準的技術委員會現在稱為 ANSI NCITS-H2。由於 SQL 標準的複雜性快速升高，ANSI 與 ISO 標準委員會在進行 SQL3（因為是第三個大改版）時同意將標準拆開成 12 個部分與一個附錄以便每個部分可以平行推進。1997 年後又定義了兩個部分。

本書內容基於大部分資料庫系統採用的 ISO 目前的 SQL 資料庫語言標準—SQL/Foundation（ISO/IEC 9075-2:2011）。ANSI 同樣也採用此 ISO 文件，因此它是真正的國際標準。必要時我們還使用 IBM DB2、Microsoft Access、Microsoft SQL Server、MySQL、Oracle，與 PostgreSQL 的最新版本以提供個別產品的語法。雖然本書大部分的 SQL 並非專屬某個產品，適當時機還是會展示產品專用的範例。

本書涵蓋的資料庫系統

雖然前面說到 SQL 有個標準，但並非表示所有 DBMS 都相同。DB-Engines 網站在 http://db-engines.com/en/ranking/relational+dbms 有列出多家 DBMS 的資訊與排行榜。

它們的排行榜顯示有六個 DBMS 最受歡迎，以下依字母順序列出（括號中是我們進行測試的版本）：

1. IBM DB2（DB2 for Linux、UNIX，與 Windows v10.5.700.368）
2. Microsoft Access（Microsoft Access 2007—與 2010、2013、2016，以及之後的版本相容）
3. Microsoft SQL Server（Microsoft SQL Server 2012—11.0.5343.0）
4. MySQL（MySQL Community Server 5.7.11）
5. Oracle Database（Oracle Database 11g Express Edition Release 11.2.0.2.0）
6. PostgreSQL（PostgreSQL 9.5.2）

這並不代表本書內容不能用於以上六家以外的 DBMS，它只是表示我們沒有在其他 DBMS 或不同版本上測試。閱讀時你會在有需要修改的地方看到建議（註記），它們只適用於上列六個 DBMS。若你使用不同的 DBMS 並在範例中遇到問題，請參考說明文件。

範例資料庫

為展示本書內容的概念，我們使用了幾個範例資料庫，包括：

1. **Beer Styles**：根據 Michael Larson 的 *Beer: What to Drink Next*（Sterling Epicure, 2014）一書內容的 89 種啤酒的資訊。
2. **Entertainment Agency**：此資料庫用於管理藝人、經紀人、客戶，與檔期。你可以使用類似的設計來處理活動或旅館預約。
3. **Recipes**：你可以使用這個資料庫儲存與管理食譜。
4. **Sales Orders**：這是典型的訂單資料庫，儲存商品資料。
5. **Student Grades**：此資料庫列出學生、選修科目，與成績。

還有幾個範例資料庫供特定方法使用，有些是由內容中的程式產生。schema 與資料可從本書的 GitHub 網站取得。

取得在 GitHub 的範例資料

許多技術書籍附帶 CD-ROM，而我們決定將範例放在 https://github.com/TexanInParis/Effective-SQL。

你可於該網站發現我們討論的六個 DBMS 的目錄，每個目錄下有對應本書十個章節的十個目錄加上範例資料庫目錄。

每個章節的目錄下有個別檔案，其名稱對應做法的編號。注意並非每個檔案都可用於各種 DBMS，細節見 README 檔案。對 Microsoft Access，範例資料庫內容見 README 檔案。

GitHub 上的根目錄有個 Listings.xlsx 檔案記錄資料庫對應的方法，此檔案中還有適用六個資料庫系統的 SQL 範例。

除了 Microsoft Access 目錄下的 .accdb 格式檔案外，每個範例資料庫目錄下都有幾個 SQL 檔案。Microsoft Access 使用 2007 格式是因為它與 12 版（2007）之後的版本都相容。有一組檔案建構各個範例資料庫的結構，另外一組檔案帶有範例資料庫的資料（注意某些方法依靠特定的資料。有些結構與資料放在章節內容中）。

注意事項

準備本書的材料時，我們有時無法將它們壓在適合頁面的 63 個字元限制內。編輯可能有誤，因此有問題時以 GitHub 上測試過的內容為準。

總結

如書名所述，本書內容有 61 個具體做法。每個做法都是獨立的；使用時應該無須參考其他做法。當然有時候某個做法的材料是基於其他做法的材料，此時我們會對背景做說明並提供交叉參考以供你檢視。

雖然每個做法如上述是獨立的，但我們認為主題可以分為以下十類：

- **第 1 章 資料模型設計**：使用設計不良的資料模型時你無法寫出有效的 SQL，這一章討論關聯式模型的基本設計。若你的資料庫設計違反這一章討論的規則，你必須找出問題並加以改正。
- **第 2 章 程式化與索引設計**：只是好的資料模型設計並不足以讓你寫出有效的 SQL。你必須確保以正確的方式實作設計，不然你會發現你以 SQL 從資料中擷取資訊的能力受到限制。這一章的內容可幫助你了解索引的重要性以及如何正確的實作索引。
- **第 3 章 不能改變設計時**：有時候，無論你多努力，你還是得處理不受你控制的外部資料。這個做法幫助你處理這種情況。
- **第 4 章 過濾與搜尋資料**：搜尋或過濾資料是 SQL 最重要的功能之一。這一章討論精確擷取資料的不同技巧。
- **第 5 章 彙整**：SQL 標準提供了彙整資料的功能。然而，你通常是問“每個客戶的加總”、“每天的訂單數量”，或“根據類別以月為單位的平均銷售”。“每個”、“分別”，與“根據”之後的部分需要特別注意。這一章討論高效能彙整的技巧，有些技巧還展示如何使用窗口函式提供更複雜的彙整。
- **第 6 章 子查詢**：使用子查詢的方式有很多種。這一章討論各種有額外彈性的 SQL 子查詢方式。

- **第 7 章 取得與分析元資料**：有時光靠資料是不夠的，你需要資料的資料，你可能還需要知道如何取得資料的資料。在某些情況下，以 SQL 取得元資料比較方便。這一章偏向個別產品，但我們希望提供足夠的資訊讓你套用在其他 DBMS 上。
- **第 8 章 笛卡兒積**：笛卡兒積是組合兩個資料表的列的產出。或許較其他連接形式更不常見，但有些問題只能以笛卡兒積回答。
- **第 9 章 對應表**：另一個實用的工具是對應表格，它通常是具有單一連續數值，或連續日期，或其他複雜值的欄的資料表以提供彙整的“樞紐”。笛卡兒積依靠資料表中的實際值，而對應表能夠涵蓋各種可能性。這一章示範以對應表解決的各種問題。
- **第 10 章 建構階層資料模型**：在關聯式資料庫中建構階層資料模型並不罕見。不幸的是，這是 SQL 的弱項。這一章討論資料正規化與容易查詢以及維護元資料間的取捨。

每個資料庫系統有不同的函式用於計算或操作日期與時間值。每個資料庫還有自己的資料型別與日期時間運算規則。由於這些差異，我們加上“時間與日期型別、操作，以及函式”這個附錄以幫助你運用日期與時間值。我們認為它精確的列出資料型別與運算操作，但我們還是建議參考你的資料庫的說明文件。

1 資料模型設計

“你無法用母豬的耳朵製作絲綢包”。這句話出自 1579 年英國諷刺作家 Stephen Gosson，它當然也適用於資料庫。若資料庫模型設計不良，你無法撰寫“有效”的 SQL。資料模型沒有以正確的關聯定義做正規化時，你會發現很難甚至是不可能以 SQL 從資料中擷取有意義的資訊。這一章討論基本關聯式模型設計。若你的資料庫設計違反這些規則，你必須找出問題並加以改正。

若你無法控制設計，你至少要知道原因並能夠向控制設計的一方說明要如何修改。你可以使用這一章的內容來說明為何很難甚至是不可能以 SQL 從資料中擷取資訊。如果你不能修改設計，在 SQL 中有些方法能夠繞開問題。若你遇到這種情況，請參考第 3 章“不能改變設計時”。

我們並不打算涵蓋資料庫設計的所有細節，只有基本部分。若你想要更深入的認識如何設計關聯式模型，請參考 Michael J. Hernandez 所著的 *Database Design for Mere Mortals 第三版*（Addison-Wesley, 2013）等設計專書。

做法 01 確定所有資料表都有主鍵

由於關聯式模型要求資料庫系統能夠從資料表的所有列中區分單一列，每個資料表都應該有一或多個欄作為主鍵。主鍵的內容必須唯一與獨特且不能為空（更多空的細節見做法 10：“建構索引時的空”）。沒有主鍵就無法確保過濾時只有零或一個列相符，但“問題”是建構沒有主鍵的資料表是合法的。事實上，具有一或多個非空且獨特的欄並不表示資料庫引擎能夠有效的使用它們，你必須透過定義主鍵以明確的告訴資料庫引擎這件事。此外，建構兩個資料表間的關聯式模型時通常不能（或不想要）沒有定義主鍵。

資料表缺少主鍵會產生各種問題，包括重複與不一致的資料、查詢跑很慢，與報表中不正確的資訊！以圖 1.1 所示的 Orders 資料表為例。

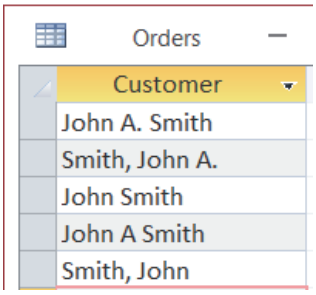

圖 1.1 不一致的資料

在圖 1.1 中，所有的值對電腦來說是獨特的，但或許它們都是同一個人，至少列 1、2、4 都是（John A. Smith）。雖然電腦處理速度比人腦快，但它們在沒有大量程式設計下是不太能判斷這種資料。雖然我們可以將 Customer 欄定義為該資料表的主鍵，但就算滿足了唯一獨特的需求，它也不是個好選擇。

所以什麼才是好的主鍵？欄應該具有下列特徵：

- 儲存獨特值
- 不能為空
- 穩定（也就是不會改變）
- 盡可能簡單（例如使用整數資料型別而非浮點數或字元，單一欄位而非多個欄位）

達成此目標的一個常見方法是使用自動產生的無意義數值作為主鍵，其名稱視所使用的關聯式資料庫管理系統（RDBMS）軟體而定，例如 IBM DB2、Microsoft SQL Server，與 Oracle 12c 的 IDENTITY、Microsoft Access 的 AutoNumber、MySQL 的 AUTO_INCREMENT，與 PostgreSQL 的 serial。在 Oracle 之前版本中必須使用 Sequence 物件執行類似的服務，但它是獨立的物件而非欄屬性。DB2、SQL Server，與 PostgreSQL 也支援 Sequence 物件。

參考完整性（RI）是關聯式資料庫中非常重要的概念。強制 RI 表示具有非空的外來鍵的子資料表中的每個列必須在父資料表中有相對應的紀錄。

在設計良好的 Orders 資料表中，客戶資訊會來自外來鍵對應 Customers 資料表主鍵。若有多個客戶姓名都是 John Smith，每個客戶列會有不同的獨特鍵值，因此能夠辨識每個訂單的客戶。

為了維持資料表間的 RI，主鍵值的改變必須串聯到關聯資料表中有關聯的子紀錄。此修改的串聯會將關聯資料表鎖住而使得同時有大量使用者的資料庫產生嚴重的問題。以圖 1.2 所示摘自 Microsoft Access 2003 的 Northwind 範例資料庫中的 Customers 資料表為例。

Results Messages

	CustomerID	CompanyName	ContactName	ContactTitle
1	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
2	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner
3	ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner
4	AROUT	Around the Horn	Thomas Hardy	Sales Representative
5	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator
6	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative

圖 1.2 Customers 資料表

此例中，我們假設有商業規則表示 CustomerID 這個文字主鍵與公司名稱相關。若有個公司改名，則 CustomerID 應該跟著修改以反映主鍵值設定的商業規則。這需要串聯修改到相關聯的資料表。若使用無意義的鍵，你可以避免修改其值且以文字欄來儲存符合該商業規則的值。

文字主鍵常被認為可以防止輸入重複資料。舉例來說，若以 CompanyName 作為主鍵，你可以確保名稱不會重複。但對 Customers 資料表的 CompanyName 建構獨特索引也可以確保名稱不會重複。完整性受到保證，但你還是可以用產生出的數值作為主鍵。這在你採用做法 2：“消除重複儲存資料”與做法 4：“每個欄只儲存一個屬性”時特別有用，它可以幫助你避免圖 1.1 所示的問題。另一方面，使用文字主鍵通常能因避免聯集所需的資料表搜尋以取得數值鍵相關值（圖 1.2 所示的 CompanyName）而簡化 SQL 陳述。

數值主鍵與文字主鍵的選擇在資料庫專家間存在爭議。我們不偏向任何一方；重點是在所有資料表上以獨特值作為主鍵。

我們還建議不要使用複合主鍵，它們因兩個理由而比較沒效率：

1. 定義主鍵時，大部分資料庫系統會強制以獨特索引定義。一個欄以上的獨特索引需要資料庫系統執行更多的工作。
2. 對主鍵執行聯集是相當常見的，但在多欄主鍵上執行則更為複雜與緩慢。

然而，在某些情況下使用多欄的主鍵是合理的。以具有 VendorID 與 ProductID 指向關聯資料表主鍵的連結產品與廠商的資料表來說，該資料表或許有其他表示廠商是該產品主要或次要供應商以及產品價格的欄。

你可以建構產生數值的欄作為人工主鍵，但你也可以組合 VendorID 與 ProductID 作為主鍵。你會以個別欄來連結到此資料表，所以定義複合主鍵比定義額外的欄或許會更有效率。你會將兩個欄位合起來定義為獨特，因此避免額外欄並定義複合主鍵是合理的。更深入的複合主鍵範例見做法 8：“3NF 不夠時，更多的正規化”。

重點摘要

- 所有的資料表都應該指定一個欄（或一組欄）作為主鍵。
- 若對不是鍵的欄有重複值的疑慮，你可以對該欄定義獨特索引以確保完整性。
- 盡可能使用簡單且不會變更的鍵。

做法 02　消除重複儲存資料

重複儲存資料會導致很多問題，包括不一致的資料；新增、修改、刪除異常；以及浪費磁碟空間。正規化是以類別拆分資訊來消除儲存重複資料的問題的程序。注意“重複”並非指某個資料表的主鍵值與另一個資料表的外來鍵之間明顯的重複，這種重複是維護資料表間關聯所必須的。我們更關注使用者在不同地方輸入相同資料的情況。

雖然我們因為篇幅關係無法深入資料庫正規化這個主題，但完整了解它對資料庫從業人員非常重要。網路上與很多書籍都有非常棒的深入討論。

正規化的一個目標是相同資料表或不同資料表中重複資料的最小化。重複儲存資料的例子見圖 1.3 所示的 Customer Sales 資料庫。

其中一個資料不一致的例子是 Tom Frank 的地址。在第二筆紀錄中，他的地址的數值部分是 7453，而第六筆資料的數值部分是 7435。類似的資料不一致也會發生在任何一欄。

插入異常是因為你無法輸入某個車型的資料，除非已經輸入一筆客戶銷售紀錄。還有，此設計要求某個客戶買更多車時要重複輸入大部分的資料。不必要的資料會浪費磁碟空間、記憶體、網路資源、甚至是資料輸入工時。此外，重複的資料輸入大幅的提升資料輸入錯誤的風險，例如圖 1.3 所示的地址數字錯誤。

CustomerSales

SalesID	CustFirstName	CustLastName	Address	City	Phone	
1	Amy	Bacock	111 Dover Lane	Chicago	312-222-1111	...
2	Tom	Frank	7453 NE 20th St.	Bellevue	425-888-9999	...
3	Debra	Smith	3223 SE 12th Pl.	Seattle	206-333-4444	...
4	Barney	Killjoy	4655 Rainier Ave.	Auburn	253-111-2222	...
5	Homer	Tyler	1287 Grady Way	Renton	425-777-8888	...
6	Tom	Frank	7435 NE 20th St.	Bellevue	425-888-9999	...

	PurchaseDate	ModelYear	Model	SalesPerson
...	2/14/2016	2016	Mercedes R231	Mariam Castro
...	3/15/2016	2016	Land Rover	Donald Ash
...	1/20/2016	2016	Toyota Camry	Bill Baker
...	12/22/2015	2016	Subaru Outback	Bill Baker
...	11/10/2015	2016	Ford Mustang GT Convertible	Mariam Castro
...	5/25/2015	2015	Cadillac CT6 Sedan	Jessica Robin

圖 1.3 單一資料表中的重複資料

舉例來說，更新異常可能發生於某人因為結婚而更改姓氏時必須以更新查詢更新該人的所有姓名紀錄，這在多人同時使用的大型資料庫上會是個挑戰。此外，這種更新只會在該人的所有姓名紀錄拼寫都完全一致（沒有不一致的資料）且沒有他人的姓名相同時才會成功。

刪除異常發生於刪除一列同時刪除本來沒有打算要刪除的資料。

圖 1.3 所示的銷售資料在邏輯上可拆分成四個資料表：

1. `Customers` 資料表（姓名、地址等）
2. `Employees` 資料表（業務員姓名、到職日期等）
3. `AutomobileModels` 資料表（車型、年份等）
4. `SalesTransactions` 資料表

這種設計可讓你輸入客戶、員工，與車輛資訊到相對應的資料表。所有的資料表都有獨特識別資料作為主鍵。`SalesTransactions` 資料表以外來鍵儲存每一筆交易的細節，見圖 1.4。

Customers

CustomerID	CustFirstName	CustLastName	Address	City	Phone
1	Amy	Bacock	111 Dover Lane	Chicago	312-222-1111
2	Tom	Frank	7453 NE 20th St.	Bellevue	425-888-9999
3	Debra	Smith	3223 SE 12th Pl.	Seattle	206-333-4444
4	Barney	Killjoy	4655 Rainier Ave.	Auburn	253-111-2222
5	Homer	Tyler	1287 Grady Way	Renton	425-777-8888

Employees

EmployeeID	SalesPerson
1	Mariam Castro
2	Donald Ash
3	Bill Baker
4	Jessica Robin

AutomobileModels

ModelID	ModelYear	Model
1	2016	Mercedes R231
2	2016	Land Rover
3	2016	Toyota Camry
4	2016	Subaru Outback
5	2016	Ford Mustang GT Convertible
6	2015	Cadillac CT6 Sedan

SalesTransactions

SalesID	CustomerID	ModelID	SalesPersonID	PurchaseDate
1	1	1	1	2/14/2016
2	2	2	2	3/15/2016
3	3	3	3	1/20/2016
4	4	4	3	12/22/2015
5	5	5	1	11/10/2015
6	2	6	4	5/25/2015

圖 1.4 依主題切開資料表

精明的讀者可能會注意到這個過程中刪除了一筆重複的客戶紀錄並留下 Tom Frank 的正確地址。

我們可以如圖 1.4 所示連接三個資料表（Customers、AutomobileModels，與 Employees）的主鍵與 SalesTransactions 資料表的外來鍵欄以建構關聯。我們以 Microsoft Access 的關聯編輯工具建構圖例所示的範例，每個關聯式資料庫以不同的方式表示資料表間的關係。

你可以用程式列 1.1 所示的方式建構虛擬資料表（查詢）來重新建構圖 1.3 所示的資料而不會因儲存重複的資料產生問題（建構虛擬資料表是簡稱為 CTE 的通用資料表運算式最好的用途，見做法 42：“可能的話，以通用資料表運算式替代子查詢”）。

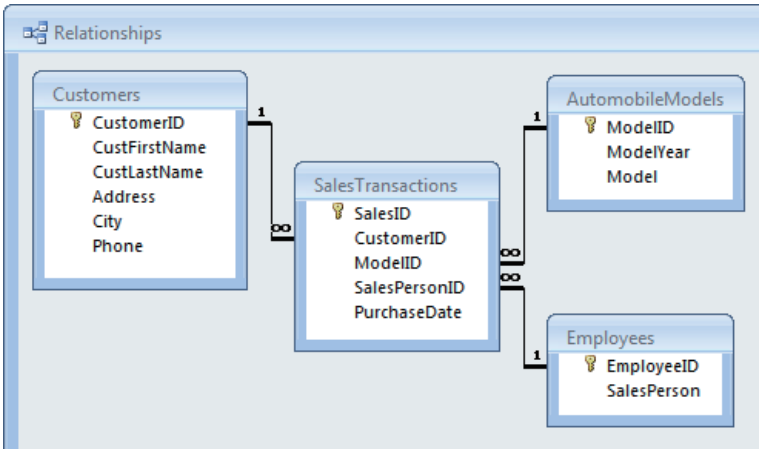

圖 1.5 以主鍵連接外來鍵欄關聯四個資料表

程式列 1.1 回傳原始資料的 SQL 陳述

```
SELECT st.SalesID, c.CustFirstName, c.CustLastName, c.Address,
  c.City, c.Phone, st.PurchaseDate, m.ModelYear, m.Model,
  e.SalesPerson
FROM SalesTransactions st
  INNER JOIN Customers c
    ON c.CustomerID = st.CustomerID
  INNER JOIN Employees e
    ON e.EmployeeID = st.SalesPersonID
  INNER JOIN AutomobileModels m
    ON m.ModelID = st.ModelID;
```

重點摘要

- 資料庫正規化的目的是消除重複資料並減少處理資料時所需的資源。
- 消除重複資料可消除新增、修改，與刪除異常。
- 消除重複資料可減少不一致的資料。

參考

如果你想要正確的設計關聯式資料庫，以下是建議書單；第一本適合新手閱讀：

- Hernandez, Michael J. *Database Design for Mere Mortals* (Addison-Wesley, 2013). ISBN-10: 0-321-88449-3.
- Fleming, Candace C., and Barbara von Halle. *Handbook of Relational Database Design* (Addison-Wesley, 1989). ISBN-10: 0-201-11434-8.

做法 03　去除重複群組

重複的資料群組在試算表中很常見。使用者通常會匯入資料到資料庫而沒有考慮到資料庫的正規化。重複資料群組的例子如圖 1.6 所示，其中 `DrawingNumber` 最多與五個 `Predecessor` 相關。此資料表在 drawing number 與 predecessor 值之間有一對多的關係。

Assignments

ID	DrawingNumber	Predecessor_1	Predecessor_2	Predecessor_3	Predecessor_4	Predecessor_5
1	LO542B2130	LS01847409	LS02390811	LS02390813	LS02390817	LS02390819
2	LO426C2133	LS02388410	LS02495236	LS02485238	LS02495241	LS02640008
3	LO329W2843-1	LS02388418	LS02640036	LS02388418		
4	LO873W1842-1	LS02388419	LS02741454	LS02741456	LS02769388	
5	LO690W1906-1	LS02742130				
6	LO217W1855-1	LS02388421	LS02769390			

圖 1.6　單一資料表中的重複群組

圖 1.6 所示的 `Predecessor` 是重複的群組，而 `ID = 3` 的紀錄有重複的 `Predecessor` 值。同樣的例子包括以 January、February、March（或 Jan、Feb、Martin）命名的欄，但重複的群組不僅限於單一屬性。舉例來說，若欄

位命名為 Quantity1、ItemDescription1、Price1、Quantity2、ItemDescription2、Price2 … QuantityN、ItemDescriptionN、PriceN，你應該將它們視為重複群組。

重複群組難以查詢並難以用屬性產生分群的報表。在圖 1.6 所示的範例中，若你需要增減 Predecessor 的數量，現在的設計需要增減資料表的欄。你還需要修改所有相關的檢視表、表單，與報表。一個有用的口訣是：

欄代價高。

列代價低。

若資料表的設計在未來需要類似資料時必須增減欄就要警惕。更好的設計應該是在有需要時增減列。舉例來說，我們建構 Predecessors 資料表並以 ID 值作為外來鍵。我們還將現有的 ID 欄重新命名為更清楚表示的 DrawingID，如圖 1.7 所示。

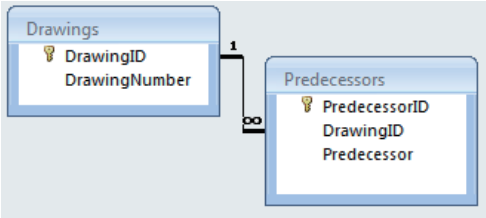

圖 1.7 適合一對多關係的正規化設計

UNION 查詢對重複群組的處理很有用。若無法建構正規化的設計，我們可以使用 UNION 查詢將唯讀的檢視表中的資料“正規化”。我們也可以使用類似的 UNION 查詢作為新增查詢的來源以將新的紀錄新增至新的 Predecessors 資料表中，如程式列 1.2 所示。

程式列 1.2 將資料正規化的 UNION 查詢

```
SELECT ID AS DrawingID, Predecessor_1 AS Predecessor
FROM Assignments WHERE Predecessor_1 IS NOT NULL
UNION
SELECT ID AS DrawingID, Predecessor_2 AS Predecessor
FROM Assignments WHERE Predecessor_2 IS NOT NULL
UNION
SELECT ID AS DrawingID, Predecessor_3 AS Predecessor
FROM Assignments WHERE Predecessor_3 IS NOT NULL
```

```
UNION
SELECT ID AS DrawingID, Predecessor_4 AS Predecessor
FROM Assignments WHERE Predecessor_4 IS NOT NULL
UNION
SELECT ID AS DrawingID, Predecessor_5 AS Predecessor
FROM Assignments WHERE Predecessor_5 IS NOT NULL
ORDER BY DrawingID, Predecessor;
```

注意事項

若需要所有資料，包括列中重複的資料，我們可以在每個 UNION 關鍵字後面加上 ALL 關鍵字，變成 UNION ALL。但此例中我們需要消除 ID 3 中重複的 Predecessor。

UNION 查詢要求每個 SELECT 陳述的欄資料型別相同，以及相同的順序。這表示它不需要在第一個實例後加上 AS DrawingID 或 AS Predecessor：UNION 查詢使用第一個 SELECT 的欄位名稱。

每個 SELECT 陳述可以有不同的 WHERE 條件。依資料而定，我們也可以排除零長度字串（ZLS）和 / 或其他不列印的格式，例如單個空白（' '）。

UNION 查詢在最後可加上一個 ORDER BY。我們可以指定序數參考，例如 ORDER BY 1, 2。這與程式列 1.2 的 ORDER BY DrawingID, Predecessor 相同。

重點摘要

- 資料庫正規化的目標是消除重複資料群組並減少 schema 的變動。
- 消除重複資料群組後，你可以使用索引來防止重複的資料並大幅簡化查詢。
- 消除重複資料群組讓設計更為彈性，因為加入新的群組只需要另一個資料列而不用改變資料表設計以增加欄。