前言

歷年來，各種程式語言興衰更迭，現役的程式語言中，很少發跡於超過十年以前。少數的例子包括 COBOL，這是一種在大型主機中仍十分常見的程式語言；其次是 Java，它誕生於 1990 年代中期，但已成為最受歡迎的程式語言之一；而 C 語言則仍受到作業系統、伺服器開發及嵌入式系統相當程度的歡迎。至於資料庫的領域，就不能不提到 SQL，它甚至可以溯源至 1970 年代。

SQL 最早的起源，是作為一種從關聯式資料庫中產生、操作及檢索資料的語言，從當時至今已逾 40 餘載。然而在過去十年左右，又興起了其他的資料平台，像是 Hadoop、Spark 和 NoSQL 等等，它們引起了相當的注目，同時也侵蝕了關聯式資料庫的市場。然而正如本書最後幾個章節中所探討的，SQL 語言已經發展成可以從各種平台檢索資料，而不限於資料表、文件或普通檔案等資料儲存方式。

為何要學習 SQL ？

無論你是否使用關聯式資料庫，只要從事資料科學、商務情報、或是其他跟資料分析有關的工作，可能都得對 SQL 有所了解，同時還必須熟悉其他的語言 / 平台，像是 Python 和 R 等等。資料無所不在，它們為數龐大、累積的速度又快，能從這些資料中理出頭緒的人，在業界可說是炙手可熱。

碁峰 www.gotop.com.tw

為何要從本書著手？

坊間許多書籍往往先將讀者視為一無所知，但這類書籍常流於淺薄。另一種書籍則屬於參考指南型，它鉅細靡遺地說明語言中每道敘述的所有面向，如果你已對自己的目標心知肚明，只缺如何達成目的的語法，那前述第二種書會很有用。

但本書居於這兩種角色之間，筆者會先從若干 SQL 語言的基礎開始，藉由基本知識，逐步進展到一些更為進階的功能，讓你真正掌握其中訣竅。此外，本書最後一章還會介紹如何查詢非關聯式資料庫中的資料，這是入門書籍鮮少觸及的題材。

本書結構

本書共分十八章、加上兩篇附錄：

第 1 章，一點背景知識

簡單介紹電腦資料庫的過往歷史，包括關聯式模型及 SQL 語言的興起。

第 2 章，建立並填製資料庫

說明如何建置本書要用到的 MySQL 資料庫及資料表，並將資料填入其中。

第 3 章，基礎查詢

介紹 `select` 敘述，並進一步展示最常見的子句（`select`、`from`、`where`）。

第 4 章，篩選

介紹可以用在 `select`、`update` 或是 `delete` 等敘述的 `where` 子句中的各種條件類型。

第 5 章，查詢多個資料表

說明如何透過資料表結合來查詢多組資料表。

第 6 章，集合的運用

說明資料集合的始末，以及如何在查詢語句中與它們互動。

第 7 章，資料的產生、操作與轉換

介紹數種可用於操作或轉換資料的內建函式。

碁峯 www.gotop.com.tw

第 8 章，分組與彙整

說明如何彙整資料。

第 9 章，子查詢

介紹子查詢（個人偏好），並說明在何處及如何應用它們。

第 10 章，再談結合

進一步探索各種類型的資料表結合動作。

第 11 章，條件邏輯

探索如何將條件邏輯（如 if-then-else）運用在 `select`、`insert`、`update` 和 `delete` 等敘述當中。

第 12 章，交易

介紹交易及其運用方式。

第 13 章，索引與約束條件

探索索引與約束條件。

第 14 章，Views

說明如何建置獨特的介面，藉以將使用者和資料複雜性區隔開來。

第 15 章，中繼資料

解說資料字典的作用。

第 16 章，分析函式

包括各種用於產生排行、小計、及其他常用於報表與分析所需數值的函式。

第 17 章，操作大型資料庫

介紹如何將超大型資料庫變得易於管理及檢閱的技術。

第 18 章，SQL 與大數據

探索 SQL 語言的轉型，理解如何從非關聯式的資料平台檢索資料。

附錄 A，範例資料庫的 ER 關係圖

展示本書中所有範例所使用的 database schema。

碁峰 www.gotop.com.tw

第一章

一點背景知識

在我們正式捲起袖子來幹活之前，先來回顧資料庫技術的過往，同時追溯一下關聯式資料庫與 SQL 語言發展的緣起，應該會有所助益。因此筆者會先從若干基礎的資料庫概念講起，同時回顧一下運算式資料儲存及檢索的歷史。

如果你是那種想趕快開始學習撰寫查詢的讀者，儘管放心地跳到第 3 章去繼續，但筆者建議各位還是在稍後轉過頭來回顧一下第 1 與第 2 兩章，這樣才能了解 SQL 語言的歷史與效用。

資料庫簡介

所謂的資料庫（*database*），說穿了不過是若干彼此有關資訊的集合。舉例來說，一本電話簿就是由住在特定區域中的所有人名、電話號碼及地址構成的資料庫。不過電話簿雖然確實是個無所不在、而且運用頻繁的資料庫，它卻有以下的缺陷：

- 要找出某人的電話號碼十分耗時，尤其是當電話簿篇幅浩繁的時候。
- 電話簿僅以姓名作為索引，因此若想反過來找出住在特定地址的人名時，雖說理論上不無可能，實務上卻不是這類資料庫該有的使用方式。

碁峯
www.gotop.com.tw

- 從電話簿印刷成篇的那一刻起，其中的資訊就開始日趨偏差，因為人們會不定期地移入或遷離電話簿所涵蓋的這個區域、或是更改電話號碼、甚至是在同一區內搬遷，都會造成電話簿資訊的失準。

電話簿所帶有的一切缺陷，同樣也會發生在其他手動維護的資料儲存系統上，譬如檔案櫃裡的病歷。由於紙本資料庫具有如此繁瑣的本質，故而在電腦剛問世之時，最先為人開發出來的應用程式之一，就是*資料庫系統*，這是一種儲存及檢索運算資料的機制。由於資料庫系統是以電子化的形式取代紙本儲存，因此它的資料檢索起來更為迅速，又可以透過多種方式進行索引，還可以對使用者隨時提供最新近的資訊。

早期的資料庫系統所管理的資料，係儲存在磁帶上。由於磁帶的數量往往遠多於可以讀取磁帶的機器，於是技術人員只有在需要特定資料時，才會裝卸磁帶。當時的電腦記憶體仍屬有限，因此若有多筆取得相同資料的請求，通常就得一再地從同一捲磁帶中讀取多次。雖說此種資料庫已比紙本資料庫大有進步，與今日的技術相較仍相去甚遠（現代的資料庫系統管理的資料動輒達數個 petabytes，可以透過伺服器叢集取得，而這些伺服器中往往又以高速記憶體暫存了多達數十個 gigabytes 的資料，不過這些都有點離題了。）

非關聯式的資料庫系統

這個小節中含有一些關聯式資料庫問世以前的資料庫系統背景知識。如果讀者們亟於開始鑽研 SQL，儘管跳過這幾頁，到下一小節讀下去。

在運算式資料庫系統問世的早期，資料會以幾種迥異於今日的方式儲存、並呈現給使用者。以*階層式資料庫系統*（*hierarchical database system*）為例，資料會以一個以上的樹狀結構來呈現。圖 1-1 展示的就是 George Blake 和 Sue Smith 兩位客戶的銀行帳戶相關資料，以樹狀結構呈現時的外觀。

碁峰 www.gotop.com.tw

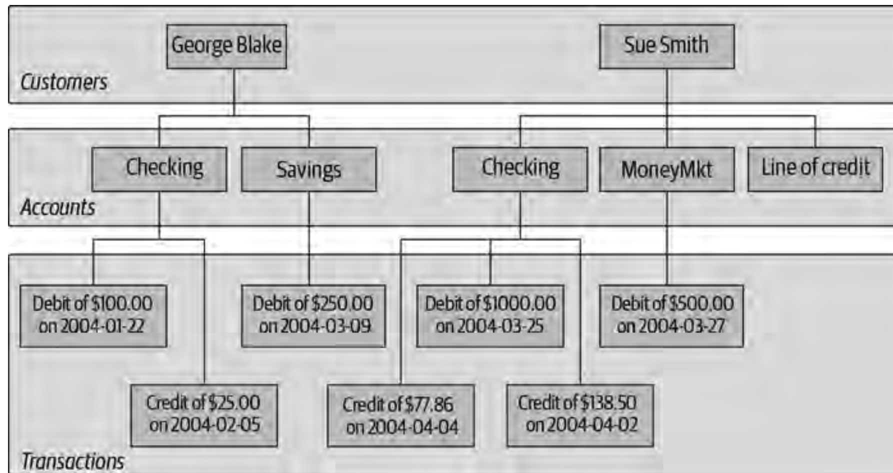

圖 1-1　帳戶資料的階層式外觀

George 與 Sue 兩人都擁有自己的資料樹，其中含有帳戶及各自的交易紀錄。而樹狀資料庫系統所提供的工具，可以找出特定客戶的資料樹、然後循線找出所需的帳戶和交易紀錄。樹中的每一個節點都會有一個上層可以追溯（但頂層就沒有上層可以追溯了）、也會有一個或多個下層可以伸展（自然底層也不會有下層可以伸展）。這種組態就是知名的**單源階層**（*single-parent hierarchy*。）

另一種常見的方式，則稱為**網路式資料庫系統**（*network database system*），它定義了一連串紀錄的集合，同時又定義了許多連結的集合，而這些連結則決定了前述不同紀錄之間的關係。圖 1-2 顯示的便是與上述相同的 George 和 Sue 的帳戶，在這種系統中的外觀。

碁峰
www.gotop.com.tw

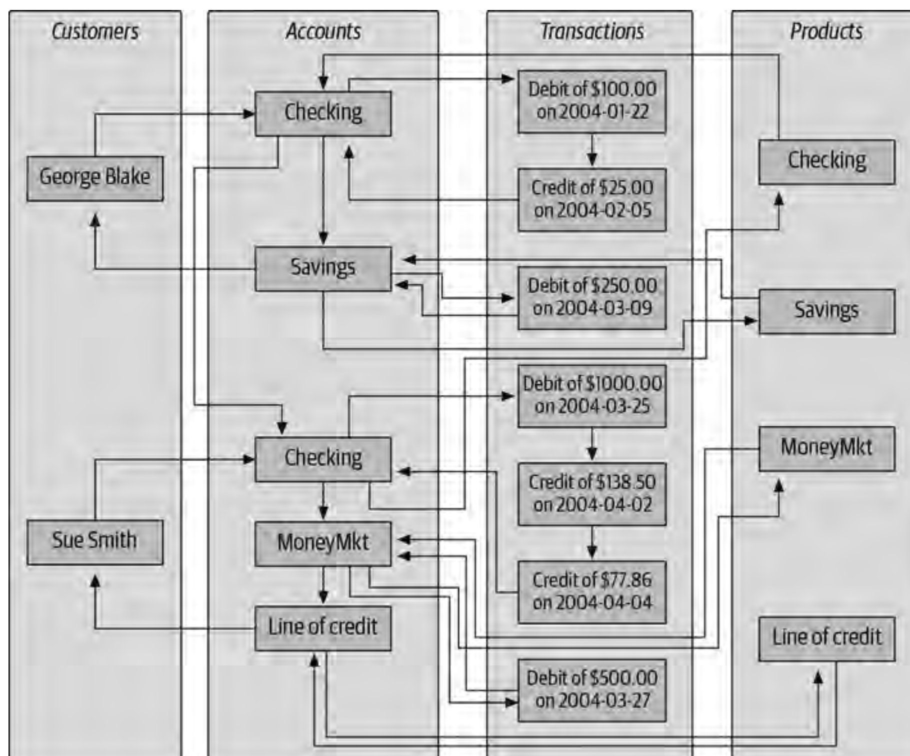

圖 1-2　帳戶資料的網路式外觀

為了找出 Sue 的金融交易帳戶（money market account）中的交易，你得透過以下的步驟：

1. 先找出 Sue Smith 的客戶紀錄。
2. 再順著連結，從 Sue Smith 的客戶紀錄找出她的帳戶清單。
3. 然後沿著帳戶鏈找下去，直到找到你找到她的金融交易帳戶為止。
4. 然後再沿著金融交易帳戶紀錄，找出交易清單

網路式資料庫系統的有趣功能之一，就是圖 1-2 中最右側所展示的產品紀錄集合。注意，每一種 product 紀錄（支票、儲蓄等等）都會指向 account 紀錄中對應該產品類別的清單。因此 account 的紀錄必可從多處取得（包括 customer 紀錄和 product 紀錄），因此網路式資料庫有時也被視為以**多源階層**（*multiparent hierarchy*）運作。

碁峯 www.gotop.com.tw

不論是階層式、還是網路式資料庫系統，至今都仍常為人所運用，不過它們多半活躍於大型主機環境。此外，階層式資料庫系統也在目錄服務（directory services）的領域大為活躍，例如微軟的 Active Directory 和開放原始碼的 Apache Directory Server 等等。然而到了 1970 年代，一種嶄新的資料呈現方式開始萌芽，這種作法更為嚴謹、然而卻相對容易理解與實現。

關聯式模型

1970 年，IBM 研究實驗室的 E. F. Codd 博士出版了題為「A Relational Model of Data for Large Shared Data Banks」（大型共用資料庫的關聯式資料模型）的論文，其中倡議資料可以用一系列的**資料表**（*tables*）來加以呈現。這裡揚棄了原本以指標在相關資料實體（entities）之間遊走的方式，而是以實體間彼此重複的資料為基礎，將位於不同資料表的紀錄連結起來。圖 1-3 便展示了 George 和 Sue 的帳戶資訊在這種概念下的呈現方式。

圖 1-3 中的四個資料表，分別呈現了到目前為止所探討的四種資料實體：`customer`、`product`、`account` 和 `transaction`。請看圖 1-3 上方的 `customer` 資料表，你會發現三個**欄位**（*columns*）：`cust_id`（其中含有客戶的 ID 序號）、`fname`（其中含有客戶的名字）、以及 `lname`（其中含有客戶的姓氏）。在 `customer` 資料表中，你會看到兩**列**（*rows*）資料，其中一筆便是 George Blake 的資料、另一筆則是 Sue Smith 的資料。一個資料表裡可以有多少個欄位，各種伺服器之間皆不盡相同，但通常都足以應付需求（以微軟的 SQL Server 為例，每個資料表最多可以有 1,024 個欄位）。至於資料表中最多可以有多少筆資料，就只是物理限制（例如磁碟機空間）、以及可維護性（例如資料表成長到什麼程度就會導致難以操作）的問題，而非資料庫伺服器自身限制的問題了。

關聯式資料庫中的每個資料表，都會含有某種資訊，可以獨一無二的識別資料表中的某筆資料（這就是所謂的**主鍵**（*primary key*）），此外還有其他可以用來完整描述資料實體的額外資訊。再度觀察 `customer` 資料表，`cust_id` 欄位，其中便含有每個客戶間彼此互異的數字；以 George Blake 為例，就可以透過客戶 ID 第 1 號來加以識別。其他客戶決不會被分配到雷同的識別碼，此外也不需透過其他資訊來找出 `customer` 資料表中 George Blake 的資料。

碁峰 www.gotop.com.tw

Customer

cust_id	fname	lname
1	George	Blake
2	Sue	Smith

Account

account_id	product_cd	cust_id	balance
103	CHK	1	$75.00
104	SAV	1	$250.00
105	CHK	2	$783.64
106	MM	2	$500.00
107	LOC	2	0

Product

product_cd	name
CHK	Checking
SAV	Savings
MM	Money market
LOC	Line of credit

Transaction

txn_id	txn_type_cd	account_id	amount	date
978	DBT	103	$100.00	2004-01-22
979	CDT	103	$25.00	2004-02-05
980	DBT	104	$250.00	2004-03-09
981	DBT	105	$1000.00	2004-03-25
982	CDT	105	$138.50	2004-04-02
983	CDT	105	$77.86	2004-04-04
984	DBT	106	$500.00	2004-03-27

圖 1-3　帳戶資料的關聯式外觀

每一種資料庫伺服器都自有一套機制，用於產生獨特的一組數字，以便作為主鍵的值來使用，因此你無須擔心如何追蹤哪些序號已經用於分配。

雖說我也可以用 `fname` 和 `lname` 兩個欄位來組成主鍵（含有兩個以上的欄位組成的主鍵，又被稱為（複合鍵，*compound key*），但很有可能發生銀行帳戶中有多人同名同姓的狀況。因此筆者會特別選擇只以 `customer` 資料表的 `cust_id` 欄位作為主鍵欄位。

碁峰 www.gotop.com.tw

在上例中，選擇 fname/lname 作為主鍵的作法，稱為**自然鍵**（*natural key*），若是選用 *cust_id* 作為主鍵，便稱為**代理鍵**（*surrogate key*）。至於要使用自然鍵或是代理鍵，完全要看資料庫設計者而定，但在以上的案例中，答案很明顯，因為人的姓氏可能會變動（例如因婚姻承續配偶姓氏），而主鍵欄位的值一旦分配、是不得再變動的。

有些資料表中還會含有可以用於瀏覽其他資料表的資訊；這便是先前提過的所謂「實體間彼此重複的資料」（redundant data）。舉例來說，account 資料表中也有一個欄位是 cust_id，它代表的是該開戶的客戶獨有的識別碼，此外還有 product_cd 欄位，這代表的則是該帳戶所屬產品類型的獨特識別碼。這兩個欄位就是所謂的**外來鍵**（*foreign keys*），至於它們的用途，就跟階層式和網路式資料庫中串接帳戶資訊的那些箭頭是一樣的。如果你正在檢視某一筆特定帳戶紀錄、並想進而查詢其開戶客戶的相關資訊，就可以透過該帳戶的 cust_id 欄位值，用它去找出 customer 資料表中對應的那筆資料（在關聯式資料庫的術語裡，這個過程便是我們熟知的**結合**（*join*）；第 3 章時會介紹結合的概念，第 5 章和第 10 章還會再深入探討到它）。

表面上看起來，將相同的資料重複多次儲存在各處，似乎顯得浪費，但是關聯式模型中的資料重複儲存方式是有明確定義的。舉例來說，在 account 資料表中加上含有客戶獨特識別碼的欄位、藉以代表開戶的客戶資料，這是正確的作法，但若是在 account 資料表中也納入客戶的姓名欄位，則是不當的作法。舉例來說，萬一有客戶改冠夫姓，你就得確認資料庫是否只有一處存有該客戶的名稱資料；不然就會有一處資料異動、它處卻未隨之異動的副作用，如此便會導致資料庫中的資料不夠精確可靠。這類資料的正確位置應該是 customer 資料表，而且只有 cust_id 這個值可以用在其他資料表中。此外，以單一欄位容納多項資訊，像是以 name 欄位同時收錄姓氏與名字，或是以 address 欄位同時收錄街道、城市、省分、郵遞區號的資訊，也是不適當的做法。將資料庫設計加以精煉、以確保各個彼此獨立的資訊片段只會出現在一處（外來鍵不算）的這個過程，就叫做**正規化**（*normalization*）。

回到圖 1-3 的四個資料表，讀者們也許會想，那要如何從這四張表找出 George Blake 的支票帳戶交易呢？首先，你得先在 customer 資料表中找到代表 George Blake 的獨特識別碼。然後再到 account 資料表中比對，找出哪一筆帳戶資料的 cust_id 欄位符合 George 的獨特識別碼，而且該筆資料的 product_cd 欄位的內容，還要和 product 資料表中 name 一欄的內容「Checking」相符合。最後到 transaction 資料表中，找出與以上那筆帳戶資料的 account_id 識別碼相符的交

碁峰 www.gotop.com.tw

易資料。聽起來也許很複雜，但你只需一行 SQL 語言的命令便能解決，這一點各位馬上就會學到。

一些名詞

筆者在先前的小節中已經提到過若干新穎的術語了，現在該來正式地定義一下。表 1-1 列出了我們會在本書後續篇幅中用到的術語，以及相關的定義。

表 1-1　術語與說明

名詞	說明
資料實體	代表資料庫使用者需要的內容。像是客戶啦、零件啦、地理位置等等，都算是資料實體。
資料欄位	儲存在資料表中的個別資訊片段。
資料列	一個由欄位構成的集合，足以完整描述一個資料實體、或是對某個實體的操作。有時也稱為一筆紀錄。
資料表	一個資料列的集合，可以存在於記憶體中（暫時存在的）或是永久性儲存體中（持續存在的）。
結果集合	非持續性資料表的別稱，通常是一道 SQL 查詢的結果。
主鍵	可以用來獨特識別資料表中每一列資料的欄位，可以由一個或多個欄位組成。
外來鍵	可以用來獨特識別其他資料表中每一列資料的欄位，可以由一個或多個欄位組成。

SQL 是什麼？

Codd 除了定義出關聯式模型之外，他還提出了一種稱為 DSL/Alpha 的語言，用來操作關聯資料表裡的資料。就在 Codd 的論文公開後不久，IBM 便按照 Codd 的觀念，委託一個團隊建構出了一套原型。該團隊打造出來的，是一套精簡過的 DSL/Alpha，並命名為 SQUARE。而 SQUARE 不斷改進的成果，便衍生出名為 SEQUEL 的語言，後來就簡稱為 SQL。雖說 SQL 一開始是用在關聯式資料庫中操作資料的語言，後來卻成長（在本書尾聲時會看到）並蛻變成一隻可以在各種資料庫技術間通用的資料操作語言。

SQL 問世已逾 40 載，而且長年以來也發生了不少變化。到了 1980 年代中期，美國國家標準協會（American National Standards Institute，ANSI）開始制定第一套 SQL 語言標準，然後在 1986 年公布。隨後又分別在 1989、1992、1999、2003、2006、2008、2011 和 2016 年推出了 SQL 的修訂標準。隨著 SQL 核心語言的改

碁峯
www.gotop.com.tw

進，都會再加上新的功能，藉以整合物件導向功能、及其他的新事物。最近的標準則專注於整合相關的技術，像是可延伸標記語言（extensible markup language，XML）和 JavaScript 物件註記寫法（JavaScript object notation，JSON）等等。

SQL 與關聯式模型的發展始終亦步亦趨，正是因為 SQL 查詢的結果形式便是資料表（在這種情境下產生的資料表又稱為**結果集合**（*result set*））。因此要在關聯式資料庫中新建永久性資料表時，只需將查詢的結果集合加以儲存即可。同理，也可以把永久性資料表和源自其他查詢的結果集合視為查詢對象，來進行查詢（這一點會在第 9 章再詳細探討）。

最後一點：SQL 並非首字母縮寫（雖然許多人仍堅持它是「結構化查詢語言」（Structured Query Language）的首字母縮寫）。但實際稱呼該語言時，其實不論是以字母稱呼（例如 S. Q. L.）、或是以單字稱呼為 *sequel*，意思都是一樣的。

SQL 敘述的種類

SQL 語言可區分為幾個互異的部分：本書要探討的部分，包括用來定義資料庫中儲存資料結構的 ***SQL 架構敘述***（*SQL schema statements*）；以及用來操作上述 SQL 架構敘述所定義資料結構的 ***SQL 資料敘述***（*SQL data statements*）；還有用於起始、結束和復原交易過程的 ***SQL 交易敘述***（*SQL transaction statements*）（第 12 章會探討交易）。舉例來說，若要在資料庫中新建一個資料表，就必須利用 SQL 架構敘述來建立資料表，而事後要再對新資料表填入（populating）資料的過程，則需要用到 SQL 資料敘述。

為了讓讀者們體會一下這些敘述的外貌，以下便是一道會建立名為 `corporation` 資料表的 SQL 架構敘述：

```
CREATE TABLE corporation
 (corp_id SMALLINT,
  name VARCHAR(30),
  CONSTRAINT pk_corporation PRIMARY KEY (corp_id)
);
```

以上敘述會建立一個資料表、其中含有兩個欄位 `corp_id` 和 `name`，而 `corp_id` 欄位則被視為資料庫的主鍵。第 2 章時我們會再進一部細談以上敘述的各項細節，像是 MySQL 中有哪些不同的資料型別等等。以下則是一道會對 `corporation` 資料表填入一筆 Acme Paper Corporation 相關資料的 SQL 資料敘述：

```
INSERT INTO corporation (corp_id, name)
VALUES (27, 'Acme Paper Corporation');
```


碁峰 www.gotop.com.tw

這道敘述會對 corporation 資料表填入一筆資料，其中則含有 corp_id 欄位的資料值 27、以及 name 欄位的資料值 Acme Paper Corporation。

最後則是一道簡單的 select 敘述，它會從剛剛建立的資料表取出資料：

```
mysql< SELECT name
    -> FROM corporation
    -> WHERE corp_id = 27;
+------------------------+
| name                   |
+------------------------+
| Acme Paper Corporation |
+------------------------+
```

所有透過 SQL 架構敘述所建立的資料庫元素，都儲存在一組特殊的**資料目錄**（*data dictionary*）資料表裡。這份「關於資料庫本身的資料」，通常又稱為**中繼資料**（*metadata*），我們會在第 15 章介紹它。資料目錄中的資料表和你自己建立的一般資料表並無差異，同樣也可以透過 select 敘述進行查詢，藉以查閱運行中資料庫中所部署的現有資料結構。舉例來說，如果你受命要產生一份報告，其中包括上個月新開的所有戶頭，

你可以在製作報告時直接引用 account 資料表中已知的欄位名稱，或是轉而查詢資料目錄，藉以得知現有的欄位集合、並在每一次執行時都動態地產生最新的相關資訊報表。

本書大部分篇幅都著重在 SQL 語言的資料處理部分，其中涵蓋 select、update、insert 和 delete 命令。第 2 章會展示 SQL 架構敘述，引導讀者們如何設計和建立若干簡單的資料表。一般來說，除了語法以外，SQL 架構敘述不太需要什麼探討，但 SQL 資料敘述雖然為數較少，但卻有許多值得鑽研的內容。因此筆者雖然會介紹許多 SQL 架構敘述，但本書大部分章節仍會專注在 SQL 資料敘述上。

SQL：非程序性語言

如果你過去曾用過程式語言，你應該已經習於事先定義變數和資料結構，並使用條件邏輯（如 if-then-else）和迴圈結構（如 do while ... end），還有把程式碼拆分成較小的、可重複利用的片段（如物件、函式、程序）等等。你的程式碼會先交由編譯器處理，而編譯好的可執行檔則會如實地執行你設計的動作（好吧，也許**有時不如預期**）。不論你使用的是 Java、Python、Scala 或其他的**程序性**（*procedural*）語言，你都可以全權控制程式的動作。

碁峯 www.gotop.com.tw

程序性語言既定義了期望中的結果應有的外貌，也定義了產生該結果所需的機制或過程。非程序性語言也會定義期望中的結果，但卻把產生結果所需的過程交給外部代理機制去處理。

但是使用 SQL 時，你必須放棄習慣控制權的一部分，因為 SQL 敘述雖然定義了必要的輸入與輸出，但執行一道敘述的方式，卻必須留給資料庫引擎中一個名為**最佳化工具**（*optimizer*）的元件去決定。最佳化工具的任務，就是檢視你的 SQL 敘述，並考量你的資料表設定、以及其中具備的索引，然後決定最有效率的執行路徑（好吧，有時也不見得是最有效率的）。大部分的資料庫引擎都允許你指定所謂的**最佳化提示**（*optimizer hints*），藉以影響最佳化工具的決策，類似的提示包括指名使用特定索引等等；但大多數的 SQL 使用者卻都選擇不要進行如此複雜的微調，而是把它留給資料庫管理員或性能調校專家去傷腦筋。

因此，你用 SQL 是無法寫出完整應用程式的。除非你只是要寫一個簡易的指令碼（script）來操作特定的資料，不然就得整合 SQL 和你偏好的程式語言。有些資料庫廠商已經為你整合了這樣的功能，像是甲骨文的 PL/SQL 語言、MySQL 的預存程序（stored procedure）語言、以及微軟的 Transact-SQL 語言等等。在這類語言中，SQL 資料敘述是語法的一部分，因此你可以不露痕跡地將資料庫查詢寫到程序式命令當中。然而，如果你使用的是 Java 或是 Python 之類的非資料庫專屬語言，你就得透過工具程式（toolkit）或 API，才能在程式碼中執行 SQL 敘述。這類工具程式多半由資料庫廠商提供，有些則來自第三方的供應商、甚至是由開放原始碼社群所提供。表 1-2 列出了若干將 SQL 整合至特定語言的既有選項。

表 1-2　SQL 的整合工具程式

Language	Toolkit
Java	JDBC（Java 資料庫連接）
C#	ADO.NET（微軟）
Ruby	Ruby DBI
Python	Python DB
Go	database/sql 套件

如果你只是想要以互動方式執行 SQL 命令，每一家資料庫廠商其實都會提供簡易的命令列工具，讓你可以從中對資料庫引擎下達 SQL 命令、並進而檢視結果。大多數的廠商還會額外提供圖形介面工具，其中一個視窗會顯示你的 SQL 命令、另

碁峰 www.gotop.com.tw

一個視窗則顯示執行後的結果。此外也有一些第三方廠商的工具，例如 Squirrel，它可以透過 JDBC 連接到各種資料庫伺服器。由於本書中的範例都是以 MySQL 資料庫示範執行的，筆者會使用 mysql 這個命令列工具，這是一個安裝 MySQL 時預設就會提供的工具，它可以執行範例中的程式、並將輸出結果格式化成為易於觀看的外觀。

SQL 的範例

筆者先前在本章開始時曾承諾，要讓讀者們體驗一下實際的 SQL 敘述，並藉以傳回支票帳戶中所有的交易紀錄。閒話休提，這就拿出來：

```
SELECT t.txn_id, t.txn_type_cd, t.txn_date, t.amount
FROM individual i
  INNER JOIN account a ON i.cust_id = a.cust_id
  INNER JOIN product p ON p.product_cd = a.product_cd
  INNER JOIN transaction t ON t.account_id = a.account_id
WHERE i.fname = 'George' AND i.lname = 'Blake'
  AND p.name = 'checking account';

+--------+-------------+---------------------+--------+
| txn_id | txn_type_cd | txn_date            | amount |
+--------+-------------+---------------------+--------+
|     11 | DBT         | 2008-01-05 00:00:00 | 100.00 |
+--------+-------------+---------------------+--------+
1 row in set (0.00 sec)
```

這裡暫時不會太過深入所有細節，而是只會說明，這筆查詢會找出 individual 資料表中與 George Blake 有關的資料列、同時也找出 product 資料表中名為「checking」產品的資料列，再藉此找出 account 資料表中含有以上客戶 / 產品組合的帳戶資料，進而取得 transaction 資料表中關於該帳戶交易資料的四個欄位。如果你剛好知道 George Blake 的客戶識別碼（customer ID）就是 8、也知道支票帳戶的識別碼是「CHK」，那你甚至還可以直接到 account 資料表中，用客戶識別碼和帳戶識別碼找出 George Blake 的支票帳戶，進而調出相關的交易紀錄：

```
SELECT t.txn_id, t.txn_type_cd, t.txn_date, t.amount
FROM account a
  INNER JOIN transaction t ON t.account_id = a.account_id
WHERE a.cust_id = 8 AND a.product_cd = 'CHK';
```

筆者會在以下幾章中一一說明以上查詢語句中的觀念（再加上更多範例），這裡不過是想讓大家先領教一下即將面對的內容罷了。

碁峯 www.gotop.com.tw

以上查詢語句中一共包含三個不同的**子句**（*clauses*）：select、from 和 where。幾乎所有你會看得到的查詢語句中，都至少會包含這三種子句，只不過還會有更多的子句可以供特定目的使用。這三種子句的角色說明如下：:

```
SELECT /* 單一或多項內容 */ ...
FROM /* 從單一或多處來源 */ ...
WHERE /* 符合一或多項條件 */ ...
```

大部分實作出來的 SQL，都會把放在 /* 和 */ 標籤中間的內容視為註解。

建構查詢語句時，首要任務是先大致地決定需要用到哪些資料表，然後將它們加入到你的 from 子句中。然後你得在 where 子句中加上判斷條件，以便從你的資料表中篩選出你真正有興趣的資料。最後則是要決定你要從這些資料表中取出哪些欄位，並放到 select 子句中。以下是一個簡單的例子，顯示如何找出姓氏為「Smith」的所有客戶：

```
SELECT cust_id, fname
FROM individual
WHERE lname = 'Smith';
```

以上查詢會在 individual 資料表中搜尋，找出所有 lname 欄位內容符合字串「Smith」的資料列，再取出該種資料列的 cust_id 和 fname 等欄位。

除了查詢資料庫，你很可能還需要為資料庫填入資料、或是加以修改。以下是一個如何在 product 資料表中插入一筆新資料的簡單例子：

```
INSERT INTO product (product_cd, name)
VALUES ('CD', 'Certificate of Depysit')
```

哎呀，似乎你把「Deposit」這個字拼錯了。不過這不是問題。你可以用 update 敘述加以修正：

```
UPDATE product
SET name = 'Certificate of Deposit'
WHERE product_cd = 'CD';
```


碁峰 www.gotop.com.tw

注意，以上的 update 敘述同時還包含一個 where 子句，就像 select 敘述的做法一樣。這是因為 update 敘述必須找出真正需要更新的資料列；在這個案例中，你指定只有 product_cd 欄位內容符合「CD」字樣的資料列才是修改的目標。由於 product_cd 欄位是 product 資料表的主鍵，你可以預期以上的 update 敘述只會修改到一筆（也可能是零筆，要看你提供的篩選值在該資料表中是否存在而定）。每當你執行一道 SQL 資料敘述時，你都會從資料庫引擎收到相關的回應，包括有多少筆資料受到你的敘述所影響。如果你使用先前提到過的 mysql 之類的命令列互動式工具，那你也會看到關於有多少筆資料會涉及相關動作的訊息：

- 你的 select 敘述找到的資料筆數
- 你的 insert 敘述所建立的資料筆數
- 你的 update 敘述所修改的資料筆數
- 你的 delete 敘述所移除的資料筆數

如果你正在使用某種含有前述工具程式的程序性語言，該工具程式必會含有某種呼叫功能（call），可以在你的 SQL 資料敘述執行過後，取得類似的執行結果資訊。一般來說，在程式每次執行敘述後檢查此類資訊、確認沒有發生什麼意料外的問題，是比較穩當的做法（例如你忘記在 delete 敘述中加上 where 子句，結果導致資料表中所有資料列都被刪掉！）。

MySQL 又是什麼？

關聯式資料庫在商用領域已經叱吒超過卅個年頭。其中最成熟及受到愛用的商業版產品包括：

- 甲骨文公司的 Oracle Database
- 微軟的 SQL Server
- IBM 的 DB2 Universal Database

這些資料庫伺服器的功能都大同小異，不過其中有些較為適於運行規模非常大、或是吞吐量極高的資料庫。有些則較善於處理物件或是超大檔案、或是 XML 等文件檔案。此外，所有這些伺服器都明確地遵守了最新版的 ANSI SQL 標準。這是好事，當筆者教大家如何寫出只須略為修改（甚至不用改）就可以在任何平台上執行的 SQL 敘述時，會把是否符合 ANSI 標準列為要點。

碁峰 www.gotop.com.tw

除了商業版資料庫伺服器，開放原始碼社群在試圖建立可行的替代用方案方面，過去廿年中也有不少進展。其中最常為人們所使用的開放原始碼資料庫伺服器，要算是 PostgreSQL 和 MySQL 了。MySQL server 是可以免費取得的，筆者也發現其下載及安裝都極為容易。有鑑於此，筆者決定在 MySQL（8.0 版）資料庫上執行本書所有的範例，並以 `mysql` 命令列工具來格式化所有的查詢結果。即就算你已在使用他種伺服器、也沒打算要換成 MySQL，筆者仍舊希望你可以安裝一套最新版的 MySQL 伺服器，並載入範例的 schema 和資料，再拿本書的資料和範例來做實驗。

然而還是要提醒大家：

> 本書重點並非 MySQL 版本的 SQL 實作。

相反地，本書的設計是要教導讀者們寫出不必修改就能在 MySQL 上執行的 SQL 敘述，甚至只須小改（甚至不用改）就也能在最近的 Oracle Database、DB2 和微軟 SQL Server 上執行的 SQL 敘述。

抽離 SQL

在本書第二版和第三版發行中間的十年中，資料庫業界發生了許多變化。雖說關聯式資料庫仍受到重用，這種現象一時也不會有太大的變化，但是新進的資料庫技術卻更能滿足像是亞馬遜和谷歌等新興業者的需求。這些技術包括 Hadoop、Spark、NoSQL 和 NewSQL，它們都是善於擴展的分散式系統，通常部署在叢集形式的伺服器上。雖說詳盡探索這些技術已經超越了本書的範圍，它們卻都與關聯式資料庫有一個共通點：那就是 SQL。

由於企業常會以多種技術來儲存資料，因此有必要將 SQL 從特定的資料庫伺服器上抽離出來，並建立一種可以跨越多種資料庫的服務。舉例來說，你有可能得從儲存在 Oracle、Hadoop、JSON 檔案、CSV 檔案及 Unix 日誌檔中的資料拼湊而成一份報表。坊間已有可以滿足上述挑戰的新一代工具問世，其中前景看好的是 Apache Drill，它是一種開放原始碼的查詢引擎，允許使用者撰寫查詢來存取任何資料庫或檔案系統中的資料。我們會在第 18 章時探討 Apache Drill。

碁峰 www.gotop.com.tw

還有哪些參考書呢？

以下四章的整體目標，是先介紹 SQL 資料敘述，其中會特別強調 `select` 敘述中的三種主要子句。此外讀者們會看到許多採用 Sakila schema 的範例（下一章便會談到），本書所有範例均採用相同架構。筆者衷心希望，藉由熟悉單一資料庫，就能讓讀者們掌握到範例的精髓，無須每次都要停下來先熟悉資料表內容。如果你覺得每次都操作同一批資料表很無趣，儘管自行建立其他資料表來擴充範例資料庫，或是自行建置資料庫，並用來做實驗。

一旦你掌握了基本知識，隨後的章節就會帶大家深入鑽研更多內容，它們大部分都彼此互相獨立。因此讀者們若是讀到覺得迷惑，儘管放心地跳過去繼續閱讀，然後晚一點再回頭來重讀不懂的部分。當你讀完本書，也練習過所有範例，應該就已具備成為一位熟練 SQL 技術人員的本錢了

如果讀者們有意繼續深入了解關聯式資料庫、以及有關運算式資料系統的發展史，或是不甘心只侷限於以上簡短的 SQL 語言簡介，以下是一些值得一讀的參考來源：

- C. J. Date 所著的《*Database in Depth: Relational Theory for Practitioners*》（O'Reilly 出版）（中文版為《深入資料庫之美學》）
- C. J. Date 所著的《*An Introduction to Database Systems*》第八版（Addison-Wesley 出版）（中文版為《資料庫系統概論》）
- C. J. Date 所著的《*The Database Relational Model: A Retrospective Review and Analysis*》（Addison-Wesley 出版）
- 維基百科在定義資料庫一文中的「Database Management System」段落（*https://oreil.ly/sj2xR*）

碁峰
www.gotop.com.tw