

CH13 SQL與資料庫

13-1 前言

- 在現實中除了檔案以外，資料庫(database)也是很重要的資料來源，而且不僅能放在本地端，更能從遠端存取
- 這一章不會深入講解資料庫本身的管理以及SQL語言，主要用意還是在展示go語言是如何連接SQL資料庫
- Go語言使用一種高階的方式來連接資料庫，也就是使用標準套件的database/sql作為API，而API底下才會連接到資料庫需要的驅動程式，例如MySQL, Postgres 等等

- 大部分資料庫都有原生的go語言驅動程式可以下載；當然也有少數需要額外套件
- 之所以要使用這種API/驅動程式架構，是為了以go語言為統一抽象介面，使任何人無須了解資料庫的溝通細節就能操作 ----你只需要在一開始匯入正確的驅動程式和登入資料庫，在這之後的控制過程就是完全一樣的
- 舉個例子，假如你有一個專案使用MySQL資料庫，但隨時間進展發現它逐漸不敷需求，想要換成AWS Athena雲端資料庫，若你在專案中使用MySQL的專屬驅動介面來寫程式，那就要花大量時間改寫成另一種驅動程式的版本了！
- 然而，若你一開始就透過database/sql介面，那麼只需要更換驅動程式即可，不僅可以節省時間，還可以免於修改程式碼而意外引入bug的風險

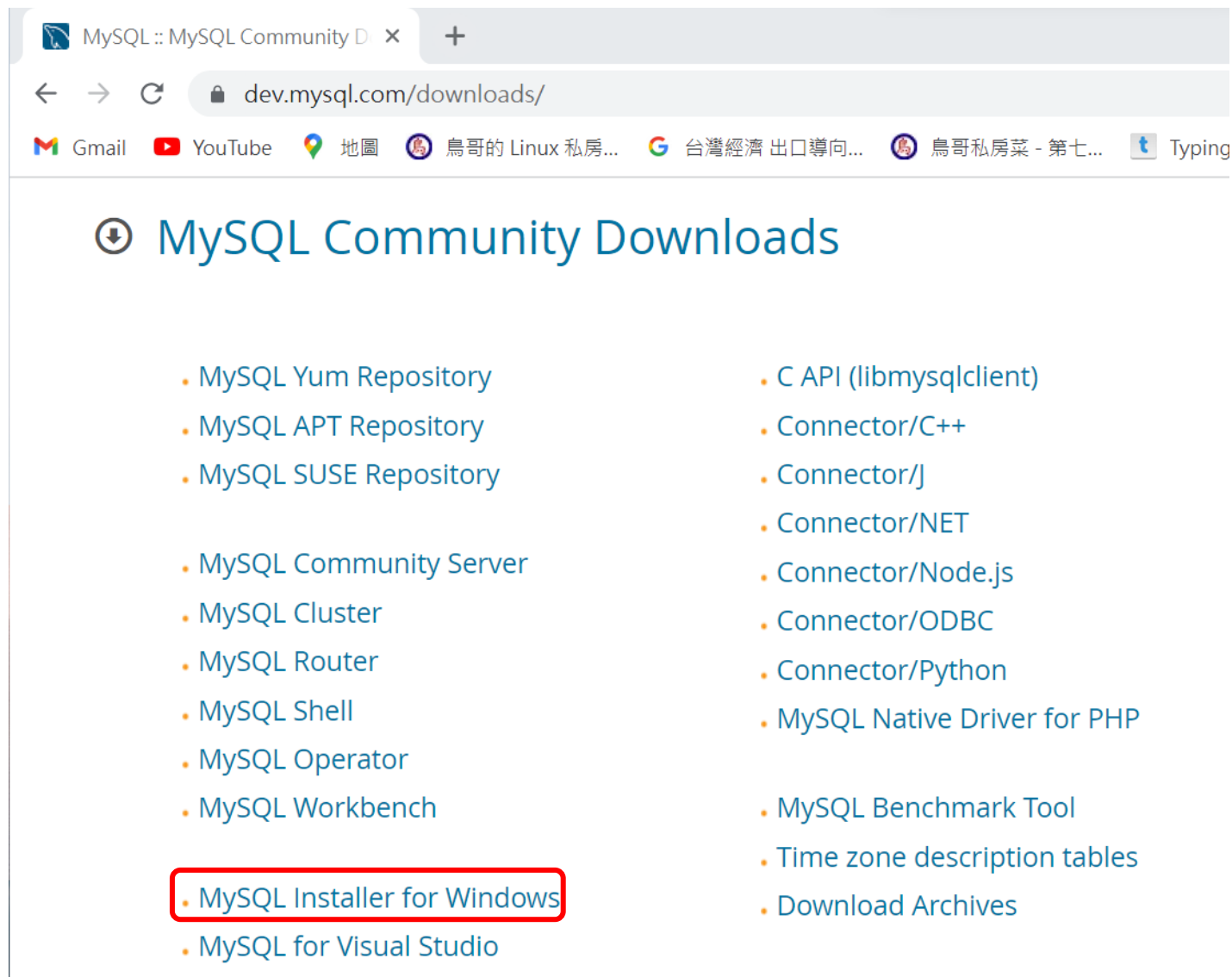
13-2 安裝MySQL資料庫

- Go語言能連接的資料庫種類很多，不過本章我們將以其中一種最受歡迎的開源資料庫MySQL為例
- 在安裝好後，系統會自動執行MySQL伺服器，讓其他應用程式能夠連結它

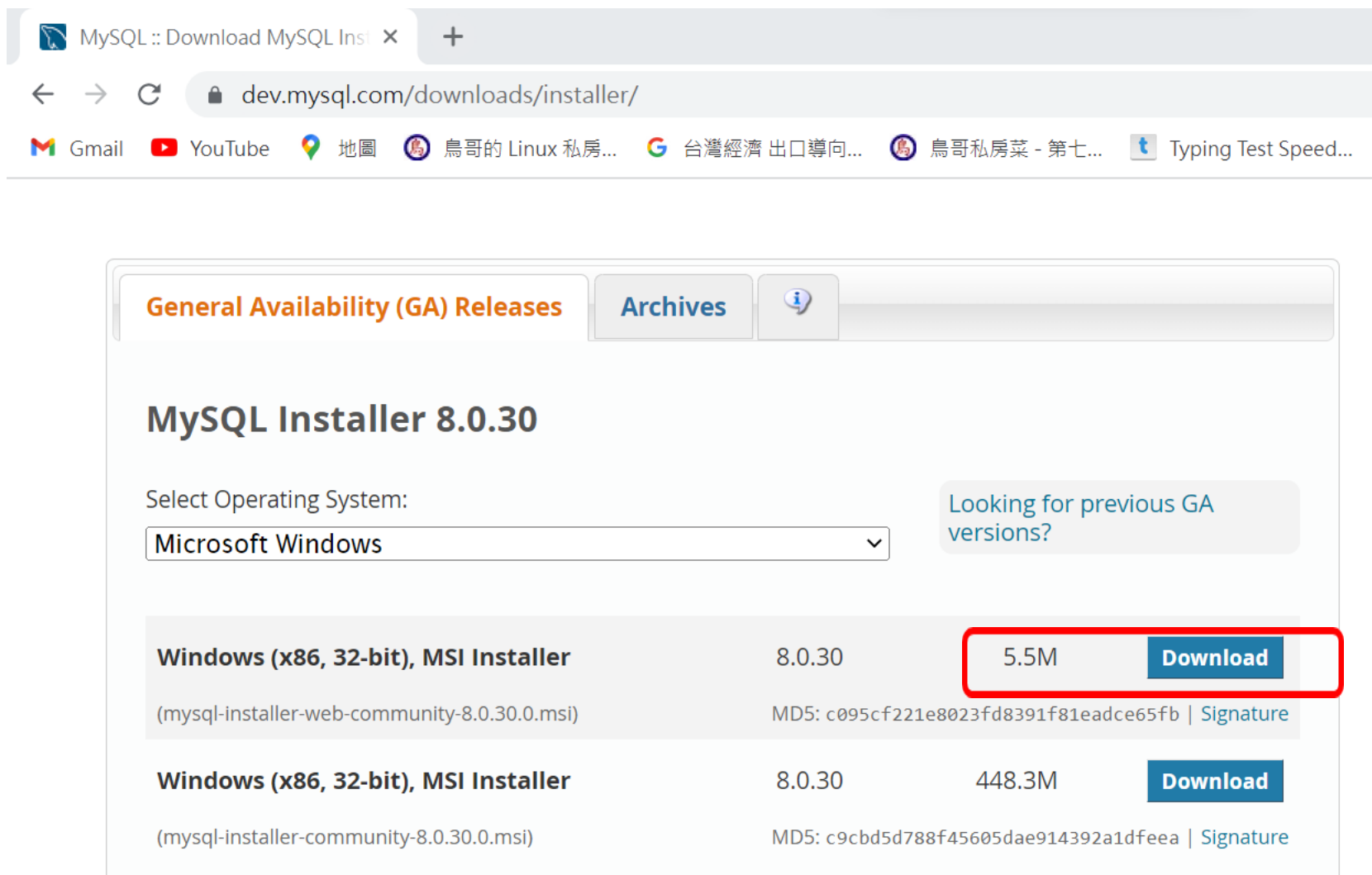
13-2-1 安裝MySQL Server

- Linux安裝MySQL相當簡單，只要在終端機輸入 `sudo apt-get install mysql-server`即可 (這裡用Ubuntu為例)
- 底下介紹Windows的安裝方法：

- 首先, 到 <https://dev.mysql.com/downloads/> 點選MySQL Installer for windows 下載MySQL Server :



- 接著安裝程式，選擇容量較小的版本 (若打算安裝所有功能，也可以安裝另一個版本)：



MySQL :: Download MySQL Installer

dev.mysql.com/downloads/installer/

Gmail YouTube 地圖 鳥哥的 Linux 私房... 台灣經濟 出口導向... 鳥哥私房菜 - 第七... Typing Test Speed...

General Availability (GA) Releases Archives

MySQL Installer 8.0.30

Select Operating System:
Microsoft Windows

Looking for previous GA versions?

Windows (x86, 32-bit), MSI Installer (mysql-installer-web-community-8.0.30.0.msi)	8.0.30	5.5M	Download
Windows (x86, 32-bit), MSI Installer (mysql-installer-community-8.0.30.0.msi)	8.0.30	448.3M	Download

MD5: c095cf221e8023fd8391f81eadce65fb | [Signature](#)

MD5: c9cbd5d788f45605dae914392a1dfeea | [Signature](#)

- 於下一個畫面點選下圖連結來下載檔案：

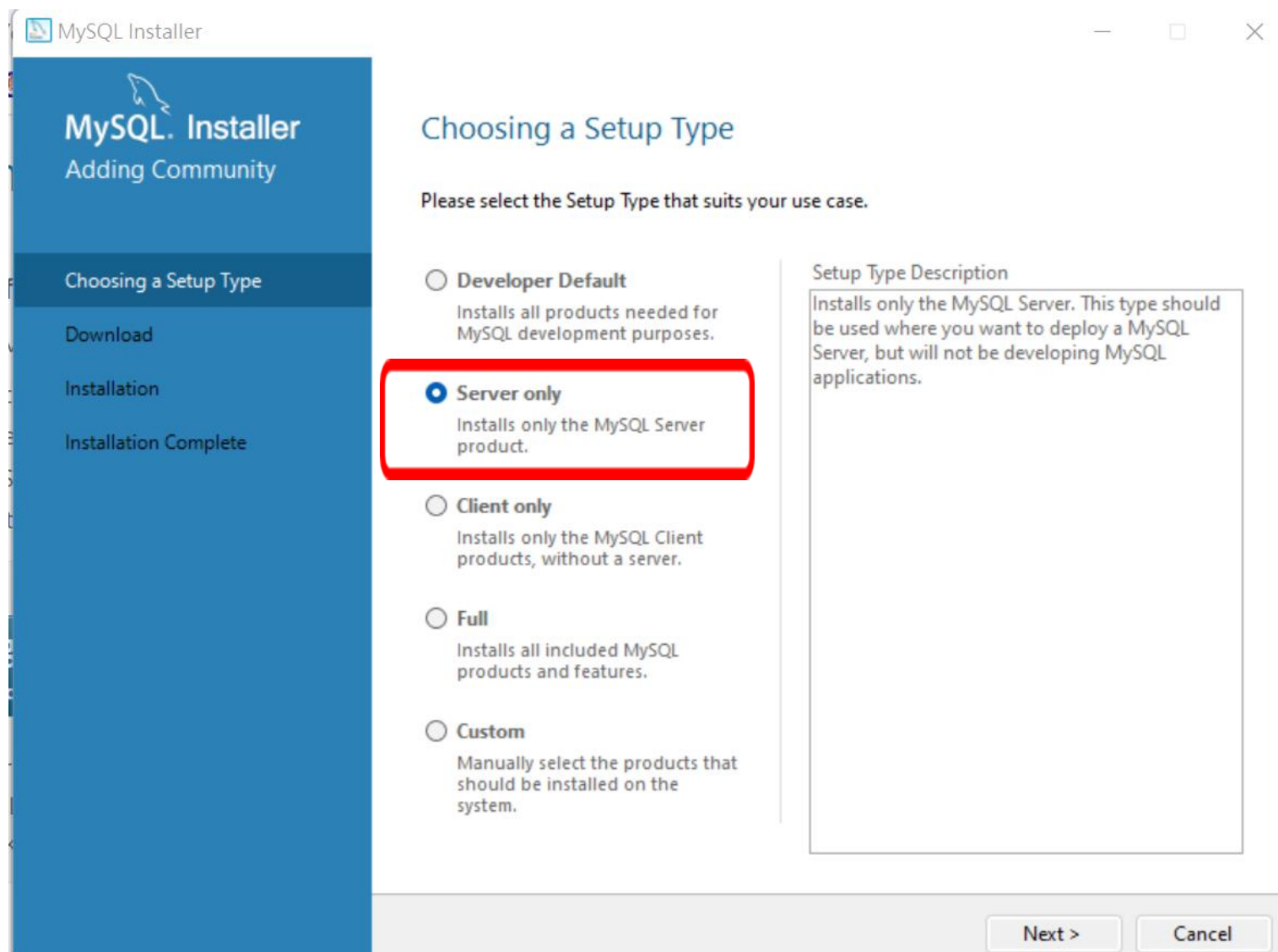
Login »
using my Oracle Web account

Sign Up »
for an Oracle Web account

MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

No thanks, just start my download.

- 啟動安裝程式，選擇**Server only** (本章只會用到這個功能，但你也可以自行安裝其他功能)



- 在下載並安裝好MySQL Server後，一直按next繼續，最後來到資料庫的帳號設定畫面：

MySQL. Installer
MySQL Server 8.0.30

Type and Networking
Authentication Method
Accounts and Roles
Windows Service
Apply Configuration

Accounts and Roles

Root Account Password
Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password:

Repeat Password:

MySQL User Accounts
Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

MySQL User Name	Host	User Role
-----------------	------	-----------

Add User
Edit User
Delete

< Back Next > Cancel

- 在畫面頂端輸入root帳號的密碼，此外你也可以新增一個在go程式中使用的使用者帳號
- 為了示範起見，這裡設定了一個使用者要user, 主機回localhost, 密碼為1234；MySQL預設的本機通訊埠為3306，我們就不更動了：

MySQL Installer

MySQL Server 8.0.30

Type and Networking

Authentication Method

Accounts and Roles

Accounts and Roles

Root Account Password

Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password:

••••

Repeat Password:

••••

Password strength: **Weak**

MySQL User Account

Please specify the user name, password, and database role.



User Name: user

Host: localhost

Role: DB Admin

Authentication: ☒ MySQL

MySQL user credentials

Password: ••••

Confirm Password: ••••

Password strength: **Weak**

OK

Cancel

and applications. Assign a role to the user that

User Role

Add User

Edit User

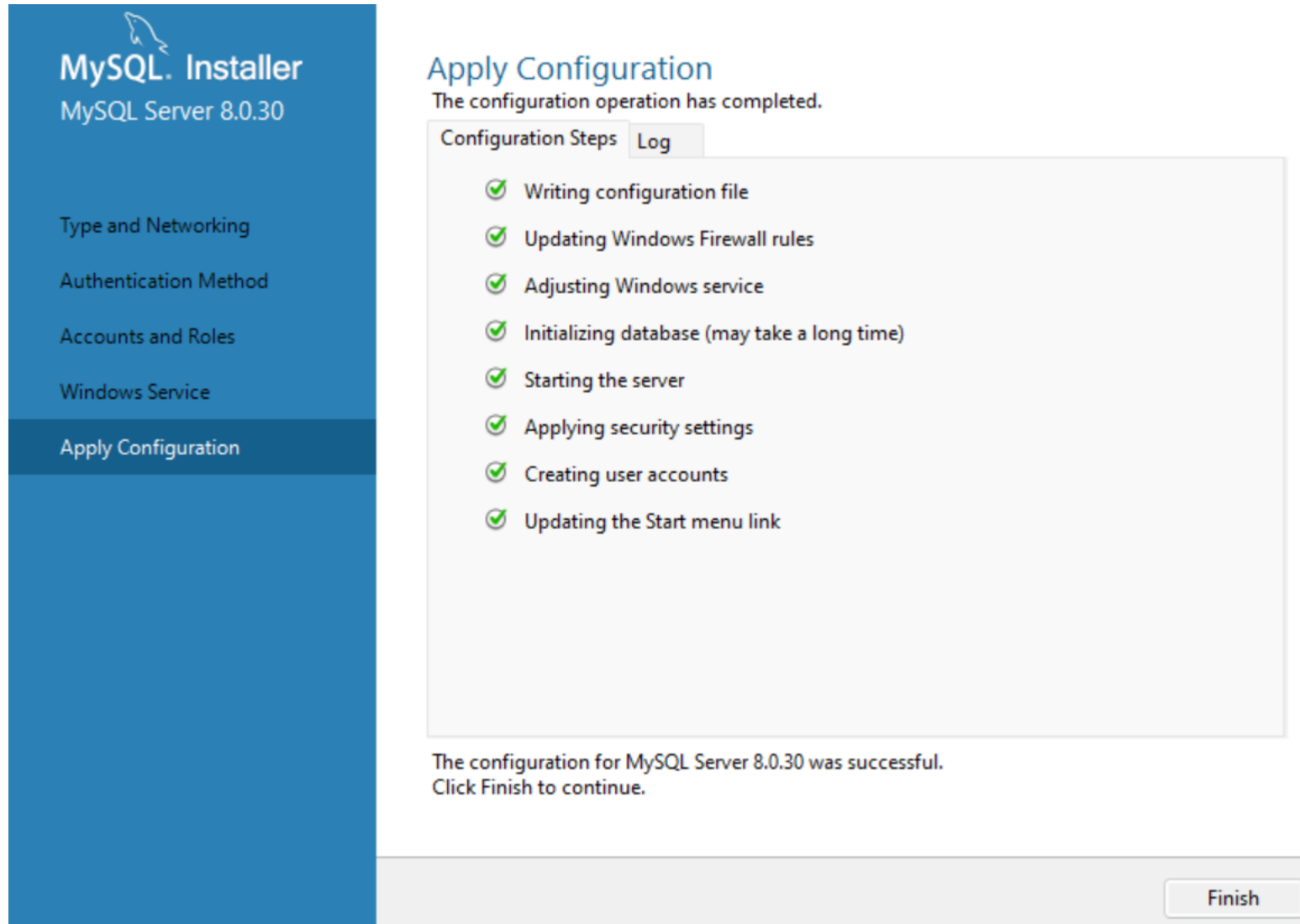
Delete

< Back

Next >

Cancel

- 在最後的畫面，讓安裝程式啟動MySQL伺服器，便能完成安裝：



13-2-2 新增資料庫使用者

- 無論是Windows還是Linux使用者，你都需要至少新增一個權限足夠的使用者，而這可以透過MySQL命令列客戶端來操作：
 - Windows使用者請執行開始選單 → MySQL → MySQL 8.0 Command Line Client → 輸入root密碼
 - Linux使用者請在終端機執行`sudo mysql -u root -p root`密碼
- 這會進入MySQL monitor的提示字元：

Enter password: ****

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 13

Server version: 8.0.30 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

- 下面來建立一個新使用者，取名為user, 密碼為1234 (若你已經在前面建立了user可以跳過這個步驟)：

```
mysql> CREATE USER 'user'@'localhost' IDENTIFIED BY '1234';
```

- 接著賦予user完整的操作權限：

```
mysql> GRANT ALL PRIVILEGES ON * . * TO 'user'@'localhost' ;
```

- 最後確保設定生效：

```
mysql> FLUSH PRIVILEGES
```

- 最後關閉主控台或在主控台輸入\q

12-2-3 建立一個MySQL資料庫

```
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.30 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE mysqldb;
Query OK, 1 row affected (0.01 sec)

mysql>
```

這要會建立一個叫做mysqldb的資料庫

12-2-4 下載go語言的MySQL驅動程式

- 你可以在以下網址找到針對不同資料庫go語言驅動程式套件：

<https://github.com/golang/go/wiki/SQLDrivers>

- 本章我們使用MySQL, 因此會使用驅動程式 <https://github.com/go-sql-driver/mysql>
- 依照該頁面指示, 你可以在主控台輸入go get來下載它：

```
PS D:\git\Golang> go get -u github.com/go-sql-driver/mysql
go: downloading github.com/go-sql-driver/mysql v1.6.0
go: added github.com/go-sql-driver/mysql v1.6.0
PS D:\git\Golang> █
```

13-3 以go語言連接資料庫

- 經過以上步驟，連接資料庫其實是整個過程中最容易的；為了連接任何資料庫，必須先準備4件事：
 - 可供連接的資料庫伺服器
 - 使用者帳號
 - 使用者密碼
 - 特定操作的權限

- 資料庫伺服器好比一棟房子，有特定位址，而使用者帳號與密碼就像開門鑰匙；至於進門後可以做甚麼，取決於他得到多少許可(權限)
- 資料庫權限包括能否查詢, 插入或移除資料，還有是否能建立或刪除表格等等

- 一旦完成了前一節的準備，就可以開始撰寫go程式了
- 首先是匯入相關套件：

```
1  package main
2
3  ∨ import (
4      "database/sql"
5      "fmt"
6
7      _ "github.com/go-sql-driver/mysql"
8  )
9
```

- 上面匯入套件中的database/sql即為go內建的資料庫API，而第二行則是匯入我們下載的MySQL驅動程式
- 注意驅動程式前面有一個底線，意思是讓該套件使用底線為別名，因為我們並不會直接使用mysql套件，只是要匯入而已(若你匯入套件有包含名稱，卻未呼叫其功能，就會在編譯時產生錯誤)
- 匯入套件後，就能準備連接資料庫：

```
10 func main() {  
11     db, err := sql.Open("mysql", "user:1234@tcp(localhost:3306)/mysql?charset=utf8")  
12 }
```

- 以上函式值得注意，因為他是database/sql套件提供的通用API，第一個參數是驅動程式名稱，第二個參數則是資料來源名稱
- 要注意的是，sql.Open()並不會真正連線到資料庫，而是回傳一個sql.DB結構給我們使用，並姊也會回傳一個error，讓我們用來檢查提共的驅動程式及資料來源名稱是否有誤：

```
13     if err != nil {  
14         panic(err)  
15     }  
16     defer db.Close()  
17     fmt.Println("sql.DB 結構已建立")  
18
```

- 接著我們檢查資料庫是否能正確連線，因為`sql.Open()`不知道你提供的帳號密碼是否正確
- 此外，若你的程式會長時間運作，也有可能遇到資料庫伺服器斷線會網路不穩問題，因此你在任何操作前都應檢查資料庫的連線狀態：

```
19     err = db.Ping()
20     if err != nil {
21         panic(err)
22     }
23     fmt.Println("資料庫連線成功")
24 }
25
```


- 用完資料庫後，你可以關閉資料庫連線，不過正常情況下不用這樣做，因為`sql.DB`會在`go`程式結束後自動關閉所有連線
- 話說回來，若資料庫會同時被多人存取，而你對資料庫的存取並不頻繁，僅限一個函式的範圍，那你應該在該函式結束時關閉資料庫連線

16

```
defer db.Close()
```

- 若要執行以上程式，得先建立go.mod並加入資料庫相關套件的路徑(參閱第8章)
- 執行正確的結果如下：

```
sql.DB 結構已建立  
資料庫連線成功
```

13-4 建立/清空和移除資料表

- 確認資料庫連線正常後，接著來建立資料表(table)
- 建立資料表的目的，是用一個抽象容器來存放彼此相關的資料；不管資料是什麼，其共通的目的都是要讓應用程式讀取和解析
- 建立資料表的SQL語法一般如下：

```
CREATE TABLE 資料表名稱 (  
  欄位1名稱 資料形別 限制,  
  欄位2名稱 資料形別 限制,  
  ....  
)
```

- SQL(Structured Query Language, 結構化查詢語言)是一種用來操作關聯式資料庫(rerlational database)系統的統一表準語言，像是MySQL, Postgres, DB2都屬於這類資料庫，因此你能用完全相同的方法操作他們
- **CREATE TABLE**是用來建立新資料表的SQL語法，你得指定資料表名稱，以及每一個欄位的名稱和型別，外加欄位可能有的限制
 - 幾種常見的資料型別如下：
 - INT (整數)
 - FLOAT (浮點數)
 - DOUBLE (雙精度浮點數)
 - VARCHAR (字串, 需指定長度)

- 欄位的限制因資料庫系統而異，不過常見的幾種如下：
 - NOT NULL (不得為NULL值)
 - UNIQUE (必須是獨一無二的值)
 - PRIMARY KEY (資料代表主鍵)
 - FOREIGN KEY (資料代表外鍵，即另一個資料表的主鍵)
- 我們不但能建立資料表，也可以清空資料表，移除其中的全部資料：

`TRUNCATE TABLE 資料表名稱;`

- 或者你可以把資料表從資料庫中移除：

`DROP TABLE 資料表名稱;`

在MySQL建立資料表

- 在這個範例中，我們要替mysqlldb資料庫新增一個資料表叫做employee, 只有兩個欄位：

Id	name
員工代號 (整數)	員工名稱(字串, 長度20)

```
1  package main
2
3  import (
4      "database/sql"
5      "fmt"
6
7      _ "github.com/go-sql-driver/mysql"
8  )
9
10 func main() {
11     db, err := sql.Open("mysql", "user:1234@tcp(localhost:3306)/mysql?charset=utf8")
12
13     if err != nil {
14         panic(err)
15     }
16     defer db.Close()
17     fmt.Println("sql.DB 結構已建立")
18
19     err = db.Ping()
20     if err != nil {
21         panic(err)
22     }
23     fmt.Println("資料庫連線成功")
24 }
```

```
25 //建立資料表的SQL指令
26 //欄位id為數字，不得為NULL且不能重複
27 //欄位name為長度20的字串
28 DBCreate := `
29 CREATE TABLE employee
30 (
31     id INT NOT NULL UNIQUE,
32     name VARCHAR(20)
33 );`
34
35 _, err = db.Exec(DBCreate) //執行SQL指令
36 if err != nil {
37     panic(err)
38 }
39 fmt.Println("表格 employee 已建立")
40 }
41
```


執行結果

```
sql.DB 結構已建立  
資料庫連線成功  
表格 employee 已建立
```

- 上面的程式碼中，首先以使用者**user**和密碼**1234**連接資料庫**mysqldb**，接著用一個字串來定義我們要使用的**SQL**指令，再用**db.Exec()**方法執行他
- **Exec()**會回傳兩個值，第一個是**SQL**指令的執行結果，另一個是**error**值：

Func (db *DB) Exec(query string, atgs ...interface{}) (Result, error)

- 執行結果會指出**SQL**指令影響了資料庫多少資料，但在此我們不需要知道，故用_跳過該回傳值
- 只要**error**值為**nil**，就代表資料表新增成功

在MySQL命令列客戶端檢視表格資料及移除之

- 在執行以上範例後，可以於MySQL客戶端命令列選擇資料庫 `mysqldb`，並執行 `SHOW TABLES;` 來瀏覽資料庫內的資料表：

```
mysql> use mysqldb
Database changed
mysql> SHOW TABLES;
+-----+
| Tables_in_mysqldb |
+-----+
| employee          |
+-----+
1 row in set (0.03 sec)
```

- 你也可以透過命令列來移除資料表：

```
mysql> DROP TABLE employee;  
Query OK, 0 rows affected (0.03 sec)
```

- 注意：
 - 每次啟用MySQL命令列客戶端時，必須用 “**use 資料庫名稱**” 才能操作該資料庫內的資料表

13-5 插入資料

- 在go語言中，對資料表插入新資料的動作分為兩個階段：先用sql.DB結構的Prepare()產生參數化查詢敘述(prepared statement)，也就是sql.Stmt結構，再透過這個來實際操作資料庫
- 需要這樣做的原因是，參數化查詢是目前公認防範SQL攻擊最有效的方法，因為資料庫會先將SQL指令編譯成位元組碼，然後才透過參數將值放進需要的地方
- 也就是說，就算傳入參數的值帶有SQL指令，他也不會被資料庫執行，這樣能提高SQL的值形效率，因為一部分的指令已經先編譯好了

在MySQL資料表插入資料

- 在以下範例會在先前建立的**employee**資料表中新增一筆資料：

id : 305

name : Sue

```
1  package main
2
3  import (
4      "database/sql"
5      "fmt"
6
7      _ "github.com/go-sql-driver/mysql"
8  )
9
10 func main() {
11     db, err := sql.Open("mysql", "user:1234@tcp(localhost:3306)/mysql?charset=utf8")
12
13     if err != nil {
14         panic(err)
15     }
16     defer db.Close()
17     fmt.Println("sql.DB 結構已建立")
18
19     err = db.Ping()
20     if err != nil {
21         panic(err)
22     }
23     fmt.Println("資料庫連線成功")
24 }
```

```
25 //準備參數化查詢敘述
26 insertStmt, err := db.Prepare("INSERT INTO employee (id, name) VALUES (?, ?);")
27 ✓ if err != nil {
28     | panic(err)
29 }
30 defer insertStmt.Close() //在程式結束時關閉參數化查詢敘述
31 _, err = insertStmt.Exec(305, "Sue") //新增一筆資料
32 ✓ if err != nil {
33     | panic(err)
34 }
35 fmt.Println("成功插入資料 305, Sue")
36 }
37
```


執行結果：

sql.DB結構已建立

資料庫連線成功

成功插入資料 305, Sue

- 來解釋一下程式碼
 - 首先是插入資料的SQL指令：`"INSERT INTO employee (id, name) VALUES (?, ?);"`
 - 為了避免SQL注入攻擊，要填入的值先寫成問號代表參數，並由`db.Prepare()`編譯和傳回`insertStmt`結構
 - 在第二步驟中，`insertStmt.Exec()`會將填入的值放進上述的INSERT指令中的問號所在位置
 - 此外請注意，參數化查詢敘述(`sql.Stmt`結構)會占用一些資源，因此在用完該結構後，應該用`Close()`關閉它來釋放資源

13-6 查詢資料

- 資料表的查詢分兩類，一種是沒有參數，從資料表中取出大量資料用的，另一種則會有篩選條件，常用來找出特定的資料
- 下面就來看看這兩種情境的範例

13-6-1 查詢並印出整個資料表的內容

- 假設你已經在資料表employee中新增了4筆資料：
 1. 305 Sue
 2. 204 Bob
 3. 631 Jake
 4. 73 Tracy
- 在以下範例中，會查詢資料表employee全部內容：

```
1  package main
2
3  ✓ import (
4      |     "database/sql"
5      |     "fmt"
6
7      |     _ "github.com/go-sql-driver/mysql"
8  )
9
10 ✓ type employee struct {
11     |     id    int
12     |     name  string
13 }
14
15 ✓ func main() {
16     |     db, err := sql.Open("mysql", "user:1234@tcp(localhost:3306)/mysql?charset=utf8")
17
18     |     if err != nil {
19         |         panic(err)
20     }
21     defer db.Close()
22     fmt.Println("sql.DB 結構已建立")
23
24     err = db.Ping()
25     ✓ if err != nil {
26         |         panic(err)
27     }
28     fmt.Println("資料庫連線成功")
29 }
```

```
30 //查詢資料表，傳回sql.Rows
31 rows, err := db.Query("SELECT * FROM employee")
32 if err != nil {
33     panic(err)
34 }
35 defer rows.Close() //在程式結束時關閉Rows
36 fmt.Println("資料表查詢成功，列出 employee 內容...")
37
38 for rows.Next() { //走走訪Rows
39     e := employee{}
40     err := rows.Scan(&e.id, &e.name) //讀出一筆資料
41     if err != nil {
42         panic(err)
43     }
44     fmt.Println(e.id, e.name) //印出資料
45 }
46 err = rows.Err() //檢查Rows有無遭遇其他錯誤
47 if err != nil {
48     panic(err)
49 }
50 }
51
```

執行結果

```
sql.DB 結構已建立  
資料庫連線成功  
資料表查詢成功，列出 employee 內容...  
73 Tracy  
204 Bob  
305 Sue  
631 Jake
```

- 在上面的程式中，首先用`db.Query()`來執行`SELECT * FROM employee`
- 和`db.Exec()`方法的差別在於，`Query()`會傳回`sql.Rows`結構，用來代表查詢結果的一列資料：

```
func (db *DB) Query(query string, args ...interface{}) (*Rows, error)
```

- 接著我們用**for**迴圈`rows.Next()`來走訪他，迴圈每執行一次`rows`就會指向下一列，這時就能用`rows.Scan()`來將該欄位的欄位賦予給變數(變數的數量必須跟欄位相同)

13-6-2 查詢符合條件的資料

- 另一種查詢資料的場合，是設下過濾條件時
- 問題是，這是另一個可能遭受SQL注入攻擊的情境，因此這裡需要再度使用`db.Prepare()`來產生參數化查詢敘述：

```
1  package main
2
3  import (
4      "database/sql"
5      "fmt"
6
7      _ "github.com/go-sql-driver/mysql"
8  )
9
10 type employee struct { //用來記錄employee一筆資料的結構
11     id    int
12     name  string
13 }
14
15 func main() {
16     db, err := sql.Open("mysql", "user:1234@tcp(localhost:3306)/mysql?charset=utf8")
17
18     if err != nil {
19         panic(err)
20     }
21     defer db.Close()
22     fmt.Println("sql.DB 結構已建立")
23
24     err = db.Ping()
25     if err != nil {
26         panic(err)
27     }
28     fmt.Println("資料庫連線成功")
29 }
```

```
30      //產生參數化查詢敘述
31      rowStmt, err := db.Prepare("SELECT name FROM employee WHERE id=?")
32  ✓    if err != nil {
33      |        panic(err)
34      |    }
35      defer rowStmt.Close()
36
37      //用參數化查詢來取出符合的單一筆資料
38      e := employee{id: 305}
39      err = rowStmt.QueryRow(e.id).Scan(&e.name)
40  ✓    if err != nil {
41      |        panic(err)
42      |    }
43      fmt.Printf("id=%v 的員工名稱為 %v", e.id, e.name)
44  }
45
```

執行結果：

```
PS D:\git\Golang\ch13\13-6-2> go run .  
sql.DB 結構已建立  
資料庫連線成功  
id=305 的員工名稱為 Sue
```

- 注意到這次我們產生rowStmt (也是sql.Stmt結構) 後, 用了QueryRow()方法來查詢
- sql.DB或sql.Stmt結構都有Query()及QueryRow()方法; 差別在QueryRow最多只回傳一筆資料(sql.Row結構, 不是Rows結構)
- 當你只要一筆特定資料時, 這樣就方便許多, 不需要再用迴圈走訪了

13-7 更新既有資料

- 想要更新資料表內的既有資料，方法和前面插入資料的動作是很像的：

```
1 package main
2
3 import (
4     "database/sql"
5     "fmt"
6
7     _ "github.com/go-sql-driver/mysql"
8 )
9
10 type employee struct {
11     id   int
12     name string
13 }
14
15 func main() {
16     db, err := sql.Open("mysql", "user:1234@tcp(localhost:3306)/mysql?charset=utf8")
17
18     if err != nil {
19         panic(err)
20     }
21     defer db.Close()
22     fmt.Println("sql.DB 結構已建立")
23
24     err = db.Ping()
25     if err != nil {
26         panic(err)
27     }
28     fmt.Println("資料庫連線成功")
29 }
```

```
30 //產生參數化查詢敘述
31 updateStmt, err := db.Prepare("UPDATE employee SET name=? WHERE id=?")
32 if err != nil {
33     panic(err)
34 }
35 defer updateStmt.Close()
36
37 //將id為204的員工名字改成Robert，並執行參數化查詢
38 e := employee{204, "Robert"}
39 updatedResult, err := updateStmt.Exec(e.name, e.id)
40 if err != nil {
41     panic(err)
42 }
43 //檢查更新時影響幾筆資料
44 updatedRecords, err := updatedResult.RowsAffected()
45 if err != nil {
46     panic(err)
47 }
48 fmt.Println("更新資料筆數:", updatedRecords)
49 }
50
```


執行結果：

```
PS D:\git\Golang\ch13\13-7> go run .  
sql.DB 結構已建立  
資料庫連線成功  
更新資料筆數： 1
```

- 可以看到整個過程跟插入資料幾乎一樣，都是先用`db.Prepare()`產生參數化查詢敘述`updateStmt`後，再用`updateStmt.Exec()`來執行漢傳入參數
- 不過，這回我們也想檢查SQL指令更新資料時引響了幾筆資料，因此會多接收`Exec()`的第一個參數，並呼叫他的`RowsAffected()`方法：

`RowsAffected()` (**int64**, error)

第一個回傳值即代表受影響的資料數

13-8 練習：FizzBuzz統計表

- 現在建立一個資料表**FizzBuzz**, 包含兩個欄位：**number**(整數) 以及 **status**(長度為**10**的字串), 我們也要在新增資料表前嘗試刪除它, 這樣若有舊的資料表就會被取代
- 接著在該資料表插入**100**筆資料, **number**欄位即**1~100**, **status**欄位則設為空字串
- 然後更新資料, 規則如右:

number	status
15的倍數	FizzBuzz
3的倍數	Fizz
5的倍數	Buzz

- 最後將status欄位是FizzBuzz的資料全部刪除，並統計FizzBuzz剩下幾筆資料
- 註：在資料表中刪除資料的SQL語法如下：

```
DELETE FROM employee WHERE id=?
```

```
1 package main
2
3 import (
4     "database/sql"
5     "fmt"
6
7     _ "github.com/go-sql-driver/mysql"
8 )
9
10 func main() {
11     db, err := sql.Open("mysql", "user:1234@tcp(localhost:3306)/mysqldb?charset=utf8")
12     if err != nil {
13         panic(err)
14     }
15     defer db.Close()
16     if err := db.Ping(); err != nil { //檢查資料庫連線
17         panic(err)
18     }
19
20     //先刪除fizzbuzz
21     //這裡我們呼略錯誤，所以就算資料表不存在也沒有問題
22     DBDrop := "DROP TABLE fizzbuzz;"
23     db.Exec(DBDrop)
24 }
```

```
25 //新建資料表
26 DBCreate := `
27 CREATE TABLE fizzbuzz
28 (
29     number INT NOT NULL,
30     status VARCHAR(10)
31 );`
32 if _, err := db.Exec(DBCreate); err != nil {
33     panic(err)
34 }
35
36 //在fizzbuzz插入100筆資料
37 DBInsert := "INSERT INTO fizzbuzz (number, status) VALUES (?, ?);"
38 insertStmt, err := db.Prepare(DBInsert) //產生參數化查詢敘述
39 for err != nil {
40     panic(err)
41 }
42 for i := 1; i <= 100; i++ {
43     if _, err := insertStmt.Exec(i, ""); err != nil {
44         fmt.Println(err)
45     }
46 }
47 insertStmt.Close()
48 fmt.Println("插入資料筆數: 100")
49
```

```
50 //更新資料表
51 DBUpdate := "UPDATE fizzbuzz SET status=? WHERE MOD(number, ?)=0 AND status='"
52 updateStmt, err := db.Prepare(DBUpdate)
53 if err != nil {
54     panic(err)
55 }
56 numbers := []int{15, 3, 5}
57 statuses := []string{"FizzBuzz", "Fizz", "Buzz"}
58 for i := 0; i < 3; i++ {
59     updatedResult, err := updateStmt.Exec(statuses[i], numbers[i])
60     if err != nil {
61         panic(err)
62     }
63     //取得每次更新的筆數
64     updatedRecords, err := updatedResult.RowsAffected()
65     if err != nil {
66         panic(err)
67     }
68     fmt.Println(statuses[i], "更新筆數:", updatedRecords)
69 }
70 updateStmt.Close()
71
```

```
72 //刪除資料表內status = "FizzBuzz" 的項目
73 DBDelete := "DELETE FROM fizzbuzz WHERE status=?"
74 deleteStmt, err := db.Prepare(DBDelete)
75 if err != nil {
76     panic(err)
77 }
78 deletedResult, err := deleteStmt.Exec("FizzBuzz")
79 if err != nil {
80     panic(err)
81 }
82 //統計刪除筆數
83 deletedRecords, err := deletedResult.RowsAffected()
84 if err != nil {
85     panic(err)
86 }
87 fmt.Println("FizzBuzz 刪除筆數:", deletedRecords)
88
89 //用SQL函式COUNT()取得資料表資料筆數
90 rowStmt, err := db.Prepare("SELECT COUNT(*) FROM fizzbuzz")
91 if err != nil {
92     panic(err)
93 }
94 //由於只有一個結果，故用QueryRow()即可
95 var count int
96 if err := rowStmt.QueryRow().Scan(&count); err != nil {
97     panic(err)
98 }
99 fmt.Println("資料表總筆數: ", count)
100 rowStmt.Close()
101 }
102
```


執行結果：

```
PS D:\git\Golang\ch13\13-8> go run .  
插入資料筆數： 100  
FizzBuzz 更新筆數： 6  
Fizz 更新筆數： 27  
Buzz 更新筆數： 14  
FizzBuzz 刪除筆數： 6  
資料表總筆數： 94
```

本章結束