

Database Management System

Lab 2: Intermediate SQL

逢甲資工 許懷中

SQL Advanced

TRANSACTIONS

安裝 MySQL

- <https://dev.mysql.com/downloads/installer/>
- 下載 web-community 版本即可
- Custom 安裝
 - MySQL Server 5.7
 - MySQL Workbench 8.0
- 若缺少 MS VC++ 套件 到下列網址安裝
 - <https://www.microsoft.com/zh-tw/download/details.aspx?id=40784>
- 安裝完成後打開 MySQL Workbench

範例資料庫

- 下載並且安裝範例資料庫
 - <https://tinyurl.com/script-world>
 - <https://tinyurl.com/script-cmdev>
- 下載範例資料庫
 - <https://tinyurl.com/data-iris-hj>

交易 (Transaction)

- Transaction 是存取資料庫的基本單位
- 四大特性
 - ACID
 - Atomicity, Consistency, Isolation, Durability
 - 原子性(單元性)、一致性、獨立性、持續性

交易的特性 (cont.)

- 原子性 (Atomicity) a.k.a 單元性
 - 交易不可分割
 - 全部做完或完全沒做
 - 銀貨兩訖



交易的特性 (cont.)

- 一致性 (Consistency)
 - 一致的狀態 (Consistent State) 沒有違反任何約束 (Constraints) 的狀態
 - 交易得令資料庫從一個 Consistent State 轉移到另一個 Consistent State
 - 例如，行內轉帳前後，總金額必須相同



交易的特性 (cont.)

- 獨立性 (Isolation)
 - 同時進行的交易，其結果與順序執行相同



交易的特性 (cont.)

- 持續性 (Durability)
 - 交易一旦完成，其結果永久 (Permanent) 存在



交易實務

- 原子性
 - 全部被執行、或者全部被滾回 (rolled back)
- 平行的交易具有隔離性
- 交易並沒有明確的開始，但是有明確的結束，結束於 `commit` 或 `rollback`
- SQL 標準，預設所有的 SQL 指令都會自動 `commit`
 - SQL 1999: `begin atomic ... end`

測試 Transaction

- 開啟第二個 Connection
- 在兩個 connection 都執行
 - **select** @@AUTOCOMMIT;
 - **set** AUTOCOMMIT=0;
 - **use** cmdev;
- 開始進行交易的指令
 - **start transaction** 或 **begin**
- 結束交易的指令
 - **commit** (正常結束，正式執行 transaction 中的指令)
 - **rollback** (滾回，將 transaction 中造成的影響還原)

Connection 1	Connection2
begin; update emp set salary=1000 where empno=7369;	
	begin; select salary from emp where empno=7369;
	select salary from emp where empno=7369 lock in share mode;
	select salary from emp where empno=7369;
commit;	
	commit; select salary from emp where empno=7369;

Connection 1	Connection2
begin; select salary from emp where empno=7369;	
	begin; update emp set salary=1100 where empno=7369;
select salary from emp where empno=7369; select salary from emp where empno=7369 lock in share mode;	
	update emp set salary=1200 where empno=7369;
	commit;
commit;	

Connection 1	Connection2
begin; select salary from emp where empno=7369 lock in share mode;	
	begin; select salary from emp where empno=7369 lock in share mode; update emp set salary=1300 where empno=7369;
commit;	
	commit;

Connection 1	Connection2
begin; select salary from emp where empno=7369 for update;	
	begin; select salary from emp where empno=7369 lock in share mode;
update emp set salary=1400 where empno=7369; commit;	
	commit;

Lock

- 分享鎖 (S, Shared Lock)
 - 擁有此鎖的 connection 可以讀
- 排他鎖 (X, Exclusive Lock)
 - 擁有此鎖的 connection 可以寫(update or insert)
- 意圖分享鎖 (IS, Intention to Share)
- 意圖排他鎖 (IX, Intention for Exclusion)
 - 預告即將進行 update
 - 確保其他 transaction 不會看到錯誤的結果

Lock (cont.)

- **select ... from ...**
 - 取讀資料快照，與 lock 無關
- **select ... from ... lock in share mode**
 - 加共享鎖 (S)
- **select ... from ... for share**
 - 加意圖共享鎖 (IS)
- **select ... from ... for update**
 - 加意圖排他鎖 (IX)

Lock (cont.)

- **update ... where ...** (加排他鎖)
 - search 中被 touch 到的資料列均加上了排他鎖
- **delete ... from ... where...** (加排他鎖)
 - search 中被 touch 到的資料列均加上了排他鎖
- **insert** (加排他鎖)
 - 對於被 insert

Lock (cont.)

	X	IX	S	IS
X	<i>Conflict</i>	<i>Conflict</i>	<i>Conflict</i>	<i>Conflict</i>
IX	<i>Conflict</i>	<i>Compatible</i>	<i>Conflict</i>	<i>Compatible</i>
S	<i>Conflict</i>	<i>Conflict</i>	<i>Compatible</i>	<i>Compatible</i>
IS	<i>Conflict</i>	<i>Compatible</i>	<i>Compatible</i>	<i>Compatible</i>

讀取上的異常狀況

- 髒讀 (dirty read)
 - 留存的資料是被修改過的
- 無法重現讀取 (non-repeatable read)
 - 兩次讀取中間曾經被修改
- 幻讀 (phantom read)
 - 兩次取得的筆數不同

預設層級的差異

- MySQL 預設的交易安全層級為‘REPEATABLE-READ’可以避免上述問題
- 檢視目前安全層級
 - **select** @@transaction_isolation;
- 變動安全層級
 - **set SESSION** transaction_isolation = 'READ-UNCOMMITTED'
- 請在變更安全層級後，開始進行下列 lab

髒讀 (Dirty Read)

Connection 1	Connection2
begin; select salary from emp where empno=7369;	
	begin; update emp set salary=1500 where empno=7369;
select salary from emp where empno=7369;	
	rollback;
select salary from emp where empno=7369; commit;	

無法重現讀取

(Non-Repeatable Read)

Connection 1	Connection2
begin; select salary from emp where empno=7369;	
	begin; update emp set salary=1600 where empno=7369; commit;
select salary from emp where empno=7369; commit;	

幻讀 (Phantom Read)

Connection 1	Connection2
begin; select salary from emp where salary between 1000 and 1500;	
	begin; update emp set salary=1600 where empno=7876; commit;
select salary from emp where salary between 1000 and 1500; commit;	

Intermediate SQL

IMPORT DATA INTO DBMS

檢視欲匯入的文檔

"Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", "Species"

5.1,3.5,1.4,0.2, "setosa"

4.9,3,1.4,0.2, "setosa"

4.7,3.2,1.3,0.2, "setosa"

4.6,3.1,1.5,0.2, "setosa"

5,3.6,1.4,0.2, "setosa"

5.4,3.9,1.7,0.4, "setosa"

4.6,3.4,1.4,0.3, "setosa"

5,3.4,1.5,0.2, "setosa"

4.4,2.9,1.4,0.2, "setosa"

4.9,3.1,1.5,0.1, "setosa"

5.4,3.7,1.5,0.2, "setosa"

4.8,3.4,1.6,0.2, "setosa"

4.8,3,1.4,0.1, "setosa"

CSV 格式

- CSV (Comma Separated Values)
 - 以逗號區隔的結構化資料 (structured data)
 - 每一列都有同樣多的資料欄
 - 以雙引號標示字串
 - 通常第一列會是欄位名稱
- TSV (Tab Separated Values)
- 只是一種格式，檔案本質上是純文字檔

CSV 的眉眉角角

Year, Make, Model,
1997, Ford, E350
2000, Mercury, Cougar

"1997", "Ford", "E350"

1997, Ford, E350, "super, luxurious truck"

1997, Ford, E350, "super, ""luxurious"" truck"

1997, Ford, E350 1997, Ford, E350

1997, Ford, E350, 4.9

1997; Ford; E350; 4, 9

檔案匯入位址

- `show global variables like '%datadir%';`
- MySQL 的安全規則限制了匯入檔案的放置位址，因此需要將檔案放置於該位址底下

建立資料庫與資料表

- **CREATE DATABASE IF NOT EXISTS irisdb
CHARACTER SET 'utf8';**
- **CREATE TABLE iris (**
 id INT NOT NULL AUTO_INCREMENT,
 Sepal_Length DECIMAL(10 , 2) NOT NULL,
 Sepal_Width DECIMAL(10 , 2) NOT NULL,
 Petal_Length DECIMAL(10 , 2) NOT NULL,
 Petal_Width DECIMAL(10 , 2) NOT NULL,
 Species varchar(15) NOT NULL,
 PRIMARY KEY (id)
);
- **USE irisdb;**

匯入資料

- **LOAD DATA INFILE** 'C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/iris.csv' **INTO TABLE** iris
FIELDS TERMINATED BY ',' ENCLOSED BY ''
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;
- **SELECT * from** iris;

Intermediate SQL

PREPARED STATEMENT

使用者變數 (User Variables)

- MySQL Database Server所提供的一種簡易的儲存資料方式
- 儲存一些簡單的資料，例如：數字或字串
- 可以在後續操作中使用

語法：

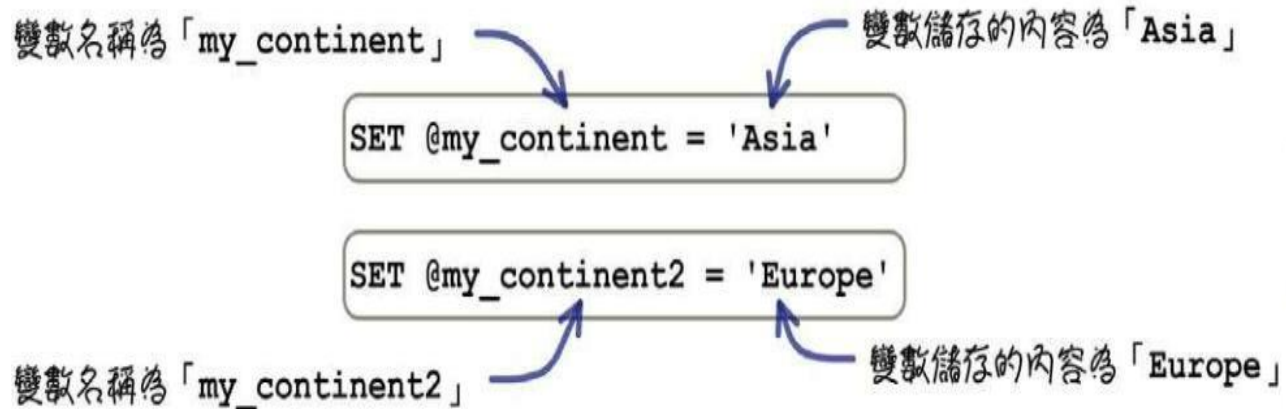
The diagram illustrates the syntax for setting user variables. It features the text `SET @變數名稱 {= | :=} 值 [, ...]` with several handwritten annotations and arrows pointing to specific parts of the syntax:

- An arrow points from the text "為變數取一個名稱" to the `@變數名稱` part of the syntax.
- An arrow points from the text "使用「=」或「:=」都可以" to the `{= | :=}` part of the syntax.
- An arrow points from the text "使用「SET」" to the `SET` keyword.
- An arrow points from the text "要指定給變數保存起來的值" to the `值` part of the syntax.
- An arrow points from the text "可以一次設定多個變數" to the `[, ...]` part of the syntax.

```
SET @變數名稱 {= | :=} 值 [, ...]
```

設定使用者變數

- 下列敘述設定兩個儲存字串資料的使用者變數：



- 設定好後可以使用 SELECT 敘述中查詢儲存內容



設定使用者變數

- 多個變數

設定一個名稱為「my_region」,儲存內容為「Eastern Asia」的變數

使用逗號

```
SET @my_region = 'Eastern Asia', @my_region2 = 'Middle Asia'
```

設定另一個變數

- 查詢以確認設定好的使用者變數：

查詢「my_region」與「my_region2」兩個變數儲存的內容

```
SELECT @my_region, @my_region2
```

@my_region	@my_region2
Eastern Asia	Middle Asia

使用SELECT (cont.)

- SELECT 可以用指定符號 := 設定變數

為變數取一個名稱 → 只能使用「:=」
使用「SELECT」 → SELECT @變數名稱 := 值 [, ...]
要指定給變數保存起來的值 → 可以一次設定多個變數

- 使用SELECT敘述，設定儲存資料的使用者變數
- 設定好以後會顯示設定的使用者變數內容

設定與查詢「my_gnp」與「my_gnp2」兩個變數

```
SELECT @my_gnp := 30000, @my_gnp2 := 5000
```

@my_gnp := 30000	@my_gnp2 := 5000
30000	5000

使用SELECT (cont.)

- 已設定好的使用者變數，可以使用在大部分的敘述中
- 如下列範例使用變數來設定查詢敘述的條件設定

```
SET @my_continent = 'Asia';  
SET @my_continent2 = 'Europe';
```

已經設定好的
使用者變數

```
SELECT Continent, Name, GNP, Population  
FROM country  
WHERE Continent IN (@my_continent, @my_continent2)
```

使用變數儲存的
內容來設定條件

Continent	Name	GNP	Population
Asia	Afghanistan	5976.00	22720000
Europe	Netherlands	371362.00	15864000
Europe	Albania	3205.00	3401200
Europe	Andorra	1630.00	78000
Asia	United Arab Emirates	37966.00	2441000

...

使用 SELECT (cont.)

- 使用SELECT敘述設定使用者變數的方式，也可以直接把查詢敘述傳回的資料儲存起來

The diagram illustrates the execution of a SQL statement to set user variables. The SQL code is as follows:

```
SELECT @max_gnp := MAX(GNP),  
       @max_population := MAX(Population)  
FROM   country
```

Annotations with arrows point to the code:

- An arrow points from the text "把GNP最大值儲存在「max_gnp」變數" to the `@max_gnp := MAX(GNP)` part of the SELECT statement.
- An arrow points from the text "把Population最大值儲存在「max_population」變數" to the `@max_population := MAX(Population)` part of the SELECT statement.
- An arrow points from the `FROM country` part of the statement to the result table below.

The result table shows the values assigned to the variables:

@max_gnp := MAX(GNP)	@max_population := MAX(Population)
8510700.00	1277558000

- 上列範例執行後所設定的使用者變數也可以用在後續的敘述中：

This diagram is identical to the one above, showing the same SQL code and result table. It includes the same annotations explaining how the SQL statement sets the user variables `@max_gnp` and `@max_population` to the maximum GNP and Population values from the `country` table.

使用 SELECT (cont.)

- 也可以拿使用者變數來運算

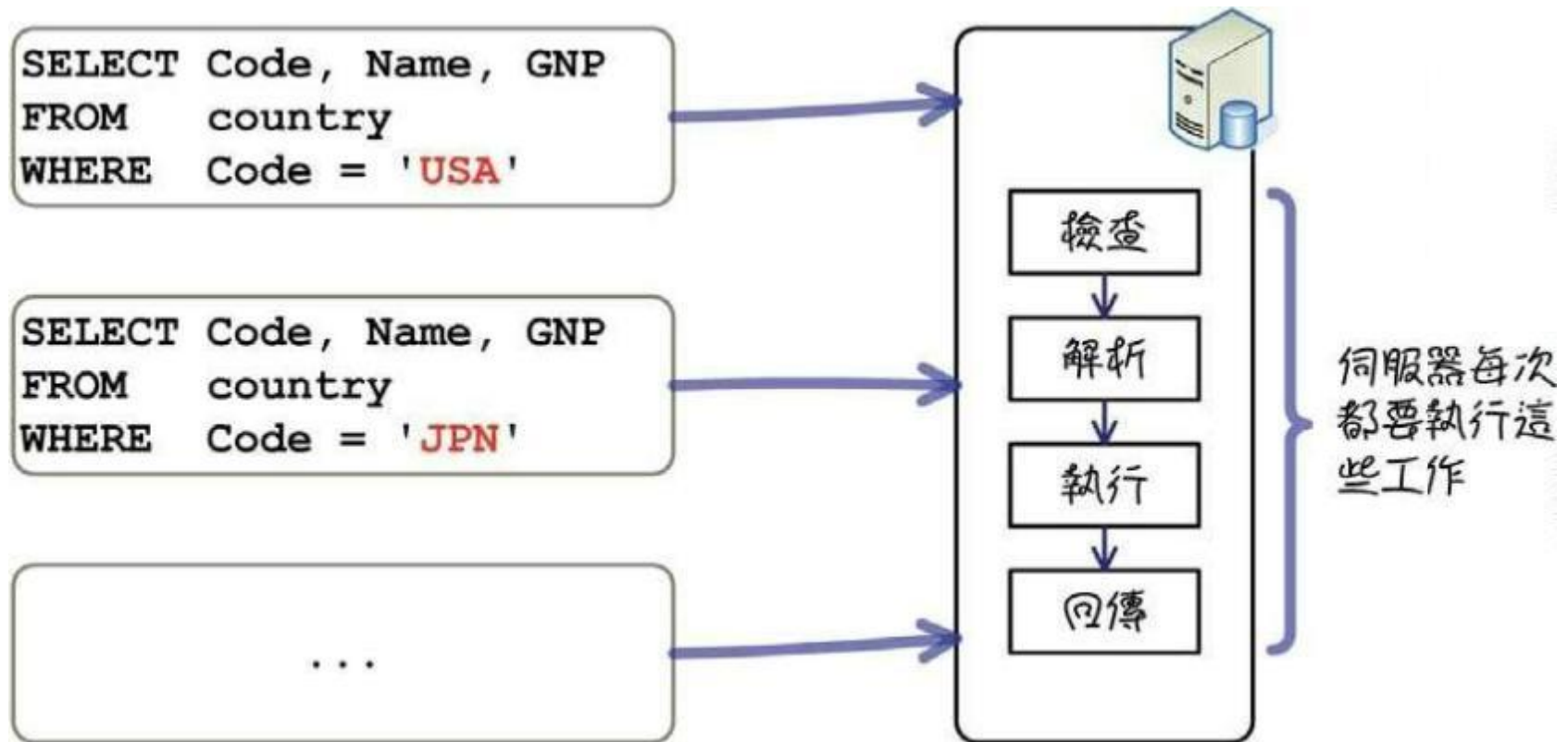
```
SELECT Name, @max_gnp - GNP, @max_population - Population  
FROM country
```

使用變數儲存的内容執行運算

Name	@max_gnp - GNP	@max_population - Population
Afghanistan	8504724	1254838000
Netherlands	8139338	1261694000
Netherlands Antilles	8508759	1277341000
Albania	8507495	1274156800
Algeria	8460718	1246087000

...

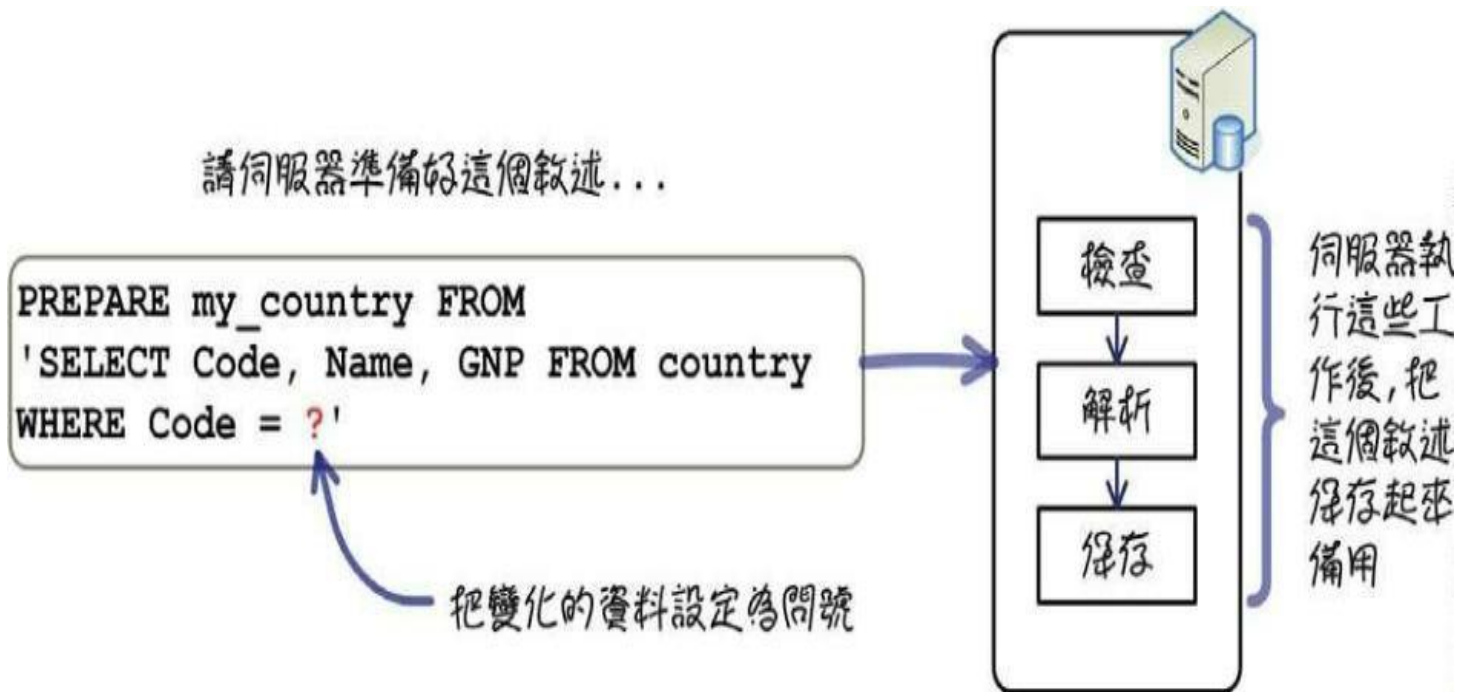
Prepared Statements 的應用



許多 Query 只有條件不同，其他大部分皆相同

Prepared Statements的應用

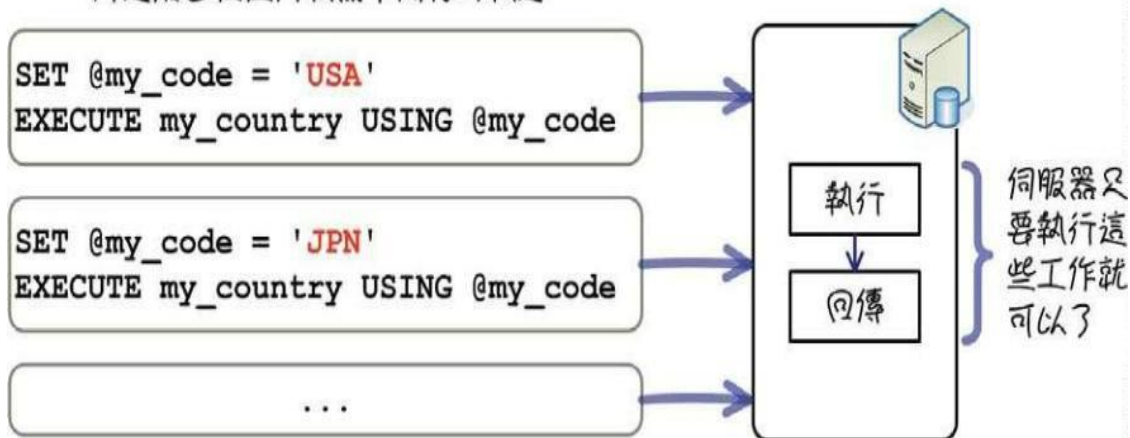
- 如果有許多相似內容的敘述要執行，可以使用 Prepared Statements 改善資料庫效率



Prepared Statements

- 要先設定好資料與 Query，再請Server執行指定的Prepared Statements
- 主要應用在資料庫應用程式中
- 要在MySQL Command Line Client才能正確的建立與執行

使用變數設定必要的資料後，再執行這個已經在伺服器準備好的敘述



建立 Prepared Statements

- 如果常需要查詢，可以建立 Prepared Statements，語法如下：

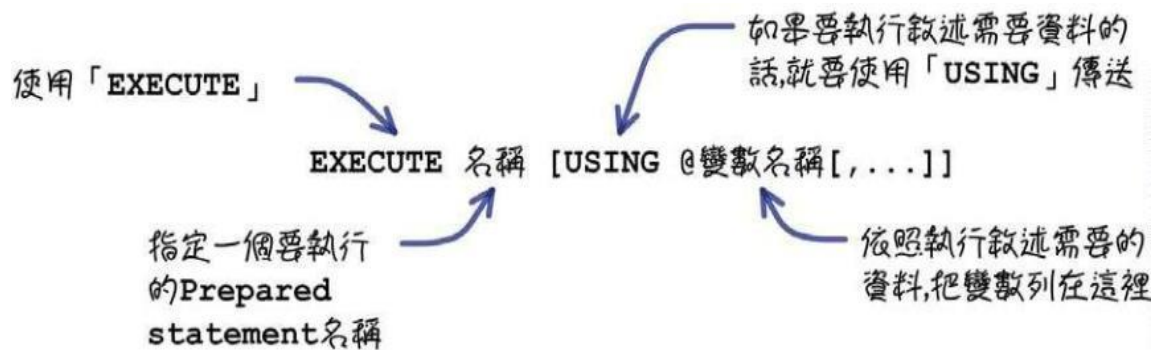
使用「PREPARE」與「FROM」
PREPARE 名稱 FROM '敘述'
在單引號或雙引號中填入要伺服器準備的敘述
取一個名稱

- 敘述中的問號是 參數標記 (parameter marker)，表示執行這個 Prepared Statements 需要的參數資料

名稱為「my_country」
mysql> PREPARE my_country FROM
-> 'SELECT Code, Name, GNP FROM world.country WHERE Code = ?';
Query OK, 0 rows affected (0.05 sec)
Statement prepared
把變化的資料設定為問號, 表示執行這個敘述需要一個資料
在「Command Line Client」中記得要在敘述結尾加一個分號
建立成功

執行 Prepared Statements

- 建立好之後，必須用 EXECUTE 執行



- 執行 Prepared Statements 不一定需要傳參數進去，而是依據其敘述是否包含問號來決定
- 若有，則每個問號需要設定相應的使用者變數
- 在 command line 以 USING 傳送資料給 Prepared Statements 使用

執行 Prepared Statements

先使用「SET」設定一個變數...

```
mysql> SET @my_code = 'USA';  
Query OK, 0 rows affected (0.00 sec)
```

使用「EXECUTE」執行「my_country」

使用「USING」傳送「my_code」變數

```
mysql> EXECUTE my_country USING @my_code;  
+-----+-----+-----+  
| Code | Name           | GNP           |  
+-----+-----+-----+  
| USA  | United States | 8510700.00    |  
+-----+-----+-----+  
1 row in set (0.00 sec)
```

傳回執行後的結果

執行 Prepared Statements

- 後續要執行此查詢，依照相同步驟依樣畫葫蘆即可

把變數內容重新設定為「JPN」

```
mysql> SET @my_code = 'JPN';  
Query OK, 0 rows affected (0.00 sec)
```

再執行一次

```
mysql> EXECUTE my_country USING @my_code;
```

Code	Name	GNP
JPN	Japan	3787042.00

傳回執行後的結果

```
1 row in set (0.00 sec)
```

移除 Prepared Statements

- 刪除指定 Prepared Statements :

使用「DEALLOCATE」或「DROP」都可以

```
{ DEALLOCATE | DROP } PREPARE 名稱
```

指定一個要刪除的Prepared statement名稱

- 範例 :

刪除「my_country」

```
mysql> DEALLOCATE PREPARE my_country;  
Query OK, 0 rows affected (0.00 sec)
```


Prepared Statements 的參數

- 在建立 Prepared Statements 時，會依照敘述需求設定參數標記，多個標記表示需要傳入多筆參數
- 以新增紀錄的敘述為例：

名稱為「new_dept」

```
mysql> PREPARE new_dept FROM  
-> 'INSERT INTO cmdev.dept VALUES (?, ?, ?)';  
Query OK, 0 rows affected (0.00 sec)  
Statement prepared
```

需要三個資料給「INSERT」
敘述使用

Prepared Statements 的參數 (cont.)

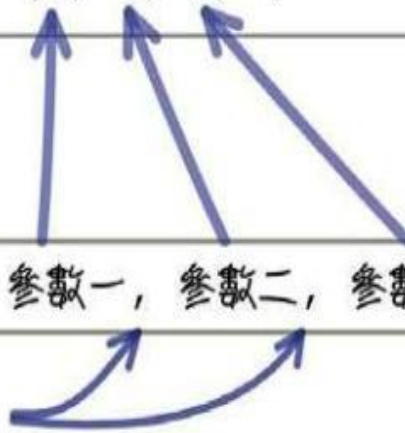
- 根據 Prepared Statements 使用的參數標記，若傳送錯誤參數資料，則會產生錯誤訊息。

```
PREPARE new_dept FROM  
'INSERT INTO cmdev.dept VALUES (?, ?, ?)';
```

依照順序傳送給Prepared
statement的問號使用


```
EXECUTE new_dept USING 參數一, 參數二, 參數三;
```

參數資料間使用逗號隔開




Prepared Statements 的參數 (cont.)

- 先把要新增部門的編號、名稱、地點設定為使用這變數
- 在執行new_dept時，傳送給它使用



```
mysql> SET @my_deptno=99, @my_dname='HR', @my_location='TAIPEI';  
Query OK, 0 rows affected (0.00 sec)
```

使用「SET」設定三個變數，
依序為部門編號、名稱與地點



```
mysql> EXECUTE new_dept USING @my_deptno, @my_dname, @my_location;  
Query OK, 1 row affected (0.10 sec)
```

執行「new_dept」並傳送三個變數



Prepared Statements 的參數 (cont.)

- 參數數量若有誤，則會產生錯誤訊息

重新設定部門編號與名稱變數

```
mysql> SET @my_deptno=101, @my_dname='IT';  
Query OK, 0 rows affected (0.00 sec)
```

執行的時候只有傳送兩個變數

```
mysql> EXECUTE new_dept USING @my_deptno, @my_dname;  
ERROR 1210 (HY000): Incorrect arguments to EXECUTE
```

產生參數錯誤的訊息

Prepared Statements 的參數 (cont.)

- 若傳送的使用者變數不存在，會自動以 NULL 值代替

mysql> EXECUTE new_dept USING @my_deptno, @my_dname, @not_exists;
Query OK, 1 row affected (0.10 sec)

使用一個不存在的變數

執行以後不會有錯誤

deptno	dname	location
101	IT	NULL

會填入「NULL」值

Intermediate SQL

STORED ROUTINES

Stored Routines

- 在 MySQL 資料庫管理系統中，會把 Stored Procedures 和 Stored Functions 合稱 Stored Routines
- Stored Function 必須要 return value 而 Stored Procedure 則無此要求
- Stored Procedure 使用上較為寬鬆而廣泛，這邊的 lab 主要教各位使用 Stored Procedure，以下簡稱 SP

Stored Routines 應用

- 在比較複雜的應用程式需求下，有些工作經常需要重複執行

刪除已經存在的表格

```
DROP TABLE IF EXISTS mycountry
```

設定要處理的國家代碼

```
SET @my_code := 'JPN'
```

查詢國家的人口數

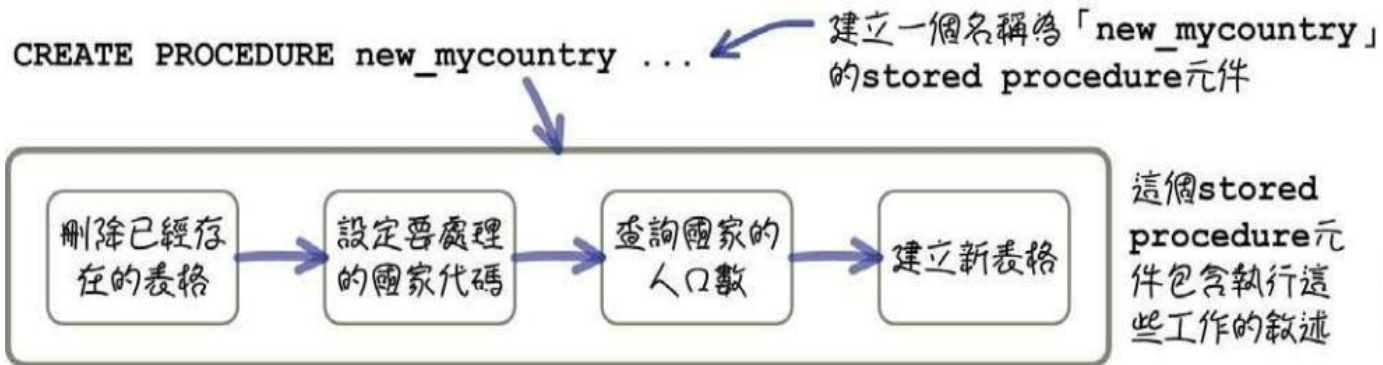
```
SELECT @pop_var := Population  
FROM    country  
WHERE   Code = @my_code
```

建立新表格

```
CREATE TABLE mycountry  
SELECT    Code, Name, GNP, Population  
FROM      country  
WHERE     Code = @my_code OR Population > @pop_var  
ORDER BY  Population
```

Stored Routines應用

- SQL敘述的特點是一次只能執行一件工作
- 所以若一組工作很常執行，就可以考慮把要執行的敘述建立為Stored procedure元件



Stored Routines 應用

- 把一組工作建立為 SP 元件後，以後要執行時，就可以呼叫該元件

`CALL new_mycountry('JPN')`

呼叫「new_mycountry」並傳入參數「JPN」

人口數比「JPN」多的國家

Code	Name	GNP	Population
JPN	Japan	3787042.00	126714000
BGD	Bangladesh	32852.00	129155000
RUS	Russian Federation	276608.00	146934000
PAK	Pakistan	61289.00	156483000
BRA	Brazil	776739.00	170115000
IDN	Indonesia	84982.00	212107000
USA	United States	8510700.00	278357000
IND	India	447114.00	1013662000
CHN	China	982268.00	1277558000

- 範例：建立人口數比USA多的國家表格時，只需傳入指定國家代碼

`CALL new_mycountry('USA')`

呼叫「new_mycountry」並傳入參數「USA」

人口數比「USA」多的國家

Code	Name	GNP	Population
USA	United States	8510700.00	278357000
IND	India	447114.00	1013662000
CHN	China	982268.00	1277558000

Stored Procedures

- SP 是一種 Stored Routines 元件，可以建立、刪除、維護，用來儲存程序，程序表示一組特定工作
- 若常常需要執行同一組工作，就可考慮將敘述建立成 SP 元件
- 可以視需要在 DB 中建立許多不同用途的 SP
- 可以包含需執行的一組工作，或必要的參數資料 (例如上列 new_country 中的國家代碼)
- 呼叫這些建立好的 SP，可以省掉很多繁複的工作

Stored Procedures (cont.)

■ 建立：

為procedure元件取一個名稱

設定呼叫procedure需要的參數資料

```
CREATE PROCEDURE 名稱 ( [參數[,...]] )  
Procedure程式碼
```

把procedure要執行的工作寫在這裡

■ 刪除：

如果指定的procedure元件存在的話就刪除;如果不存在的話,也不會出現錯誤訊息

```
DROP PROCEDURE [IF EXISTS] 名稱
```

指定要刪除的procedure元件名稱

■ 呼叫：

指定要呼叫的procedure名稱

依照需要提供參數資料

使用「CALL」

```
CALL Procedure名稱( 參數[,...])
```

管理 Stored Routines

- Stored Routines元件中可以包含許多要執行的 SQL 敘述
- 也可以包含宣告與設定變數，和控制執行流程的指令
- Stored Routines 元件有一點類似開發應用程式用的程式語言
- 但不像程式語言那麼複雜，且大部分都是跟資料庫相關的SQL敘述。

以 SQL Script 建立 Stored Procedures

- 由於 SP 中通常會包含許多需要的敘述，通常會使用 SQL Script 來建立 Stored Procedures
- SQL Script 也就是將想執行的 SQL 指令集中同一檔案中

SQL Script範例

- 以cmdev.sql為例：

以「--」開始的話,表示這行是註解,只是用來說明用的,而不是要執行的敘述

```
-- MySQL DB Development Certification Guide
-- Database : cmdev

set character_set_client='big5';
set character_set_connection='big5';
set character_set_results='big5';

DROP DATABASE IF EXISTS cmdev;

CREATE DATABASE cmdev CHARACTER SET big5;

USE cmdev;
...
```

每一個敘述後面都要使用分號表示敘述的結尾,MySQL稱為「delimiter」

Delimiters in SQL Script

- MySQL使用分號做為預設的delimiter
- 可以使用DELIMITER指令，修改預設的符號
- 範例：



The diagram illustrates the use of delimiters in a MySQL script. It shows a sequence of SQL commands, with blue arrows pointing to specific parts and Chinese annotations explaining their purpose.

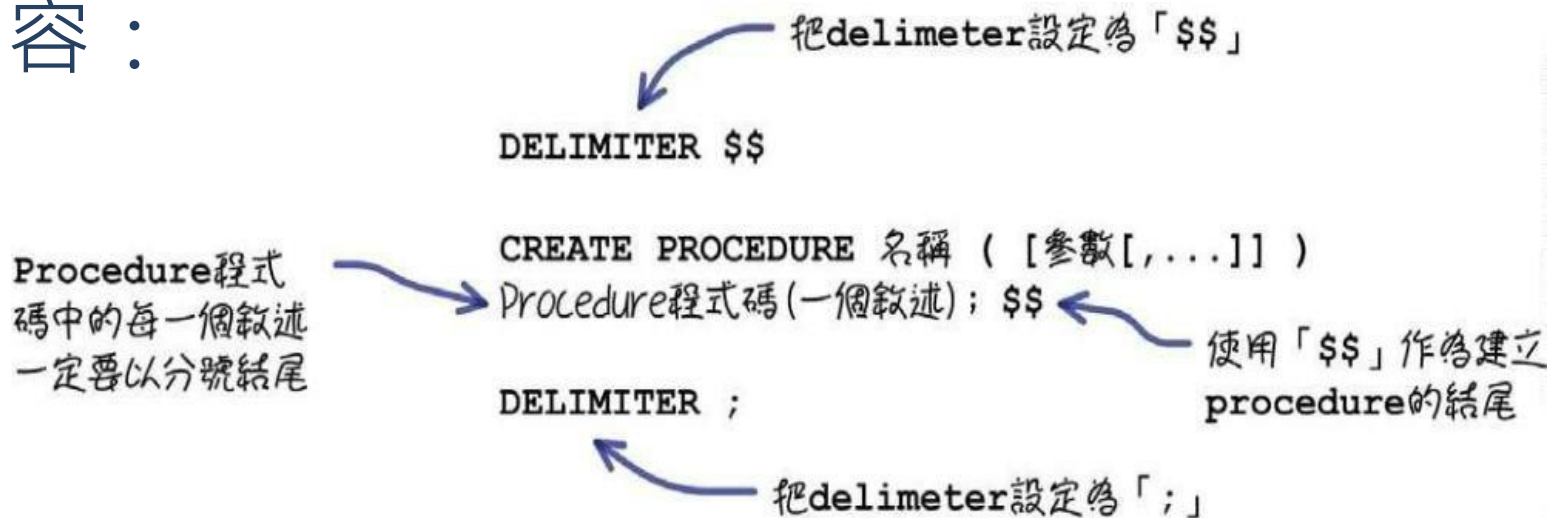
```
DELIMITER $$  
  
set character_set_client='big5'$$  
set character_set_connection='big5'$$  
set character_set_results='big5'$$  
  
DROP DATABASE IF EXISTS cmdev$$  
  
CREATE DATABASE cmdev CHARACTER SET big5$$  
  
USE cmdev$$  
...  
DELIMITER ;
```

Annotations:

- 使用「DELIMITER」把delimiter從「;」設定為「\$\$」 (Use 「DELIMITER」 to set the delimiter from 「;」 to 「\$\$」)
- 每一個敘述後面都要使用「\$\$」 (Each statement must use 「\$\$」 at the end)
- 通常在結尾會把delimiter設定為預設的「;」 (Usually at the end, the delimiter is set to the default 「;」)

為了 SP 修改 Delimiter

- 在一般應用時通常不會修改預設 Delimiter
- 但在建立 Stored Routines 元件的 SQL Script 檔案中就一定要使用
- 下列是建立 Stored procedures 元件的基本內容：



以 SQL Script 建立 SP

1. 在MySQL Query Browser中選擇功能表 File > New Script Tab，接下來可以輸入下列敘述建立Stored procedures：

```
DELIMITER $$
```

```
CREATE PROCEDURE show_countries ( )
```

```
SELECT * FROM country; $$
```

```
DELIMITER ;
```

指定名稱為「show_countries」

不需要參數的話，左右
括號中就沒有任何東西

這個procedure要執行的
敘述，最後一定要使用分號

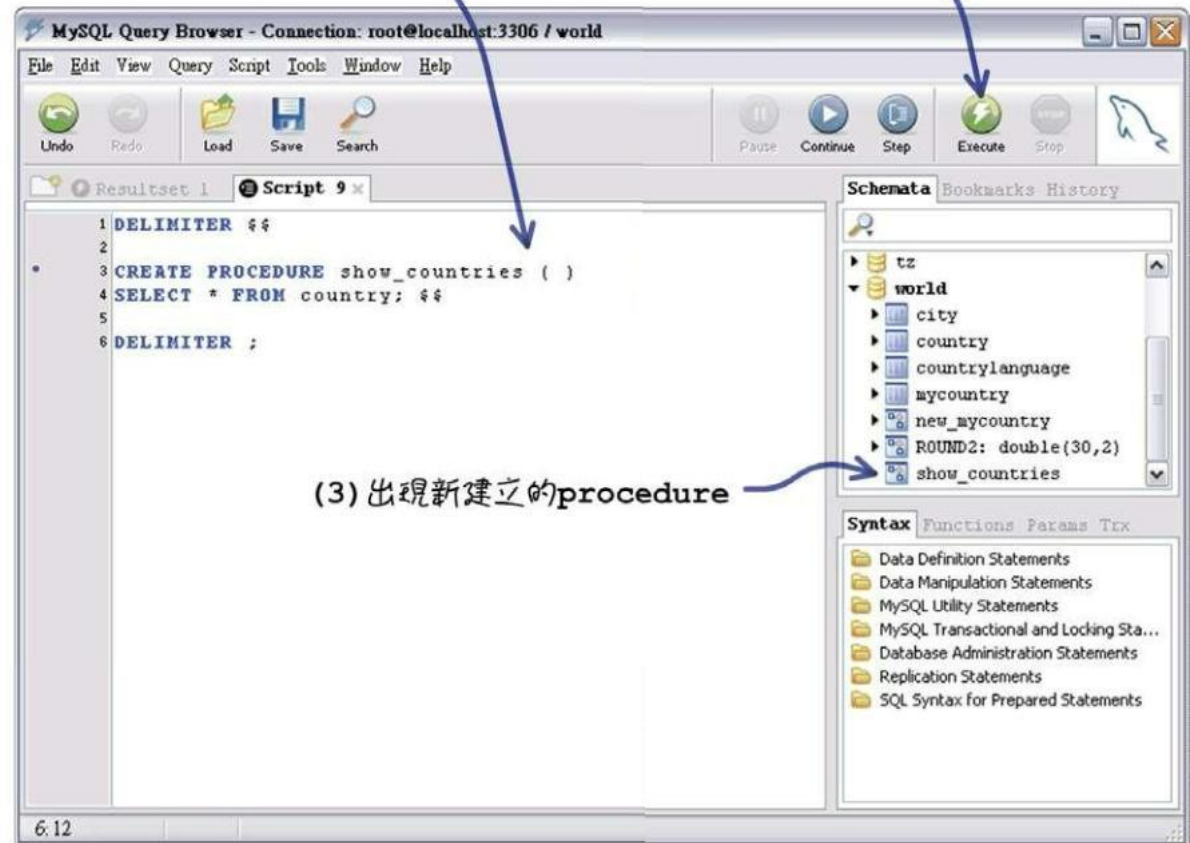
建立procedure的結尾

建立 SP 元件

2. 建立後執行

(1) 完成建立procedure的敘述

(2) 選擇「Execute」



建立多敘述之 SP

- 包含多個敘述的 SP，需要用BEGIN 與 END：

```
DELIMITER $$  
  
CREATE PROCEDURE 名稱 ( [參數[,...]] )  
BEGIN  
    Procedure程式碼(多個敘述)  
END $$  
  
DELIMITER ;
```

包含多個敘述時，一定要放在「BEGIN」與「END」之間

在「END」後面要使用「\$\$」作為建立procedure的結尾

建立多敘述之 SP (cont.)

- 範例：一次查詢國家、語言、城市三個表格的數量

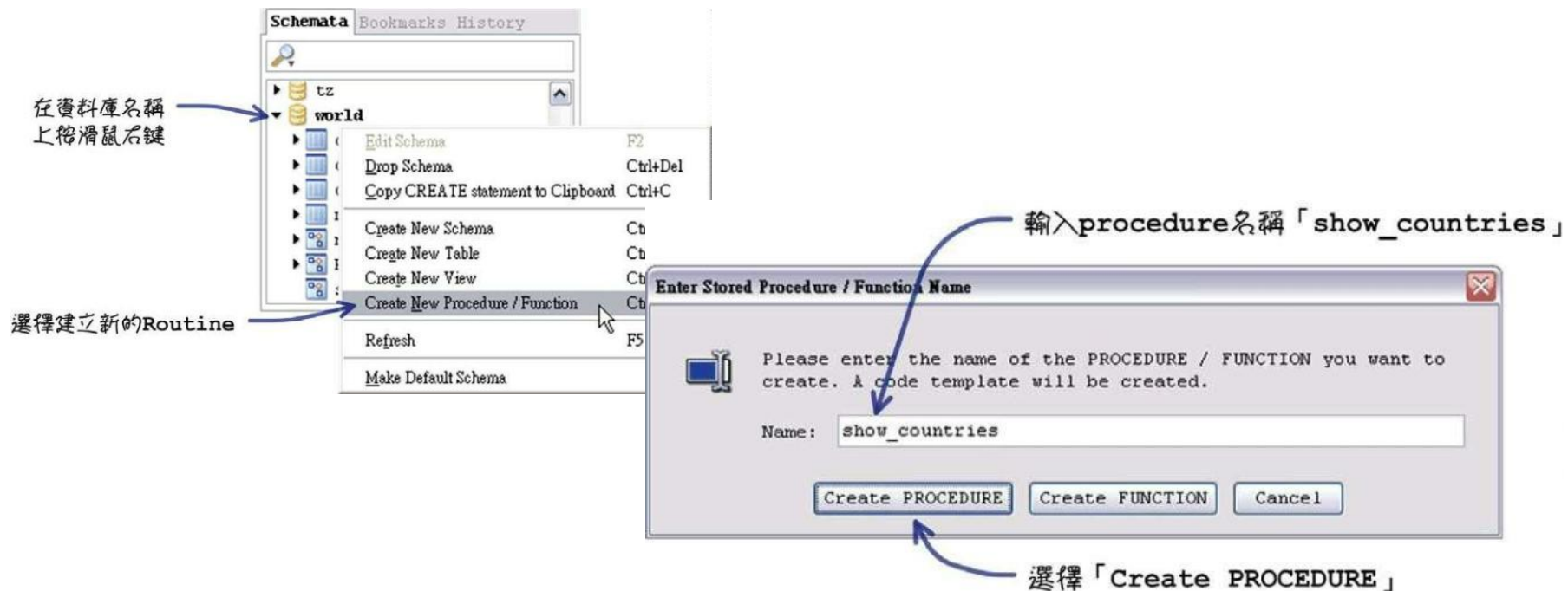
```
DELIMITER $$  
  
CREATE PROCEDURE my_world_count ( )  
BEGIN  
    SELECT COUNT(*) countrycount FROM country;  
    SELECT COUNT(*) languagecount FROM countrylanguage;  
    SELECT COUNT(*) citycount FROM city;  
END $$  
  
DELIMITER ;
```

指定名稱為「my_world_count」

包含多個敘述

管理 SP

- 除了使用 SQL script 建立需要的 Stored procedures 之外，也可以使用 MySQL Query Browser 來管理



管理 SP (cont.)

- MySQL Query Browser 會準備樣版
- 在 BEGIN 與 END 之間輸入需要執行的敘述再執行此 SQL script

先刪除舊的procedure

```
DELIMITER $$
```

```
DROP PROCEDURE IF EXISTS `world`.`show_countries` $$
```

```
CREATE PROCEDURE `world`.`show_countries` ()
```

```
BEGIN
```

```
END $$
```

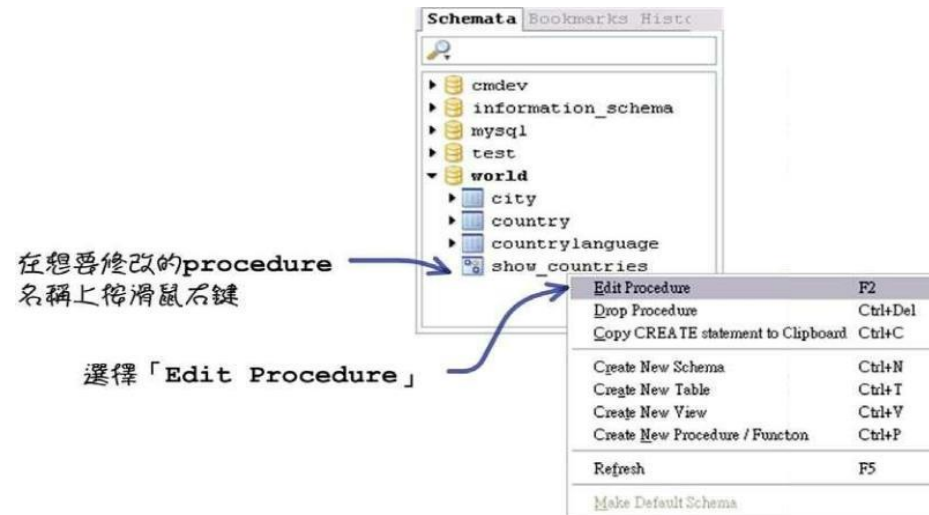
```
DELIMITER ;
```

MySQL會自動幫你把名稱
使用單引號包起來

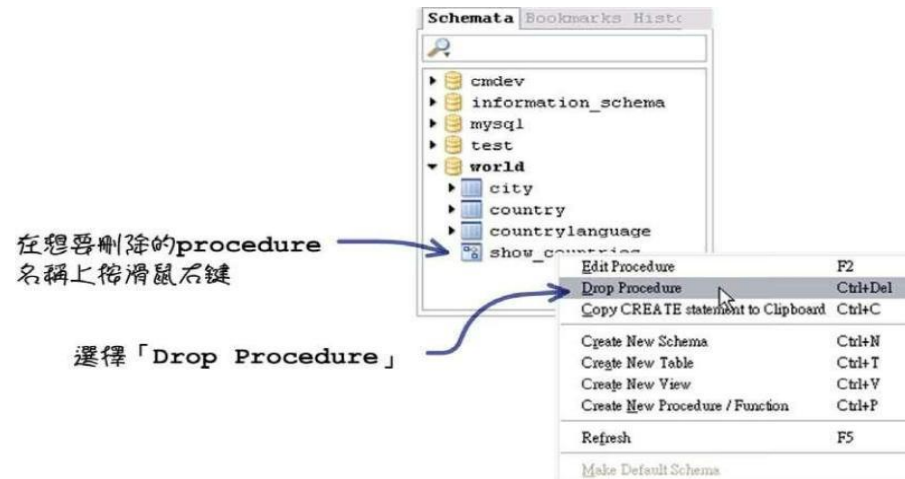
把這個procedure要執行的工作
填在「BEGIN」和「END」之間

管理 SP (cont.)

■ 修改：



■ 刪除：

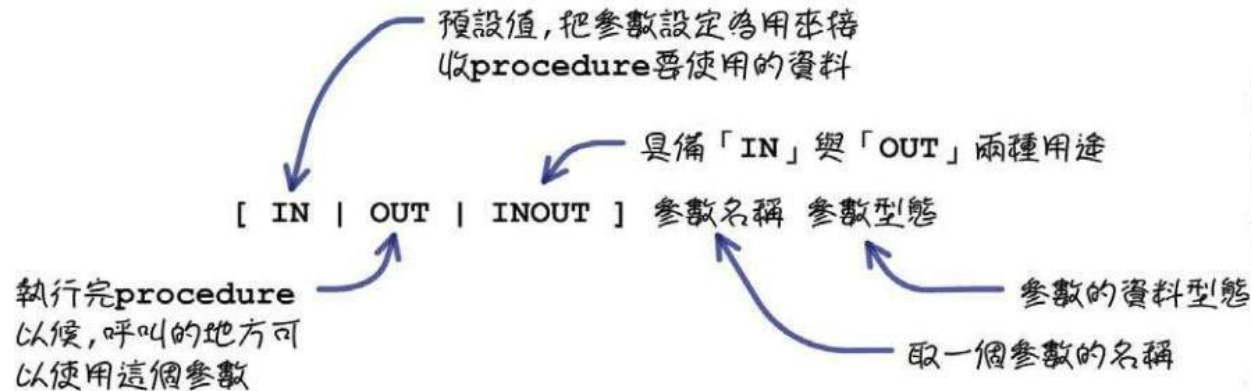


Stored Routines 的參數

- Stored routines 可以使用參數 (parameters) 讓使用者傳送資料給 Stored routines 使用

Stored Procedures 的參數

- Procedure參數定義，除了名稱、型態、順序，還有各參數用途



- IN：輸入用參數
- OUT：輸出用參數
- INOUT：同時有 IN 跟 OUT 兩種用途

參數範例

```
CREATE PROCEDURE test_param ( IN pi_in INT,  
                             OUT po_out INT,  
                             INOUT pio_inout INT )  
  
BEGIN  
    SELECT pi_in, po_out, pio_inout;  
  
    SET pi_in = 99, po_out = 99, pio_inout = 99;  
  
END
```

設定為「IN」,「OUT」與「INOUT」三種參數

顯示三個參數的值

設定三個參數的值

參數範例 (cont.)

- 呼叫procedures時，依照定義的參數個數、型態來傳送資料

```
CALL test_param ( 1 )
```

這個procedure需要三個參數，這裡卻只有給一個

! Incorrect number of arguments for PROCEDURE world.test_param; expected 3, got 1

- 呼叫procedures時傳送的參數資料，因不同用途有不同限制

「IN」參數可以指定一個值或變數名稱

「OUT」或「INOUT」參數只能指定一個變數名稱

```
CALL test_param (參數一, 參數二, 參數三)
```

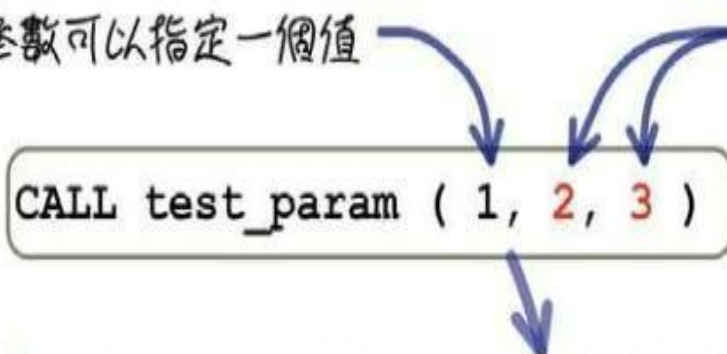
```
... ( IN pi_in INT, OUT po_out INT, INOUT pio_inout INT )  
BEGIN  
...  
END
```

參數範例 (cont.)

- 違反參數用途規定，發生錯誤

「IN」參數可以指定一個值

「OUT」或「INOUT」參數
不可以使用一個值傳送參數

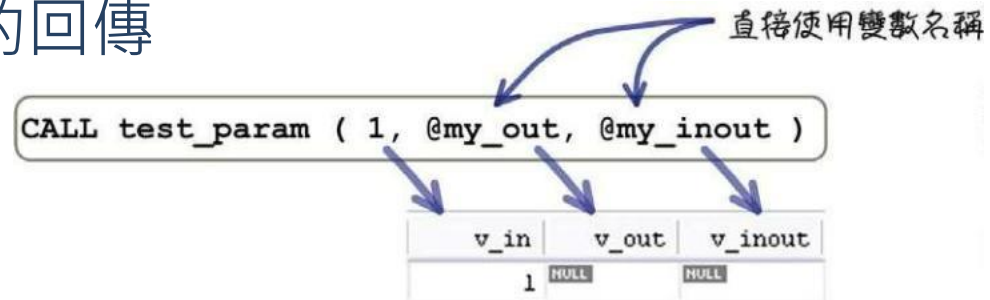


```
CALL test_param ( 1, 2, 3 )
```

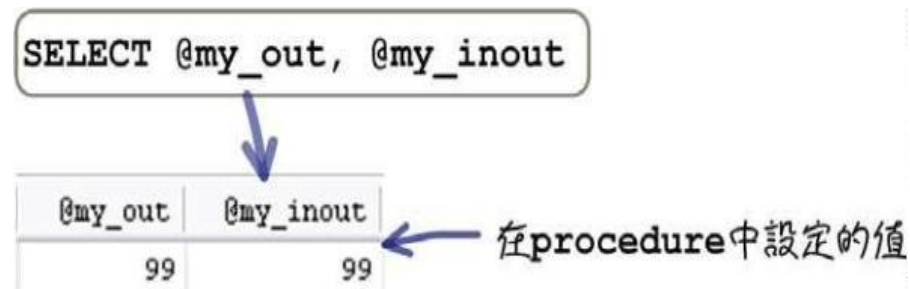
! OUT or INOUT argument 2 for routine world.test_param
is not a variable or NEW pseudo-variable in BEFORE
trigger

參數範例 (cont.)

- 在呼叫 SP 時，OUT 與 INOUT 必須指定變數作為參數，因為 OUT 與 INOUT 在執行完成後會回傳資料，使用變數才能接收 SP 的回傳



- 執行 SP 以後，指定給 OUT 與 INOUT 的變數，會儲存 SP 中所設定的值



參數範例 (cont.)

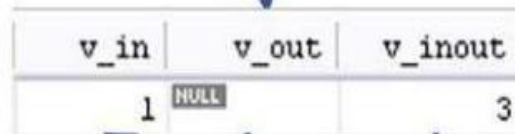
- 如果在呼叫 SP 之前，先把參數資料設定為使用者變數，在指定給參數使用：

先設定好三個使用者變數

```
SET @my_in = 1, @my_out = 2, @my_inout = 3
```

三個參數都使用變數

```
CALL test_param ( @my_in, @my_out, @my_inout )
```



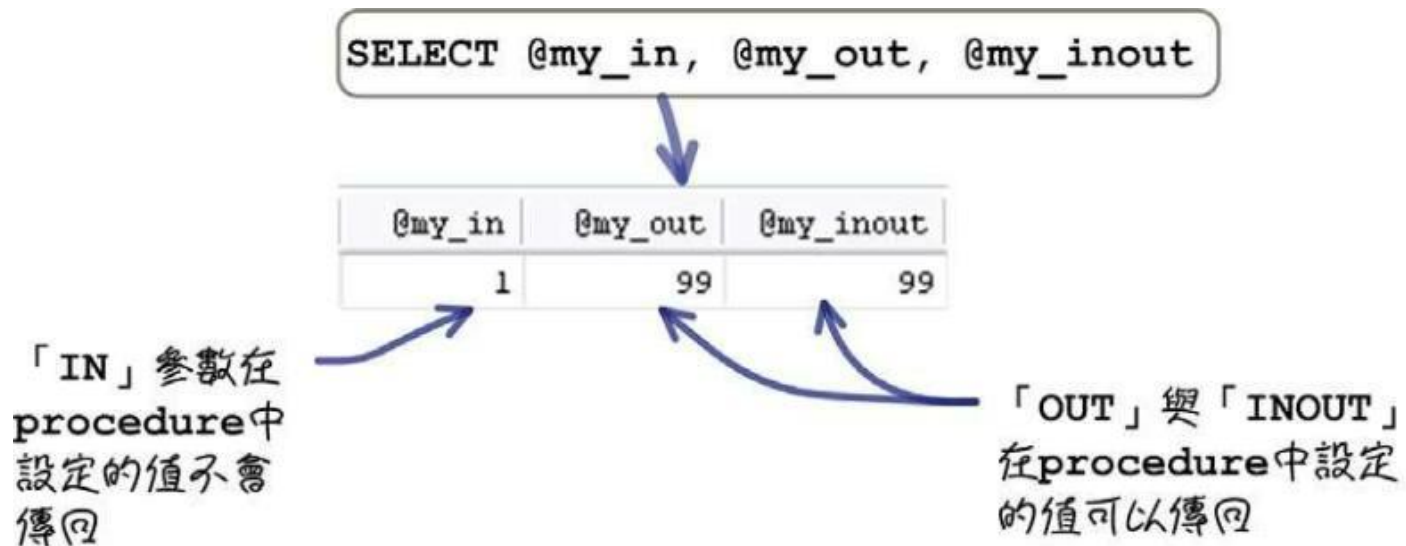
v_in	v_out	v_inout
1	NULL	3

「OUT」參數不能接收傳送過來的參數值

「IN」與「INOUT」參數可以接收傳送過來的參數值

參數範例 (cont.)

- 執行上列呼叫 SP 的敘述後，會發現設定為 OUT 用途的變數值是不會傳入 SP
- 此外，也會發現設定為 IN 用途的變數沒有接收回傳資料的功能



參數範例 (cont.)

- 只是用來設定查詢條件使用，不需要回傳資料的參數適合設定為 IN

這個procedure需要接收一個洲名的參數，它會顯示指定洲名的國家數量

洲名的參數用來設定條件用的，所以設定為「IN」

```
CREATE PROCEDURE country_count ( IN pi_con VARCHAR(26) )  
BEGIN  
    SELECT COUNT(*) FROM country WHERE Continent = pi_con;  
END
```

使用接收到的參數設定查詢條件

- 先設定好使用者變數儲存洲名，再呼叫：

```
SET @my_con = 'Europe'
```

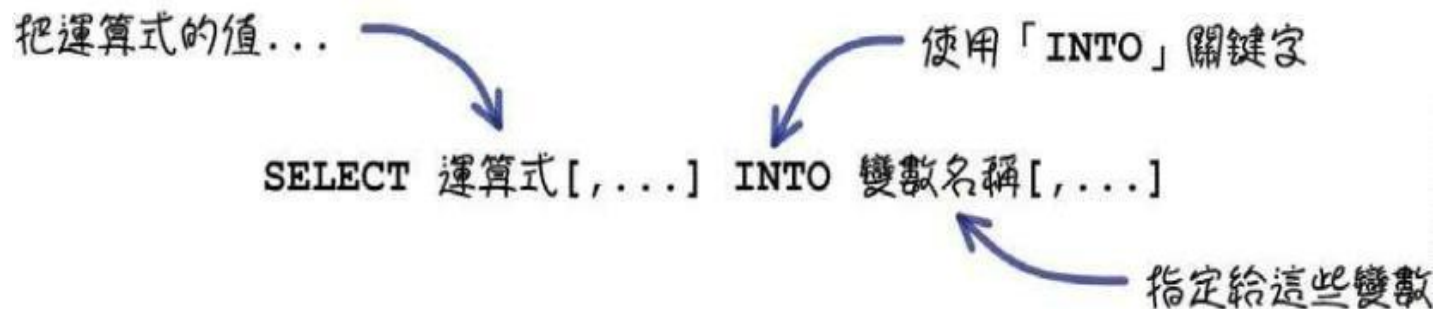
先設定好一個儲存洲名的使用者變數

```
CALL country_count( @my_con )
```

使用設定好的使用者變數作為參數

Stored Routines 的參數

- MySQL提供一種特別的查詢敘述給 Stored Routines 內使用
- 一般查詢敘述是用來回傳需要的資料用的
- 而 INTO 這個查詢敘述可以將 SELECT 敘述中取得的指定給變數



參數範例

這個procedure需要接收一個
洲名與回傳用的數量參數,它
會傳回指定洲名的國家數量

```
CREATE PROCEDURE country_count2  
  ( IN pi_con VARCHAR(26), OUT po_count INT )  
BEGIN  
  SELECT COUNT(*) INTO po_count  
  FROM    country  
  WHERE   Continent = pi_con;  
END
```

數量的參數用來回傳用的,
所以設定為「OUT」

把查詢得到的數字指定
給「po_count」變數

參數範例 (cont.)

- 呼叫 `country_count2` 時，要提供洲名與接收國家數量的變數名稱，執行該 SP 以後，使用者變數就會儲存國家數量

