MySQL 入門

1

林新德

shinder.lin@gmail.com

1. 關聯式資料庫和 SQL

- 關於資料
- 什麼是資料庫
- 資料庫管理系統種類
- 關聯式資料庫
- SQL (Structured Query Language)

1.1 關於資料

伙食採買單據

雞蛋	1盒	100
高麗菜	1顆	120
啤酒	6罐	240
小雞麵	3包	60

一堆採買單據，如何管理？

1. Excel
2. 記帳 App
3. 其它？

如果你想知道：

1. 上個月花費
2. 去年的月平均花費
3. 蔬菜佔月花費的比例
4. 零食花費是否太高

面臨到的問題：

1. 有記錄採買時間？
2. 有依照時間排列？
3. 項目有分類？
4. 同分類如何加總計算？

擴大到公司需求：

1. 公司員工資料
2. 產品資料（分類）
3. 月銷售（月營業額）
4. 客戶資料
5. 訂單
6. 會計資料

1.2 什麼是資料庫

- 廣義：許多資料的集合（資料可能是有整理過、組織化的）。包含：Excel 檔、文字檔（CSV、JSON、XML）等。
- 狹義：資料庫管理系統裡，所存放的資料集合。
- 一般我們提到「資料庫」時，有時指的是「資料庫管理系統」。
- 「資料庫管理系統」是管理資料的「軟體」。

1.3 資料庫管理系統種類

- 1. 關聯式資料庫管理系統
  - Relational database management system (RDBMS)
  - 使用 「結構化查詢語言」Structured Query Language (SQL)
- 2. 非關聯式資料庫管理系統
  - NO-SQL (Not Only SQL)
  - 較為著名:
    - MongoDB (document base)
    - Redis (key-value)
- RDBMS 優點可靠性高、準確性高。
- No-SQL 優點快速、擴充性高。
- 兩者不是二選一，而是相輔相成。

1.4 關聯式資料庫

- 包含多個資料表。
- 資料表和資料表之間，以欄位對應的方式產生關聯。
- 下游的資料表的外鍵，對應到上游資料表的主鍵。
- 常見的關聯式資料庫
  - 主從式架構:
    - Microsoft SQL Server
    - Oracle
    - PostgreSQL
    - MySQL
    - MariaDB
  - 輕量級（檔案式）:
    - Microsoft Access
    - SQLite

1.5 SQL (Structured Query Language)

伴隨關聯式資料庫一起從 1970 年代開始發展至今。

SQL 在 1986 年被 ANSI 標準化，1987 年被 ISO 標準化。

然而，每家 RDBMS 為了額外增加自家產品 SQL 的功能，而加入非標準的語法，進而造成各家 SQL 語法的獨特性。

SQL 依據功能細分：

1. 資料定義语言 Data Definition Language (DDL)
   CREATE, ALTER, DROP, TRUNCATE
2. 資料查詢語言 Data Query Language (DQL)
   SELECT
3. 資料操作语言 Data Manipulation Language (DML)
   INSERT, UPDATE, DELETE
4. 資料控制语言 Data Control Language (DCL)
   GRANT, REVOKE
5. 事務控制語言 Transaction Control Language (TCL)
   COMMIT, ROLLBACK

2. 建立資料庫和資料表

- SQL 關鍵字不區分大小寫，大寫或小寫都可以。
- 每個 SQL 敘述 (statement) 以分號為結束符號。
- 單行註解以 -- 為開頭。
- 多行註解同 C 語言的 /* */ 。

2.1 建立資料庫

```
-- 建立資料庫語法
CREATE DATABASE 資料庫名稱;

/* 使用反引號 (backtick) 用來標示資料庫名稱 ( 非標準 SQL ) */
CREATE DATABASE `my_db_01`;

-- 加上預設的文字編碼 ( 建議設定 )
CREATE DATABASE `my_db_02`
  DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;

-- 如果不存在這個資料庫才建立
CREATE DATABASE
  IF NOT EXISTS `my_db_03`
  DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
```

2.2 使用資料庫

- 在使用任何操作資料表的語法前，必須先「選擇使用」要操作的資料庫。

```
-- 呈現所有資料表的名稱
SHOW DATABASES;

-- 選擇使用某個資料庫
USE 資料庫名稱;
```

2.3 建立資料表

- 建立資料表之前，應先規劃。（之後將討論資料表規劃方式）
- **MySQL** 使用反引號 **(backtick)** 標示
  - 資料庫名稱
  - 資料表名稱
  - 資料欄位名稱
- 若沒有使用到 **MySQL** 的關鍵字或禁用字，可以省略反引號。

```
-- 語法說明，非正確 SQL 語法
CREATE TABLE 資料表名稱 (
  欄位名稱一   資料類型 [是否可為空值] [DEFAULT 預設值],
  欄位名稱二   資料類型 [是否可為空值] [DEFAULT 預設值],
  欄位名稱三   資料類型 [是否可為空值] [DEFAULT 預設值]
);

-- 描述資料表
DESCRIBE 資料表名稱;
DESC     資料表名稱;

-- 刪除資料表
DROP TABLE 資料表名稱;
```

2.4 建立資料表基本用法

```
-- 建立名為 my_friends 的資料表
CREATE TABLE my_friends (
    sid      int,
    name     varchar(100),
    birthday date,
    mobile   varchar(30)
);

-- 查看資料表結構 (structure, schema)
DESCRIBE `my_friends`;

-- 刪除資料表
DROP TABLE `my_friends`;
```

2.5 設定主鍵和自動累增

```
-- 加入主鍵
ALTER TABLE `my_friends`
    ADD PRIMARY KEY (`sid`);

-- 加入主鍵後再變更欄位設定加入 AUTO_INCREMENT
ALTER TABLE `my_friends`
    CHANGE `sid`
    `sid` int NOT NULL AUTO_INCREMENT;
```

2.6 建立資料表同時設定主鍵

```
-- 以比較完整的方式建立資料表
CREATE TABLE my_friends
(
    sid         int           NOT NULL AUTO_INCREMENT,
    name        varchar(100) NOT NULL,
    birthday    date,
    mobile      varchar(30),
    created_at timestamp     NOT NULL DEFAULT CURRENT_TIMESTAMP,
    -- 設定主鍵
    PRIMARY KEY (sid)
) DEFAULT CHARSET=utf8mb4;
```

2.7 常用資料類型

★ int

4-byte 整數：若允許負數，範圍由 -2,147,483,648 至 2,147,483,647。

★ bigint

8-byte 整數。

★ decimal

定點數 (M, D) - (M) 最大為 65 位數 (預設 10)，(D) 小數位數最大為 30 (預設 0)。

★ float

浮點數，範圍為 -3.402823466E+38 至 -1.175494351E-38、0 及 1.175494351E-38 至 3.402823466E+38。

★ date

日期的有效範圍為 1000-01-01 至 9999-12-31。

★ **datetime**

日期與時間組合的有效範圍為 1000-01-01 00:00:00 至 9999-12-31 23:59:59。

★ **timestamp**

時間戳記以自 1970-01-01 00:00:00 UTC 計算的秒數儲存。

目前 MySQL 8.0.37 依然有 2038-01-09 03:14:07 UTC 的限制，可以使用 bigint 存放 timestamp 來處理，或改用 datetime 來儲存。

★ **varchar**

可變長度 (0-65,535) 的字串，實際的最大長度需視資料列大小限制而定。**一定要設定最大長度**。

★ **text**

文字欄位，最大長度為 65,535 (2^16 - 1) 個字元，儲存時會附加 2 個位元組在最前面用來記錄長度。

★ **mediumtext**

文字欄位，最大長度為 16,777,215 (2^24 - 1) 個字元，儲存時會附加 3 個位元組在最前面用來記錄長度。

3. 基本 CRUD

- Create, Read, Update, Delete 簡略的講法。
- 新增、讀取、修改、刪除
- 增刪修（中文變更資料的簡略講法，排除了讀取）

```
-- 查詢所有欄位、所有筆數的資料
SELECT * FROM my_friends;
```

3.1 新增資料

```
-- 新增一筆資料語法

INSERT INTO 資料表名稱 (欄位串列) VALUES (對應到各欄位要存放的值);
```

- 沒有標示在「欄位串列」裡的欄位：
  - 1. 若為主鍵，並設定自動累加時，則自動填入流水號。
  - 2. 若有預設值則填入預設值。
  - 3. 若無設定預設值，但欄位可填入 **NULL** 時，則填入 **NULL**。
  - 4. 若無預設值也不能填入 **NULL** 值，則會發生錯誤。

```
INSERT INTO `my_friends`
    (`name`, `birthday`, `mobile`)
VALUES
    ('林小新', '1995-09-20', '0935666888');
```

- 在新增資料的語法中，「值」通常使用'單引號'標示，也可以使用"雙引號"標示。
- 注意，欄位名稱只能使用反引號標示或者省略反引號，不可以使用'單引號'或"雙引號"標示欄位名稱。

```
INSERT INTO my_friends
    (name, birthday, mobile)
VALUES
    ("李大明", "1993-02-15", "0935777888");
```

```sql
-- 一次新增多筆資料
INSERT INTO `my_friends`
    (`name`, `birthday`, `mobile`)
VALUES
    ('陳小華', '1996-08-21', '0935666777'),
    ('王小花', '1997-07-22', '0935666555'),
    ('丁小雨', '1998-06-23', '0935666333');
```

3.2 刪除資料

- 刪除資料的動作是相對比較少的操作，因為資料刪掉就沒有了。
- 有些網站系統會以某欄位值決定是否顯示，讓一般用戶以為該項目已刪除，但實際上並沒有。這類的作法稱為「假刪除」或「軟刪除」。

```
-- 先查詢資料
SELECT * FROM `my_friends`;

-- 先以主鍵為條件，查詢欲刪除的該筆資料
SELECT * FROM `my_friends` WHERE sid = 1;

-- 再以相同的條件做刪除
DELETE FROM `my_friends` WHERE sid = 1;
```

★★ 若沒有設定 WHERE 子句，將造成所有資料被刪除的悲劇。

3.3 修改資料

```sql
-- 先查詢資料，再依主鍵修改資料
SELECT * FROM `my_friends` WHERE sid = 7;

-- 修改某一筆單一個欄位的值
UPDATE `my_friends`
SET `name`='吳小迪'
WHERE sid = 7;

-- 修改某一筆兩個欄位的值
UPDATE `my_friends`
SET `name`='吳大迪',
    `birthday`='1998-12-24'
WHERE sid = 7;
```

★★ 若沒有設定 WHERE 子句，將造成所有資料被修改的悲劇。

4. 基本查詢

- 下載範例資料表的 SQL, 並在 MySQL Workbench 執行:
- https://github.com/shinder/mmmh57-php/blob/master/db-training/shin01.sql

- 學習 SQL 主要有兩個方向：
  - 1. 資料分析。
  - 2. 建構網站所需的資料表規劃、接收表單資料寫入、資料的呈現。

- 本課程將以網站需求的 SQL 為主要討論內容。

4.1 只查詢某些欄位

- ★ 查詢時，欄位名稱不區分大小寫。
- 一般在命名欄位名稱時，建議使用全小寫 **Snake style** 的方式，同時使用簡短且有說明功能的欄位名稱。

- ★ 在欄位數較多，且資料量較大時，「只查詢有需要的欄位」，可以減少不必要的效能損耗及頻寬花費。

```
-- 只查詢 name 和 birthday 兩個欄位
SELECT name, birthday FROM my_friends;

-- 注意！查詢時，欄位名稱不區分大小寫
SELECT NAME, BIRTHDAY FROM my_friends;
```

4.2 ORDER BY 排序

```sql
-- 以生日欄位做升冪排序
SELECT * FROM my_friends ORDER BY birthday ASC;
SELECT * FROM my_friends ORDER BY birthday;

-- 以生日欄位做降冪排序
SELECT * FROM my_friends ORDER BY birthday DESC;
```

4.3 WHERE 條件查詢

```sql
-- 使用主鍵讀取單筆
SELECT * FROM my_friends WHERE sid = 7;

-- 使用關係運算子
SELECT * FROM my_friends
WHERE birthday >= '1996-01-01';

-- 使用關係運算子和邏輯運算子
SELECT * FROM my_friends
WHERE birthday >= '1996-01-01'
  AND birthday <= '1998-01-01’;

-- 使用 BETWEEN
SELECT * FROM my_friends
WHERE birthday BETWEEN '1996-01-01' AND '1998-01-01';
```

4.4 分頁查詢

```sql
-- 讀取前 3 筆
SELECT * FROM my_friends LIMIT 3;

-- 跳過 2 筆讀取 3 筆
SELECT * FROM my_friends LIMIT 3 OFFSET 2;

-- 跳過 2 筆讀取 3 筆
SELECT * FROM my_friends LIMIT 2, 3;

-- 10 筆為 1 頁
SELECT * FROM address_book LIMIT  0, 10; -- 第 1 頁

SELECT * FROM address_book LIMIT 10, 10; -- 第 2 頁

SELECT * FROM address_book LIMIT 20, 10; -- 第 3 頁
```

4.5 使用運算符號及函式

- MySQL 數值相關函式
- https://dev.mysql.com/doc/refman/8.4/en/numeric-functions.html

```
SELECT
    book_name,
    price                AS 原價,
    ROUND(price * 1.05) AS 含稅價格 -- 使用欄位別名
FROM products LIMIT 0, 10;
```

- MySQL 日期和時間相關函式
- https://dev.mysql.com/doc/refman/8.4/en/date-and-time-functions.html

```
-- DATE_FORMAT() 依格式輸出日期
-- NOW() 取得當下的時間
SELECT DATE_FORMAT(NOW(), "%c 月 %e 日 %W") AS 今天;

SELECT name, birthday,
       DATE_FORMAT(birthday, "%w %W") AS 星期幾
FROM address_book LIMIT 0, 10;

-- DATEDIFF() 兩個日期相差幾天
-- FLOOR() 正數時去除小數部份
-- YEAR() 取得西元年
SELECT ab_id, name, birthday,
    FLOOR(DATEDIFF(NOW(), birthday)/365) AS 年齡,
    YEAR(NOW())- YEAR(birthday)          AS 年齡2
FROM address_book LIMIT 0, 10;
```

- MySQL 字串相關函式
- https://dev.mysql.com/doc/refman/8.4/en/string-functions.html

```
-- CONCAT() 字串串接
SELECT
    ab_id,
    CONCAT(name, ': ', mobile) AS 連絡資訊
FROM address_book LIMIT 0, 10;

-- CHAR_LENGTH() 是取得 utf8 字元長度
-- LENGTH() 是取得使用 byte 數量
SELECT author, CHAR_LENGTH(author), LENGTH(author) FROM products;
```

5. 鍵和索引 (Keys, Indexes)

Key 指的是欄位的特殊性質，以下為常見的鍵。

主鍵 (Primary Key)：

用來識別資料項目的主要欄位（可以是多個欄位），必須每一筆的主鍵都不同，且不為 NULL。

唯一鍵 (Unique Key)：

資料不能重複，但可以為 NULL。

索引鍵 (Index)：

資料可以重複，也可以為 NULL。可以加快查詢，但佔用比較多的磁碟空間。

組合鍵 (Composite Key)：

可以使用兩個欄位或以上，組成「主鍵」、「唯一鍵」或「索引鍵」。

外鍵 (Foreign Key)：

下游的資料表（子表）用來對應到上游資料表（父表）的主鍵，用以描述兩資料表的關係。

★★ 一般提到的「索引」包含「主鍵」、「唯一鍵」和「索引鍵」。

5.1 查看索引的設定

- 查看索引在 MySQL 有以下兩種方式，但不包含使用 DESCRIBE。

```
-- 查看資料表的所有索引
SHOW INDEXES IN `my_friends`;

-- 查看完整的資料表定義，包含索引的設定
SHOW CREATE TABLE my_friends;
```

5.2 索引的設定和解除

```
-- 移除主鍵前應該先取消 AUTO_INCREMENT
ALTER TABLE `my_friends` CHANGE `sid` `sid` INT NOT NULL;
ALTER TABLE `my_friends` DROP PRIMARY KEY;

-- 加入唯一鍵 (自動決定唯一鍵名稱)
ALTER TABLE `my_friends` ADD UNIQUE(`name`);
-- 加入唯一鍵 (指定唯一鍵名稱)
ALTER TABLE `my_friends` ADD UNIQUE KEY `unique_name`(`name`);
-- 移除唯一鍵
ALTER TABLE `my_friends` DROP INDEX `unique_name`;

-- 加入索引鍵 (自動決定索引鍵名稱)
ALTER TABLE `my_friends` ADD INDEX(`name`);
-- 加入索引鍵 (指定索引鍵名稱)
ALTER TABLE `my_friends` ADD KEY `my_name`(`name`);
-- 移除索引鍵
ALTER TABLE `my_friends` DROP INDEX `my_name`;
```

5.3 設定外鍵

- 外鍵是一種為了讓資料一致的設定。
- 有沒有設定外鍵，和能不能使用 JOIN 合併查詢沒有直接的關係。
- 設定外鍵，除了宣告子表和父表的關係，同時也約定了「當父表對應的資料〖修改〗或〖刪除〗時，子表的資料要如何因應」。

```
-- 建立資料表時，同時設定外鍵
CREATE TABLE `orders2` (
    `order_id`    int       NOT NULL AUTO_INCREMENT,
    `member_id`   int       NOT NULL,
    `amount`      int       NOT NULL,
    `ordered_at` datetime NOT NULL,
    PRIMARY KEY (`order_id`),
    KEY           `member_id` (`member_id`),
    FOREIGN KEY (`member_id`) REFERENCES `members` (`member_id`)
);
```

```sql
-- 查看完整的資料表定義
SHOW CREATE TABLE orders2;

-- 移除外鍵
ALTER TABLE orders2
    DROP FOREIGN KEY `orders2_ibfk_1`;

-- 指定名稱設定外鍵
ALTER TABLE orders2
    ADD FOREIGN KEY `自訂外鍵名稱` (`member_id`)
    REFERENCES `members` (`member_id`);

-- 移除外鍵
ALTER TABLE orders2
    DROP FOREIGN KEY `自訂外鍵名稱`;
```

外鍵的協議針對兩個操作：UPDATE 和 DELETE。

而協議裡分成 4 種模式，但實際上只有 3 種，RESTRICT 和 NO ACTION 行為是一樣的。

ON DELETE 模式：

RESTRICT（限制）：預設模式。當刪除父表資料時，若子表有對應的資料，則不允許刪除。

CASCADE（逐層傳遞）：當刪除父表資料時，若子表有對應的資料，則一起刪除。

SET NULL：當刪除父表資料時，若子表有對應的資料，則子表該筆資料保留，而外鍵欄設定為 NULL。

NO ACTION：與 RESTRICT 相同。

ON UPDATE 模式：

RESTRICT（限制）：預設模式。當變更父表主鍵欄位值時，若子表有對應的資料，則不允許變更。

CASCADE（逐層傳遞）：當變更父表主鍵欄位值時，若子表有對應的資料，則子表該資料外鍵一起變更。

SET NULL：當變更父表主鍵欄位值時，若子表有對應的資料，則子表該資料外鍵設定為 NULL。

NO ACTION：與 RESTRICT 相同。

```sql
-- 指定外鍵的協議模式
ALTER TABLE `orders2`
    ADD FOREIGN KEY `orders2_fk_member`(`member_id`)
    REFERENCES `members`(`member_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE;
```

6. 深入條件篩選

- **NULL** 查詢
- 日期範圍
- 是否在數列裡
- 模糊查詢

6.1 NULL 查詢

- **NULL** 表示是「空」的，或未決定的，但不是數值 0 也不是空字串。

```
-- 新增一筆包含空值的資料
INSERT INTO my_friends (name, birthday, mobile)
VALUES ("蘇小丙", NULL, "0935777858");

-- IS NULL 判斷是否為空值，不可以使用 =
SELECT * FROM my_friends WHERE birthday IS NULL;

-- IS NOT NULL 判斷是否不為空值，不可以使用 !=
SELECT * FROM my_friends WHERE birthday IS NOT NULL;

-- 下式不會包含 birthday 為 NULL 的資料
SELECT * FROM my_friends WHERE
        birthday >= '1996-01-01' OR birthday <= '1996-01-01';
```

- 如果要以某特定值來取代 **NULL**，可以使用 **COALESCE()** 函式。

```
SELECT
    *,
    COALESCE(birthday, '不知道')
FROM my_friends;
```

6.2 日期範圍

- 使用 **DATE_SUB()** 或 **DATE_ADD()** 取得該日期的前後範圍，單位可以是 **DAY**、**MONTH**、**YEAR**。
- 使用情境如，訂購後七天尚未繳款的訂單查詢。

```
SELECT *,
       DATE_ADD(birthday, INTERVAL 1 MONTH) AS 滿月
FROM my_friends;
```

6.3 是否在數列裡

```sql
-- 欄位名稱 IN (集合值): 包含在集合裡的資料
SELECT product_id, book_name, price
FROM products
WHERE price IN (450, 500, 550);

-- 欄位名稱 NOT IN (集合值): 不包含在集合裡的資料
SELECT product_id, book_name, category_id
FROM products
WHERE category_id NOT IN (1, 3);
```

6.4 模糊查詢

```
-- % 為萬用字元表示「沒有字元」或「一個以上任何字元」
-- _ 為萬用字元表示一個任何字元
SELECT *
FROM products
WHERE book_name LIKE '%java%';

-- 使用 Regular Expression
SELECT book_name
FROM products
WHERE book_name REGEXP '[0-9]{2,}';

SELECT book_name
FROM products
WHERE book_name REGEXP '\\d{2,}';
```

7. 彙總函式 (Aggregate Functions)

- 彙總函式會根據一組值（欄位）來執行計算，並傳回單一值，經常搭配 SELECT 敘述的 GROUP BY 子句使用。
- GROUP BY: 以某欄位中相同的值為一組的概念。
- 彙總函式會忽略 NULL 值。
- 常用的彙總函式：
  - COUNT(): 計算數量
  - AVG(): 求平均值
  - MAX(): 取最大值
  - MIN(): 取最小值
  - SUM(): 加總
- 以下連結為 MySQL 彙總函式的官方文件。
- https://dev.mysql.com/doc/refman/8.4/en/aggregate-functions.html

7.1 DISTINCT 和 GROUP BY 的差別

- SELECT DISTINCT 的功能是去除重複的值。
- GROUP BY 則有群組起來的意思，經常搭配「彙總函式」使用。

```
-- 去除重複的值
SELECT DISTINCT category_id
FROM products;

-- 相同值的群組起來變成一項
SELECT category_id
FROM products
GROUP BY category_id;
```

7.2 彙總函式搭配 GROUP BY

```sql
-- 計算總數，注意計數的欄位，若裡面有 NULL 則不列入計算
SELECT COUNT(sid), COUNT(1), COUNT(birthday) FROM my_friends;

-- 依 category_id 分組，計算數量
SELECT category_id, COUNT(1) FROM products
GROUP BY category_id;

-- 依年次，計算人數
SELECT YEAR (birthday) year, COUNT(1) 數量
FROM address_book
GROUP BY year ORDER BY year;

-- 依年次和月份，計算人數
SELECT YEAR (birthday) year, MONTH(birthday) month, COUNT(1) 數量
FROM address_book
GROUP BY year, month ORDER BY year, month;
```

```sql
-- 求平均值
SELECT AVG(price) FROM products;

-- 依 category_id 分組，求平均值
SELECT category_id, AVG(price) FROM products
GROUP BY category_id;

-- 取最大值和最小值
SELECT MAX(price), MIN(price) FROM products;

-- 依 category_id 分組，取最大值和最小值
SELECT category_id, MAX(price), MIN(price)
FROM products GROUP BY category_id;

-- 加總
SELECT SUM(price) FROM products;

-- 依 category_id 分組，加總
SELECT category_id, SUM(price)
FROM products
GROUP BY category_id;
```

7.3 HAVING

- 彙總函式計算的欄位不能使用 **WHERE** 判斷，應該使用 **HAVING**，而且要放在 **GROUP BY** 之後。

```sql
-- 求出生人數大於 65 人的年份
SELECT
    YEAR(birthday) year,
    COUNT(1) num
FROM address_book
-- WHERE num > 65 -- 錯誤的作法
GROUP BY year
HAVING num > 65
ORDER BY year;
```

```sql
-- 經過轉換的欄位也是要在 HAVING 判斷
SELECT
    YEAR(birthday) year,
    COUNT(1) num
FROM address_book
GROUP BY year
HAVING year > 1995 AND num < 65
ORDER BY year;
```

8. CASE 運算式

- CASE 運算式使用於條件判斷，類似一般程式語言的 if/else 和 switch/case。
- 以 CASE 開頭，END 結尾，在 END 後通常會設定別名。
- WHEN 和 THEN 是成對的。
- WHEN 後放的是條件式，THEN 後是條件成立時對應的值。
- ELSE 放在所有 WHEN/THEN 之後，表示之前的條件都沒有吻合時，欲回應的值。

```sql
-- 以 CASE 將價格分成「高、中、低」
SELECT product_id, book_name, price,
       CASE
           WHEN price < 500 THEN '低'
           WHEN price < 600 THEN '中'
           ELSE '高'
       END AS 價位
FROM products;

-- 依「高、中、低」分類群組計算數量
SELECT
    CASE
       WHEN price < 500 THEN '低'
       WHEN price < 600 THEN '中'
       ELSE '高'
    END  AS 價位,
   COUNT(1) AS 數量
FROM products
GROUP BY 價位;
```

9. 合併查詢 (JOIN)

- 表和表之間的關聯
- Cross Join 交叉連結
- Inner Join 內部連結
- 合併多張表
- Left Outer Join 左側外部連結
- Self Join 自我連結

9.1 表和表之間的關聯

表和表之間可以使用「外鍵」來描述。

某表的外鍵通常是用來對應到另一張表的「主鍵」，表示此表是另一張表的「子表」，也就是有附屬的關係。例如，「訂單名細」為「訂單」的子表。

表和表之間的資料項，會因為不同關係也有不同的對應數：

1. 一對一

   例如，旅遊公司的司機只負責管理一台遊覽車時，司機資料表和遊覽車資料表間的關係。

2. 一對多

   例如，一個分類項目可以包含很多產品。

3. 多對多

   例如，商品和標籤的關係。一個商品可以被標示多個標籤；一個標籤也可以用來標示多個商品。

   一個會員可以加入多個商品到喜愛清單；一個商品也可以被許多會員喜愛。

9.2 Cross Join 交叉連結

- 第一張表的每個項目，皆對應到第二張表的每個項目一次。
- 很少需要用到 **CROSS JOIN**，但藉由交叉連結可以看出 **DB** 底層做 **JOIN** 大概的方式。

```
SELECT * FROM categories JOIN products;

-- 結果同 CROSS JOIN
SELECT * FROM categories, products;
```

9.3 Inner Join 內部連結

- 下游的表（child table）的外鍵要對應到上游的表（parent table）的主鍵。
- 語法上，每個 JOIN 都必須有對應的 ON。ON 關鍵字後面接條件式，用來比對外鍵和主鍵。

```
-- 商品歸屬的分類
SELECT p.product_id, p.book_name, c.name AS category_name
FROM products AS p
         INNER JOIN categories AS c
                    ON p.category_id = c.category_id;

-- Inner Join 時，兩張表的順序沒有差別
SELECT p.product_id, p.book_name, c.name category_name
FROM categories c
         JOIN products p
              ON p.category_id = c.category_id;
```

9.4 合併多張表

```
-- 訂單、會員：查看訂單的會員
SELECT o.*,
       m.email    訂購人電郵,
       m.nickname 訂購人
FROM orders o
         JOIN members m
              ON o.member_id = m.member_id;
```

```sql
-- 訂單、會員、訂單明細
SELECT o.order_id,
       m.email      訂購人電郵,
       m.nickname   訂購人,
       od.quantity 數量,
       od.product_id
FROM orders o
         JOIN members m
              ON o.member_id = m.member_id
         JOIN order_details od
              ON o.order_id = od.order_id;
```

```sql
-- 訂單、會員、訂單明細、商品
-- 指定某筆訂單
SELECT o.order_id,
       m.email      訂購人電郵,
       m.nickname   訂購人,
       od.quantity 數量,
       p.book_name 商品名稱
FROM orders o
         JOIN members m
              ON o.member_id = m.member_id
         JOIN order_details od
              ON o.order_id = od.order_id
         JOIN products p
              ON od.product_id = p.product_id
WHERE o.order_id = 10;
```

9.5 Left Outer Join 左側外部連結

- 以左側的表格（第一張表）為主要的表格，主表的所有資料都必須出現，沒有對應到第二張表的部份以 **NULL** 為內容。

```
-- products 裡的項目都必須出現
SELECT p.product_id, p.book_name, c.name category_name
FROM products p
         LEFT OUTER JOIN categories c
                         ON p.category_id = c.category_id;

-- categories 裡的項目都必須出現
SELECT p.product_id, p.book_name, c.name category_name
FROM categories c
         LEFT JOIN products p
                   ON p.category_id = c.category_id;
```

9.6 Self Join 自我連結

★ 如何將樹狀的結構轉換成資料表？

```
-- 查詢分類上層的分類名稱
SELECT c1.*, c2.name parent_name
FROM categories c1
         LEFT JOIN categories c2
                   ON c1.parent_id = c2.category_id;
```

10. Union 聯集

- Join 是兩張表的欄位並排做查詢；Union 則是兩張衍生表串接起來。
- 使用 Union 時，兩張衍生表的欄位數必須一樣，欄位名稱以第一張表為主。
- Union 也可以換成 Union all 使用，兩者的差異在於 Union 會去掉重複的部份，Union all 則不會。

```
SELECT 1 my_type, category_id, name FROM categories
UNION
SELECT 2, product_id, book_name FROM products;
```

11. Subquery 子查詢

- 某個查詢是另一個查詢的一部份，稱為「子查詢」。

- 子查詢大概可以區分成三種形式：
  - 1. 子查詢結果（衍生表）只有一欄一列，視為一個值。
  - 2. 子查詢結果為一欄多列，視為值的列表（list）。
  - 3. 子查詢結果為多欄多列，視為一般資料表。

- ★ 衍生資料表（Derived Table）：某個查詢的結果。

11.1 衍生表視為一個值

```sql
-- 查詢價格大於平均值的商品
SELECT product_id, book_name, price
FROM products
WHERE price > (
    SELECT AVG(price) FROM products
);
```

11.2 衍生表視為一列表

```sql
-- 會員 3 曾經買過的商品
SELECT product_id, book_name, price
FROM products
WHERE product_id IN (
    SELECT DISTINCT product_id
    FROM orders o
        JOIN order_details od
            ON o.order_id = od.order_id
    WHERE o.member_id = 3
);
```

11.3 衍生表視為一般資料表

```
-- 計算每年出生的人數
-- 然後再算平均值
SELECT AVG(年出生數) 平均年出生數 FROM
    (
        SELECT
            YEAR(birthday) year,
            COUNT(1) 年出生數
        FROM address_book
        GROUP BY year
    ) nt;
```

```sql
-- 在商品列表分頁中呈現，已登入會員加至最愛的項目
SELECT
    p.product_id, p.book_name,
    CASE
        WHEN mf.product_id > 0 THEN 1
        ELSE 0
    END fav
FROM products p
LEFT JOIN (
    SELECT product_id FROM favorites WHERE member_id = 7
) mf
    ON p.product_id = mf.product_id;
```

12. Normalization 正規化

- 正規化是數據庫設計中的一個重要過程，用於組織數據以減少冗餘和依賴性。
- 主要是減少不必要的重複資料記錄。重複的資料會浪費磁碟空間，並產生維護方面的問題。
- 其中包括根據設計來保護數據的規則，建立數據表和建立這些數據表之間的關聯性，並藉由消除備援和不一致的相依性，讓資料庫更有彈性。
- 1NF：單純化資料。
- 2NF：消除部分依賴性，即每個非主鍵屬性完全依賴於主鍵。
- 3NF：消除傳遞依賴性，即每個非主鍵屬性僅直接依賴於主鍵，而不依賴於其他非主鍵屬性。

12.1 第一正規化

第一正規化（1NF, The First Normal Form）：

每個欄位 (Cell) 的值都只能是單一值 (Atomic Value)。

不符合 1NF

訂單編號	訂單日期	訂購人	訂購人電郵	商品名稱	商品單價	商品數量
5	2016-03-25 12:19:05	大明	ming@test.com	Web CSS網頁設計大全	580	1
				Java互動網站實作 -JSP與資料庫	620	2
10	2016-06-01 11:15:53	小新	shin@test.com	圖解C++程式設計	560	1
				圖解資料結構-使用JAVA	420	1
				Visual C# 2008網路遊戲程式設計	480	1
11	2017-10-03 13:49:20	大明	ming@test.com	Linux Device Driver Programming 驅動程式設計	690	2
				ASP.NET 3.5初學指引-使用Visual Basic 2008	650	1
				Visual C# 2008網路遊戲程式設計	480	2

```sql
-- 呈現的表格未符合第一正規化
SELECT
    o.order_id 訂單編號,
    o.ordered_at 訂單日期,
    m.nickname 訂購人,
    m.email 訂購人電郵,
    GROUP_CONCAT(p.book_name SEPARATOR "\n") 商品名稱,
    GROUP_CONCAT(p.price  SEPARATOR "\n") 商品單價,
    GROUP_CONCAT(d.quantity  SEPARATOR "\n") 商品數量
FROM orders o
         JOIN order_details d ON o.order_id=d.order_id
         JOIN products p ON d.product_id=p.product_id
         JOIN members m ON o.member_id=m.member_id
GROUP BY o.order_id;
```

12.2 第二正規化

- 必須符合第一正規化。
- 每個非主鍵屬性都完全依賴於主鍵：也就是說，不能有部分依賴性。部分依賴性是指非主鍵屬性只依賴於主鍵的一部分。如果主鍵是由多個欄位組成的複合鍵，則任何非主鍵欄位不能只依賴於其中的一部分。
- 例如，表格 Enrollment（註冊）的欄位：StudentID, CourseID, StudentName, CourseName。
- 在這個表格中，主鍵是組合鍵（StudentID, CourseID）。如果 StudentName 只依賴於 StudentID，而 CourseName 只依賴於 CourseID，那麼這個表格就違反了 2NF。

12.3 第三正規化

- 必須符合第二正規化。
- 每個非主鍵屬性都僅依賴於主鍵：即不存在傳遞依賴性。傳遞依賴性是指非主鍵屬性依賴於其他非主鍵屬性，而這些非主鍵屬性又依賴於主鍵。
- 例如，表格 Employee（員工）的欄位：EmployeeID, DepartmentID, DepartmentName。
- 如果 DepartmentName 依賴於 DepartmentID，並且 DepartmentID 依賴於 EmployeeID，那麼 DepartmentName 也間接依賴於 EmployeeID，這就是一種傳遞依賴性，違反了 3NF。

12.4 正規化練習

訂單編號	訂單日期	訂購人	訂購人電郵
5	2016-03-25 12:19:05	大明	ming@test.com
10	2016-06-01 11:15:53	小新	shin@test.com
11	2017-10-03 13:49:20	大明	ming@test.com

明細流水號	訂單編號	商品名稱	商品單價	商品數量
4	5	Web CSS網頁設計大全	580	1
5	5	Java互動網站實作 –JSP與資料庫	620	2
9	10	圖解C++程式設計	560	1
10	10	圖解資料結構–使用JAVA	420	1
11	10	Visual C# 2008網路遊戲程式設計	480	1
12	11	Linux Device Driver Programming 驅動程式設計	690	2
13	11	ASP.NET 3.5初學指引–使用Visual Basic 2008	650	1
14	11	Visual C# 2008網路遊戲程式設計	480	2

13. Transactions 交易

- 在關聯式資料庫管理系統（RDBMS）中，交易（Transaction）是一個或多個 SQL 操作組成的邏輯單元，這些操作作為一個整體被執行。交易確保數據庫的數據在操作過程中保持一致性和完整性。交易有以下幾個特性，通常簡稱為 ACID 特性：

  - 原子性（Atomicity）：交易中的所有操作若不是「全部完成」，就是失敗並回滾（rollback）。這意味著交易是不可分割的最小單元，不能只完成其中的一部分。

  - 一致性（Consistency）：交易在完成後，數據庫必須從一個一致的狀態轉換到另一個一致的狀態。這意味著交易應該遵守所有的數據完整性約束和規則。

  - 隔離性（Isolation）：交易的操作應該相互隔離，彼此不應互相影響。即使多個交易同時執行，它們的執行結果應該與按某一順序依次執行這些交易的結果一致。

  - 持久性（Durability）：一旦交易完成並提交（commit），其對數據的改變應該永久地保存在數據庫中，無論後續是否發生系統崩潰等問題。

在 一般 RDBMS，交易的基本操作流程：

1. 開始交易（Start Transaction）：標記交易開始。某些資料庫會在執行第一個 SQL 語句時自動開始交易。
2. 回滾（Rollback）：在還沒有完成交易內要完成的所有敘述時，可能因為發生例外，進而取消交易中的所有操作，將數據庫恢復到交易開始前的狀態。
3. 提交（Commit）：將交易中的所有操作結果永久保存到數據庫中，完成整個交易動作。

交易的使用情境範例：

1. 金融帳戶兩個帳戶轉帳過程。
2. 電商將購物車資料轉移到訂單資料。
3. 大量新增資料，交易可快速完成。

```
-- 變更敘述結束符號為 // 主要是為了建立預存程序
DELIMITER //
-- 建立預存程序 (Stored Procedures)
CREATE PROCEDURE TA_TRY(IN p1_id INT, IN p2_id INT)
BEGIN
    START TRANSACTION;
    -- 為了測試，先將金額加入
    UPDATE products SET price=price+500 WHERE product_id=p2_id;
    -- 在「預存程序」裡才能使用 IF/ELSE
    IF (SELECT price>=500 FROM products WHERE product_id=p1_id) THEN
        -- 餘額足夠時
        UPDATE products SET price=price-500 WHERE product_id=p1_id;
        -- 另外記錄過程的敘述
        COMMIT; -- 提交
    ELSE
        SELECT '餘額不足' message;
        ROLLBACK;  -- 回復交易前狀態
    END IF;
END //
-- 回復敘述結束符號為 ;
DELIMITER ;
-- 呼叫時使用 CALL TA_TRY(23, 22);
```